PROJETO E IMPLEMENTAÇÃO DE UM PROCESSADOR COM CONJUNTO DE INSTRUÇÕES UTILIZANDO VHDL

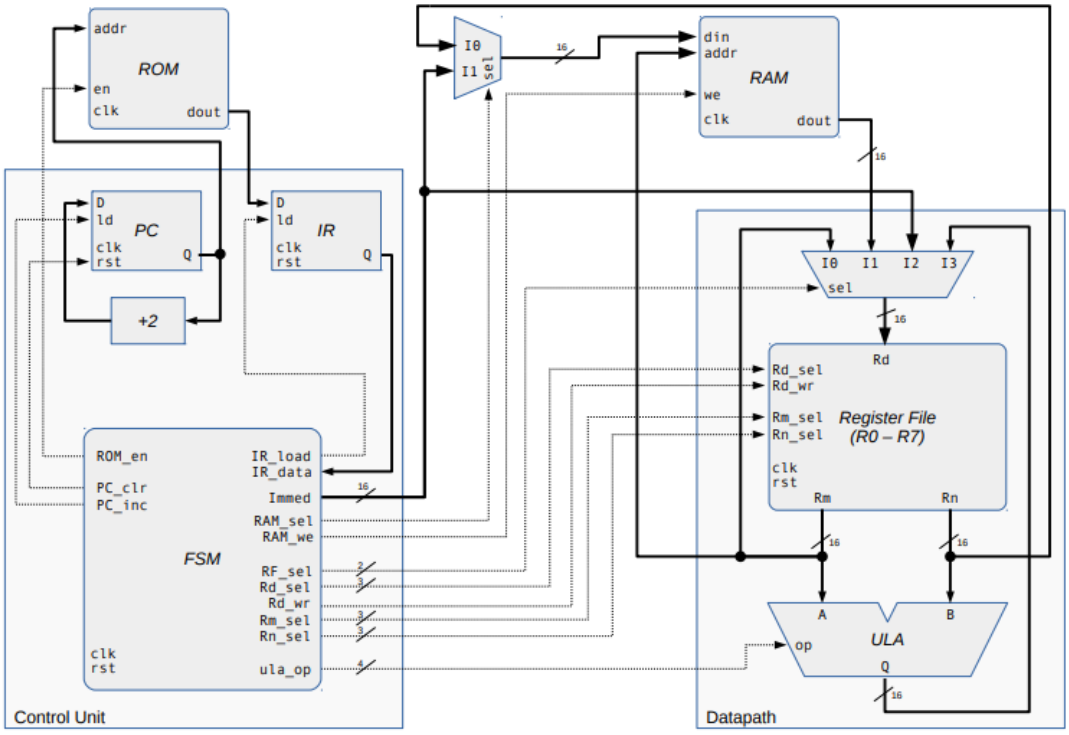
Universidade Federal do Ceará - Campus Quixadá

Jonas Bezerra da Costa Máximo[1]

Resumo: Este trabalho possui como objetivo central a implementação de instruções em VHDL para o processador desenvolvido em sala de aula. As instruções implementada pela equipe serão push, pop, jumpers, in e out.

Palavras-chave: vhdl, processador e instruções.

1 PROCESSADOR ORIGINAL



.

