Universidade Estadual Paulista
“Júlio de Mesquita Filho”
Câmpus de Rio Claro
IGCE

Projeto de microprocessadores

Relatório Final da Disciplina

Grupo:

Gabriel Crescencio
Murilo Geraldini
Ramon Varela

1. Objetivos

O objetivo deste trabalho era o desenvolvimento de um aplicativo console que aceite um conjunto de comandos enviados pelo usuário. Cada comando recebido pelo console deve refletir uma determinada ação na placa DE2 Altera.

2. Introdução

Este projeto envolveu a criação de uma aplicação que interage com a placa DE2 da Altera. A comunicação com o console é implementada por meio da UART, e o aplicativo utiliza a linguagem de montagem Nios II, aproveitando os recursos da DE2, que também é um meio computacional.

Quando o programa é iniciado ele pede para o usuário entrar com um comando. Os comandos são compostos por dois números inteiros, que definem que ação será realizada, por exemplo piscar um determinado led vermelho na placa, ou apagá-lo.

Como desafio, também tivemos a oportunidade de implementar comandos diversos aos requeridos no trabalho.

3. Cronograma

Planejamento	
26/10	Parsing
	Função para acender o 7-seg
9/11	Função para acender o 7-seg
	Leds
	Chaves
16/11	Chaves
	Cronômetro
23/11	Cronômetro
	Erros / Extras
30/11	
7/12	Apresentação

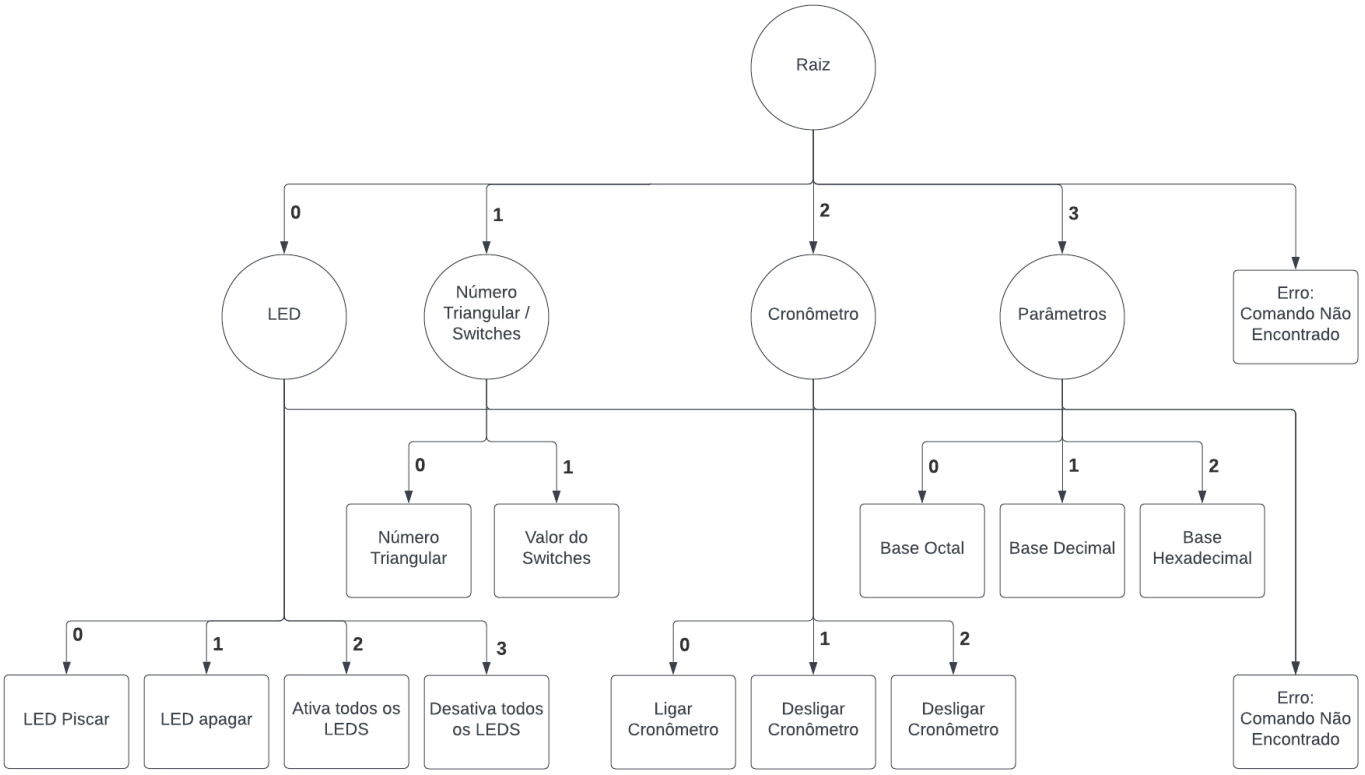
Real	
26/10	Parsing
	Número Triangular / Chaves
	Função para acender o 7-seg
9/11	Função para acender o 7-seg
	Cronômetro
16/11	Leds
	Erros
23/11	Erros / Extras
30/11	Erros / Extras
	Apresentação

