DATASHEET

MIPS MICROCONTROLLER

Luis F. de Deus[1] - 201520865

Tiago Knorst[2] - 201422292

[1]Centro de Tecnologia – Universidade Federal de Santa Maria (UFSM)
97.105-220 – Santa Maria – RS – Brasil

Engenharia de Computação

dedeus.f.l@gmail.com (1)

tiago.knorst@ecomp.ufsm.br (2)

SUMÁRIO

SUMÁRIO 2
1) *Overview* 3
1) Diagrama 4
2) Periféricos 5
a. Memória de Instruções (ROM) 5
b. Memória de Dados (RAM) 5
c. Porta de Entrada/Saída 6
d. *Programable Interrupt Controller* (PIC) 8
e. Bootloader 10
f. Transmissor Serial (TX) 11
g. Receptor Serial (RX) 11
h. Timer 13
3) Registradores do co-processador 15
4) Chamadas de Sistema 16
5) Tutorial 19
i. Geração de arquivos 21
ii. Configuração do terminal serial 23
iii. Configuração do modo de programação do microcontrolador 26
iv. Execução de uma aplicação 28

1) Overview

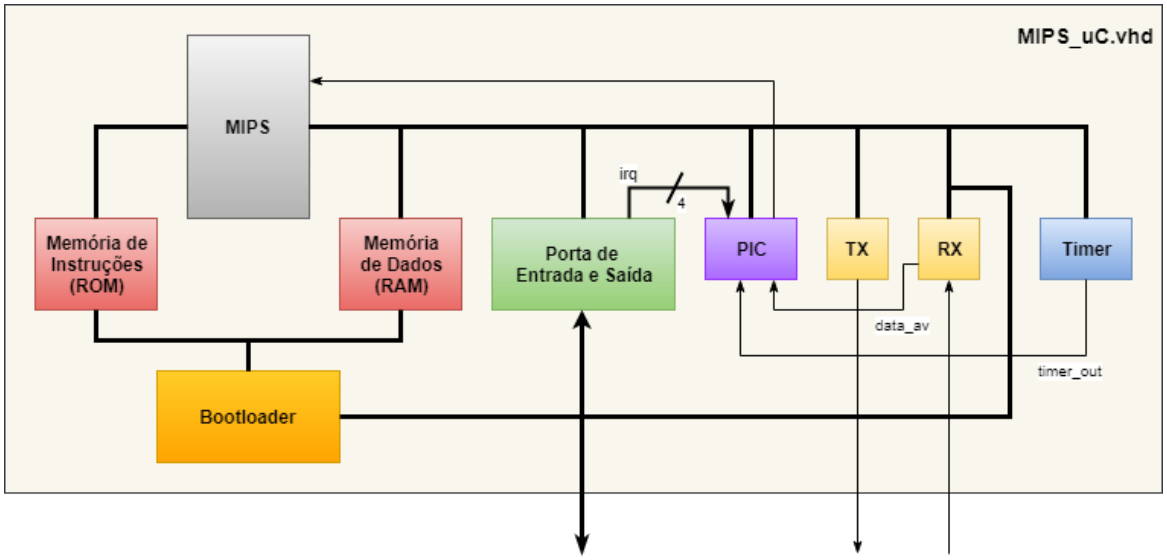


Figure 1 - Diagrama de blocos

Table 1 - Mapa de memória

Mapa de Memória		
Registrador	**Periférico**	**Endereço**
-	Memória de Dados	0x00000000 - 0x0fffffff
PORT_ENABLE	IO_PORT	0x10000000
PORT_CONFIG	IO_PORT	0x10000001
PORT_DATA	IO_PORT	0x10000002
PORT_IRQ	IO_PORT	0x10000003
IRQ_ID_ADDR	PIC	0x20000000
INT_ACK_ADDR	PIC	0x20000001
MASK_ADDR	PIC	0x20000002
TX	UART	0x30000000
RX	UART	0x40000000
RATE_FREQ_BAUD	UART	0x40000000
COUNTER	TIMER	0x50000000

2) Diagrama

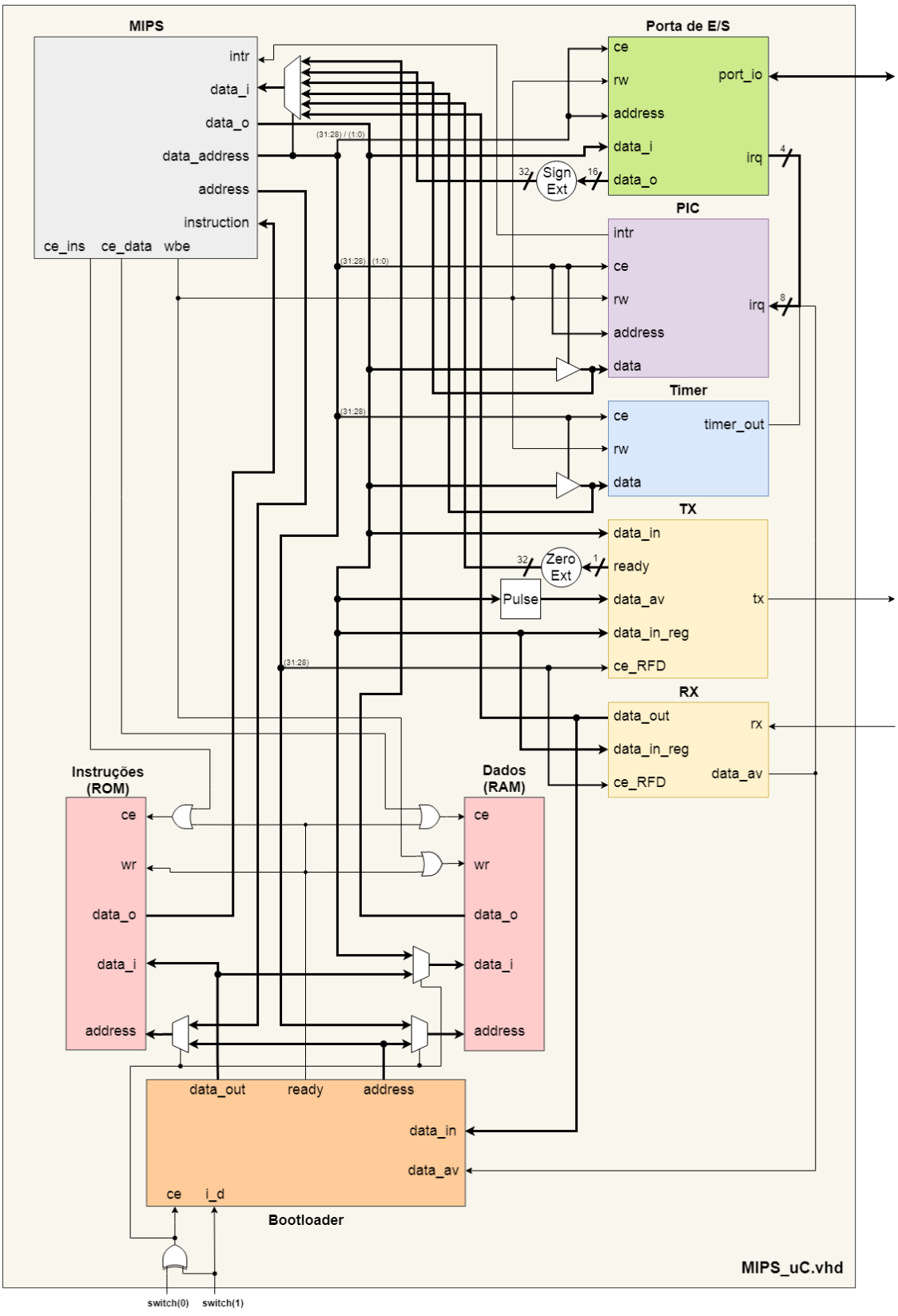


Figure 2 - Diagrama interno detalhado

3) Periféricos

Nesta seção serão abordados os periféricos do microcontrolador. É definido como periférico todo e qualquer bloco de circuito que envia ou recebe dados, ou interage com o processador, tipicamente os circuitos ficam na periferia do processador.

