Documento de Arquitetura

Interface E/S - Problema 2

Universidade Estadual de Feira de Santana

Build 2.0a

Histórico de Revisões

Date	Descrição	Autor(s)
03/05/2018	Relatório Desenvolvido ao Final do Projeto	Nadine Cerqueira Marques Valmir Vinicius de Almeida Santos

SUMÁRIO

1 | Introdução

1. Propósito do Documento

Este documento tem por objetivo descrever a arquitetura do projeto Interface E/S - Problema 2, incluindo a apresentação da visão geral do sistema, bem como os componentes utilizados para sua construção.

2. Acrônimos e Abreviações

Sigla	Descrição
DTE	*Data Terminal Equipment*
DCE	*Data Terminating Equipment*
LCD	*Liquid Crystal Display* (Disolay de Cristal Líquido)
UART	*Universal Asynchronous Receiver Transmiter* (Transmissor/Receptor Assíncrono Universal)

3. Definições

Termo	Descrição
RS232	Padrão de protocolo para troca serial de dados binário entre um DTE e um DCE. É geralmente implementado por uma UART.
Comunicação Serial	Uma forma de troca de dados entre sistemas digitais na qual múltiplos *bits* são transmitidos sequencialmente, um por unidade de tempo. [1]
Buffer	Área de armazenamento temporário [2]
FIFO	Um *buffer first-in-first-out* que funciona como um armazenamento elástico entre dois subsistemas. Um subsistema guarda dados no *buffer* e o outro subsistema recupera dados e os remove do buffer. A ordem de recuperação de dados é a mesma em que o dado foi armazenado [3]
Baud Rate	Taxa de transmissão de dados
LCD	*Liquid Crystal Display - Display* de Cristal Líquido
Mapeamento de Memória	Método para realização de entrada e saída de dados entre um processador e dispositivos periféricos em um sistema computacional

4. Objetivo do Projeto

O sistema desenvolvido tem por objetivo realizar a comunicação entre o processador e dispositivos de E/S. Além disso, um outro requisito do projeto é uma aplicação para teste de erro, com o uso do cálculo de CRC, a fim de validar essa comunicação.

2 | Visão Geral da Arquitetura e Descrição do Projeto

1. Arquitetura do Projeto

A visão geral da arquitetura do projeto é apresentada nesta seção. Além disso, a implementação dos controladores para comunicação com dispositivos de E/S é descrita.

A arquitetura geral do projeto, representada pela figura 2.1, é composta pelo processador NIOS II/e, que se conecta aos dispositivos de entrada e saída. Além disso, as especificações do projeto, como qual versão do NIOS foi utilizada, são apresentadas na tabela a seguir.

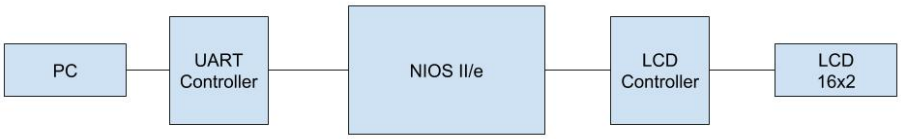

Figura 2.1: Visão Geral da Arquitetura do Projeto

Especificações do Projeto

Processador	NIOS II *Economy*
Memórias Associadas	*On Chip Memory* com 4096 *bytes*. FIFO com 2048 *bytes*.
Periféricos E/S	Interface RS232 e LCD 16x2
Placa	Cyclone IV E EP4CE6E22C8 com *clock* de 50 MHz

1.1. Interface com Memória

Para realizar a comunicação do processador NIOS com os componentes periféricos de *hardware*, desenvolvidos em Verilog, foi necessário definir um tipo de interface entre estes diferentes dispositivos. Existiam, inicialmente, três possíveis técnicas diferentes para executar essa etapa do projeto: mapeamento de memória, instruções customizadas e interrupção. A técnica adotada no contexto do projeto desenvolvido foi a de mapeamento de memória, por ser facilmente implementado e atender a todos os requisitos esperados.

A implementação da interface com mapeamento de memória foi realizada a partir do componente PIO (*Parallel I/O*), instanciado através da ferramenta Qsys, integrante do *software* Quartus. Nesse processo de instanciamento define-se a largura do barramento, o endereço a ser mapeado e a direção do fluxo de dados (*Input*, *Output* ou *Bidir*). As saídas geradas a partir do componente são, então, conectadas adequadamente aos periféricos implementados.