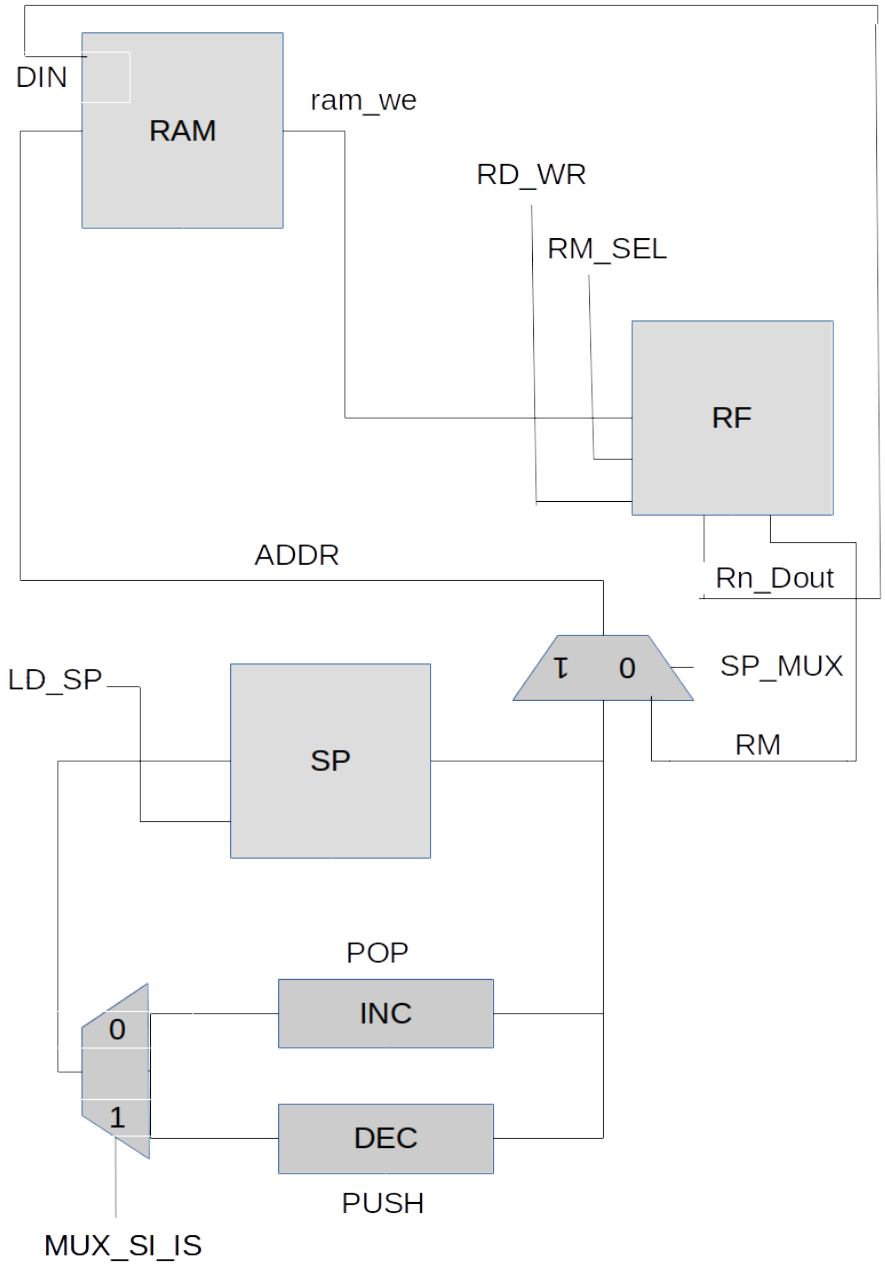
[1] UFC, jonasbezz@alu.ufc.br

2 PILHA

São estruturas de dados do tipo LIFO (last-in first-out), onde o último elemento a ser inserido, será o primeiro a ser retirado. Assim, uma pilha permite acesso a apenas um item de dados - o último inserido. Para processar o penúltimo item inserido, deve-se remover o último.

2.1 POP

O POP e a instrução de enviar da mamória para um registrador POP R(numero do registrador)



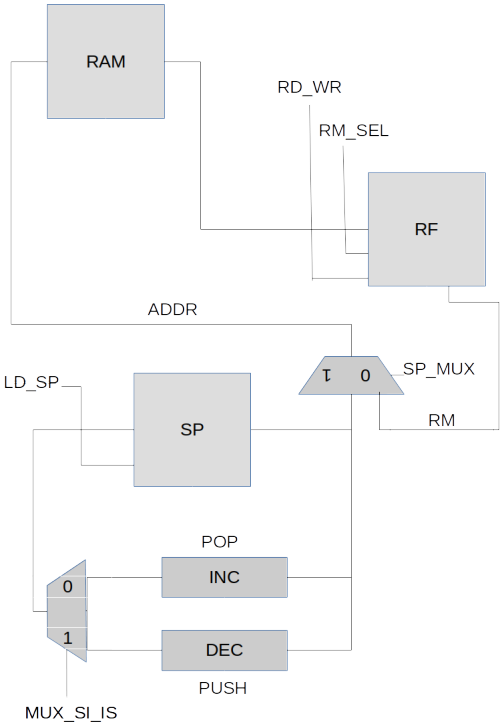
Possui um estado que é a anterior ao do pop (espera_pop) na qual ele seta(1) no *mux_sp_is* para selecionar a soma e o *ld_sp* para habilitar a gravação no registrador da pilha (*reg_sp*) no qual é

utilizado para diminuir o ponteiro da pilha para ele possa decrementar para o endereço que tenha dados, caso não existisse esse estado iria desempilhar o endereço errado pois a pilha aponta sempre para o próximo endereço vazio. O próximo estado será *sp_pop* no qual será tratado os sinais.