As questões abordadas serão como quais são os periféricos, suas funcionalidades, características, registradores e endereços. São eles:

- Memória de Instruções (ROM);
- Memória de Dados (RAM);
- Porta de Entrada/Saída;
- *Programable Interrupt Controller* (PIC);
- *Bootloader*;
- Transmissor Serial (TX);
- Receptor Serial (RX);
- *Timer*.

a. Memória de Instruções (ROM)

A memória de instruções implementada no microcontrolador é do tipo ROM (*Read-Only Memory*) em termos de execução, ela armazena as instruções referentes ao Kernel do microcontrolador e a aplicação do usuário, devidamente separados por áreas.

Possui palavras de 32 *bits* de informação, e o endereçamento é somente a *word*, não suporta endereçamento a *byte*. A escrita na memória se dá apenas quando o *bootloader* está ativo, o que é fora do modo de execução do microcontrolador, para isso o modo de programação deve estar selecionado, então será possível o carregamento de um novo conjunto de instruções, ou seja, uma nova aplicação no sistema.

b. Memória de Dados (RAM)

A memória de dados implementada no microcontrolador é do tipo RAM (*Random Access Memory*), possui suporte a leitura e escrita, ela armazena os dados do Kernel do microcontrolador e os dados da aplicação do usuário, devidamente separados por áreas.

Possui palavras de 32 *bits* de informação, e o endereçamento é somente a *word*, não suporta endereçamento a *byte*. A escrita da memória se dá de duas formas, através do processador ou através do *bootloader*, entretanto somente o processador pode ler dados da memória.

Por meio do processador, para leitura ou escrita faz-se uso de endereços com os primeiros quatro *bits* (31:28) formando o número zero (0) enquanto a escrita através do

bootloader somente quando o mesmo estiver ativo, no modo de programação, abaixo é mostrado um código exemplo da utilização da memória de dados.

```
   .text
   .globl  main
######################FUNÇÃO PRINCIPAL#######################
main:
#Leitura da memória
la $t0, 0x00000000    #Carrega em T0 endereço 0X0000000
lw $t1, ($t0)    #Carrega em T1 dado referente a posição 0X0000000 da memória

#Escrita na Memória
la $t0, 0x00000004   #Carrega em T0 endereço 0X0000004
sw $t1, ($t0)    #Escreve na posição 0X0000004 da memória o valor de T1
#########################################################################
.data
Ex: .word 4
#########################################################################
#End of Program
```

c. Porta de Entrada/Saída

Com a finalidade de receber ou mandar dados ao mundo exterior ao microcontrolador, foi implementada um sistema de entrada e saída (IO_PORT), contendo dezesseis portas bidirecionais, com suporte a interrupção, que irão trazer ou levar dados ao interior do microcontrolador.

O sistema de entrada e saída é configurável via *software*, para ter acesso ao periférico do IO_PORT deve-se usar endereços iniciados com os primeiros quatro *bits* (31:28) formando o número um (1). Então através de quatro registradores internos de largura de dezesseis *bits*, onde cada *bit* configura individualmente cada pino do microcontrolador assim será possível a configuração do IO_PORT, no fim desta seção um código exemplo é mostrado, são eles:

- PORT_ENABLE;
- PORT_CONFIG;
- PORT_DATA;

- PORT_IRQ.

O registrador PORT_CONFIG, é responsável pela configuração se o respectivo pino será de entrada ou saída do microcontrolador, ele é acessado através do endereço 0x10000001, onde efetuar a escrita neste endereço com o *bit* 1 será respectivo a configuração do pino como entrada, e escrever o *bit* 0 será respectivo a configuração de saída.

O registrador PORT_ENABLE, como o nome sugere, irá ser responsável por habilitar ou desabilitar o pino em questão, ele é acessado através do endereço 0x10000000, onde efetuar a escrita neste endereço com o *bit* 1 será respectivo a habilitar o pino, e escrever o *bit* 0 será respectivo a desabilitar o pino.

O registrador PORT_DATA, é responsável por levar ou trazer dados do microcontrolador, ele é acessado através do endereço 0x10000002 e é possível escrever qualquer dado nele com a instrução *store word* (sw), ou em caso de leitura, também é possível efetuar a leitura com a instrução *load word* (lw)

Por fim, o registrador PORT_IRQ, é responsável por habilitar as interrupções referentes ao pino em questão, ele é acessado através do endereço 0x10000003, onde efetuar a escrita neste endereço com o *bit* 1 será respectivo a habilitar a interrupção do pino, e escrever o *bit* 0 será respectivo a desabilitar a interrupção do pino, frisando que as interrupções são referentes ao um evento vindo de fora para o microcontrolador, então para fazer uso deste recurso é necessário o pino estar configurado como entrada.

```
.text
.globl  boot
.globl  main
.eqv        ADD_PORTIO_CONFIG 0x10000001
.eqv        ADD_PORTIO_ENABLE 0x10000000
.eqv        ADD_PORTIO_DATA 0x10000002
#------------------ INICIO DO KERNEL-----------------------------
boot:
            # PORT CONFIG
            li    $t1, 0xf000# Configura 4 bits mais significativos como entrada
            sw    $t1, ADD_PORTIO_CONFIG#    ||     12 bits menos significativos como saida
            # PORT ENABLE
            li    $t1, 0xffff     # Habilita todos os 16 bits da porta para o uso
            sw    $t1, ADD_PORTIO_ENABLE# Salva config. no end. do reg. de habilitação
            j     main
#------------------- FIM DO KERNEL--------------------------------
```

```
main:
        li    $t0, 5# Exemplo de dado a ser salvo
        sw    $t0, ADD_PORTIO_DATA     # Salva na porta de entrada e saída
        lw    $t0, ADD_PORTIO_DATA     # Carrega no reg. t0 o conteúdo da porta
        j     main
```

d. *Programable Interrupt Controller* (PIC)

Como anteriormente comentado, através do periférico de Entrada e Saída, é possível configurar alguns pinos como suporte a interrupção, no entanto faz-se necessário um bloco de *hardware* intermediário entre a entrada e o processador, com a finalidade de controlar caso houver mais de uma interrupção simultânea.

Pensando nisso foi implementado o controlador de interrupção programável (PIC), o qual tem por objetivo gerenciar as interrupções colocando prioridades, e também podendo mascarar uma determinada interrupção por um determinado tempo.

Com o mascaramento de uma ou mais interrupções é possível deixar um pedido de interrupção para ser tratado posteriormente, como por exemplo, uma seção crítica de código onde na maioria das vezes faz-se necessário mascarar várias ou todas as interrupções.

O PIC é acessado através de endereços iniciado com os primeiros quatro *bits* (31:28) formando o número dois (2), possuindo três registradores usados para a interação com o periférico, são eles:

- IRQ_ID_ADDR;
- INT_ACK_ADDR;
- MASK_ADDR.

O registrador IRQ_ID_ADDR é utilizado para armazenar o número da interrupção que foi gerada, ele é acessado através do endereço 0x20000000, o sistema deve fazer uma leitura deste número, salvando-o em algum registrador ou variável, para posteriormente enviar a sinalização que a determinada interrupção foi tratada.

O registrador INT_ACK_ADDR tem como finalidade sinalizar se a interrupção já foi ou não tratada, ele é acessado pelo endereço 0x20000001, onde o sistema deve fazer uma escrita com o número da interrupção, neste endereço para sinalizar ao PIC que a interrupção foi tratada.