4. Desenvolvimento

A entrada do usuário é feita através do Terminal UART. Cada caractere é armazenado em um buffer na memória até que o carácter ‘\n’ seja identificado, quando o programa passa para a fase de análise (parsing) da entrada do usuário. A análise é feita caractere por caractere e aproveita-se da

forma como os comandos foram definidos para utilizar uma estrutura de árvore de decisão decidindo qual comando deve ser acionado.

Ao analisar os comandos, percebe-se que os comandos começados em 0 referem-se a comandos de LED, comando começado em 1 é o de número triangular, os comandos começados em 2 são comandos do cronômetro e os comandos começados em 3 são comandos voltados à definição de parâmetros do sistema (no caso foram implementados comandos de troca de base dos displays de 7 segmentos). Dessa forma cria-se uma árvore onde as folhas são os comandos em si e cada aresta corresponde ao caractere necessário para alcançar aquele nível.



Os comandos de LED não geram erro do tipo “comando não encontrado” uma vez que os caracteres, além dos dois utilizados no comando, são considerados o número do LED que deve começar ou parar de piscar, ou seja, a presença ou não de um ou mais espaços em branco depois do comando não é relevante para o analisador nesse caso. Por exemplo, tanto o comando “0000” quanto “00 00” ou até mesmo “00 00” marcam o mesmo LED para piscar.

Os outros comandos, entretanto, geram erro de “comando não encontrado” caso exista qualquer caractere no buffer além dos dois caracteres que definem o comando. Isso acontece porque, após fazer a leitura completa da entrada do usuário (que acaba apenas quando o programa lê um ‘\n’), o programa adiciona 0 no fim do buffer, indicando que esse é o final da cadeia de caracteres, dessa forma, o programa identifica que um comando é invalido caso o terceiro caractere seja diferente de 0.

Além do buffer com 128 bytes, 4 bytes de memória estão destinados a armazenar os LEDs ativos, 4 bytes de memória são utilizados como contador do temporizador, 4 bytes estão destinados a informações do cronômetro e 4 bytes destinados ao valor atual que está sendo mostrado no display de 7 segmentos. O contador do temporizador foi definido como 4 bytes apenas com fim de simplificar o código, uma vez que 1 bit seria o suficiente para satisfazer sua função que é de, caso o contador seja par, ele apaga todos os LEDs e caso seja ímpar ele liga todos os LEDs ativos e soma

um ao cronômetro que conta de segundo em segundo. Os 4 bytes destinados ao cronômetro, utilizam-se de 16 bits para armazenar o valor atual do cronômetro e 1 bit para identificar se o cronômetro está ou não ligado. Por fim, os 4 bytes para o valor atual dos displays são necessários para efetuar a troca de base dos displays, fazendo com que a troca possa ser realizada instantaneamente e não apenas na próxima chamada de um comando.

Inicialmente, o parser verificava o carácter em si, ou seja, ele verificava se o carácter era 0x30 (para 0), 0x31 (para 1), etc. porém isso fazia o código menos legível e por isso foi feito a refatoração do código para que ele verificasse o número em si, convertendo de carácter para número subtraindo 48. Outra dificuldade foi a conversão dos números de hexadecimal para decimal no display 7 segmentos. A implementação do código que fazia essa conversão depois permitiu a implementação da troca de base, uma vez que o novo código é genérico. Outra decisão foi de condensar as duas sub-rotinas de ativar e desativar LEDs em uma única onde a escolha de ativar ou desativar o LED é decidido por passagem de parâmetro, uma vez que as duas sub-rotinas compartilhavam grande parte do código. Isso faz com que a manutenção da sub-rotina seja mais fácil e o código fique mais limpo.