Deste estado de espera do pop o próximo estado será o *sp_pop*, no qual seleciona o registrador de instrução e seta(1) *sp_mux* o qual fará o chaveamento para o endereço da ram, desabilita o *ld_sp* pois como o ponteiro da pilha já foi decrementado no estado anterior não e necessário o fazê-lo novamente, habilito a gravação no RF *rd_wr* em (1), e feito o chaveamento na saída da RAM com o *rf_sel* em (01) e o mux *mux_in* no qual fará a entrada no RF onde no *rd_sel* foi selecionado o registrador referente a posição dos bits (10 ao 8) da instrução. após isso o próximo estado será a busca de uma nova instrução.

2.2 PSH

O PSH e uma instrução que emvia o dado do registrador para memória PSH R(numero do registrador

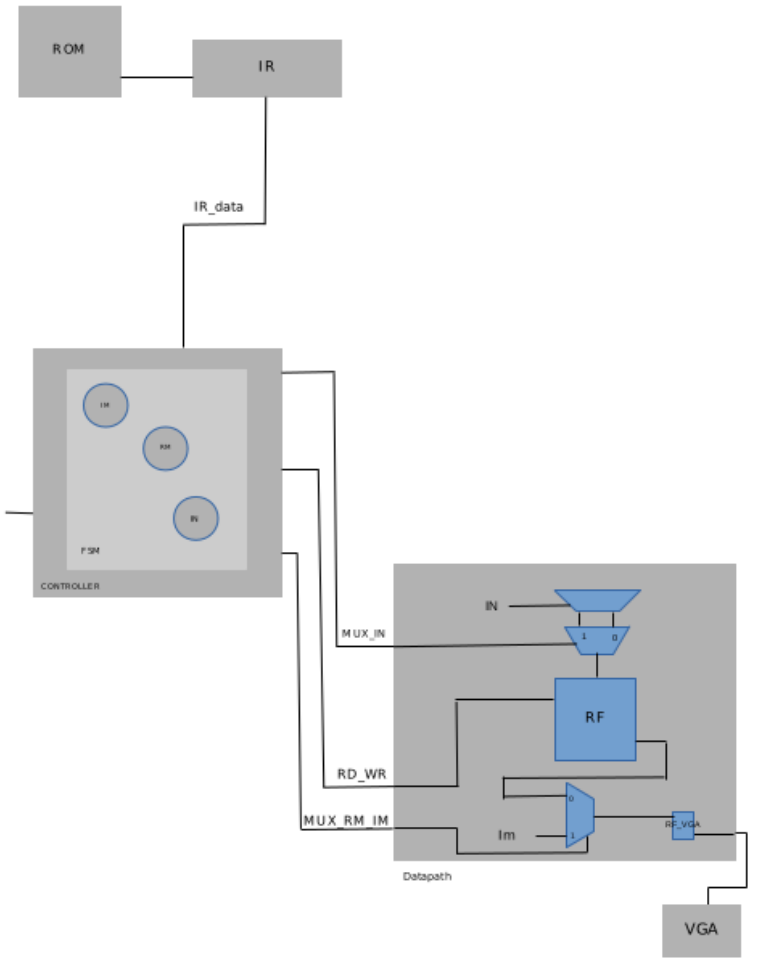


Funcionamento do PSH se da pelo *sp_push* que e o estado que vai enviar os dados do registrador e vai empilhar na memória, com isso e necessário enviar os sinais *Rn_sel* para o RF na qual vai selecionar o registrador que será indicado com os bits (4 ao 2) da instrução . e necessário também habilitar a escrita na ram *RAM_we*(1), enviando o sinal (0) para o *ram_sel* o qual chavear o sinal do rn do RF para a entrada de dados da RAM, selecionará (o) no *mux_sp_is*

para que o endereço possa ser decrementado após o dado ser enviado e o *ld_sp*(1) tem que ser habilitado para que possa ser gravada a decrementarão do

3 I/O

I/O e a entrada é saída de dados, foi implementada com três estados, um para entrada *IN* e dois para a saída, registrador *RM* e a do Imediato *IM*



.

