**Display LCD SPI**

**Objetivos:**

* Entender a comunicação entre o display e FPGA via SPI
* Desenhar formas livres no display
* Serializar uma imagem e mostrar no display

**Funcionamento**

A comunicação SPI é simples, se trata de um protocolo mestre-escravo, onde o mestre inicia a comunicação. O protocolo tem 4 modos, duas variações para borda de subida ou descida e estado inativo do SCK em zero ou um. Com as duas condições são 4 modos. O princípio básico do SPI é o registrador de deslocamento, a cada borda de clock os bits entram no registrador e a cada 8 bits o dado pode ser usado pelo mestre ou escravo. Os dados do mestre saem do MOSI para o escravo e os dados do escravo vão para o mestre pelo MISO.

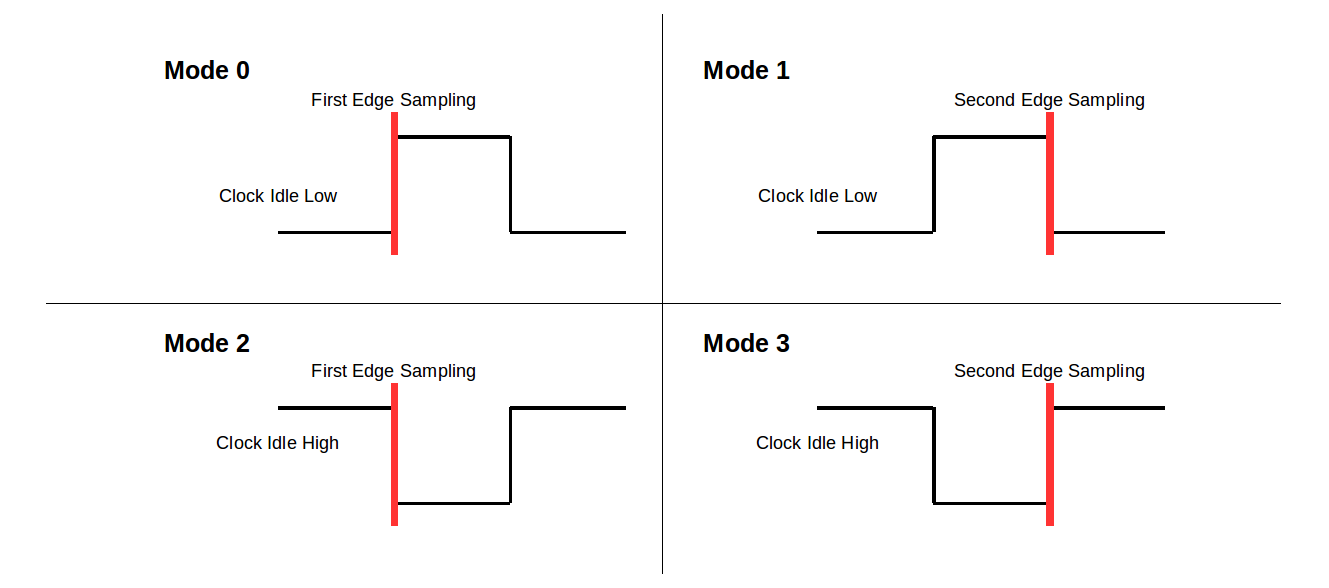


Figura 1 – Modos SPI.

O modelo do display é ILI9341, todos seus comandos podem ser encontrados na *datasheet* porém, o mais interessante deles é o de mudar a quantidade de bits por pixel, fazendo uma troca entre velocidade de transferência e resolução da imagem. Além do protocolo SPI, usa-se uma linha adicional, chamada DC para determinar se o byte enviado é comando ou dado/argumento. O display funciona no modo 0 dos SPI, onde a linha SCK fica em estado lógico baixo em inatividade e recebe os bits na borda de subida. Cada argumento do comando é um dado portanto, os comandos são definidos como constantes de 9 bits, onde o primeiro determina se é comando ou argumento.

A primeira linha da arquitetura com *o\_sck* é um seletor para desligar o clock de saída no estado de parado. A segunda define um vetor de 24 bits para ser enviado ao display. Segundo o protocolo SPI devemos enviar 8 bits mas, da forma que configuramos o display está com a maior resolução de pixels (RGB 666), onde só 6 dos 8 bits são relevantes portanto é definido esse vetor com todos os bits, que será iterado posteriormente para enviar pixel a pixel.

Este módulo possui um divisor que normalmente é definido para um e pode ser alterado na *entity.* Pode se colocar um PLL no diagrama de blocos antes do clock de entrada deste módulo para acelerar a escrita de pixels.

Antes do envio dos comandos há o processo desenhar, onde é possível alterar a cor a ser enviada para cada posição linha x coluna que acontece na borda de descida. Aqui será possível usar sua lógica própria para desenhar sem alterar muitas coisas. Como o modo SPI utilizado é zero, então temos que alterar os bits na borda de descida, pois no próximo clock de subida os dados já estarão na saída.

