MANUAL DO R8_uC, MICROCONTROLADOR COM NÚCLEO R8

ELC1094 - PROJETO DE PROCESSADORES

Julio Vicenzi
Victor O. Costa

Professor: Everton A. Carara

Santa Maria, 12/07/2019

Sumário

Visão geral **2**

Arquitetura expandida - Tabela de instruções **3**
Registradores especiais 4

Periféricos **7**
PortA - GPIO (Endereço de periférico: 0x0) 7
PIC - (Endereço de periférico: 0x1) 9
TX - Transmissor serial UART (Endereço de periférico 0x2) 11

Chamadas de sistema **15**
print_string 15
integer_to_string 16
integer_to_hexstring 16
string_to_integer 17
read_buffer 17

Tutorial **18**
Organização do diretório 18
Report de síntese 18
Utilização de recursos 18
Temporização 19
Guia passo-a-passo 20
Configuração do Terminal 20
Programação serial 20
Executando a aplicação exemplo 20

Visão geral

O microcontrolador R8_uc é formado por 6 periféricos e um processador, usando a placa FPGA nexys 3 para a sua prototipação e implementação. A arquitetura é baseado no R8 original de 16 bits, com uma expansão para o suporte de interrupções de hardware e software, assim como operações de multiplicação e divisão.

Para entrada e saída de dados, o microcontrolador usa uma porta serial. É possível carregar programas através dela, quando o microcontrolador se encontra em modo de programação.

O microcontrolador suporta 16 pinos de entrada e saída que podem ser configurados via software, possibilitando a habilitação de uso, modo de entrada e geração de interrupção individualmente para cada pino.

O controlador de interrupção com prioridade (PIC) tem a função de criar uma interface fácil para o tratamento de múltiplas interrupções em fila em ordem de prioridade, assim como habilitando e desabilitando interrupções específicas.

A frequência do clock é de 25 MHz e a memória RAM suporta 65534 KB (32767 palavras de 16 bits).

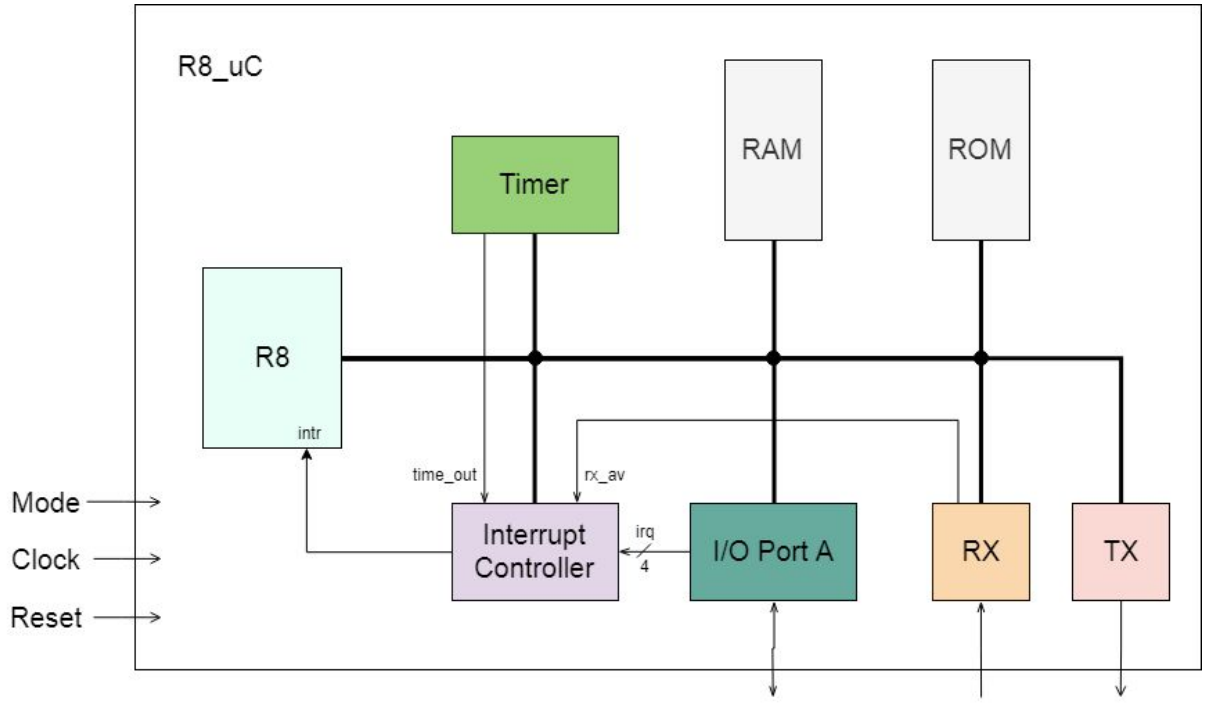

Figura 1. Esquemático geral do microcontrolador.