E por fim o registrador MASK_ADDR tem como finalidade armazenar a configuração de quais interrupções estão habilitadas e quais estão desabilitadas (mascaradas), ele é acessado através do endereço 0x20000002, onde o usuário ou o sistema deve fazer uma escrita neste endereço com o *bit* 1 sinalizando quais interrupções ele deseja habilitar, as demais com o *bit* 0 irão ficar desabilitadas.

```
.text
.globl  boot
.globl  main


.eqv        ADD_PIC_IRQ_ID 0x20000000
.eqv        ADD_PIC_INT_ACK 0x20000001
.eqv        ADD_PIC_MASK 0x20000002
# INICIO DO KERNEL
boot:
        # ISR ADDRESS CONFIG
        la    $t0, InterruptionServiceRoutine
        mtc0  $t0, $31# Move endereço de memória do tratamento da ISR para o registrador da ISR c0[31]
        # PIC CONFIG
        li    $t0, 0xf0# Habilita bits de 7 a 4 e mascara bits de 3 a 0
        sw    $t0, ADD_PIC_MASK# Salva configuração no endereço respectivo ao registrador de mascara do PIC
        j     main


InterruptionServiceRoutine:
        #  [...]  SALVAMENTO DE CONTEXTO
        # LEITURA DO NUMERO DA INTERRUPCAO
        lw    $s0, ADD_PIC_IRQ_ID# Leitura do registrador de identificador da interrupção
        sw    $s0, irq_id_addr# Armazena identificador para posterior envio do ACK
        sll   $t0, $s0, 2# Parametriza indice da jump table (address = index * 4)
        lw    $s1, irq_handlers($t0)# Carrega no registrador s1 o endereço do handler da interrupção
        jalr  $s1# Salta para o endereço lido


        # ENVIO DO ACK APOS TRATAMENTO DA INTERRUPCAO
        lw    $s0, irq_id_addr# Carrega identificador da interrupçao
        sw    $s0, ADD_PIC_INT_ACK# Envia identificador para o acknowledgment da interrupção
        #  [...]  RETOMADA DE CONTEXTO
        eret
```

```
Handler_0:
        # CODIGO DO HANDLER 0
        jr    $ra
Handler_1:
        # CODIGO DO HANDLER 1
        jr    $ra
#  [...]  SUPORTE A ATÉ 8 HANDLERS
# FIM DO KERNEL
main:
        #  [...]  PROGRAMA DO USUARIO
.data
   irq_id_addr:.word 0
   irq_handlers:.word Handler_0 Handler_1# [...]
```

e. Bootloader

O periférico chamado de *bootloader* é responsável por fazer a troca das memórias, ou seja, o usuário deseja trocar da aplicação "A" para a aplicação "B", ele precisa carregar as instruções e os dados da nova aplicação nas respectivas memória de instrução e memória de dados.

A transferência das memórias se dará a partir de comunicação serial, por meio de um terminal que conte com a função de enviar arquivos (e.g. Hyperterminal), é indiferente a ordem de envio, mas é sugerido que se envie primeiro dados e depois instruções como boa prática. Além da comunicação serial será necessário o uso dos *slides switch's* e dos *push-button's* da FPGA. Para selecionar o modo de programação do microcontrolador faz-se necessário o uso das *slide switch's* onde a combinação "01" é usada para transferir a memória de dados, a combinação "10" é usada para transferir a memória de instruções, a combinação "11" é um *by-pass* usado para trocar entre as configurações e a combinação "00" é de execução normal do microcontrolador, sendo o LSB o *slide switch* T9 e o MSB o T10. Para a tarefa de envio é necessária um conjunto de etapas:

1. Selecionar o modo programação da memória de dados na placa, os *slides switch's* devem estar na combinação "01" para envio de dados;
2. Resetar o microcontrolador através do *push-button* B8;
3. Resetar o *bootloader* para inicializar o deslocamento de endereço, através do *push-button* D9*;*
4. Enviar via terminal serial o arquivo binário da memória de dados;

5. Atenção para o LED indicador na FPGA, quando o mesmo se apagar, a transmissão estará concluída;
6. Selecionar o modo *by-pass* com as *slides switch's* na combinação "11";
7. Selecionar o modo programação da memória de instruções, os *slides switch's* devem estar na combinação "10" para envio de instruções;
8. Resetar o *bootloader* para inicializar o deslocamento de endereço, através do *push-button* D9*;*
9. Enviar via terminal serial o arquivo binário da memória de instruções;
10. Atenção para o LED indicador na FPGA, quando o mesmo se apagar, a transmissão estará concluída;
11. Selecionar o modo de execução do microcontrolador com a combinação "00";
12. Resetar o microcontrolador através do *push-button* B8.

f. Transmissor Serial (TX)

Com a finalidade de uma melhor interação com o usuário, foi implementado no microcontrolador, o suporte a comunicação serial de oito *bits*, através do protocolo RS232. O protocolo RS232 é muito utilizado em comunicações em geral, o qual é caracterizado por enviar inicialmente um *start bit* (0), após é enviado os dados serialmente *Data* (8) e por fim encerra a comunicação enviando um *stop bit* (1).

O bloco TX é responsável pelo envio de dados serialmente para fora do microcontrolador, onde, por exemplo, será ligado a um bloco RX de um equipamento (e.g. Notebook). O periférico TX é acessado através de endereços com os primeiros quatro bits (31:28) formando o número três (3). É possível configurar a velocidade da comunicação serial via *software*, através de um registrador chamado RATE_FREQ_BAUD, que possui largura de 32 *bits*, este registrador é acessado através do endereço com os primeiros quatro bits (31:28) formando o número quatro (4).

O valor que é escrito no registrador RATE_FREQ_BAUD é definido pela razão entre o *clock* do periférico em Hertz e a velocidade da comunicação em *Baud Rate*, no fim da seção seguinte é abordado um código exemplo envolvendo a comunicação serial.

g. Receptor Serial (RX)

O bloco RX é responsável pela recepção de dados serialmente de um transmissor externo para dentro do processador, onde, por exemplo, um bloco TX de um equipamento (e.g. Notebook) esta enviando dados. O periférico RX possui uma saída denominada *data_av* que é conectada ao controlador programável de interrupções (PIC), ou seja, quando um dado válido chegou ao sistema, gera uma interrupção no microcontrolador, que é tratada assim que possível, e o usuário pode usufruir destes dados através de uma função do sistema (e.g. *read*).

O bloco RX é acessado através de endereços com os primeiros quatro *bits* (31:28) formando o número quatro (4). É possível configurar a velocidade da comunicação serial via *software*, através de um registrador chamado RATE_FREQ_BAUD, que possui largura de 32 *bits*, este registrador é acessado através do endereço com os primeiros quatro *bits* (31:28) formando o número quatro (4), a coincidência de endereços não causa problema devido ao fato de que, quando a instrução associada ao endereço for uma instrução de leitura o microcontrolador vai tratar como a leitura do barramento de saída do bloco RX, e quando for uma instrução de escrita, o microcontrolador vai tratar como configuração do registrador RATE_FREQ_BAUD.

O valor que é escrito no registrador RATE_FREQ_BAUD é definido pela razão entre o *clock* do periférico em Hertz e a velocidade da comunicação em *Baud Rate*.

```
.text
.globl  boot
.globl  main


.eqv        ADD_TX0x30000000
.eqv        ADD_UART0x40000000
.eqv        SPEED_UART0x00000208#   9600 bps -> 5Mhz/9600 = (208)16
#.eqv       SPEED_UART0x00000104#  19200 bps
# INICIO DO KERNEL
boot:
            la    $t0, InterruptionServiceRoutine
            mtc0  $t0, $31
            # SPEED UART CONFIG
            li $t1, SPEED_UART
            sw $t1, ADD_UART# Configuração da velocidade dos módulos TX e RX
            j main
InterruptionServiceRoutine:
            #  [...]  SALVAMENTO DE CONTEXTO
            #  [...]  IDENTIFICAÇÃO DA INTERRUPÇÃO
            jalr  $t0# Sub-rotina ao respectivo handler
            #  [...]  RETOMADA DE CONTEXTO
            eret
Handler_RX:
            lw    $t0, ADD_UART# Leitura do dado recebido via módulo RX
            sw    $t0, variable# Armazenamento do dado em uma variavel
            jr    $ra
# FIM DO KERNEL
```

```
main:
        li    $t0, 5# Dado a ser enviado via módulo TX
WaitReadyTX:
        lw    $t1, ADD_TX# Leitura do bit ready do módulo TX
        beq   $t1, $zero, WaitReadyTX# Pooling do bit ready
        sw    $t0, ADD_TX# Envio do dado
        j     main
.data
        variable:.word 0
        irq_handlers:.word Handler_RX  # Jump Table   [...]
```

h. Timer

Por fim, o ultimo periférico do microcontrolador é um bloco de *timer*, usado para implementar tarefas periódicas. O bloco foi implementado para efetuar a contagem até um valor previamente configurado pelo usuário, o *timer* possui uma saída denominada *time_out* conectada ao controlador programável de interrupções (PIC) assim que atingir o valor definido, acontecerá uma interrupção no microcontrolador que irá desviar sua rotina para tratar a tarefa periódica.

