

Executando o exemplo CL hello_world com a GUI Vivado

Descrição

No exemplo hello_world tem-se um software onde estão definidos os endereços dos registradores Hello world e Virtual LED e onde são utilizadas as funções de escrita (fpga_pci_poke) e leitura (fpga_pci_peek) desses registradores. Além disso, do lado do hardware tem-se o design, onde também estão definidos os endereços dos registradores, e onde são implementados processos de leitura e escrita dos registradores de acordo com a lógica proposta pelo exemplo hello_world. Por exemplo, o software escreve um valor no registrador Hello world, por meio da função fpga_pci_poke, o processo implementado no design ler o dado no registrador, realiza o Swapped, escreve o novo valor e então o software ler o novo valor do registrador por meio da função fpga_pci_peek.

Nesta prática vamos implementar uma soma de dois valores, a partir do exemplo hello_world. Serão instanciados dois registradores com os valores a serem somados e um registrador com o resultado da soma, além da lógica no lado do software e do hardware.

Objetivos de Aprendizagem

- Instanciamento de registradores.
- Implementação do somador no design.
- Implementação do somador no software.

Parte 1:

1. Faça um Fork do repositório <https://github.com/aws/aws-fpga.git> para um repositório da sua conta do Github
- 2.
3. Definir os endereços dos registradores no lado do hardware
 - a. Abra o arquivo /hdk/cl/examples/common/design/cl_common_defines.vh
 - b. Neste arquivo estão definidos os endereços dos registradores utilizados no exemplo hello_world da seguinte maneira:

```
`define HELLO_WORLD_REG_ADDR 32'h0000_0500
`define VLED_REG_ADDR        32'h0000_0504
```
 - c. Adicione os três registradores necessários para a implementação, nos endereços seguintes.
4. Feito isso, teremos que modificar o arquivo de implementação do design. Abra o arquivo /hdk/cl/examples/cl_hello_world/design/cl_hello_world.sv

- a. Perceba que a partir da linha 49 estão definidos os wires. Adicione os wires necessários.

A partir da linha 243 até a linha 263 está implementado o processo de resposta de leitura. Perceba que quando a leitura realizada é do registrador HELLO_WORLD_REG_ADDR, o valor a ser escrito será hello_world_q_byte_swapped[31:0].

```
259      rdata <= (araddr_q == `HELLO_WORLD_REG_ADDR) ? hello_world_q_byte_swapped[31:0]:
260              (araddr_q == `VLED_REG_ADDR          ) ? {16'b0, vled_q[15:0]} :
261              `UNIMPLEMENTED_REG_VALUE ;
---
```

O hello_world_q_byte_swapped[31:0] esta definido no processo logo abaixo, a partir da linha 270 até a linha 282.

```
270  always_ff @(posedge clk_main_a0)
271      if (!rst_main_n_sync) begin                // Reset
272          hello_world_q[31:0] <= 32'h0000_0000;
273      end
274      else if (wready & (wr_addr == `HELLO_WORLD_REG_ADDR)) begin
275          hello_world_q[31:0] <= wdata[31:0];
276      end
277      else begin                                // Hold Value
278          hello_world_q[31:0] <= hello_world_q[31:0];
279      end
280
281  assign hello_world_q_byte_swapped[31:0] = {hello_world_q[7:0],  hello_world_q[15:8],
282                                              hello_world_q[23:16], hello_world_q[31:24]};
```

Esse processo define que no sinal do reset o wire hello_world_q receberá o valor de 32'h0000_0000 (será zerado), quando houver o sinal de leitura e os dados forem do registrador HELLO_WORLD_REG_ADDR, então o wire hello_world_q receberá a informação contida no registrador. Caso não ocorra nenhuma dessas duas situações, o wire hello_world_q apenas manterá seu valor anterior. E finalmente, o valor atribuído ao wire hello_world_q_byte_swapped será o dado reorganizados, de forma que os bits menos significativos passam a ser mais significativos e vice-versa.

- b. A partir dessa implementação, insira o trecho de código que define o processo da soma de dois valores de entrada.
5. Feita as modificações no design, precisamos agora modificar o software para realizarmos a escrita dos valores que serão somados e a leitura do resultado.
 - a. Abra o arquivo
hdk/cl/examples/cl_hello_world/software/runtime/test_hello_world.c
 - b. Primeiro, é preciso declarar os registradores e seus endereços, da mesma forma que foram declarados no arquivo cl_common_defines.vh. Nas linhas 40 e 41 adicione a declaração dos registradores.