Arquitetura expandida - Tabela de instruções

Instrução	FORMATO DA INSTRUÇÃO				
	15 – 12	**11 - 8**	**7 - 4**	**3 - 0**	**AÇÃO ; *flags***
ADD Rt, Rs1,Rs2	**0**	R *target*	R source1	R source2	Rt ← Rs1 + Rs2; *wnz*; *wcv*
SUB Rt, Rs1,Rs2	**1**	R *target*	R source1	R source2	Rt ← Rs1 - Rs2; *wnz*; *wcv*
AND Rt, Rs1,Rs2	**2**	R *target*	R source1	R source2	Rt ← Rs1 *and* Rs2; *wnz*
OR Rt, Rs1,Rs2	**3**	R *target*	R source1	R source2	Rt ← Rs1 *or* Rs2; *wnz*
XOR Rt, Rs1,Rs2	**4**	R *target*	R source1	R source2	Rt ← Rs1 *xor* Rs2; *wnz*
ADDI Rt, *cte8*	**5**	R *target*	Constante (*unsigned*)		Rt ← Rt + ("00000000" & constante); *wnz*; *wcv*
SUBI Rt, *cte8*	**6**	R *target*	Constante (*unsigned*)		Rt ← Rt - ("00000000" & constante); *wnz*; *wcv*
LDL Rt, *cte8*	**7**	R *target*	Constante		Rt ← RtH & constante
LDH Rt, *cte8*	**8**	R *target*	Constante		Rt ← constante & RtL
LD Rt, Rs1,Rs2	**9**	R *target*	R source1	R source2	Rt ← PMEM (Rs1+Rs2)
ST Rt, Rs1,Rs2	**A**	R *target*	R source1	R source2	PMEM (Rs1+Rs2) ← Rt
SL0 Rt, Rs1	**B**	R *target*	R source1	**0**	Rt ← Rs1[14:0] & 0 ; *wnz*
SL1 Rt, Rs1	**B**	R *target*	R source1	**1**	Rt ← Rs1[14:0] & 1 ; *wnz*
SR0 Rt, Rs1	**B**	R *target*	R source1	**2**	Rt ← 0 & Rs1 [15:1]; *wnz*
SR1 Rt, Rs1	**B**	R *target*	R source1	**3**	Rt ← 1 & Rs1 [15:1]; *wnz*
NOT Rt, Rs1	**B**	R *target*	R source1	**4**	Rt ← not (Rs1) ; *wnz*
NOP	**B**	-	-	**5**	nenhuma ação
HALT	**B**	-	-	**6**	suspende seqüência de ciclos de busca e execução
LDSP Rs1	**B**	**0**	R source1	**7**	SP ← Rs1 (inicializa o apontador de pilha)
LDISRA Rs1	**B**	1	R source1	**7**	regISR SP ← Rs1 (inicializa o apontador de ISR)
LDTSRA Rs1	**B**	2	R source1	**7**	regTSR SP ← Rs1 (inicializa o apontador de TSR)
RTS	**B**	-	-	**8**	PC ←PMEM(SP+1) ; SP←SP+1
RTI	**B**	-	**0**	**B**	PC ←PMEM(SP+1) ; SP←SP+1; InterruptStatus ←0
POP Rt	**B**	R *target*	-	**9**	Rt ←PMEM(SP+1) ; SP←SP+1
POPF	**B**	-	**2**	**B**	regFlags ← PMEM(SP+1) ; SP←SP+1
PUSH Rt	**B**	R *target*	-	**A**	PMEM(SP)←Rt ; SP←SP-1
PUSHF	**B**	-	**1**	**B**	PMEM(SP)← x"000" & regFlags; SP←SP-1
JUMP_INTR	**B**	**0**	**F**	**B**	PMEM(SP)←PC ; SP←SP-1; PC← regISR; InterruptStatus ←1
MFH Rt	**B**	R *target*	**3**	**B**	*Rt ← regHIGH*
MFL Rt	**B**	R *target*	**4**	**B**	*Rt ← regLOW*

MFC Rt	**B**	R *target*	**5**	**B**	*Rt causeRegister*
SWI	**B**	**0**	**E**	**B**	PMEM(SP)PC ; SPSP-1; PC regTSR; trapStatus 1; regCause 8;
JMPR Rs1	**C**	0	R source1	**0**	PC PC + Rs1 (não depende de flag de estado)
JMPNR Rs1	**C**	0	R source1	**1**	if (*n* =1) PC PC + Rs1
JMPZR Rs1	**C**	0	R source1	**2**	if (*z* =1) PC PC + Rs1
JMPCR Rs1	**C**	0	R source1	**3**	if (*c* =1) PC PC + Rs1
JMPVR Rs1	**C**	0	R source1	**4**	if (*v* =1) PC PC + Rs1
JMP Rs1	**C**	0	R source1	**5**	PC Rs1 (não depende de flag de estado)
JMPN Rs1	**C**	0	R source1	**6**	if (*n* =1) PC Rs1
JMPZ Rs1	**C**	0	R source1	**7**	if (z =1) PC Rs1
JMPC Rs1	**C**	0	R source1	**8**	if (*c* =1) PC Rs1
JMPV Rs1	**C**	0	R source1	**9**	if (*v* =1) PC Rs1
JSRR Rs1	**C**	0	R source1	**A**	PMEM(SP)PC ; SPSP-1; PC PC+ Rs1
JSR Rs1	**C**	0	R source1	**B**	PMEM(SP)PC ; SPSP-1; PC Rs1
MUL Rt, Rs1	**C**	*1*	R source1	**R source2**	*RegHIGH (Rs1* Rs2) [31:16]* *RegLOW (Rs1 * Rs2) [15:0]*
DIV Rt, Rs1	**C**	*2*	R source1	**R source2**	*RegHIGH (Rs1 % Rs2)* *RegLOW (Rs1 / Rs2)*

JMPD desloc	**D**	**-**	Deslocamento (10 bits)		PC PC + **ext_sinal &** desloc
JMPND desloc	**E**	**0**	Deslocamento (10 bits)		if (*n* =1) PC PC + **ext_sinal &** desloc
JMPZD desloc	**E**	**1**	Deslocamento (10 bits)		if (z=1) PC PC + **ext_sinal &** desloc
JMPCD desloc	**E**	**2**	Deslocamento (10 bits)		if (*c* =1) PC PC + **ext_sinal &** desloc
JMPVD desloc	**E**	**3**	Deslocamento (10 bits)		if (*v* =1) PC PC + **ext_sinal &** desloc

JSRD desloc	**F**	Deslocamento (12 bits)	PMEM(SP)PC ; SPSP-1; PC PC + **ext_sinal &** desloc

Registradores especiais

O R8 inclui agora registradores especiais para dar suporte a interrupções e operações de multiplicação e divisão.