O bloco *timer* é acessado através de endereços com os primeiros quatro *bits* (31:28) formando o número cinco (5). Através do endereço 0x50000000 o usuário poderá configurar o valor da contagem do *timer* em ciclos, e através do endereço 0x50000001 o usuário conseguirá *resetar* o periférico *timer*. Inicialmente o usuário deve *setar* normalmente através do endereço 0x50000000 o valor do *set point* de contagem do *timer*, quando ocorrer a interrupção, é necessário *resetar* o periférico pelo endereço 0x50000001, e novamente configurar o valor o valor do *set point* de contagem.

```
.text
.globl  boot
.globl  main

.eqv    ADD_TIMER_DATA0x50000000
.eqv        ADD_TIMER_RESET0x50000001

# INICIO DO KERNEL
boot:
            # ISR ADDRESS CONFIG
            la    $t0, InterruptionServiceRoutine
            mtc0  $t0, $31
            j main

InterruptionServiceRoutine:
            #  [...]  SALVAMENTO DE CONTEXTO
            #  [...]  IDENTIFICAÇÃO DA INTERRUPÇÃO
            jalr  $t0# Sub-rotina ao respectivo handler
            #  [...]  RETOMADA DE CONTEXTO
            eret
Handler_Timer:
            sw    $zero, ADD_TIMER_RESET# Reseta timer
            #  [...]  CODIGO DA TAREFA TEMPORIZADA
            jr    $ra
# FIM DO KERNEL
main:
            li    $t0, 1000
            sw    $t0, ADD_TIMER_DATA# Configura timer para contar 1000 ciclos de clock
            j main

.data
            #Jump Table
            irq_handlers:.word Handler_Timer
```

4) Registradores do co-processador

Como anteriormente comentado, o microcontrolador tem suporte a tratamento de interrupções, que são eventos externos, ou seja, vindos de fora do microcontrolador (e.g. Apertar de um botão), mas também o microcontrolador dá suporte a eventos internos (e.g. *overflow*).

Com a finalidade de ajudar no tratamento das rotinas de interrupção e exceção, foram adicionados alguns registradores extras no microcontrolador, estes registradores possuem funções específicas dentro do tratamento das interrupções e exceções e por isso são classificados como sendo parte de um co-processador e por executarem funções de *Kernel* são denominados, parte do co-processador 0, ou C0.

Foram implementados quatro registradores no co-processador 0, todos eles de largura de 32 *bits*, são eles:

- EPC_REGISTER;
- ISR_AD_REGISTER;
- ESR_AD_REGISTER;
- CAUSE_REGISTER.

O registrador EPC_REGISTER, deriva do nome *Exception PC*, e é mapeado como sendo o registrador 14 do co-processador 0 (C0[14]), ele tem a função de armazenar o endereço de retorno, da interrupção ou da exceção, ou seja, quando houver uma interrupção ou exceção primeiramente o microcontrolador salva o PC atual no EPC_REGISTER, para então desviar para a rotina de tratamento.

O registrador ISR_AD_REGISTER, deriva do nome *Interruption Service Routine,* e é mapeado como sendo o registrador 31 do co-processador 0 (C0[31]), ele tem a função de armazenar o endereço de início da rotina de tratamento das interrupções, ou seja, quando houver uma interrupção o desvio do programa acontecerá para o endereço que estiver no registrador ISR_AD_REGISTER.

O registrador ESR_AD_REGISTER, é o complementar do anterior, para tratamento das exceções, deriva do nome *Exception Service Routine,* e é mapeado como sendo o registrador 30 do co-processador 0 (C0[30]), ele tem a função de armazenar o endereço de início da rotina de tratamento das exceções, ou seja, quando houver uma exceção o desvio do programa acontecerá para o endereço que estiver no registrador ESR_AD_REGISTER.

Por fim, o registrador CAUSE_REGISTER, como o nome sugere é um registrador de “causa”, ele é mapeado como sendo o registrador 13 do co-processador 0 (C0[13]), possui a função de armazenar a “causa” da exceção, todas as exceções mapeadas pelo microcontrolador possuem um número característico são elas:

- 1: *Invalid Instruction*;
- 8: SYSCALL (instrução);
- 12: *Overflow* em complemento de 2;
- 15: *Division-by-zero.*

Através do número contido no registrador CAUSE_REGISTER, é possível saber o motivo que gerou a exceção, e então tratá-la adequadamente.

5) Chamadas de Sistema

O microcontrolador, possui em uma área de memória específica, o seu *Kernel* implementado, o qual consiste em algumas funções que não são diretamente acessíveis ao usuário, que faz seu uso restritamente a partir das chamadas de sistema (e.g. *syscall*).

No *Kernel* do microcontrolador foram implementadas cinco funções do sistema, são elas:

- *void PrintString(char *string);*
- *char IntegerToString(int n)*;
- *char IntToHexString(int n)*;
- *int Read(char *buffer, int size)*;
- *int StringToInteger(char *string)*;

A função *PrintString*, recebe como paramento o ponteiro de uma *string*, com o ponteiro tem-se o endereço inicial da *string* então é possível percorrer até o final que é indicado pelo caractere "\0". A função *PrintString* tem por objetivo enviar todos os caracteres de uma *string* para o modulo de transmissão serial (TX), quando o usuário chamar a *PrintString*, a função irá ficar em *loop* enviando os caracteres da *string* para o modulo TX, até que encontre o caractere "\0" que indica o fim da *string*, e consequentemente o fim do envio, a função não tem retorno.

A função *IntegerToString*, como o nome sugere, tem por objetivo converter um número inteiro, no seu equivalente em ASCII, cada número possui um equivalente em ASCII, o propósito da *IntegerToString* é retornar ao usuário uma *string* formada pelos equivalentes ASCII do número passado como parâmetro, ela é comumente utilizada quando é necessário fazer uma *PrintString* de um número.

Do que se refere a função *IntToHexString*, ela é uma evolução da função *IntegerToString*, esta função recebe como parâmetro um número inteiro, e da suporte a números de 0 até 15 como um único caractere ASCII (e.g. 12 -> C -> ASCII). A função então tem por objetivo receber um número inteiro, e retornar ao usuário uma *string* formada pelos equivalentes ASCII, dando suporte a números na base 16 (hexadecimais).

Uma das funções mais comuns nos microcontroladores são as funções de leitura (e.g. *scanf*), no microcontrolador MIPS foi implementado a função *Read*, que tem por objetivo efetuar a leitura do que foi recebido no modulo de recepção serial (RX) (e.g. *Keyboard Read*). A função que recebe como parâmetros, um ponteiro para uma *string* aonde vai ser copiado o conteúdo recebido e um inteiro com o número de caracteres que devem ser copiados, ao ser chamada a função verifica se a tecla <ENTER> já foi pressionada, caso contrário a função retorna zero, caso afirmativo, o <ENTER> já foi pressionado, inicia-se a cópia do que o usuário digitou, para a *string* passada por parâmetro, a copia termina caso chegue no número de caracteres que o programador solicitou via parâmetro ou caso a função encontre o caractere "\0" indicando o fim dos dados.

Por fim, a última função do sistema é a *StringToInteger*, como o nome sugere é a inversa da *IntegerToString*, ou seja, tem por objetivo converter uma *string*, no seu equivalente numérico inteiro. Comumente utilizada após a função *Read*, devido ao fato de que a recepção de dados, sejam eles números ou *strings*, chegam ao microcontrolador todos codificados em ASCII, então para fazer uso de números em outras partes do código, faz-se necessário a conversão para inteiro. A função *StringToInteger* recebe como parâmetro um ponteiro de uma *string*, e retorna ao usuário o equivalente numérico inteiro.