A máquina de estados possui quatro estados: parado, comandos, comando de escrever e pixels. A partir do estado parado, caso a entrada *i\_enable* seja zero, passamos para o próximo estado, esse estado define os padrões dos sinais e prepara para serem usados nos próximos estágios. É uma boa prática ter um estado de reset ou estado padrão. O estado comando itera sobre o *array* de vetores de bits de 9 bits. Saindo do estado parado para comando, põe o CS em zero, define o *sck\_enable* para um colocando o clock para a saída no SCK. Esse estado inicia definindo o comando a enviar conforme o índice de comandos definidos em parado, será iterado do maior número para o menor 13 para 0. Itera-se a cada borda de clock o bit no comando definido e põe para saída em MOSI. A cada 8 bits alteramos o índice do comando e ao chegar em zero vamos para o próximo comando. O comando é enviar é o vetor de 7 a 0 da constante dos comandos. O bit 8 vai sempre para a linha DC, indicando para o display se é dado ou comando.

O estado de *comando\_escrever* envia o comando para escrever os pixels em si, itera-se o valor em RGB, a cada borda de clock sobre os bits e envia para MOSI, mantendo CS e SCK ativos. Daí vamos para o estado de enviar os pixels, serão 320x240 pixels, onde cada um é enviado como tendo 8 bits, mas só 6 serão usados. Cada pixel vai de 23 a zero, são 24 bits enviados por vez para o MOSI, a cada pixel incrementa-se a linha e a cada linha incrementa-se a coluna. O valor em RGB já foi mudado no processo anterior. Ao final retorna-se ao estado do comando de escrever pixels e retorna para os pixels, sempre nessa alternância. A forma que o display funciona, exige que após escrever em toda a tela, é preciso enviar o comando de escrever novamente para escrever na memória de vídeo.

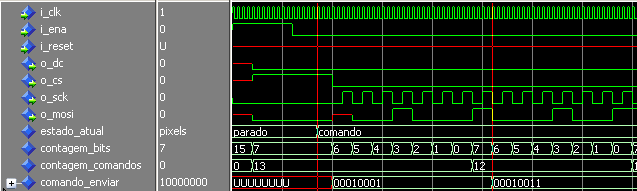


Figura 2 – Estados inicial e de comandos.

Concluindo, no modo parado define-se alguns sinais a serem usados e daí vai para o estado de comandos. Nele itera-se a cada borda de clock bit a bit, e a cada 8 bits muda-se o índice do comando. Nota-se que o SCK tem largura maior que o *i\_clk* pois temos um divisor por 2, a cada borda de descida trocamos o bit e a cada borda de subida o display lê esse bit.

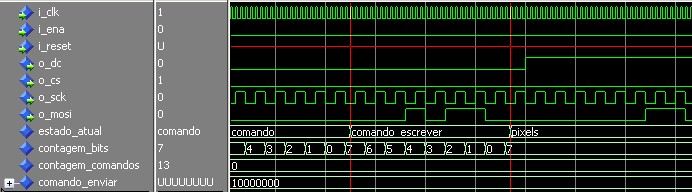


Figura 3 – Estados de escrever na memória e enviar pixels.

Após o índice de comando zerar junto com os bits a enviar, inicia-se o comando de escrever, seguindo a mesma lógica, mas separado dos outros comandos para permitir alternância entre enviar o comando de escrever e a escrita de pixels.

**Procedimentos:**

1. Copie o código em anexo, rode e acompanhe a explicação do professor
2. Crie um quadrado ou retângulo dentro do processo desenhar.
3. Mova um quadrado dentro da dela, seja alterando linhas, colunas ou ambos.
4. Serialize uma imagem conforme o slide e use substitua o *process* de desenho por essas linhas, não esqueça de criar uma constante escala = 4

R <= mem\_data(17 downto 12);

G <= mem\_data(11 downto 6);

B <= mem\_data(5 downto 0);

mem\_addr <= std\_logic\_vector(to\_unsigned(((240/escala) \* linha/escala)) + (coluna/escala), 15));

**Pinout**

Header GPIO1, siga à risca de cima para baixo ou siga o manual e defina conforme desejar, respeitando as alimentações 5V e 3.3V nos lugares corretos. Os números são apenas para facilitar a contagem e localização.

|  |  |
| --- | --- |
| H12 - DC | 1 |
| H14 - CS | 2 |
| E14 - MOSI | 3 |
| F15 - MISO | 4 |
| F12 - SCLK | 5 |
| 5V - LED | GND |
| 1 | ... |
| 2 | ... |
| 3 | ... |
| 4 | ... |
| 5 | ... |
| 6 | ... |
| 7 | ... |
| 8 | ... |
| 3.3V - VCC | ... |

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity display is

    port(

        i\_clk, i\_miso : in std\_logic;

        o\_cs, o\_sck, o\_mosi, o\_dc : out std\_logic;

        o\_leds : out std\_logic\_vector(7 downto 0);

        mem\_addr : out std\_logic\_vector(14 downto 0);

        mem\_data : in std\_logic\_vector(17 downto 0)