Registrador	Função
HIGH	Parte alta da multiplicação ou resto da divisão
LOW	Parte baixa da multiplicação ou resultado da divisão
TSRA	Endereço do tratador de traps
ISRA	Endereço do tratador de interrupções
CAUSE	Causa da trap

O registrador cause indica o que causou o acionamento da interrupção de software.

Cause	Trap
1	Invalid Instruction
8	SWI
12	Signed Overflow
15	Division by zero

Exemplo de uso de HIGH e LOW com operações de divisão e multiplicação usando MFH e MFL.

```
.code
       ldh r5, #0
       ldl r5, #20
       ldh r6, #0
       ldl r6, #3
       mul r5, r6      ;  HIGH <- 0, LOW <- 60
       mfh r7          ;  r7 <- HIGH
       mfl r8          ;  r8 <- LOW
       div r5, r6      ; HIGH <- (20 % 3) = 2 , LOW <- (20 / 3) = 6
       mfh r7          ;  r7 <- HIGH
       mfl r8          ;  r8 <- LOW
.endcode
```

Exemplo do uso de ISRA e TSRA para definir os endereços dos tratadores de interrupção e traps, e tratador de trap usando registrador CAUSE.

```
.code
       ldh r0, #ISR
       ldl r0, #ISR
       ldisra r0      ; carrega o endereço do handler de interrupção

       ldh r0, #TSR
       ldl r0, #TSR
       ldtsra r0      ; carrega o endereço do handler de trap

       ;...

TSR:
       push r0
       ;...
       pushf

       mfc    r5              ; r5 <- CAUSE
       ldh r8, #tsr_handlers
       ldl r8, #tsr_handlers
       ld r8, r8, r5           ; Carrega o tratador apropriado
       jsr r8                 ; executa o tratamento
```

```
        popf
        ;...
        pop r0
        rti
.endcode
.data
        ;estutura básica do vetor de handler de traps
        tsr_handlers: db #0, #invalid_instruction_handler, #0, #0, #0, #0, #0, #0,
        #SWI_handler, #0, #0, #0, #signed_overflow_handler, #0, #0, #div_by_zero_handler
.enddata
```

Periféricos

O acesso aos periféricos e seus registradores pelo processador R8 é dado através das instruções de leitura e escrita da memória (**LD** e **ST**). O espaço de endereços está dividido em dois, um para acesso da memória, e um espaço de endereçamento de periféricos. O bit mais significativo do endereço indica qual dos espaços ele se encontra, sendo que 0 indica memória e 1 indica periféricos. Para os periféricos, os bits 7 a 4 indicam o número do periférico, enquanto os bits 3 a 0 indicam o registrador a ser acessado.

Bits	15	14:8	7:4	3:0
Memória	0	Endereço físico de memória		
Periférico	1	Don't care	Endereço de periférico	Endereço de registrador

A tabela abaixo lista todos os periféricos disponíveis, seus endereços de acesso, e se eles estão habilitados a escrita e/ou leitura:

Periférico	Registrador	Endereço	Valor de reset	Bits de dados	write/read
PortA	Data	0x8002	0x0000	16	wr
PortA	Enable	0x8000	0x0000	16	wr
PortA	Config	0x8001	0xFFFF	16	wr
PortA	IrqEnable	0x8003	0x0000	16	wr
PIC	highPriorityReq	0x8010	0x00	8	r
PIC	ACK	0x8011	-	-	w
PIC	Mask	0x8012	0x00	8	wr
PIC	Irq_reg	0x8013	0x00	8	r
TX	Data_in	0x8020	0x00	8	w
TX	reg_freq_baud	0x8021	0x00d9	16	w
TX	ready	0x8020	0b0	1	r
RX	reg_freq_baud	0x8030	0x00d9	16	w
RX	Data_out	0x8030	0x00	8	r
Timer	Counter	0x8040	0x0000	16	wr

Todos os registradores que podem ser lidos que contém menos 16 bits de dados são estendidos para 16 bits com zeros.

PortA - GPIO (Endereço de periférico: 0x0)

A PortA possibilita, através de software, o controle de 16 pinos individuais como entrada ou saída, assim como a possibilidade de habilitação de interrupção. Ele é composto

por 4 registradores, sendo 3 com fins de configuração, e 1 que mantém os dados de cada pino.

Registrador	Endereço	Função
Config	0bXX01	Configura pinos como entrada (1) ou saída (0)
Enable	0bXX00	Habilita (1) e desabilita (0) os pinos
Data	0bXX10	Contém os dados de cada pino
IrqEnable	0bXX11	Habilita (1) e desabilita (0) interrupções dos pinos

Registrador Data:

Cada bit de 0 até 15 do registrador pode ser configurado como entrada ou saída, dependendo dos conteúdos dos registrador Config. Quando um bit estiver configurado para entrada, os dados que chegam ao pino serão gravados neste registrador. Quando um bit estiver configurado para saída, os dados contidos nele serão disponível na saída.

Quando uma operação de leitura é realizada no registrador Data, todos os seus bits atuais são lidos, incluindo de pinos não habilitados ou configurados como saída.

Para uma operação de escrita, apenas os bits que se encontram habilitados e configurados como saída serão modificados.

A configuração do portA deve ser feita com a escrita em seus registradores, Enable, Config, e IrqEnable.

Registrador Config:

Cada bit do registrador config indica o modo de cada bit respectivo do registrador Data, sendo que 1 indica ENTRADA e 0 indica SAÍDA.