Abaixo é mostrado um código exemplo, contendo tudo que é necessário para chamar as funções do sistema.

```
.text
.globl  boot
.globl  main


# INICIO DO KERNEL
boot:
        # ESR ADDRESS CONFIG
        la    $t0, ExceptionServiceRoutine
        mtc0  $t0, $30# Move endereço de memória do tratamento da ESR para o registrador da ISR c0[30]
        j main


ExceptionServiceRoutine:
        #  [...]  TRATAMENTO DAS EXCEÇÕES E SYSCALLS
        eret
# FIM DO KERNEL
```

```
main:
	# PRINT STRING
	la	$a0, string# Parâmetro da função = endereço da string
	li	$v0, 0# Parâmetro para selecionar a função PrintString
	syscall

	# INTEGER TO STRING
	lw	$a0, integer# Parâmetro da função = inteiro a ser convertido
	li	$v0, 1# Parâmetro para selecionar a função IntegerToString
	syscall

	# INTEGER TO HEXADECIMAL STRING
	lw	$a0, integer# Parâmetro da função = inteiro a ser convertido
	li	$v0, 2# Parâmetro para selecionar a função IntToHexString
	syscall

	# READ
while_read_zero:
	la	$a0, string# Parâmetro 1 da função = endereço da string
	li	$a1, 6# Parâmetro 2 da função = tamanho da string
	li	$v0, 3# Parâmetro para selecionar a função Read
	syscall

	beq	$v0, $zero, while_read_zero

	# STRING TO INTEGER
	lw	$a0, string# Parâmetro da função = endereço da string
	li	$v0, 4# Parâmetro para selecionar a função StringToInteger

	syscall

	j main
.data
	integer:.word 5
	string:.asciiz "teste"
```

6) Tutorial

Referindo-se à pasta do projeto, denominada MIPS_MICROCONTROLLER, dentro dela encontra-se o arquivo "mips_uc.bit" usado para carregar o microcontrolador na FPGA e outras duas pastas, denominadas "sim" e "src".

Na pasta "sim" encontram-se os arquivos de simulação do microcontrolador, como o *test bench* e arquivos texto, já na pasta "src" encontram-se os arquivos de código fonte do projeto, usados na implementação, as pastas "ASM" e "VHDL" e o arquivo "pinos.ucf" que contém a declaração dos pinos da FPGA que se conectam no microcontrolador (e.g. *clock*, IO_PORT).

Dentro da pasta "ASM" encontra-se os arquivos binários e o código fonte *assembly*, já dentro da pasta "VHDL" encontram-se as descrições VHDL dos periféricos do processador (e.g. UART_TX), a pasta "Memory" que contém as descrições VHDL utilizadas para fazer as memórias, a pasta "PIC" que contém as descrições VHDL usadas para o Controlador de Interrupções Programável, e por fim a pasta "MIPS" que contém as descrições VHDL do processador MIPS, na Figure 3 é possível ver a hierarquia de diretórios.

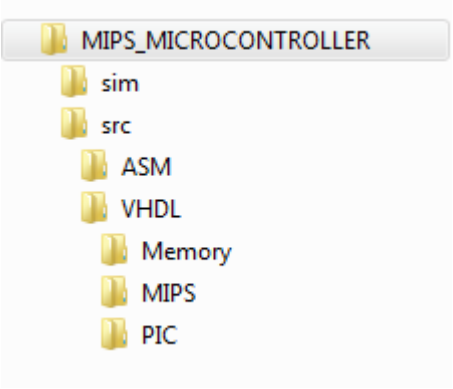
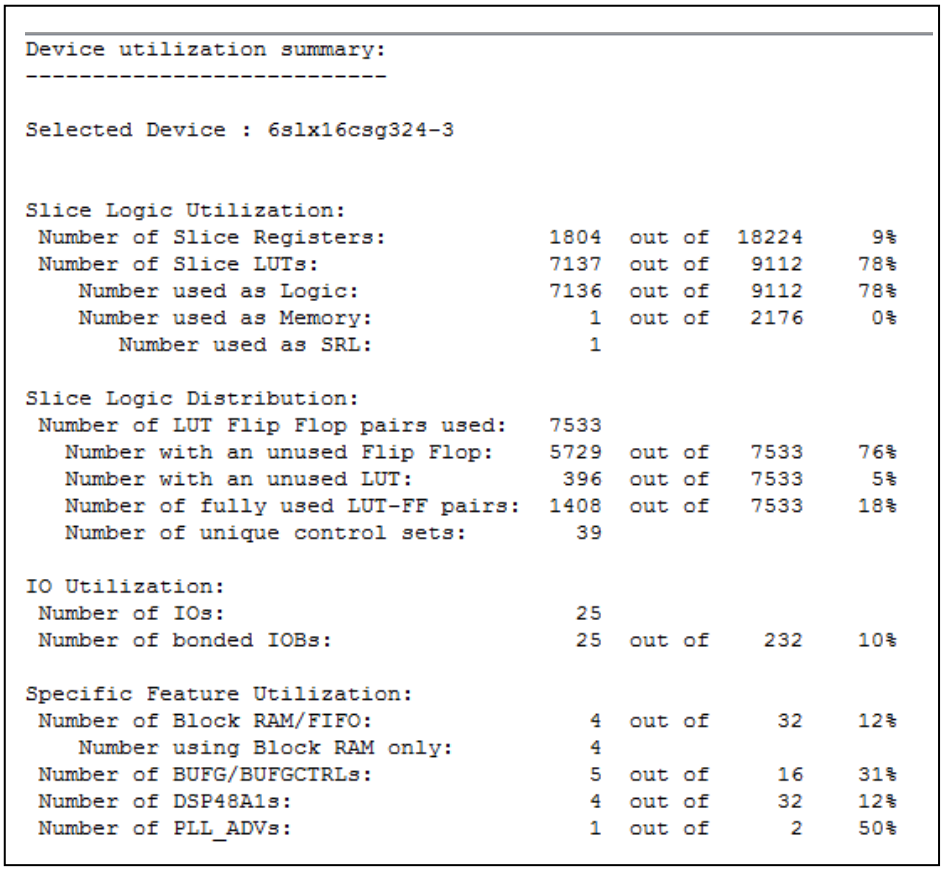


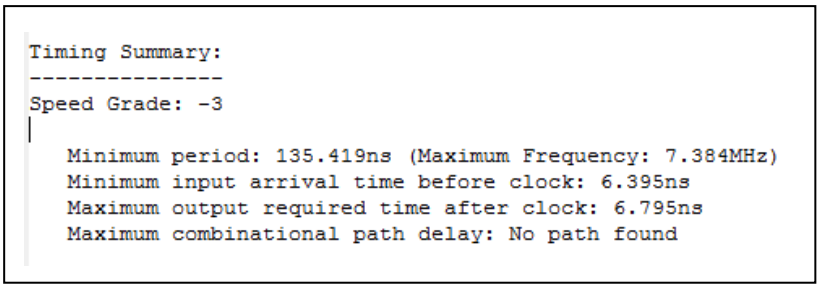
Figure 3 - Árvore de diretórios

O projeto do microcontrolador foi sintetizado na ferramenta ISE Design Suite versão 14.7 da empresa Xilinx, o microcontrolador possui uma frequência máxima de 7 MHz, entretanto usualmente foi projetado para operar em 5 MHz de *clock*. Figure 4 e Figure 5 é possível analisar os *reports* de síntese, referentes à área utilizada (*Device Utiliztion*) e a frequência (*Timing Summary*) sucessivamente.



```
Device utilization summary:
---------------------------

Selected Device : 6slx16csg324-3


Slice Logic Utilization:
 Number of Slice Registers:            1804  out of  18224     9%
 Number of Slice LUTs:                 7137  out of   9112    78%
    Number used as Logic:              7136  out of   9112    78%
    Number used as Memory:                1  out of   2176     0%
       Number used as SRL:                1

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:   7533
   Number with an unused Flip Flop:    5729  out of   7533    76%
   Number with an unused LUT:           396  out of   7533     5%
   Number of fully used LUT-FF pairs:  1408  out of   7533    18%
   Number of unique control sets:        39

IO Utilization:
 Number of IOs:                          25
 Number of bonded IOBs:                  25  out of    232    10%

Specific Feature Utilization:
 Number of Block RAM/FIFO:                4  out of     32    12%
    Number using Block RAM only:          4
 Number of BUFG/BUFGCTRLs:                5  out of     16    31%
 Number of DSP48A1s:                      4  out of     32    12%
 Number of PLL_ADVs:                      1  out of      2    50%
```

Figure 4 - Device Utilization



```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 135.419ns (Maximum Frequency: 7.384MHz)
   Minimum input arrival time before clock: 6.395ns
   Maximum output required time after clock: 6.795ns
   Maximum combinational path delay: No path found
```

Figure 5 - Timing Summary

Para melhor exemplificar a utilização dos recursos do microcontrolador, será abordado nesta seção um tutorial passo-a-passo da utilização de uma aplicação previamente desenvolvida.

