Projeto Final de Lógica Reconfigurável

Alunos:	**RA:**
Giovanni Luiz Zanetti	1729748
Rafael Gama Palone	1720449
Rafael Hideo Toyomoto	1722085

Link do projeto: https://drive.google.com/drive/folders/1fO4uxA6P64ob_zwdKJK-oxmPT_8mOjCx

Objetivo

Nosso projeto consiste em um servidor que coleta informações de clientes pela rede, e salva em um *SD Card*. Em paralelo, nosso sistema possui a funcionalidade de exibir os dados coletados em forma de um gráfico.

Um exemplo de integração com outros projetos seria qualquer sistema de sensores, que coletará os dados e enviará pela rede.

Implementação

Estudo do exemplo DE2_NET

A implementação foi realizada sobre o exemplo do próprio CD da placa, o projeto **DE2_NET**. Nesse exemplo já temos no **Qsys** uma configuração preparada com diversos componentes, incluindo a *cpu* **NIOS II** com seus componentes necessários para implementação do **MicroC/OS-II RTOS** e o *DM9000A* para acesso a rede.

Ainda de componentes, há um IP de uma controladora *VGA*, que será utilizado para nossa implementação de acesso a esse recurso ao plotar o gráfico das amostras salvas. Para os *switchs* que serão utilizados como entrada de comandos pelo usuário, são utilizados componentes nativos do **Qsys** , os *Parallel Input/Output (PIO)* para mapear esses *switchs* e já realizar as tratativas necessárias de *hardware* para uma leitura consistente de dados.

Por fim, já vem incluído componentes nativos do **Qsys** para acesso aos pinos de controle da entrada de **SD Card** da placa. Esses componentes acabaram sendo desabilitado e foi incluído um IP da própria Altera para realizar essa tarefa (*Altera University Program SD Card IP Core*).

Pela parte de *software*, o código exemplo se trata de um servidor de resposta a requisições pelo protocolo *Telnet*. Essa implementação se utiliza do **MicroC/OS-II RTOS** para criar um sistema *multi-thread* que controla requisições via rede com a biblioteca **Nichestack TCP/IP Stack**, em paralelo ao acionamento de *LEDs* e o

Display de 7 segmentos em resposta aos comandos enviados de um cliente conectado. Essa ideia será mantida para nossa implementação, realizando o processo de atualizar o *VGA* e leitura dos *switchs* em paralelo ao tratamento de requisição de clientes.

Por fim, será necessário incluir as bibliotecas fornecidos no próprio exemplo em nosso projeto, visando utilizar as implementações já prontas para acesso aos componentes comentados anteriormente.

Abaixo será apresentado uma lista dos arquivos no projeto no *Eclipse* e seus significados.

- **alt_error_handler.h** e **alt_error_handler.c** : Arquivo de tratamento de erros ocorridos no **MicroC/OS-II**, na **Nichestack TCP/IP Stack** e no próprio exemplo.
- **basic_io.h** : Macros alto nível para executar funcionalidade dos *GPIO* incluídos, do *Display* de 7 segmentos e dos *Timers*.
- **dm9000a_regs.h**, **dm9000a.h** e **dm9000a.c** : Funções do *DM9000A*.
- **iniche_init.c** : Arquivo com o código principal (*main*) que inicializará todo o sistema e sua primeira *thread* (*SSSInitialTask*). Essa *SSSInitialTask* será responsável por inicializar todas as demais *threads* do sistema e será encerrada logo após esse processo.
- **led.c** (não utilizado) : Arquivos com as *threads* de acesso aos I/O do exemplo.
- **my_threads.c** : Nosso arquivo com as *threads* de controle do VGA (*DisplayUpdateVGATask*) e leitura dos *switches* (*DisplayReadButtonsTask*).
- **network_utilities.h** e **network_utilities.c** : Funções auxiliares para o uso do **Nichestack TCP/IP Stack**.
- **simple_socket_server.h** e **simple_socket_server.c** : Arquivo com a *thread* de servidor e funções auxiliares a essa *thread*. Nesse arquivo também está a implementação das duas funções utilizadas para inicialização de estruturas necessárias para o projeto (Semáforos, Filas, …) e das *threads* secundárias.
- **VGA.h** e **VGA.c** : Funções para o uso da VGA.
- **sdcard/my_sdcard.h** e **sdcard/my_sdcard.c** : Funções de acesso a memória das amostras recebidas e controle de carregamento e salvamento desses dados no *SD Card*.

Socket

Primeiramente, o programa foi levemente modificado para simplificar a forma como a placa aguarda um cliente, permanecendo em apenas uma thread essa funcionalidade, porém sem aviso de tentativas de conexão enquanto a placa está ocupada com uma conexão já aberta. No próprio exemplo há comentários que dão a entender que é possível realizar uma implementação que suporte multi-cliente, porém essa abordagem não será considera para esse projeto.

SD Card

A opções por não utilizar o componente nativo do exemplo para o controle dessa funcionalidade foi realizada após a tentativa de se utilizar ela junto ao seu código exemplo para escrita no *SD Card*. A leitura era possível sem problemas, porém a escrita acabou não funcionando, e com algumas pesquisas foi possível confirmar que haviam outras pessoas com o mesmo problema, visto que esse componente foi apenas utilizados nos exemplos com a finalidade de leitura (Como no exemplo que utilizada o *SD Card* para ler uma música e toca-lá).