Registrador Enable:

Cada bit do registrador enable indica se o bit respectivo está habilitado (1) ou desabilitado (0) para ser utilizado. É recomendável que este seja o último registrador a ser gravado antes do uso do portA.

Registrador IrqEnable:

Cada bit do registrador habilita (1) ou desabilita (0) a interrupção do processador. Apenas bits configurados como ENTRADA e habilitados podem gerar interrupções. É necessário que a saída irq do portA esteja devidamente conectada a uma entrada irq do PIC. Por padrão, apenas os bits 3 a 0 podem ser usados para geração de interrupção.

Exemplo da configuração do portA.

```
.code
       ; Configuramos o pino 0 como entrada
       ; Configuramos o pino 1 como saída
       ; Desabilitamos o restante dos pinos
       ; Invertemos o valor do pino 1 sem alterar os outros bits de regData

       xor r0,r0,r0
    ldh r8, #80h              ;
       ldl r8, #01h           ; r8 <= endereço de PortA regConfig
```

```
        ldh r9, #00h            ;
        ldl r9, #01h            ; Apenas o bit 0 é habilitado como
        st r9, r8, r0           ;  entrada (restante são zerados)


        ldh r8, #80h            ;
        ldl r8, #03h            ; r8 <=  endereço de PortA irqEnable
        ldh r9, #00h            ;
        ldl r9, #00h            ; Nenhum bit é habilitado para interrupção
        st r9, r8, r0           ;

        ldh r8, #80h            ;
        ldl r8, #00h            ; r8 <=  endereço de PortA regEnable
        ldh r9, #00h            ;
        ldl r9, #03h            ; Habilitamos o bit 0 e 1
        st r9, r8, r0           ;

        ldh r8, #80h
        ldl r8, #02h            ; r8 <= endereço de PortA regData
        ld r9, r9, r0           ; carregamos o conteúdo de PortA
        not r9, r9              ; invertemos o bit de saída 1
        st r9, r9, r0           ; escrevemos o valor de bit 1 invertido para a saída
.endcode
```

PIC - (Endereço de periférico: 0x1)

O PIC é o controlador de interrupções com prioridades do processador, composto por 4 registradores.

Registrador	Endereço	Função
Request Number	0bXX00	Indica o número da interrupção que foi requisitada
ack	0bXX01	Limpa uma interrupção tratada
mask	0bXX10	Habilita e desabilita cada interrupção
Irq	0bXX11	porta de entrada de interrupções

Até 8 requisições de interrupção podem ser feitas através da entrada irq. Os pedidos de interrupção de entrada ao PIC podem ser modificados em R8_uc.vhd. Por padrão, temos a seguinte conexão de pedidos de requisição:

Bit de irq	7	6	5	4	3	2	1	0
Requisição	PortA(3)	PortA(2)	PortA(1)	PortA(0)	-	RX	Timer	-

A prioridade da interrupção é dada começando do bit 0 de irq sendo a mais prioritário e o bit 7 se menor prioridade. A habilitação de cada uma dessas entradas pode ser configurada pelo registrador Mask, sendo 1 habilita a interrupção e 0 desabilita as interrupções respectivas. É recomendado que Mask seja o último registrador a ser habilitado antes do programa principal de qualquer aplicação.

Quando uma interrupção é requisitada do processador, o registrador Request Number deve ser lido, indicando qual periférico fez a chamada. No final do tratamento da interrupção, o mesmo Request Number deve ser novamente escrito no endereço ack, permitindo com que outras requisições sejam feitas.

Exemplo de tratamento de interrupção usando a interface PIC, para interrupções genéricas:

```
        .code
        boot:
                ;Inicializamos o endereço do tratamento de interrupções no boot:
                ldh r0, #ISR
                ldl r0, #ISR
                ldisra r0

                ; Inicializamos o vetor de interrupções
                ; Cada posição do vetor contém o endereço do driver específico que trata
                ; a interrupção do periférico apropriado
                ; Assim, um periférico que interrompe em irq(0),
                ; tem o endereço de seu driver em irq_handlers[0]
                ldh r8, #irq_handlers
                ldl r8, #irq_handlers
                xor r0, r0, r0
        ; Interrupção irq(0) tem como tratador a função generic_handler

                ldh r9, #generic_handler
                ldl r9, #generic_handler

                ldh r8, #irq_handlers
                ldl r8, #irq_handlers
                addi r0, #1
                ldh r9, #unused_handler       ; Interrupção irq(1) não é utilizada
                ldl r9, #unused_handler       ;

                ; E assim carregamos todos os 8 elementos com drivers apropriados
                ; ...

                ; Configuramos o registrador MASK do PIC
                ldh r8, #80h
                ldl r8, #12h            ; endereço de MASK
                ldh r9, #00h
                ldl r9, #01h           ; Apenas a interrupção irq(0) está habilitada
                st r9, r8, r0

                ; Começamos a executar o programa principal
                jmpd #main

        ISR:
                push r0
                ; ...
                push r15
                pushf          ; Tratador de interrupções salva todo o contexto do processador

                xor r0, r0, r0
                ldh r8, #80h
                ldl r8, #10h           ; Fazemos leitura do registrador Request_Number
                ld r10, r8, r0                 ; r10 <- Request Number
                ldh r8, #irq_handlers
                ldl r8, #irq_handlers   ; Carregamos o endereço do tratador de interrupção
                ld r8, r8, r10         ; r8 <- MEM[irq_handlers + Request_Number]
```

```
               jsr r8          ; Executamos a função de tratamento de interrupção
                               ; do periférico adequado

        ; Ao retornarmos da interrupção, mandamos o Request_Number da interrupção
        ; tratada para ACK
               xor r0, r0, r0
               ldh r8, #80h
               ldl r8, #11h           ; Endereço de ACK
               st r10, r8, r0            ; ACK <- Request_Number

               popf
               pop r15
               ; ...
               pop r0          ; Retornamos ao contexto anterior á interrupção
               rti

       generic_handler:
               ; Função que trata interrupção genérica
               rts

       unused_handler:
               ; É boa prática o uso de uma função para entradas de irq
               ; que não são utilizadas no programa atual
               ; Assim, caso essa interrupção for ativada por acidente, o programa
                ;pode continuar a funcionar normalmente
               rts

       main:
               jmpd #main

       .endcode
       .data
               ; Vetor com 8 posições, cada uma com o endereço de uma função que
                ; trata a requisição de número associado.
               ; É recomendável que este seja carregado durante o boot do programa,
                ;mas é possível definir as funções iniciais na sua declaração
                ; Um vetor padrão tem os seguinte handlers:
               irq_handlers: db #unused_handler, #timer_handler, #rx_handler, #unused_handler,
#portA_0_handler, #portA_1_handler, #portA_2_handler, #portA_3_handler
       .enddata
```