A aplicação tem por objetivo a interação com o usuário, para isso será utilizada a comunicação serial, onde o usuário enviará informações ao microcontrolador através de um terminal serial, que para demonstração será utilizado o HyperTerminal, ou qualquer outro terminal serial que tenha suporte a enviar arquivos (e.g. GTK Terminal). Será

necessário uma série de etapas para a configuração do terminal e da programação da FPGA, que será considerada o modelo Nexys 3 ilustrada na Figure 6.

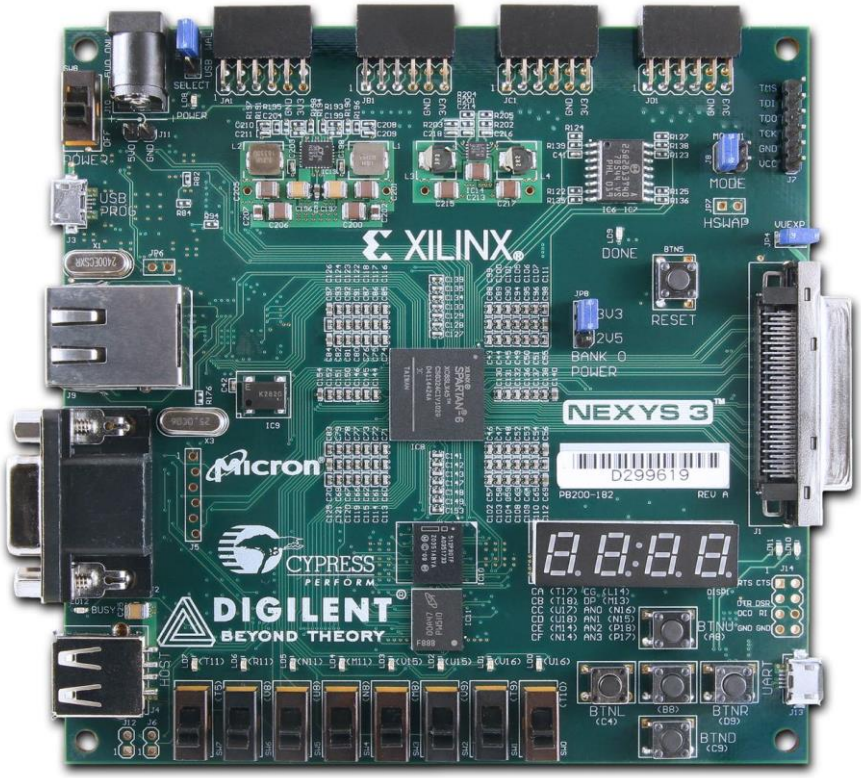


Figure 6 - FPGA Nexys 3

i. Geração de arquivos

Primeiramente é necessária a criação dos arquivos binários que contém as instruções e os dados da aplicação. Com o código aberto no MARS, selecione a opção marcada em vermelho na Figure 7.

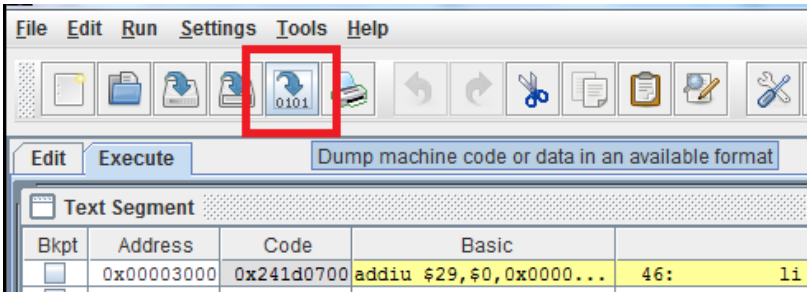


Figure 7 - Geração de arquivos de memória

Após abrir a caixa de dialogo selecione “.text” para instruções (cuidando o *range* de endereços) e o formato *binary* como ilustra a Figure 8, e clique em *dump to file*.

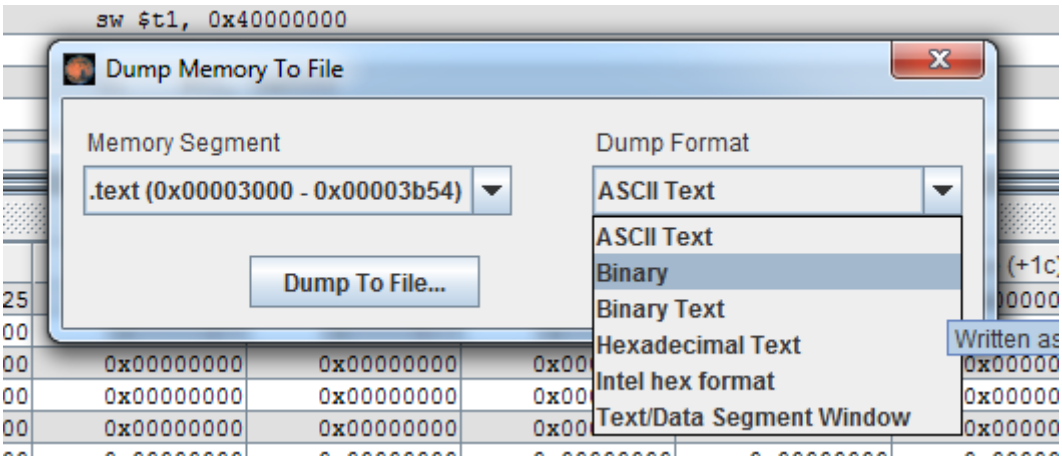


Figure 8 - Configuração de geração

Após abrir a janela de salvamento, salve o arquivo colocando o nome de sua preferência seguido da extensão “.bin” como ilustra a Figure 9.

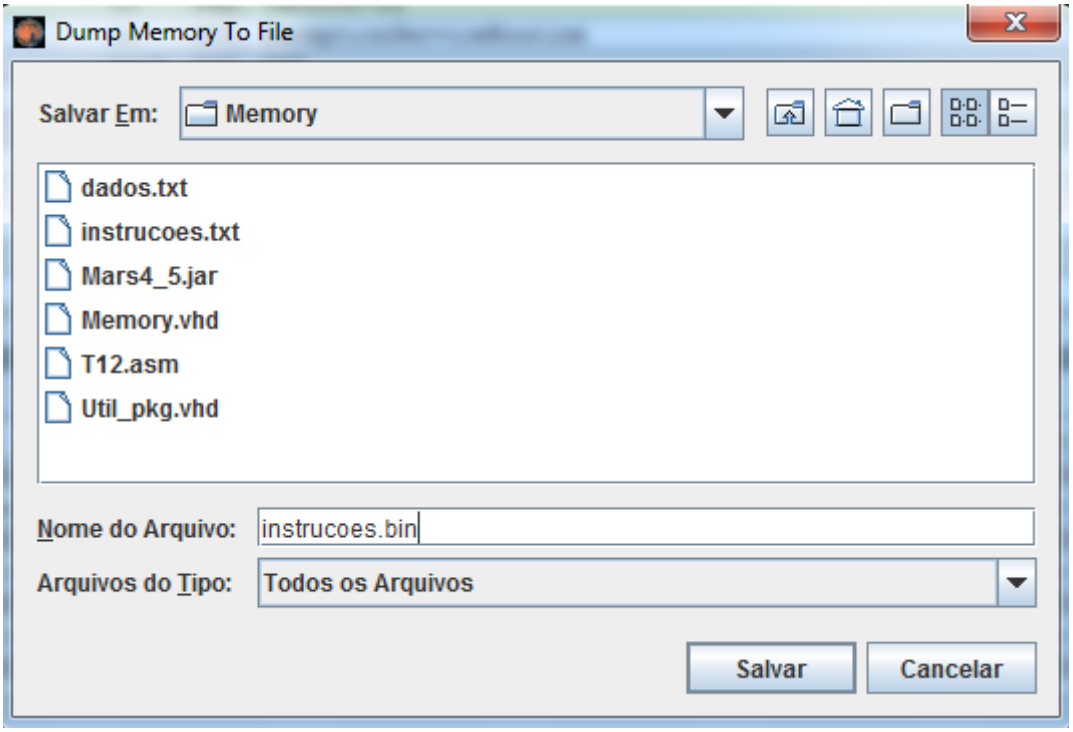


Figure 9 - Salvando arquivo

Repita o processo para os dados, alterando conforme a Figure 10 demonstra.