1.2. Controlador da UART

O controlador da interface de transmissão de dados serial foi implementador em Verilog. Ele foi desenvolvido para controlar a comunicação entre o computador e o processador NIOS. Assim sendo, esse módulo comanda em qual modo de operação a UART operará de acordo com o *byte* de controle apresentado na figura 2.2.

VELOCIDADE (7-6)	STOP (5)	HANDSHAKE (4)	TAMANHO_DADOS (3-2)	PAR/ÍMPAR (1)	PARIDADE (0)

Figura 2.2: Parâmetros de Controle

Na figura 2.3 a seguir, retirada da ferramenta RTL *Viewer* do *software* Quartus, pode-se ter uma visão geral dos componentes do controlador da UART. São apresentados os geradores de *baud rates*, dois multiplexadores 4:1, o primeiro para informar ao módulo de recepção da UART quantos *bits* de dados cada pacote da mensagem terá e o segundo para informar o *baud rate* da comunicação. Além disso, é possível identificar o módulo de recepção da UART e o controlador da FIFO.

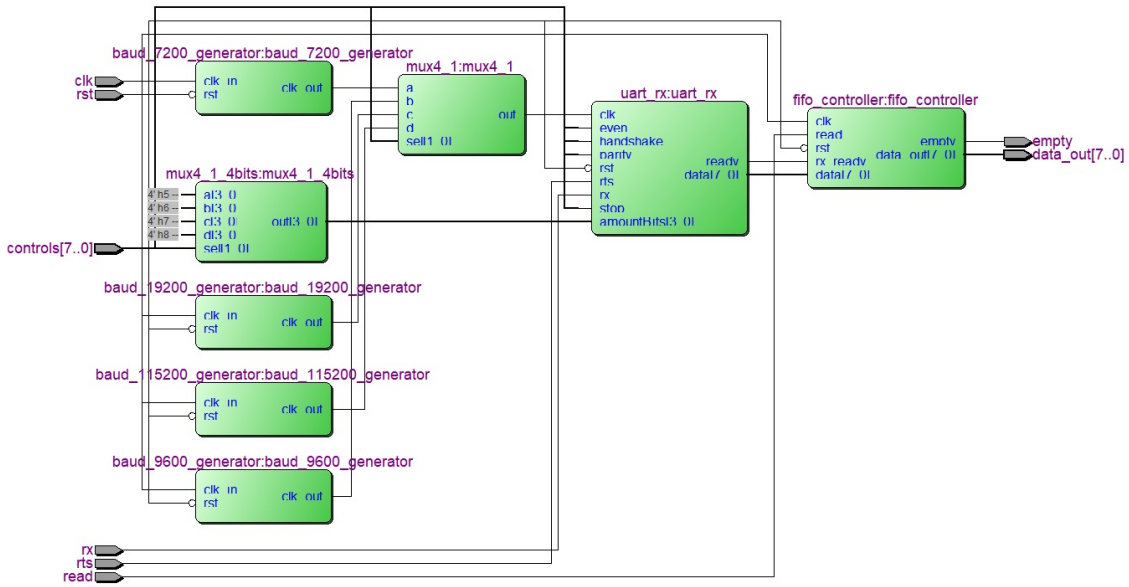

Figura 2.3: Representação RTL do Controlador da UART

Módulo de Recepção da UART

O módulo de recepção da UART é o componente que, definitivamente, receberá dados provenientes da interface serial. Ele operará de acordo com os parâmetros de controle predefinidos em seu controlador.

Além disso, esse módulo foi implementado por meio de uma máquina de estados finitos com os seguintes estados: *idle*, *receiving*, *done*, *checking* e *failed*. A máquina permanece no estado *idle* até que o *bit* de *start* seja recebido. Após isso ocorrer, a máquina transita para o estado *receiving*. Nesse estado há a recepção dos *bits*. A partir desse estado é possível ir para o estado de *done* ou para o estado de *checking*, caso haja *bits* de paridade. Nesse estado, há o teste de paridade e a máquina pode mudar para o estado de *done*, caso não haja erro, e para o estado *failed*, caso a paridade esteja incorreta. Ao final, a máquina retorna ao estado *idle* até que o novo *bit* de *start* seja identificado.

Módulo de Controle da FIFO

O controlador da FIFO atua de forma a promover a escrita ou leitura de dados nesse componente. Para isso, esse módulo gera pulsos de escrita ou leitura de acordo com as demandas de chegada de novos dados pelo módulo de recepção da UART ou requisição de leitura de dados por meio do processador NIOS.

