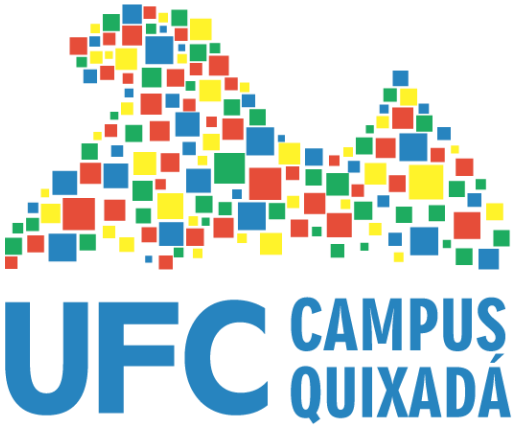
UARTsim

Um Simulador de Dupla Decodificação do Protocolo UART



UNIVERSIDADE FEDERAL DO CEARÁ

Pedro Botelho e Ítalo Moraes
Bacharelando em Engenharia de Computação

Introdução

- Proposta → Implementar em ARM Assembly um programa para realizar traduções entre códigos UART e ASCII
  - UART → Universal Asynchronous Receiver/Transmitter
  - ASCII → American Standard Code for Information Interchange
- UARTsim → Simulador do protocolo UART para processadores de arquitetura ARM
- Duas operações suportadas:
  - UART → ASCII
  - ASCII → UART
- Simula a comunicação entre um pino TX(transmitter) e um pino RX(receiver)
- No caso da tradução UART → ASCII
  - Arquivo com os pulsos UART → Pulsos vindos de TX
  - Programa atua como RX
  - Ao final do programa → Comunicação Serial em forma de caracteres

Pedro Botelho e Ítalo Moraes
Bacharelando em Engenharia de Computação

- Programa dividido em arquivos para melhor organização
- Arquivos de cabeçalho → Permite que os procedimentos possam ser chamadas em outro arquivo
  - inc/copycat.lib
  - inc/uart_sim.lib
- Arquivos de procedimentos → Procedimentos para manejo dentro do programa
  - src/main.s
  - src/copycat.s
  - src/uart_sim.s
- Arquivo binário → Executável somente em processadores ARM
  - bin/uart_sim
- Arquivos objetos → Utilizáveis pata objdump
  - build/main.o
  - build/copycat.o
  - build/uart_sim.o

- Todos os arquivos de cabeçalho utilizam a diretiva .GLOBAL <procedimento> para permitir o acesso
- Copycat
  - Código primeiramente implementado como laboratório 08
  - Melhorado para atender as necessidades do projeto
  - Funções para manejo de inputs do usuário e arquivos
  - Principais procedimentos:
    - **_getInput** → Obtém a entrada pelo **STDIN**
    - **_openFile** → Abre um arquivo baseado em permissões
    - **_closeFile** → Fecha um arquivo
    - **_fileToString** → Copia dados de um arquivo para a memória
    - **_stringToFile** → Copia dados da memória para um arquivo
    - **_printString** → Imprime uma string requisitada no **STDOUT**
    - **_clearString** → Limpa uma string, preenchendo o espaço previamente utilizado com zeros
- Procedimento Principal
  - Chamada de procedimentos
  - Lógica de Flags para melhorar densidade de código
  - Tratamento de nomes de arquivos inválidos

- UARTsim
  - Procedimentos utilizados na decodificação UART-ASCII
  - Assim com a main utiliza funções da biblioteca Copycat
  - Principais procedimentos:
    - **_decodeUART** → Decodifica pulsos UART em caracteres ASCII
    - **_decodeCHAR** → Codifica caracteres ASCII em pulsos UART
    - **_getOption** → Obtém a opção inicial do usuário e garante que a opção escolhida seja válida
- São utilizadas diretivas de pré-processamento para organização do código
- Cabeçalhos comentados para cada procedimento
- Makefile utilizado na automação de compilação
  - Makefile silencioso, simples e eficiente
  - Compila os arquivos separados em pastas e salva o executável em bin/ e os objetos em build/
  - Compila o código e separa os arquivos em pastas → **make it**
  - Executa o programa → **make run**
  - Realiza a depuração do código → **make debug**

- Procedimentos geralmente retornam com a Zero flag ativada
- Sistema conta com interface construída baseado em uma padronização → UARTsim: <Tipo do Comando>: <Comando>\n<Ação>
- Utilização de códigos de escape ANSI
  - Letras verdes → Execução norma
  - Letras vermelhas → Houve um erro
- Dois tipos de erros
  - Tratáveis → Podem ser tratados no programa, como por exemplo pedindo ao usuário que digite novamente a entrada
  - Não Tratáveis → Não podem ser tratados pelo programa, como erros no arquivo de entrada. Esse erro acarreta em um **Erro Fatal**
- Exemplo de Mensagem de Erro Não-Fatal:
  - \n\033[1;37mUARTsim\033[0m: \033[1;31mInvalid Input File\033[0m: Specified input file doesn't exist. Try again.\n
- Exemplo de Mensagem de Erro Fatal:
  - \n\033[1;37mUARTsim\033[0m: \033[1;31mFatal error\033[0m: Invalid UART pulse.\nAborting.\n\n

Funcionamento

- Ao início do programa deve-se informar qual operação será realizada
  - UART → ASCII
  - ASCII → UART
- Logo após deve-se informar um arquivo de entrada existente e condizente com a operação
- O arquivo de saída não necessariamente deve existir, mas seu nome deve ser válido
- Com a tecla <Enter> é feito o avanço para a próxima etapa
- Não deve-se utilizar espaços no nome de arquivo
- Utilização da quebra de linha somente após pelo menos um caractere ser inserido
- Nos arquivos de entrada é importante que não tenha quebras de linha de nenhuma forma
- Códigos UART fora do padrão imposto pelo protocolo acarreta em um **Erro Fatal**

Tradução UART → ASCII: _decodeUART

```
char* origem
char* destino
int count = 9
char uart
char ascii

ENQUANTO count >= 0
        uart = [origem]
        origem = origem + 1
        SE count == 9 E uart != 0x30, ERRO
        SE count == 9 E uart == 0x30, LOOP
        SE count == 0 E uart != 0x31, ERRO
        SE count == 0 E uart == 0x31, BREAK
        SE uart < 0x30 OU uart > 0x31, ERRO

        uart = uart & 1
        ascii = ascii << (8 - count)
        uart = uart | ascii

        count = count -1

*destino = ascii
destino = destino + 1
```

Pedro Botelho e Ítalo Moraes
Bacharelando em Engenharia de Computação

Tradução ASCII → UART: _decodeCHAR

```
char* origem
char* destino
int count = 8
char ascii
char uart

*destino = 0x30
destino = destino + 1

ENQUANTO count > 0
        ascii = [origem]
        origem = origem + 1

        uart = ascii & 1
        uart = uart + 0x30
        *destino = uart
        destino = destino + 1

        ascii = ascii >> 1
        count = count -1

*destino = 0x31
destino = destino + 1
```

Pedro Botelho e Ítalo Moraes
Bacharelando em Engenharia de Computação

Conclusão

- Desenvolvedores:
  - Pedro Henrique Magalhães Botelho– 471047
  - Francisco Ítalo de Andrade Moraes - 472012
- Disciplina:
  - Arquitetura e Organização de Computadores II
- Orientador:
  - Roberto Cabral Rabêlo Filho
- Repositório no Github:
  - https://github.com/pedrobotelho15/UARTsim

- Obrigado pela atenção!