3.1 IN

A entrada será feita através dos 4 switch presentes na placa e salvo no registrador de destino. no controller_FSM temos o estado *d_io_in* referente a entrada no qual ele seleciona o registrador de destino *rd_sel* com os bits (10 ao 8) da instrução, habilitando a gravação no RF com o *rd_wr*(1), foi feito um chaveamento com mux do *RF_sel* o *mux_in*(1) para possibilitar a entrada do dado no *RD* do *RF*.

3.2 OUT

Para a saída do registrador temos no controller_FSM o estado *d_io_out_rm* que vai tratar os seguintes sinais *mux_rm_im*(0) que fará o chaveamento com o imediato. *rd_sel* com os bits (7 ao 5) da instrução selecionarão o registrador que irá para a saída.

Para a saída do imediato temos no controller_FSM o estado *d_io_out_im* que vai tratar os seguintes sinais *mux_rm_im*(1) que fará o chaveamento com o registrador, e o sinal de saída

immed x"00"concatenado com os bits (10 ao 8) e os bits (4 ao 0) da instrução. Temos também um registrador(RF_VGA) para poder controlar a saa do OUT, caso contrário, sempre vai haver saída no OUT

4 JMP

o jump serve para para pular para uma determinada parte do código podendo usar uma comparação

4.1 CMP

No ciclo de instrução se a instrução decodificada for uma comparação então ela é do tipo ULA, será enviada para o estado *exec_ula*. E na *ULA* acontece ativação das flags.

As flags são ativadas comparando valores dos registradores isso ocorre na *ULA* onde as flags s_flag_z e ativada quando o valor de um registrador e igual ao outro e a s_flag_c e ativada quando o primeiro registrador tem valor menor que o segundo com essa informação podemos manipular os jumps. Elas são iniciadas com zero, e se nada for ativado o sinal continua a anterior.

4.2 JUMPERS

No processo de decodificação, temos que se o bit dez da instrução vier em um (o bit mais significativo do imediato) o número está todo negativo, então tivemos que "1111111"concatenado com os bits do 10 ao 2 para que conseguíssemos os 16 bits idem para o número positivo só que com zeros.

Temos que o sinal *s_jmp_op* são os bits 1 ao 0 da instrução os quais são usados para decodificarmos qual o tipo de JMP

temos o mux *s_pc_ctrl* no qual se for 0 o pc incrementa normalmente se for 1 temos que o *s_pc_dout* e somado com o *imediato* menos 1 pois o estado do *ROM_en*(1) está ativado no estado *espera_rom* para que o pc não incemente 1 posição a mais

4.2.1 JMP

(s_jmp_op "00") para tratarmos esse JMP apenas incrementamos o pc com *s_pc_ctrl*(1) que funciona como já citado. Após isso o próximo estado e *espera_rom* e habilitado o *PC_inc*(1) o qual e usado para incrementa o pc e o sincroniza, pois caso não existisse o pc estaria em instruções posteriores e nesse estado e habilitado também o *ROM_en*(1) para a próxima instrução.

4.2.2 JEQ

(s_jmp_op "01") com o acionamento das flags $flag\_z$(1) e $flag\_c$(0) o *s_pc_ctrl*(1) e funciona como já citado. caso as flags estejam diferentes *PC_inc*(0) não e ativado e *s_pc_ctrl*(0) também não e ativado o pc não e incrementado e ocorrerá uma nova busca.

4.2.3 JLT

(s_jmp_op "10") com o acionamento das flags $flag\_z$(0) e $flag\_c$(1) e funciona como já citado. caso as flags estejam diferentes *PC_inc*(0) não e ativado e *s_pc_ctrl*(0) também não e

ativado o pc não e incrementado e ocorrerá uma nova busca.

4.2.4 JGT

(s_jmp_op "11") com o acionamento das flags $flag\_z(0)$ e $flag\_c(0)$ e funciona como já citado. caso as flags estejam diferentes $PC\_inc(0)$ não e ativado e $s\_pc\_ctrl(0)$ também não e ativado o pc não e incrementado e ocorrerá uma nova busca.