O parser não desconsidera espaços antes do código, o que faz com que, por exemplo, a string “ 0001” retorne erro ao invés de executar a função de pisca LED para o LED 01. Além disso, quando backspace é apertado, apesar de apagar o carácter no terminal, o programa não apaga o carácter no buffer, o que faz com que mesmo que um comando pareça correto no terminal ele resultará em erro. Por último, o buffer possui um limite de 127 caracteres (uma vez que o último byte da memória é reservado para o valor 0) porém o terminal continua imprimindo as novas teclas apertadas mas não as salva no buffer. Esses foram os comportamentos não esperados encontrados no código e que são pontos de melhora no mesmo.

5. Comandos

Os comandos desenvolvidos para o programa estão na tabela abaixo:

Comandos		
LEDs	00 xx	Pisca o LED xx (00 a 17)
	01 xx	Apaga o LED xx (00 a 17)
	02	Pisca todos os LEDs
	03	Apaga todos os LEDs
Triangular / Switches	10	Calcula o número triangular definido nos switches e mostra no display de 7 segmentos
	11	Mostra o valor definido nos switches no display de 7 segmentos
Cronômetro	20	Inicia ou reinicia o cronômetro
	21	Para e cronômetro e zera
	22	Pausa o cronômetro
Base numérica do 7 segmentos	30	Octal
	31	Decimal
	32	Hexadecimal
Tabela 1: Comandos		

5.1. Explicando os comandos e sua implementação

Como já mencionado anteriormente, os comandos são compostos de dois caracteres (números inteiros) podendo, dependendo do comando, ser seguidos por outros caracteres. O primeiro caractere inteiro define o grupo de funções a que o comando pertence e o segundo define sua função específica.

5.1.1. LEDs

Quando o primeiro caractere é “0”, um comando relativo aos LEDs vem a seguir.

“00 xx”

Este comando deve ser seguido de mais dois caracteres inteiros formando um valor de 0 a 17, valores dos LEDs vermelhos disponíveis na placa utilizada. Esse comando faz com que o LED escolhido pisque no padrão de meio segundo aceso e meio segundo apagado.

Para seu funcionamento utilizamos a posição de memória que havíamos reservado para setar os LEDs ativos, ou seja, que devem piscar. Salvamos, nesta região da memória, valores referentes aos LEDs ativos na forma de máscara de bits, por exemplo “1001”, define como ativos os bits 00 e 03.

“01 xx”

Este comando, analogamente ao comando anterior deve fazer com que o LED selecionado pare de piscar e permaneça apagado.

Também utiliza a região da memória referente aos LEDs ativos, mas transformando em 0s os valores 1s referentes aos LEDs que devem ser desativados.

“02”

Este comando ajusta o valor da memória de LEDs ativos para que todos eles sejam setados como 1s.

“03”

Analogamente ao anterior, este comando ajusta o valor para que todas as posições de LEDs sejam estados com 0s.

5.1.2 Triangular / Switches

Esse conjunto de funções é ativado quando o primeiro caractere é o “1”.

“10”

Este comando deve ler os valores ajustados nas primeiras 8 chaves do conjunto de switches, calcular o número triangular e mostrar o resultado no display de 7 segmentos.

“11”

Como o anterior, deve ler o valor das chaves, mas deve mostrar esse valor diretamente no display de 7 segmentos.

5.1.3 Cronômetro

Quando o primeiro caractere é igual a “2” ativamos o conjunto de funções de cronômetro.

“20”

Este comando inicia o cronômetro mostrado no display de 7 segmentos, respeitando a base definida. O valor do cronômetro é atualizado a cada segundo, modificando seu valor.

“21”

Com este comando o cronômetro é parado e zerado, perdendo o valor atual dele.

“22”

Entrando com este comando o cronômetro é pausado. Seu valor é guardado e ele pode ser reiniciado com o código “20”.

5.1.4 Base numérica

O primeiro caractere igual a “3” ativa o conjunto de funções de mudança de base numérica do display de 7 segmentos. Como padrão a base inicial é decimal.

“30”

Este comando modifica a base do display de 7 segmentos para octal.

“31”

Este comando modifica a base do display de 7 segmentos para decimal.

“32”

Este comando modifica a base do display de 7 segmentos para hexadecimal.