Assim, foi utilizado uma outra biblioteca da própria Altera, a “Altera University Program SD Card Avalon Interface”, que pode ser testada em um projeto separado, onde foi possível realizar as operações tanto de escrita, quanto de leitura no *SD Card*. Na documentação dessa biblioteca é comentado que há um limite de tamanho dos SD Card suportados, sendo ele de 2GB.

Para integrar esse componente ao nosso projeto, foi necessário encontrar seus arquivos e colocar na pasta *ip* no diretório raíz, incluir esse componente no **Qsys** e realizar modificações no arquivo *top-level* do projeto (*de2_net.v*).

Na parte do código, para evitar escritas demasiadas sobre o *SD Card* e evitar o desgaste desse componente, toda armazenagem das amostras é realizada na própria memória do sistema, que tem sua inicialização realizada através da leitura inicial do *SD Card*, e seu estado atual salvo quando solicitado pelo usuário (*switch SW1*). Para acessar essa memória foi criado um semáforo (*SDCardSem*) para que não ocorram operações de leitura e escrita simultaneamentes o que poderia causar um erro de leitura se ocorresse preempção (sessão crítica).

Arquivos relacionados ao uso do SD card foram inseridos na pasta *sdcard*, as funções relacionadas ao mesmo estão presentes no arquivo *my_sdcard.h*. Somente os arquivos relacionados ao *SD Card* foram inseridos em uma pasta separada pois eram os únicos que inicialmente não pertenciam ao projeto.

VGA

Como já comentado, para a VGA foi utilizado o próprio IP de controladora *VGA* fornecido no projeto exemplo, onde a aplicação da biblioteca sobre o código de servidor ocorreu sem problemas. Para fins de melhora da usabilidade da aplicação, foram criadas 2 novas threads no programa para tratar da *VGA*, uma que monitora os switches da placa para detectar um *input* do usuário para atualização do gráfico, e a outra para tratar de atualizar efetivamente o *VGA*. A *thread* de *input* de usuário será responsável por preencher uma fila (*VGACommandQ*) com os comandos lidos, e a *thread* do VGA irá consumir os elementos dessa fila para realizar suas operações, entrando em espera quando a fila estiver vazia.

Toda implementação comentada acima se encontra no arquivo *my_threads.c*.

Compilação

1. Refazer a geração do .qip no QSYS (corrigir possíveis problemas de path com os arquivos dos IPs)
2. Recompilar o projeto.
3. Abrir a janela *Programmer* e rodar o projeto.
4. Abrir o eclipse e selecionar o pasta *software* no diretório raiz como *workspace* (deverá encontrar o projeto “PepinoBoris” e “PepinoBoris_bsp”).
5. Selecionar o projeto “PepinoBoris_bsp”, abrir o editor de opções de BSP do NIOS II e regerar as referências (arrumar os paths para os arquivos de *header* e *source* dos IPs). **ATENÇÃO:** A maioria dos arquivos já foram modificados para utilizar path relativo, como o *Makefile* principal do projeto.
6. Executar o comando “build” no projeto “PepinoBoris_bsp” (possíveis erros em alguns arquivos de biblioteca podem acabar ocorrendo, apenas ignorar).
7. No projeto “PepinoBoris”, encontrar o arquivo **simples_socket_example.h** e modificar seus defines para se conectar a rede (IP fixo, gateway e máscara de rede).
8. Executar o comando “build” no projeto “PepinoBoris”.
9. Executar o projeto. Recomenda-se utilizar o *Debug* para melhor visualização dos passos iniciais de configuração do projeto.
10. No console do NIOS II, será solicitado um identificador de 9 dígitos. Pode se utilizar “123123123”.
11. Esperar o DHCP atribuir um IP à placa. Pode ser que ocorra erro na inicialização do componente DMA9000, sendo necessário refazer o processo de execução do projeto. Esse problema pode ser observado pelo conector *ethernet* da placa, que deverá ter uma luz verde acesa.

Teste

1. Junto a pasta do projeto no *Google Drive*, há um arquivo chamado **ClienteSocket2.jar**. Baixar esse programa para se comunicar com a placa de testar o envio de amostras artificiais.
2. Executar o **ClienteSocket2.jar**. Ao abrir o programa, haverá 2 campos para preencher (IP e porta). Colocar o IP atribuído a placa pelo DHCP e a porta 30.
3. Selecionar uma forma de onda ao lado direito.
4. Clicar em conectar.
5. Voltar ao eclipse e apertas F8 (*Resume*) caso esteja em modo *Debug* (Há um breakpoint logo após a detecção de conexão).
6. Esperar alguns segundos para a placa recolher algumas amostras.

7. Utilizar o *Switch* SW0 para atualizar o gráfico pelo VGA (conectar a um monitor).
8. Utilizar o *Switch* SW1 para salvar o estado atual no SD Card (o SD Card deverá ser introduzido antes de inicializar a execução do programa).