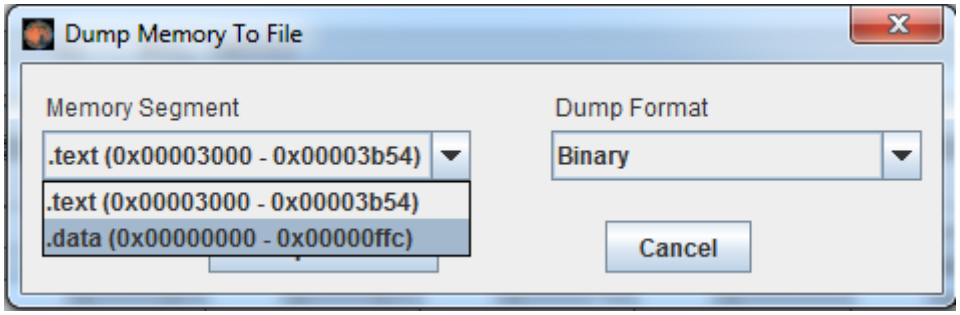


Figure 10 - Gerando memória de dados

ii. Configuração do terminal serial

Com os arquivos binários gerados, é necessário a configuração do terminal serial que irá fazer a transferência da aplicação para o microcontrolador. Como anteriormente mencionado será abordado a configuração do HyperTerminal Figure 11.

Figure 11 - Ícone HyperTerminal

Após abrir o programa, irá aparecer uma caixa de diálogo, é necessário colocar um nome e um ícone para a conexão, como mostra a Figure 12, entretanto, ambos não afetam em nada.

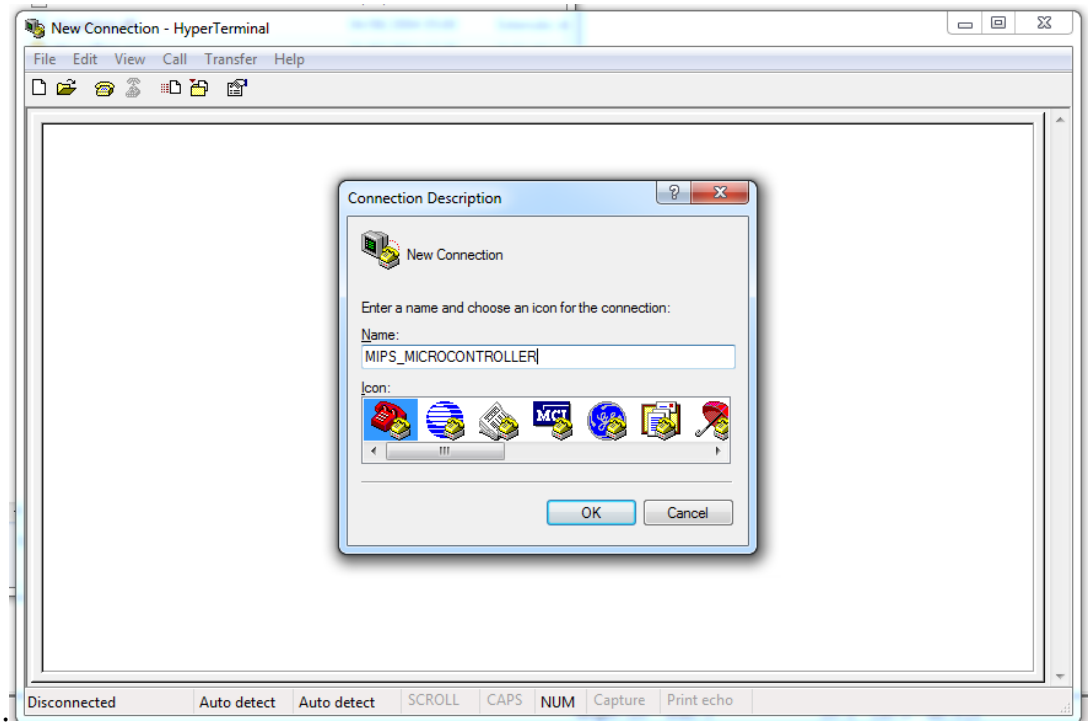


Figure 12 - Iniciando conexão

A próxima configuração, diz respeito a porta USB que está conectada a FPGA, abrindo o gerenciador de dispositivos é possível ver em qual porta encontra-se a FPGA, como ilustra a Figure 13 selecione a porta COM que está conectada a FPGA.

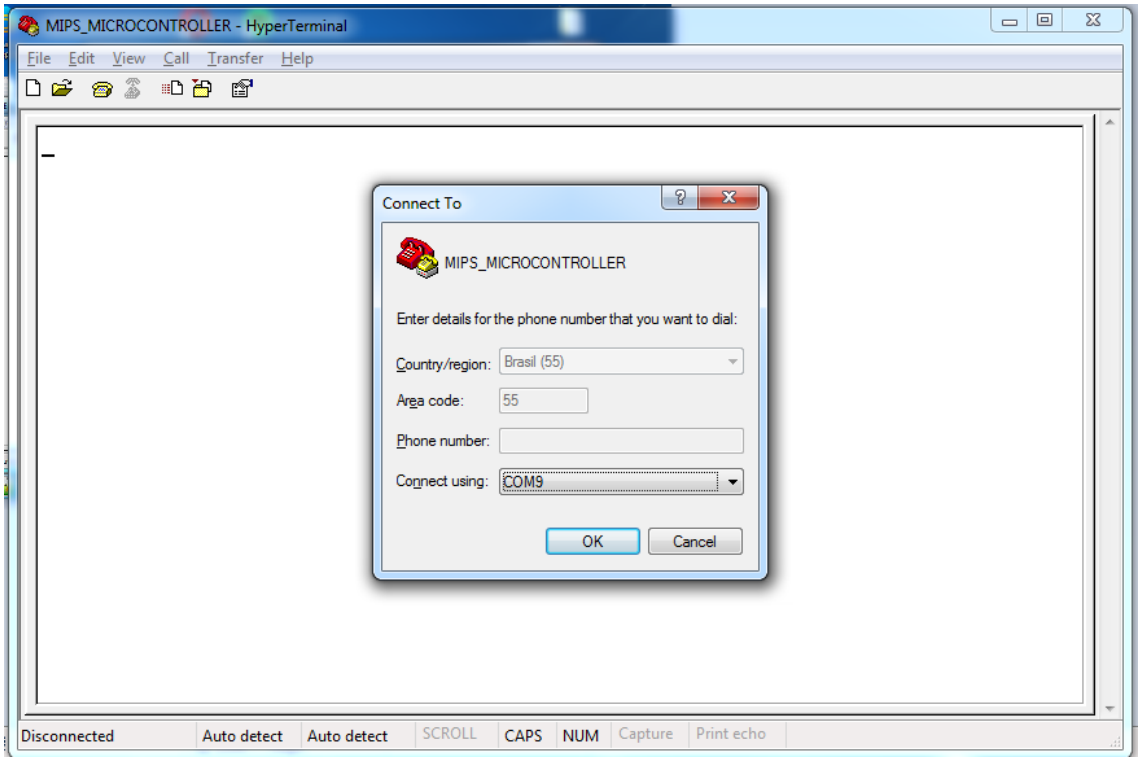


Figure 13 - Configuração de porta COM

Prosseguindo, as configurações da porta, em “Bits por segundo” selecione 9600, que é referente a velocidade de comunicação, em “Bits de dados” selecione 8, referente a quantos bits o programa deve enviar, obviamente é necessário estar igual ao suporte do microcontrolador, que recebe 8 bits por vez, ou 1 byte, em “bits de parada” selecione 1, referente ao protocolo de comunicação, e por fim em “Controle de fluxo” selecione nenhum, como ilustra a Figure 14.