5 INSTRUÇÕES

Grupo	Instrução	Operação	Tipo	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PSH Rn	[SP] = Rn; SP - -	PILHA	0	0	0	0	0	-	-	-	-	-	-	Rn_2	Rn_1	Rn_0	0	1
	POP Rd	SP++; Rd = [SP]	PILHA	0	0	0	0	0	Rd_2	Rd_1	Rd_0	-	-	-	-	-	-	1	0
1	CMP Rm, Rn	Z = (Rm = Rn)? 1 : 0 ; C = (Rm < Rn)? 1 : 0	ULA	0	0	0	0	0	-	-	-	Rm_2	Rm_1	Rm_0	Rn_2	Rn_1	Rn_0	1	1
	JMP #Im	PC = PC + #Im	DESVIO	0	0	0	0	1	Im_8	Im_7	Im_6	Im_5	Im_3	Im_3	Im_2	Im_1	Im_0	0	0
	JEQ #Im	PC = PC + #Im, se Z = 1 e C = 0	DESVIO	0	0	0	0	1	Im_8	Im_7	Im_6	Im_5	Im_3	Im_3	Im_2	Im_1	Im_0	0	1
	JLT #Im	PC = PC + #Im, se Z = 0 e C = 1	DESVIO	0	0	0	0	1	Im_8	Im_7	Im_6	Im_5	Im_3	Im_3	Im_2	Im_1	Im_0	1	0
	JGT #Im	PC = PC + #Im, se Z = 0 e C = 0	DESVIO	0	0	0	0	1	Im_8	Im_7	Im_6	Im_5	Im_3	Im_3	Im_2	Im_1	Im_0	1	1
2	IN Rd	Rd = IO_read(7 . . 0)	E/S	1	1	1	1	-	Rd_2	Rd_1	Rd_0	-	-	-	-	-	-	0	1
	OUT Rm	IO_write = Rm	E/S	1	1	1	1	0	-	-	-	Rm_2	Rm_1	Rm_0	-	-	-	1	0
	OUT #Im	IO_write = #Im	E/S	1	1	1	1	1	Im_7	Im_6	Im_5	0	0	0	Im_4	Im_3	Im_2	Im_1	Im_0
3	SHR Rd, Rm, #Im	Rd = Rm >> #Im	ULA	1	0	1	1	-	Rd_2	Rd_1	Rd_0	Rm_2	Rm_1	Rm_0	Im_4	Im_3	Im_2	Im_1	Im_0
	SHL Rd, Rm, #Im	Rd = Rm << #Im	ULA	1	1	0	0	-	Rd_2	Rd_1	Rd_0	Rm_2	Rm_1	Rm_0	Im_4	Im_3	Im_2	Im_1	Im_0
	ROR Rd, Rm	Rd = Rm >> 1; Rd(MSB) = Rm(LSB)	ULA	1	1	0	1	-	Rd_2	Rd_1	Rd_0	Rm_2	Rm_1	Rm_0	-	-	-	-	-
	ROL Rd, Rm	Rd = Rm << 1; Rd(LSB) = Rm(MSB)	ULA	1	1	1	0	-	Rd_2	Rd_1	Rd_0	Rm_2	Rm_1	Rm_0	-	-	-	-	-

16 Bits 8 Registradores

Instrução	Operação	Tipo	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOP	nop	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HALT	halt	HALT	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
MOV Rd, Rm	Rd = Rm	MOV	0	0	0	1	0	Rd_2	Rd_1	Rd_0	Rm_2	Rm_1	Rm_0	-	-	-	-	-
MOV Rd, #Im	Rd = #Im	MOV	0	0	0	1	1	Rd_2	Rd_1	Rd_0	Im_7	Im_8	Im_5	Im_4	Im_3	Im_2	Im_1	Im_0
STR [Rm], Rn	[Rm] = Rn	STORE	0	0	1	0	0	-	-	-	Rm_2	Rm_1	Rm_0	Rn_2	Rn_1	Rn_0	-	-
STR [Rm], #Im	[Rm] = #Im	STORE	0	0	1	0	1	Im_7	Im_6	Im_5	Rm_2	Rm_1	Rm_0	Im_4	Im_3	Im_2	Im_1	Im_0
LDR Rd, [Rm]	Rd =[Rm]	LOAD	0	0	1	1	-	Rd_2	Rd_1	Rd_0	Rm_2	Rm_1	Rm_0	-	-	-	-	-
ADD Rd, Rm, Rn	Rd = Rm + Rn	ULA	0	1	0	0	-	Rd_2	Rd_1	Rd_0	Rm_2	Rm_1	Rm_0	Rn_2	Rn_1	Rn_0	-	-
SUB Rd, Rm, Rn	Rd = Rm - Rn	ULA	0	1	0	1	-	Rd_2	Rd_1	Rd_0	Rm_2	Rm_1	Rm_0	Rn_2	Rn_1	Rn_0	-	-
MUL Rd, Rm, Rn	Rd = Rm * Rn	ULA	0	1	1	0	-	Rd_2	Rd_1	Rd_0	Rm_2	Rm_1	Rm_0	Rn_2	Rn_1	Rn_0	-	-
AND Rd, Rm, Rn	Rd = Rm and Rn	ULA	0	1	1	1	-	Rd_2	Rd_1	Rd_0	Rm_2	Rm_1	Rm_0	Rn_2	Rn_1	Rn_0	-	-
ORR Rd, Rm, Rn	Rd = Rm or Rn	ULA	1	0	0	0	-	Rd_2	Rd_1	Rd_0	Rm_2	Rm_1	Rm_0	Rn_2	Rn_1	Rn_0	-	-
NOT Rd, Rm	Rd = ¬Rm	ULA	1	0	0	1	-	Rd_2	Rd_1	Rd_0	Rm_2	Rm_1	Rm_0	-	-	-	-	-
XOR Rd, Rm, Rn	Rd = Rm xor Rn	ULA	1	0	1	0	-	Rd_2	Rd_1	Rd_0	Rm_2	Rm_1	Rm_0	Rn_2	Rn_1	Rn_0	-	-