Modos de Operação da UART

Os modos de operação da UART são definidos pelos parâmetros de controle supracitados e podem ser os apresentados na tabela 2.1:

Baud Rate	7200, 9600, 19200, 115200
Stop Bit	1 ou 2 *bits* de *stop*
Tamanho do Pacote de Dados	5, 6, 7 ou 8 bits
Paridade	Presente ou não
Paridade	Ímpar ou par

Tabela 2.1: Tabela com parâmetros de controle

1.3. Controlador do LCD

Assim como o módulo UART, o controlador do *display* LCD foi implementado através da linguagem de descrição de *hardware* Verilog. Este módulo foi projetado com foco em executar as duas rotinas essenciais para o funcionamento do *display* (inicialização e escrita), assim como proporcionar ao usuário diferentes modos de operação. A interface entre o módulo e o processador NIOS foi realizada por meio da técnica de mapeamento de memória. O LCD que foi utilizado é um *display* 16x2, modelo AC-162B, fabricado pela Ampire Co. Ltd.

Modo de Operação

O primeiro passo executado durante o processo de desenvolvimento do módulo consistiu na definição dos modos de operação que seriam adotados. Nesse sentido, decidiu-se que o usuário poderia escolher entre utilizar ou não o cursor que existe no LCD, obtendo-se, assim, os dois modos de operação: cursor ativado e cursor desativado.

Para a configuração do modo de operação desejado utiliza-se o endereço de memória 0x21b0, adotado exclusivamente para este objetivo. Assim, é necessário armazenar, neste endereço, o valor decimal 1 nas situações em que o cursor deve estar ativado. Por outro lado, o valor 2 deve ser armazenado para desativamento do cursor.

A diferença de implementação entre esses modos é originada durante a rotina de inicialização do *display*, quando um dos comandos enviados para o componente varia de acordo com o modo de operação selecionado. Essa rotina de inicialização será descrita com mais detalhes na próxima seção.

Inicialização

A rotina de inicialização do *display* consiste, basicamente, no envio de uma sequência de comandos, responsáveis por definir configurações iniciais para o funcionamento adequado do componente. Entre o envio de duas instruções distintas é necessário, ainda, inserir um certo intervalo de tempo (*delay*). Tanto os comandos quanto os atrasos estão especificados em um *datasheet* disponibilizado pelo fabricante. Essa sequência de tarefas foi implementada em *hardware* através de uma máquina de estados finita do tipo Moore.

Uma representação gráfica da MEF projetada pode ser visualizada na figura 2.4.

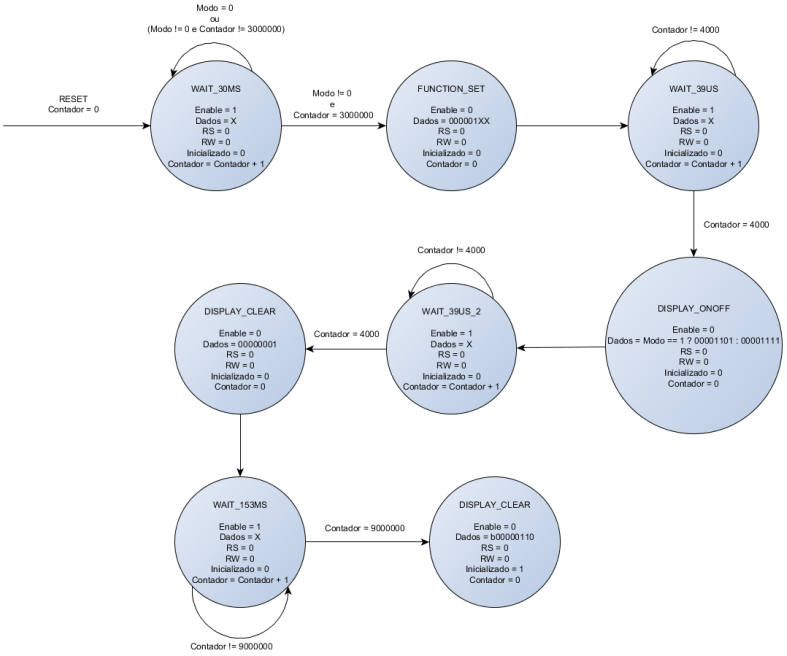

Figura 2.4: MEF - Inicialização LCD

Conforme pode ser visualizado na imagem, o módulo possui cinco saídas. Quatro das saídas (Enable, RS, RW e Dados) correspondem às entradas essenciais para comunicação com o *display*. Por outro lado, a entrada Inicializado indica que a rotina de inicialização foi completada.