TX - Transmissor serial UART (Endereço de periférico 0x2)

O transmissor serial UART tem taxa de transferência configurável, e é composto por 3 registradores.

Registrador	Endereço	Função	Operação
tx_data	0bXX00	Contém os dados a serem transmitidos	ST
reg_freq_baud	0bXX01	Configuração de taxa de baud da UART	ST
ready	0bXX00	Indica a disponibilidade do TX	LD

O registrador freq_baud ser configurado antes de qualquer comunicação serial ser iniciada. Ele contém a configuração de taxa de transmissão de dados. O valor de configuração é dado por:

$$Rate = floor(\frac{Frequência\ do\ clock}{Baud\ rate})$$

A frequência de clock padrão é de 25 MHz. Para uma baud rate de 115200, temos portanto um valor de 217 (0x00D9). Este é o valor que o registrador se encontra depois de um reset.

O registrador ready contém 1 bit e indica se o TX está pronto para fazer a transmissão de um novo dado. É recomendável sempre fazer pooling do registrador ready antes de se tentar enviar algo pelo transmissor, para garantir a transferência correta dos dados.

O registrador Data_in recebe os 8 bits menos significativos de dados e os manda serialmente.

É importante notar que o endereço de Data_in é o mesmo de Ready. Isto significa que operações de leitura farão acesso ao registrador ready, enquanto operações de escrita farão acesso ao registrador Data_in.

Exemplo da configuração do transmissor para uma baud rate de 9600 (considerando o clock padrão de 25MHz) e o envio de um dado.

```
.code
    ldh r8, #80h
    ldl r8, #21h               ; endereço de Baud_rate do TX
    ldh r9, #0Ah
    ldl r9, #2Ch              ; floor(25000000/9600) = 2604
    st r9, r8, r0             ; configura baud rate para 9600 bps

    wait_for_ready_signal:
        ld r7, r9, r0                                  ; lê o sinal de ready
        addi r7, #0                                    ; while(ready != 1) {}
        jmpzd #wait_for_ready_signal
    ldh r5, #AAh
    ldl r5, #FFh                       ; carrega dados para serem mandados
    st r5, r9, r0             ; escreve em Data_in (0xFF)
.endcode
```

RX - Receptor serial UART (Endereço de periférico 0x3)

O Receptor serial UART tem taxa de transferência configurável, e é composto por 2 registradores. Da mesma maneira que RX, a taxa de transferência precisa ser configurada. É recomendado que ambos sejam configurados em conjunto, e sempre com os mesmo valores.

Registrador	Endereço	Operação	Função
freq_baud	0bXX00	ST	Configuração de taxa de baud da UART
Data_out	0bXX00	LD	Contém os dados recebidos

RX recebe dados seriais e gera uma requisição de interrupção toda vez que recebe um byte. Por padrão, o RX manda a requisição na entrada irq(2) do PIC. No tratamento dessa interrupção é necessário apenas a leitura do registrador Data_out.

É importante notar que ambos os registradores são acessados pelo mesmo endereço, entretanto reg_freq_baud é acessado em operações de escrita, e Data_out é acessado em operações de leitura.

Exemplo de um tratador de interrupção que lê os dados, e configuração de uma taxa de transferência de 9600.

```
.code
       ldh r8, #80h
       ldl r8, #30h            ; endereço de Baud_rate do RX
       ldh r9, #0Ah
       ldl r9, #2Ch           ; floor(25000000/9600) = 2604
       st r9, r8, r0          ; configura baud rate para 9600 bps

       jmpd #main             ; executa o código principal

RX_handler:
       ;função executada quando a interrupção do RX é tratada
       xor r0, r0, r0
       ldh r8, #80h   ; r8 <- endreço de Data_out
       ldl r8, #30h
       ld r5, r8, r0  ; r5 <- dados de RX
       rts
main:
       ;...
.endcode
```

Timer - (Endereço de periférico 0x4)

O timer é um periférico formado por 1 registrador de contador.

Registrador	Endereço	Função
counter	0bXXXX	Número de ciclos de clocks a serem contados

Quando um valor é escrito em counter, a cada ciclo de clock ele é subtraído. Quando o contador chegar em zero, uma interrupção é requisitada no PIC (por padrão, em irq(1). Uma vez que o timer gerou a interrupção, ele aguarda por um novo valor ser escrito no counter.

Em geral, o tratador da interrupção do timer deve fazer sua tarefa periódica e depois então configurar um novo valor ao contador.

Para um clock padrão de 25MHz, cada unidade do counter representam 40 ns.