- c. Perceba que entre as linhas 236 e 256 a escrita e a leitura do registrador `HELLO_WORLD_REG_ADDR` foram implementadas, usando as funções `fpga_pci_poke` e `fpga_pci_peek`. Com base nesse código, vamos utilizar a função `fpga_pci_poke` para escrever os valores de `x` e `y` nos registradores `X_REG_ADDR UINT64_C(0x508)` e `Y_REG_ADDR UINT64_C(0x50C)`, e a função `fpga_pci_peek` para ler o resultado da soma que estará armazenada no registrador `Z_REG_ADDR UINT64_C(0x510)`.
- d. Para isso, abaixo da função descrita acima (linha 257), adicione o seguinte código:

```
//adder
/* write a value into the mapped address space */
uint32_t x,y;
x=2;
y=3;
expected = x+y;
printf("Writing 0x%08x to X register (0x%016lx)\n", x,
X_REG_ADDR);
rc = fpga_pci_poke(pci_bar_handle, X_REG_ADDR, x);

fail_on(rc, out, "Unable to write to the fpga !");
printf("Writing 0x%08x to Y register (0x%016lx)\n", y,
Y_REG_ADDR);
rc = fpga_pci_poke(pci_bar_handle, Y_REG_ADDR, x);

fail_on(rc, out, "Unable to write to the fpga !");

/* read it back and print it out; you should expect the byte
order to be
* reversed (That's what this CL does) */
rc = fpga_pci_peek(pci_bar_handle, Z_REG_ADDR, &value);
fail_on(rc, out, "Unable to read read from the fpga !");
printf("==== Entering peek_poke_example =====\n");
printf("register: 0x%x\n", value);

printf("==== Entering peek_poke_example =====\n");
printf("registers x and y: 0x%x\n 0x%x\n", x,y);
if(value == expected) {
    printf("TEST PASSED");
    printf("Resulting value matched expected value 0x%x. It
worked!\n", expected);
}
else{
    printf("TEST FAILED");
    printf("Resulting value did not match expected value 0x%x.
Something didn't work.\n", expected);
}
```

6. Feito isso, atualize o seu repositório do github com as modificações realizadas (commit + push).
7. Realize o mesmo procedimento descrito na [Prática 1- Criação de uma Amazon FPGA Image \(AFI\) do exemplo CL hello_world](#). Porém, no passo de download do HDK e SDK (Parte 1- item 2) substitua a url do repositório da aws para a url do repositório que foi realizado o fork do projeto e as modificações. Por exemplo:

```
git clone https://github.com/vanros/aws-fpga.git $AWS_FPGA_REPO_DIR
```
8. o resultado apresentado após a execução do teste será o seguinte:

```
==== Starting with peek_poke_example ====
Writing 0xefbeadde to HELLO_WORLD register (0x0000000000000500)
==== Entering peek_poke_example ====
register: 0xdeadbeef
TEST PASSEDResulting value matched expected value 0xdeadbeef. It worked!
Writing 0x00000002 to X register (0x0000000000000508)
Writing 0x00000003 to Y register (0x000000000000050c)
==== Entering peek_poke_example ====
register: 0x5
==== Entering peek_poke_example ====
registers x and y: 0x2
0x3
TEST PASSEDResulting value matched expected value 0x5. It worked!
Developers are encouraged to modify the Virtual DIP Switch by calling the linux shell command t
monstrate how AWS FPGA Virtual DIP switches can be used to change a CustomLogic functionality:
$ fpga-set-virtual-dip-switch -S (slot-id) -D (16 digit setting)
```

Referências

- Amazon Web Services. Hardware Development Kit (HDK) e Software Development Kit (SDK) [internet]. [Acesso em: 26 dez. 2017]. Disponível em:
https://github.com/aws/aws-fpga/blob/master/hdk/docs/IPI_GUI_Vivado_Setup.md
- ALMEIDA, Bruno. **Acessando modo gráfico da sua instância EC2**. 2013. Disponível em:
<<http://blog.rivendel.com.br/2013/09/13/acessando-modo-grafico-da-sua-instancia-ec2/>>. Acesso em: 20 fev. 2018.