Escrita de Dados

O outro submódulo do controlador LCD é o responsável por executar as operações para efetuar a escrita dos caracteres na memória do display. Este submódulo possui dois inputs: Inicializado e Entrada. O primeiro indica se a rotina de inicialização já foi corretamente executada, de modo que o componente está pronto para armazenar caracteres. Por outro lado, Entrada possui tamanho de 72 bits e deve receber os dados a serem, efetivamente, armazenados no display. Como cada caractere é representado por 1 byte, poderão ser escritos, no máximo, 9 caracteres.

A configuração de cada caractere a ser escrito é realizada a partir de nove endereços de memória mapeados para este objetivo. Estes endereços são 0x2160, 0x2150, 0x2140, 0x20e0, 0x20f0, 0x2100, 0x2110, 0x2120 e 0x2130, sendo o primeiro endereço aquele no qual deve ser armazenado o caractere mais significativo.

Assim como a inicialização, a escrita de dados é realizada através de uma MEF do tipo Moore, graficamente representada na figura 2.5.

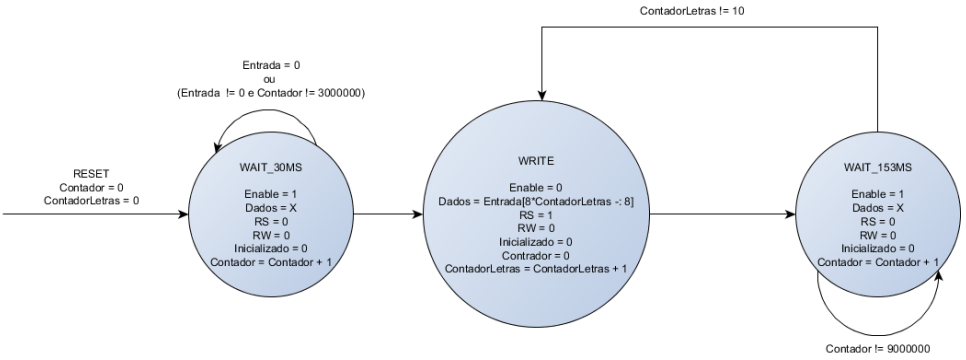

Figura 2.5: MEF - Escrita LCD

As únicas saídas do submódulo são as essenciais para comunicação com o LCD (Enable, RS, RW e Dados).

2. *Software* para Envio de Dados

A fim de enviar dados do computador para o procesador, foi desenvolvido um *software* em Python utilizando a biblioteca PySerial. O *software* implementado lê um arquivo .dat com os dados a serem enviados, organizados em 251 linhas com 4 *bytes* cada. A partir dessa leitura o programa envia, *byte* a *byte*, os dados pela porta serial do computador. Além disso, foi definido um tempo de espera entre o envio dos *bytes* de 10/(*Baud Rate*), que é o tempo de envio de cada pacote de dados, composto por 10 *bits*.

3 | Resultados

1. Comunicação Serial

Ao final do projeto, foram enviados do computador, por meio do *software* implementado, a mensagem de 1 kB e seu CRC. Com a ferramenta Altera Monitor, presente no *software* Quartus, foi possível identificar os dados recebidos pelo módulo de recepção armazenados na FIFO.

A figura 3.1 apresenta alguns dos dados recebidos pelo processador. Nessa figura pode-se perceber que os dados são recebidos de forma correta, porém há alguns dados entre eles que não existem no arquivo original. Esse dados podem ser sido originados, provavelmente, por algum erro no módulo de recepção.