Exemplo de configuração de timer para interrupção a cada 4 µs.

```
.code
boot:
        ; habilita interrupção do timer irq(1)
        xor r0, r0, r0
        ldh r8, #80h
        ldl r8, #12h            ; r8 <- endereço de MASK do PIC
         ldh r9, #00h
        ldl r9, #01h           ; Habilita apenas o bit 1
        st r9, r8, r0

        ; timer é incializado no boot após a habilitação de interrupções
        ldh r8, #80h           ;
        ldl r8, #40h           ; r8 <- endereço de timer counter
        ldh r9, #03h           ;
        ldl r9, #E8h           ;
        st r9, r8, r0          ; counter <- 1000 (equivalente a 4 us)

        jmpd #main             ;executa o programa principal

timer_handler:
        ; ... executa código periódico
        ; Reconfigura o timer para interromper novamente em 4 us
        xor r0, r0, r0
        ldh r8, #80h           ;
        ldl r8, #40h           ; r8 <- endereço de timer counter
        ldh r9, #03h           ;
        ldl r9, #E8h           ;
        st r9, r8, r0          ; counter <- 1000 (equivalente a 4 us)
        rts

main:
        ; ... código principal
.endcode
```

Chamadas de sistema

A instrução SWI dá acesso ao processador a uso de chamadas de sistema. Na atual implementação, o registrador r14 é utilizado para especificar a função do sistema a ser chamada.

Função	Valor
print_string	0
integer_to_string	1
integer_to_hexstring	2
string_to_integer	3
read_buffer	4

Pela convenção de programação, durante uma chamada de sistemas, r1 e r2 são usados para a passagem de argumentos, e o r3 e r4 são utilizados para retorno de função.

print_string

Descrição:

Manda uma string com terminador nulo para a porta serial TX. Uma string é um vetor contendo um caractere ASCII por palavra, com uma última palavra em zero indicando terminação. Atenção: a configuração das portas seriais RX e TX e suas velocidades de transferência deve ser feito no início do programa. Caso não configurado, as portas tem taxa padrão de 115200 bps.

Argumentos:

R1 <- endereço da string

Retorno:

A função não tem valores de retorno.

Exemplo mandando uma string “olá”.

```
       ldh r1, #string
       ldl r1, #string
       ldh r14, #0
       ldl r14, #0
       swi             ; print_string
       ;o terminal serial conectado ao microcontrolador
       ;deve receber "olá"

; string contém "olá" em ASCII
string db #6fh, #6ch, #e1h, #0
```

integer_to_string

Descrição:

Transforma um número inteiro de 16 bits em uma string de caracteres ASCII com terminador nulo, incluindo um sinal + ou -. É importante que a string tenha tamanho de no mínimo 7 palavras.

Argumentos:

R1 <- valor inteiro

R2 <- endereço de string

Retorno:

R3 <- endereço de string

Exemplo de conversão do número 50 para uma string “+50”.

```
       ldh r14, #0
       ldl r14, #2    ; integer_to_string
       ldh r1, #0
       ldl r1, #50
       ldh r2, #string
       ldl r2, #string
       swi                     ; chama integer_to_string(50,&string)
    ;string agora contém:
    ; "+50"

; string usada para conversão
string db #0h, #0h, #0h, #0h, #0h, #0h, #0h
```

integer_to_hexstring

Transforma um inteiro de 16 bits em uma string de sua representação hexadecimal em caracteres ASCII com terminador nulo, inicializada por “0x”. É importante que a string tenha tamanho de no mínimo 7 palavras.

Argumentos:

R1 <- valor inteiro

R2 <- endereço de string

Retorno:

R3 <- endereço de string

Exemplo de conversão do número 50 para uma string “0x32”.

```
       ldh r14, #0
       ldl r14, #2    ; integer_to_hexstring
       ldh r1, #0
       ldl r1, #50
       ldh r2, #string
       ldl r2, #string
       swi                     ; chama integer_to_hexstring(50,&string)
    ;string agora contém:
    ; "0x32"

; string usada para conversão
```

```
string db #0h, #0h, #0h, #0h, #0h, #0h, #0h
```

string_to_integer

Transforma uma string de caracteres em um número inteiro sem sinal. Apenas os números entre 0 e 32767 podem ser convertidos. Quaisquer outros caracteres ou valores valores presentes na string farão com que a função falhe.

Argumentos:

R1 <- endereço de string

Retorno:

R3 <- Valor do inteiro convertido se a função foi realizada com sucesso ou -1 para falhas

Exemplo do uso de string_to_integer convertendo “50” para um valor inteiro 50.

```
       ldh r14, #0
       ldl r14, #3    ; string_to_integer
       ldh r1, #string
       ldl r1, #string
       swi
       ;r3 <- 50

; string contém 50 em ASCII
string db #35h #30h #0
```

read_buffer

A função tenta fazer uma leitura do buffer de transmissão serial. O buffer fica disponível quando um caractere ‘\n’ ou ‘\r’ (tecla enter) é recebido pelo handler de interrupção do receptor serial. Caso o buffer estiver disponível, a função copia o número de caracteres especificados para o vetor passado. Caso o buffer não estiver disponível, a função retorna 0.

Argumentos:

R1 <- endereço do vetor

R2 <- número de palavras a serem copiadas

Retorno:

R3 <- Se a função foi realizada com sucesso, retorno é o número de caracteres copiados, se a função falha, o retorno é 0.

Exemplo do uso de read_buffer para leitura de 50 caracteres para o vetor vector.

```
       ldh r1, #vector
       ldl r1, #vector
       ldh r2, #0
       ldl r2, #50
       ldh r14, #0
       ldl r14, #4    ; read buffer
   read_loop:
        swi                  ; chama read read_buffer(&vector, 50)
        addi r3, #0    ; espera read_buffer retornar algo válido
        jmpzd #read_loop
```

Tutorial

Organização do diretório

A árvore de diretórios do projeto pode ser visualizado na Figura 1 abaixo.
Na pasta **“Src”** encontra-se todos os .vhd, seus respectivos .tcl e o .ucf. Em **“Terminal”** estão os arquivos correspondentes ao hyperterminal para Windows XP. Na pasta “**Simulator**” encontram-se o .arq e o .java correspondentes à lista de instruções e ao simulador, respectivamente. Como o nome indica, estão em “**Assembly**” os .asm e em “**Bitstream**” o .bit.