6 PROJECT SUMMARY

tempo e consumo de energia

Timing — **Setup** | Hold | Pulse Width

Worst Negative Slack (WNS):	11.794 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	84

Implemented Timing Report

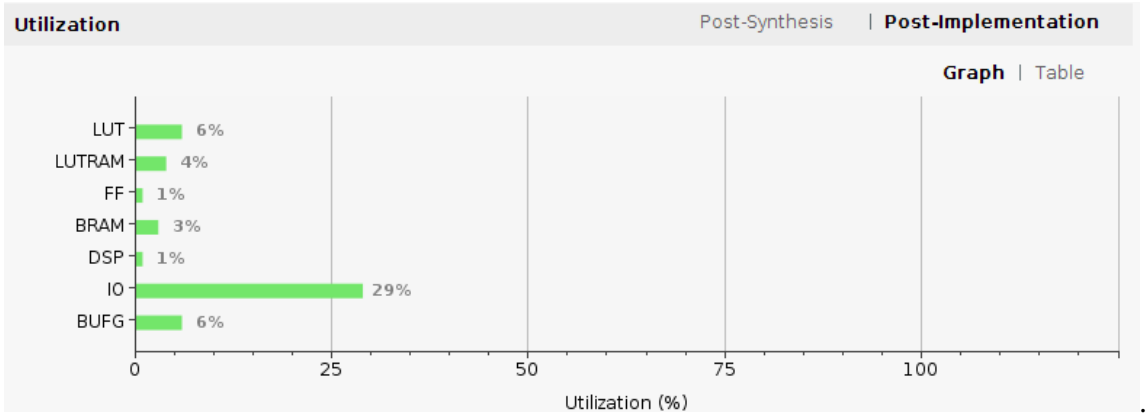
Power — **Summary** | On-Chip

Total On-Chip Power:	**0.105 W**
Junction Temperature:	**26,2 °C**
Thermal Margin:	58,8 °C (5,0 W)
Effective θJA:	11,5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

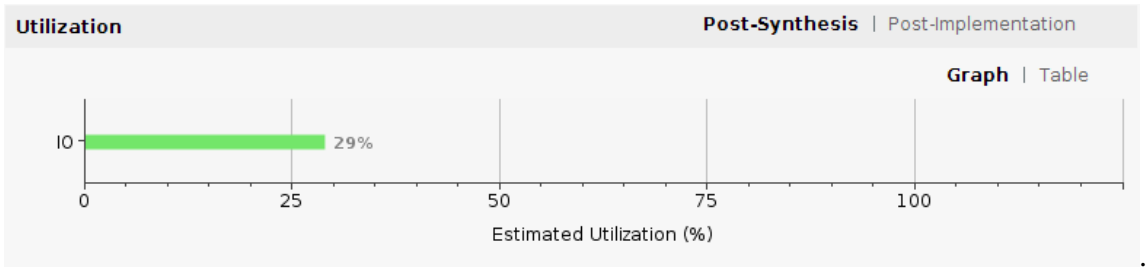
Implemented Power Report

componentes utilizados

1.png 1.png



.



.