Memory

Goto memory address | Address (hex): 9c | Go | Query All Devices

	+0x0	+0x4	+0x8	+0xc
0x00000090	38400126	003fec06	003fff06	4c757465
0x000000a0	69207072	6120656e	74726172	2065206e
0x000000b0	616f2076	6f752073	61697220	4f732071
0x000000c0	7565206e	616f2070	65727465	ffffffed
0x000000d0	2c206575	20646576	6f6c7669	20416369
0x000000e0	646f206e	6f206d65	74616c2c	20636175
0x000000f0	ffffffd9	66656974	6f206c65	74616c54
0x00000100	65746f20	42616978	6f207465	20657370
0x00000110	72656d65	ffb52072	65737069	72612051
0x00000120	75656d20	70697261	20746120	6e61206d
0x00000130	69726120	6461206d	b56e6861	20666972
0x00000140	6d612045	6e74616f	206d6520	65737065
0x00000150	72612072	65637570	ffffffb9	206f2066
0x00000160	6f6c6567	6f205365	20636f6d	b5676f20
0x00000170	6e616f20	6d6f7272	652c206e	ffffe361
0x00000180	20636169	2c206e61	6f207465	6e746120
0x00000190	6120736f	72746520	576f6f64	73746f63
0x000001a0	6b206e75	6d20666c	6f77206d	65746f64
0x000001b0	69636f20	446f7220	6e616f20	65207072
0x000001c0	61207175	656d2071	7565722c	20646f72
0x000001d0	20652070	72612071	75656d20	706f6465
0x000001e0	2045206e	6f73736f	20646573	74696e6f
0x000001f0	20652075	6d612063	61697861	20646520
0x00000200	73757270	72657361	204c656f	70617264
0x00000210	6f206f75	205a6562	72613f20	4d652064
0x00000220	697a3a20	43652071	75657220	ffffff90
0x00000230	70726564	61646f72	206f7520	70726573
0x00000240	613f2041	7373696d	2c206f20	50657263
0x00000250	ffffb969	2070656c	61206265	69726164
0x00000260	61206174	65206120	736f7274	65206d65
0x00000270	2064697a	6572204d	656e696e	6f2c2076

Figura 3.1: Dados recebidos pelo processador

2. *Display* LCD

O módulo concebido para controle do display LCD funcionou conforme planejado durante a fase de implementação, os testes realizados serviram para demonstrar que é possível utilizar o componente de acordo com o modo de operação selecionado. Na imagem 3.2 é possível perceber o *display* funcionando corretamente no modo de operação que possui cursor ativado. Por outro lado, a imagem 3.3 evidencia o funcionamento do LCD no modo de operação com cursor desativado.

Figura 3.2: ***Display*** **LCD - Cursor Ativado**

Figura 3.3: ***Display*** **LCD - Cursor Desativado**

3. Área Total Ocupada pelo Circuito

A partir de relatórios gerados pelo *software* Quartus foi possível obter dados sobre a área que o projeto implementado consome em função dos elementos que o compõem.

Área Ocupada em Função dos Elementos Internos do Dispositivo

A tabela a seguir é composta pelas por elementos internos do dispositivo e suas respectivas quantidades ao final do projeto.

Elemento	Quantidade
LE	3.011
LAB	236
Registradores	1.473
Memória	60.416 *bits* e 101.376 *bits* de implementação

Área Ocupada por Elementos Lógicos por Modo

Além disso, esse relatórios ainda apresentam a quantidade de elementos lógicos utilizados em cada modo de operação, como mostra a tabela a seguir.

Modo	Quantidade
Normal	2.605
Aritmético	278

Representação da Área Total Ocupada Pelo Circuito

A figura 3.4 a seguir, gerada pela ferramenta *Chip Planner* do *software Quartus*, apresenta a área total ocupada pelo circuito do projeto.