    );

end entity;

architecture rtl of display is

    type t\_commands is array (natural range<>) of std\_logic\_vector(8 downto 0);

    constant sequencia\_de\_comandos : t\_commands (9 downto 0) := (

    "000010001", -- sleep out

    "000010011", -- normal mode on

    "000100000", -- inversion off

    "001010001", -- brightness

    "111111111", -- max brightness

    "000100110", -- gama

    "100000001", -- curva de gama 1

    "000111010", -- formato de pixel

    "101100110", -- formato 18 bits

    "000101001"); -- display on

    type estados is (parado, comando, comando\_escrever, pixels);

    signal estado\_atual, proximo\_estado : estados := parado;

    signal coluna, deslocamento\_coluna : integer range 0 to 320 := 0;

    signal linha, deslocamento\_linha : integer range 0 to 320 := 0;

    signal pixel : integer range 0 to 24 := 0;

    signal sck\_enable : std\_logic := '0';

    signal R, G, B : std\_logic\_vector(5 downto 0);

    signal RGB : std\_logic\_vector (0 to 23);

    constant constante\_comando\_escrever : std\_logic\_vector (7 downto 0) := "00101100";

begin

    o\_sck <= i\_clk when sck\_enable = '1' else '0';

    RGB <= R & "11" & G & "11" & B & "11";

    desenhar : process(linha, coluna, deslocamento\_linha, deslocamento\_coluna, i\_clk, pixel)

        variable conta\_tempo : integer := 0;

        variable conta\_linha : integer := 0;

    begin

        if falling\_edge(i\_clk) then

            if conta\_tempo < 5e6 then

                conta\_tempo := conta\_tempo + 1;

            else

                conta\_tempo := 0;

                if conta\_linha < 319 then

                    conta\_linha := conta\_linha + 20;

                else

                conta\_linha := 0;

                end if;

            end if;

            if (linha > conta\_linha) and (linha < conta\_linha + 20) then

                R <= "111111";

                G <= "000000";

                B <= "000000";

            else

                R <= "000000";

                G <= "000000";

                B <= "000000";

            end if;

        end if;

    end process;

    passador\_maquina : process(i\_clk, estado\_atual, proximo\_estado)

    begin

        if rising\_edge(i\_clk) then

            estado\_atual <= proximo\_estado;

        end if;

    end process;

    comandos : process(i\_clk, estado\_atual, linha, coluna, pixel, i\_miso)

        variable contagem\_bits : integer range 7 downto 0;

        variable contagem\_comandos : integer range 0 to sequencia\_de\_comandos'length;

        variable comando\_enviar : std\_logic\_vector (8 downto 0);

    begin

        if falling\_edge(i\_clk) then

            case estado\_atual is

            when parado =>

                o\_mosi <= '0';

                o\_leds <= not "00000000";

                o\_dc <= '0';

                o\_cs <= '1';

                sck\_enable <= '0';

                contagem\_bits := 7;

                contagem\_comandos := sequencia\_de\_comandos'length - 1;

                pixel <= 0;

                coluna <= 0;

                linha <= 0;

                if i\_ena = '0' then

                    proximo\_estado <= comando;

                end if;

            when comando =>

                o\_cs <= '0';

                sck\_enable <= '1';

                o\_mosi <= comando\_enviar(contagem\_bits);

                comando\_enviar := sequencia\_de\_comandos(contagem\_comandos);

                if comando\_enviar(8) = '1' then

                    o\_dc <= '1';

                else

                    o\_dc <= '0';

                end if;

                if contagem\_bits > 0 then

                    contagem\_bits := contagem\_bits - 1;

                else

                    contagem\_bits := 7;

                    if contagem\_comandos > 0 then

                        contagem\_comandos := contagem\_comandos - 1;

                    else

                        proximo\_estado <=  comando\_escrever;

                    end if;

                end if;

            when comando\_escrever =>

                o\_dc <= '0';

                o\_cs <= '0';

                sck\_enable <= '1';

                o\_mosi <= constante\_comando\_escrever(contagem\_bits);

                if contagem\_bits > 0 then

                    contagem\_bits := contagem\_bits - 1;

                else

                    contagem\_bits := 7;

                    proximo\_estado <=  pixels;

                end if;

            when pixels =>

                sck\_enable <= '1';

                o\_dc <= '1';

                o\_cs <= '0';

                o\_mosi <= RGB(pixel);

                if pixel = 23 then

                    pixel <= 0;

                    if coluna = 239 then

                        coluna <= 0;

                        if linha = 319 then

                            linha <= 0;

                            proximo\_estado <=  comando\_escrever;

                        else

                            linha <= linha + 1;

                        end if;

                    else

                        coluna <= coluna + 1;

                    end if;

                else

                    pixel <= pixel + 1;

                end if;

            end case;

        end if;

    end process;

end rtl;