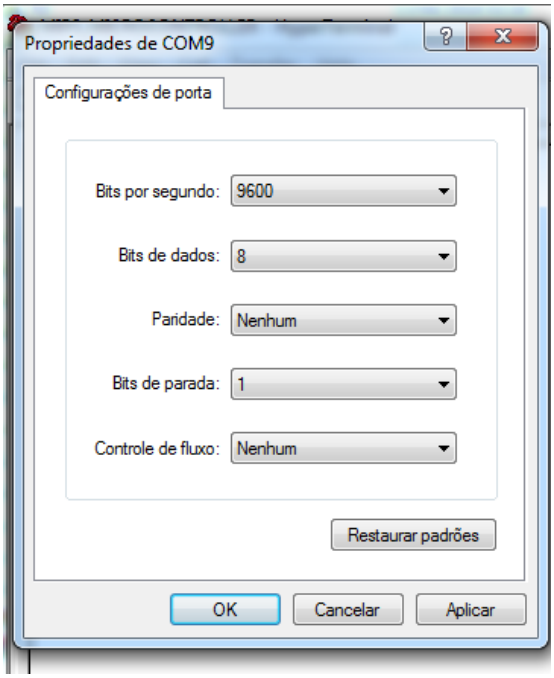


Figure 14 - Configuração da comunicação

Feito isso, o terminal serial estará apto a fazer comunicação com o microcontrolador.

iii. Configuração do modo de programação do microcontrolador

Com o link de comunicação configurado e pronto, é possível transferir os arquivos binários da aplicação via terminal, para o microcontrolador. Primeiramente, é necessário entrar no modo de programação do microcontrolador.

1º. Selecionar o modo programação da memória de dados na placa, os *slides switch's* devem estar na combinação "01", onde P0 deve estar em 0 e P1 deve estar em 1, como ilustra a Figure 15, para envio de dados;

Figure 15 - Slides Switch's FPGA

2º. *Resetar* o microcontrolador através do *push-button* "RM" indicado em vermelho na Figure 16;

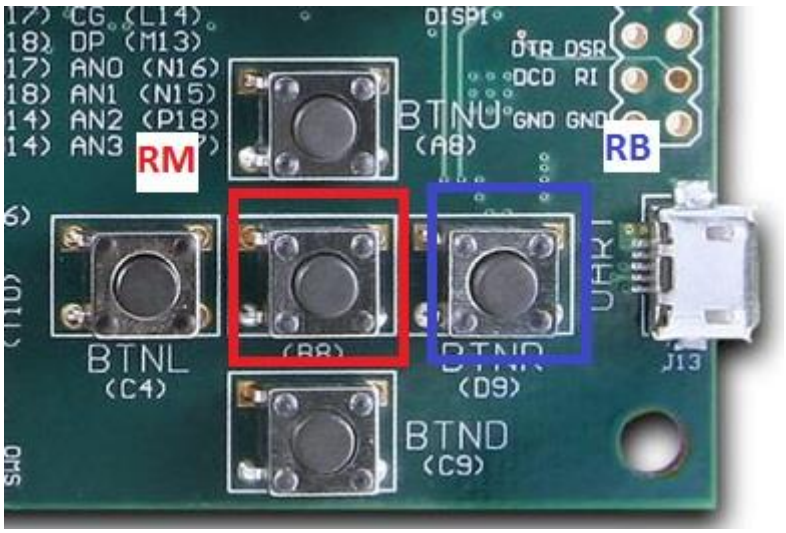


Figure 16 - Push-Button's FPGA

3º. Resetar o *bootloader* para inicializar o deslocamento de endereço, através do *push-button*, "RB" indicado em azul na Figure 16*;*

4º. Enviar via terminal serial o arquivo binário da memória de dados, como ilustra a Figure 17;

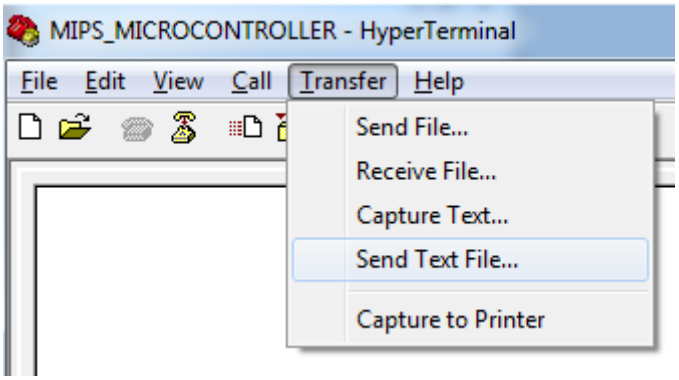


Figure 17 - Enviando arquivos via terminal

5º. Atenção para o LED indicador na FPGA, quando o mesmo se apagar, a transmissão estará concluída;

6º. Selecionar o modo *by-pass* com as *slides switch's* na combinação "11";

7º. Selecionar o modo programação da memória de instruções, os *slides switch's* devem estar na combinação "10", onde P0 deve estar em 1 e P1 deve estar em 0, para envio de instruções;

8º. *Resetar* o *bootloader* para inicializar o deslocamento de endereço, através do *push-button* "RB" indicado em azul na Figure 16;

9º. Enviar via terminal serial o arquivo binário da memória de instruções;

10º. Atenção para o LED indicador na FPGA, quando o mesmo se apagar, a transmissão estará concluída;

11º. Selecionar o modo de execução do microcontrolador com a combinação "00";

12º. *Resetar* o microcontrolador através do *push-button*, "RM" indicado em vermelho na Figure 16.

iv. Execução de uma aplicação

Ao resetar o microcontrolador no passo 12, como a aplicação carregada tem o objetivo de interação com o usuário, será voltada para o algoritmo *bubble sort*, que tem por objetivo ordenar um numero "x" de elementos de um *array*.

Inicialmente na tela do terminal irá aparecer a mensagem "Informe a String: ", solicitando ao usuário que informe uma *string* qualquer de até 80 caracteres, após digitar e ao apertar a tecla <ENTER> o microcontrolador retornará a *string* invertida, e informará a mensagem "Número de Elementos: " requisitando ao usuário que lhe informe o número de elementos (números), que farão parte da aplicação, ao digitar (e.g. 11) e apertar a tecla <ENTER>, o programa irá informar a mensagem "Ordenação (0=Decr / 1=Cres): " solicitando ao usuário que informe o tipo de ordenação, decrescente ou crescente, depois de escolher e apertar <ENTER> o usuário poderá começar a digitar os elementos que compõem o *array*, um número de cada vez, por fim, ao informar o último número, o microcontrolador irá retornar ao terminal, o *array* inicial que foi digitado pelo usuário, e o *array* ordenado, e voltará para o início da aplicação, a Figure 18 exemplifica uma execução.

```
Informe a String: All alone now Except for the memories Of what we had and what
we knew
String Invertida: wenk ew tahw dna dah ew tahw fO seiromem eht rof tpecxE won en
ola llA

Numero de Elementos: 11
Ordenacao (0=Decr / 1=Cres): 1
Array[0]: 4654
Array[1]: 1324
Array[2]: 7563
Array[3]: 1237
Array[4]: 6954
Array[5]: 1438
Array[6]: 4218
Array[7]: 32
Array[8]: 488
Array[9]: 3244
Array[10]: 1987
Array Inicial: 4654 1324 7563 1237 6954 1438 4218 32 488 3244 1987
Array Ordenado: 32 488 1237 1324 1438 1987 3244 4218 4654 6954 7563

Informe a String: _
```

Figure 18 - Exemplo de Aplicação