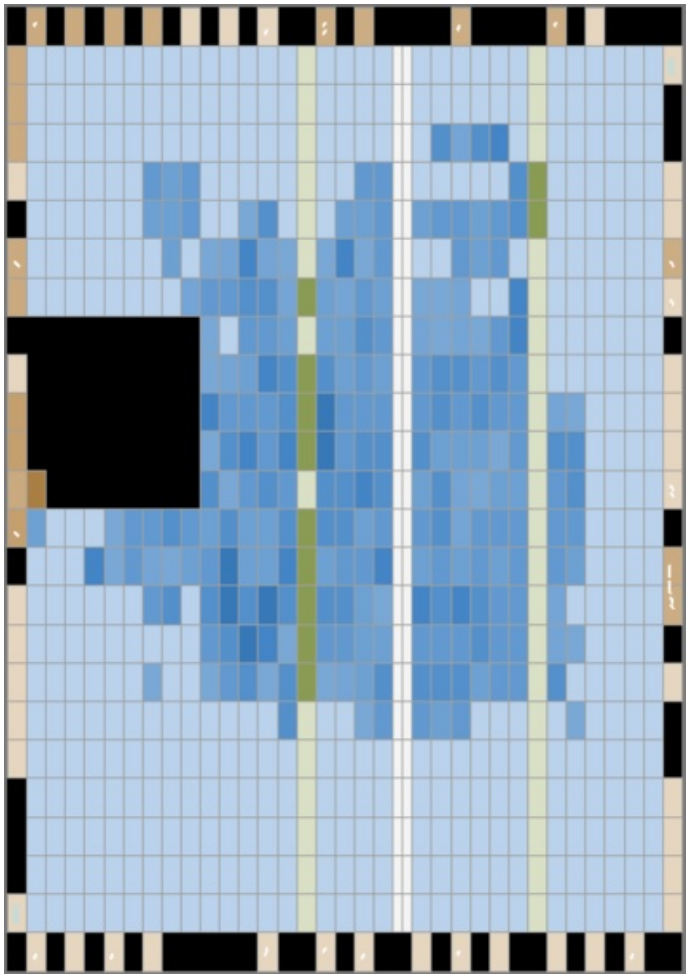

Figura 3.4: **Área Total Ocupada pelo Circuito**

4. Caminho Crítico do Circuito

A partir da geração de relatórios com tabelas por meio da ferramenta *TimeQuest Timing Analyzer*, presente no *software* Quartus, foi possível observar o caminho do sistema com maior custo em termos de tempo. Assim, verificou-se que esse caminho crítico se dá entre registradores presentes na interface JTAG e que consome 99.204ns.

Esse caminho foi identificado no circuito por meio da ferramenta *Chip Planner* e é apresentado na figura a seguir, extraída da própria ferramenta:

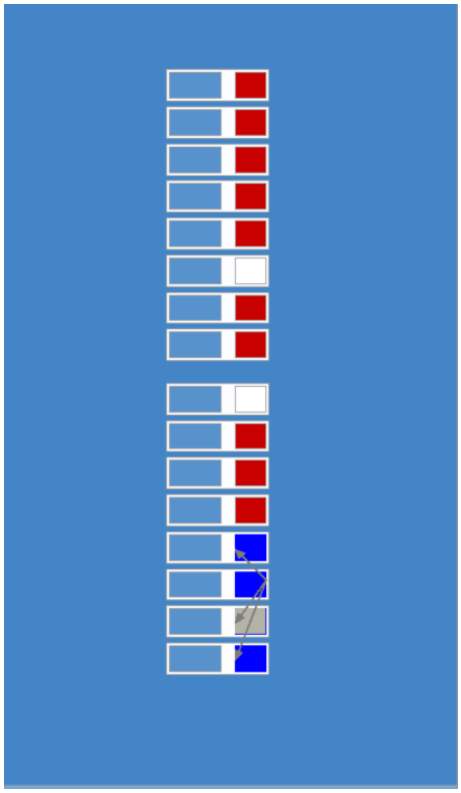

Figura 3.5: **Caminho Crítico do Circuito**

4 | Conclusão

A concepção do sistema para recebimento do CRC-32, cálculo e exibição em LCD foi fundamental para suscitar novos conhecimentos, como o desenvolvimento de módulos com linguagem de descrição de *hardware* e interface entre processador e periféricos, por exemplo. Além disso, a implementação do produto também proporcionou a obtenção de aprendizagem relacionada ao funcionamento do protocolo RS-232 e de *displays* LCD.

Apesar de não ter sido completamente implementado, o produto obtido ao final do processo de desenvolvimento conseguiu realizar as operações básicas de interfaceamento e pleno funcionamento do módulo de controle do LCD, por exemplo. Entretanto, o módulo UART não atende completamente aos requisitos inicias, pois não implementa *handshake*, envio de dados da placa para o computador, além de possuir problemas na recepção de dados. Possíveis trabalhos futuros devem ser ocorrer no sentido de corrigir essas deficiências e, além disso, incrementar os componentes que já funcionam corretamente, adicionando, por exemplo, mais opções de modos de operação para o *display* LCD.

1. Referências

[1] D. Summerville, *Embedded Systems Interfacing for Engineers Using the Freescale HCS08 Microcontroller: Digital and analog hardware interfacing. II.* Embedded Systems Interfacing for Engineers Using the Freescale HCS08 Microcontroller, Morgan & Claypool Publishers, 2009.

[2] I. M. F. Ann McHoes, *Understanding Operating Systems.* Course Technology, 7 ed., 2013.

[3] P. Chu, *RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability.* Wiley - IEEE, Wiley, 2006.