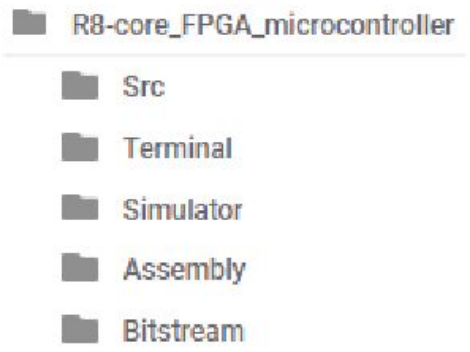

Figura 2. Estrutura de diretórios

Report de síntese

Essa seção é destinada aos dados de síntese do microcontrolador na FPGA 6slx16csg324-3 da Xilinx.

Utilização de recursos

Dos 18224 registradores disponíveis, foram utilizados 719, correspondendo a 3% dos disponíveis. Em termos de LUTs, foram ocupadas 2182, cerca de 23% das 9112 disponíveis. Dos 2488 pares LUT flip flop utilizados, apenas 413 (16,6%) foram completamente utilizados, o que significa que há a possibilidade de adicionar hardware sem aumentar a utilização de recursos.
A utilização pode ser visualizada na Figura 3 abaixo.

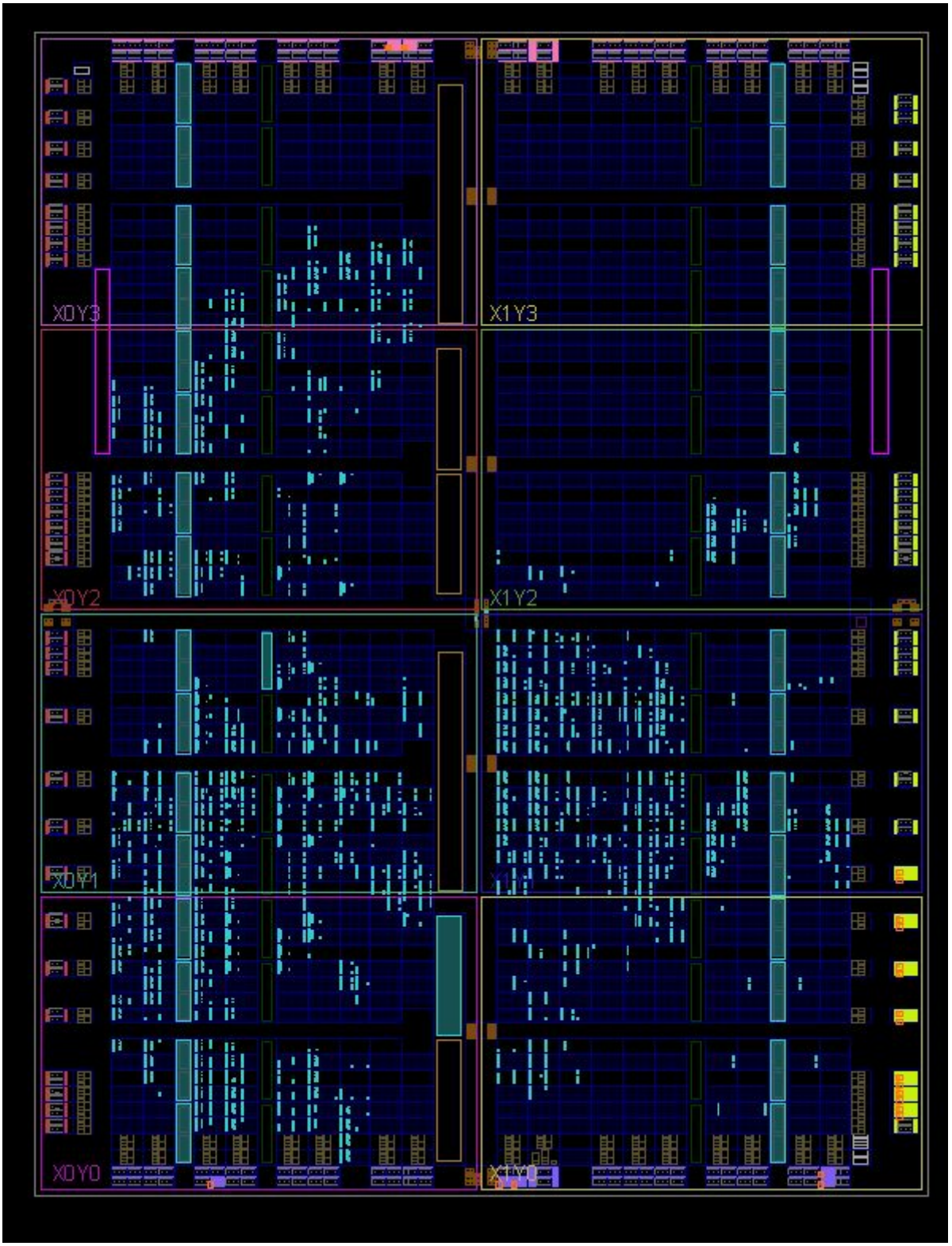

Figura 3 - Utilização extraída pós mapeamento.

Temporização

De acordo com o report de temporização, o caminho crítico do R8_uC é de 48,16 ns, correspondendo a uma frequência máxima de 20,76 MHz.

Guia passo-a-passo

Essa seção descreve como operar a escrita da memória RAM do microcontrolador via serial, considerando o software Hyperterminal e a placa Nexys 3 da Xilinx.

Configuração do Terminal

Para realizar a programação do R8_uC, é necessário enviar o programa montado através de um terminal serial que se comunique com o microcontrolador através do protocolo RS232. Neste guia será utilizado como terminal o Hyperterminal.
Com o .asm do programa a ser enviado já montado utilizando o simulador Java em /Simulator, deve-se abrir o Hyperterminal. Após a escolha de um nome arbitrário, é possível configurar a comunicação, preenchendo os campos da seguinte maneira:

- Bits per Second: inserir a baud rate desejada (padrão: 115200)
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow control: None

Programação serial

Com a comunicação configurada, deve-se conectar a placa Nexys 3 ao um computador utilizado a entrada UART. Com isso feito, **switch T5** da placa deve ser levantado, fazendo o microcontrolador entrar em modo de programação. Para enviar um programa, deve-se acessar o menu “Transfer” do Hyperterminal, e “Send text file”, escolhendo o arquivo .bin resultante da montagem. É possível saber quando ocorreu o término da transferência através do led de RX da placa.

Executando a aplicação exemplo

Para voltar ao modo de execução, o switch T5 deve ser baixado. Como aplicação exemplo, deve-se enviar para o R8_uC o .bin correspondente ao programa read_bubles_timer, que executa paralelamente um Bubble Sort parametrizado pelo usuário e um timer com contador. Com isso feito, é possível interagir com o mesmo através do próprio terminal, determinando os parâmetros de operação do Bubble Sort. e através dos botões up e down (A8 e C9), como incremento e decremento do contador. Para resetar o processador, deve-se utilizar o switch T10 da placa.

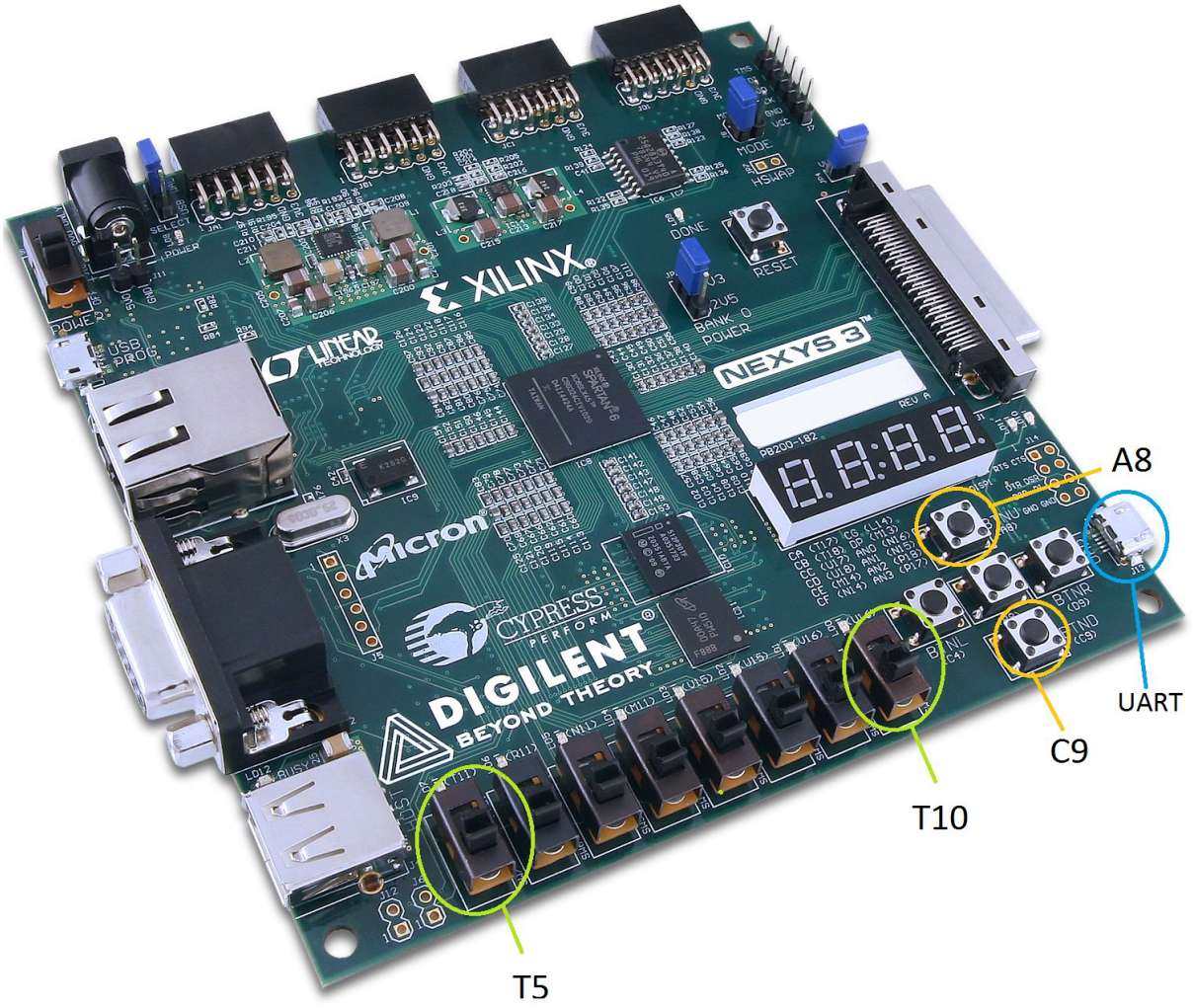

Figura 4. Placa Nexys 3 e os elementos citados.