Documentación

Proyecto 3_ Parte I: FIFO

El concepto de FIFO se refiere a un tipo de memoria o buffer que almacena datos en el orden en que llegan: el primer dato en entrar es el primero en salir. Las FIFO son maquinas que el software de simulación de hardware VIVADO permite agregar. El proceso para añadirlo es sencillo pero se deben tener en cuenta varios conceptos

- Modelo de la FPGA que se va a utilizar
- Velocidad del reloj
- Entadas deseadas
- Una correcta instanciación de la maquina

Proceso de añadido FIFO

Para añadir una FIFO usando el IP Catalog de VIVADO, se debe de entrar al IP Catalog y buscar la opción de FIFO Generator. Una vez ahí aparecerá la Figura_1 y se podrá configurar la definición de la FIFO

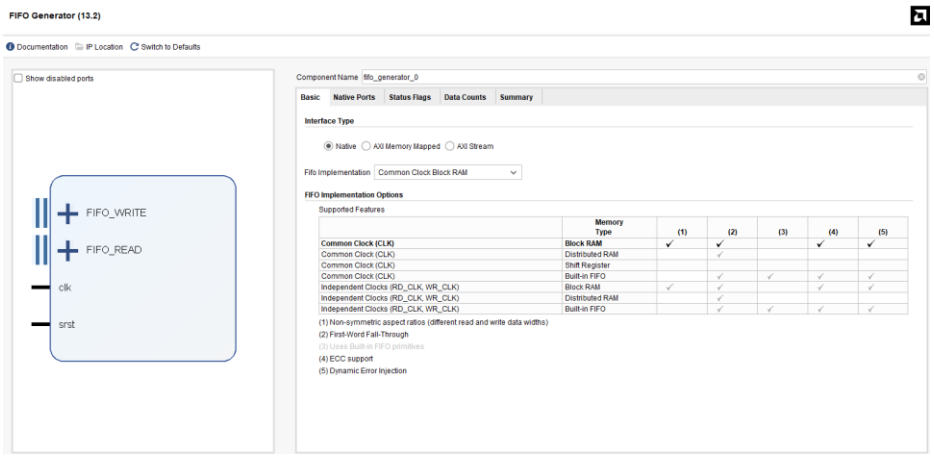

Figura_1: Generación del FIFO

Módulo 1: FIFO_Prueba

Una vez escogido el tipo de reloj deseado, el nombre para la FIFO (que en este caso se le ha colocado el nombre de FIFO_Prueba) los puertos, las banderas etc. Se va a generar el FIFO como un módulo más dentro de los documentos de VIVADO, a través del archivo .veo que se puede encontrar en la pestaña de IP Sources se puede encontrar la platilla para instanciar la FIFO de manera correcta dentro del código que usamos, la instancia para este caso se puede apreciar en la Figura_2

```
// The following must be inserted into your Verilog file for this
// core to be instantiated. Change the instance name and port connections
// (in parentheses) to your own signal names.

//----------- Begin Cut here for INSTANTIATION Template ---// INST_TAG
FIFO_Prueba your_instance_name (
  .clk(clk),                // input wire clk
  .srst(srst),              // input wire srst
  .din(din),                // input wire [7 : 0] din
  .wr_en(wr_en),            // input wire wr_en
  .rd_en(rd_en),            // input wire rd_en
  .dout(dout),              // output wire [7 : 0] dout
  .full(full),              // output wire full
  .overflow(overflow),      // output wire overflow
  .empty(empty),            // output wire empty
  .underflow(underflow),    // output wire underflow
  .data_count(data_count)   // output wire [8 : 0] data_count
);
// INST_TAG_END ------ End INSTANTIATION Template ---------
```

Figura_2: Instancia para una FIFO

Módulo 2: FIFO_TOP

Este es el módulo donde se van a instanciar tanto la FIFO como el módulo de reloj que también se ha de añadir al proyecto, se resalta que aquí se define si usar un reloj simulado o el reloj del módulo, ya que para conceptos de simulación el reloj del módulo no funciona solo para el caso de la síntesis. Este módulo TOP se puede apreciar en la Figura_3.

```
module FIFO_TOP (
    input  logic       clk,
    input  logic       srst,
    input  logic [7:0] din,
    input  logic       wr_en,
    input  logic       rd_en,
    output logic [7:0] dout,
    output logic       full,
    output logic       overflow,
    output logic       empty,
    output logic       underflow,
    output logic [8:0] data_count
);
logic cable_reloj;
logic cable_bloqueado;
```

Figura_3: FIFO_TOP

Proceso de añadido Clock

Para añadir una Clock usando el IP Catalog de VIVADO, se debe de entrar al IP Catalog y buscar la opción de Clock Wizard. Una vez ahí aparecerá la Figura_4 y se podrá configurar la definición de la Clock.

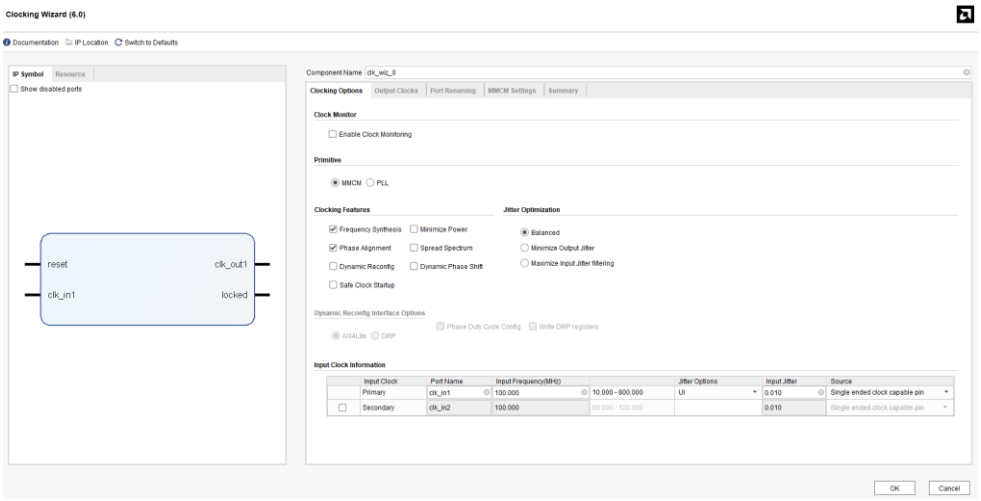

Figura_4: Generación del Clock

Módulo 3: Clock_16MHz

Seguidamente y de la misma forma que la FIFO, una vez escogido el nombre para la clock (que en este caso se le ha colocado el nombre de clock_16MHz) los puertos, las frecuencias etc. Se va a generar el Clock como un módulo más dentro de los documentos de VIVADO, a través del archivo .veo que se puede encontrar en la pestaña de IP Sources se puede encontrar la platilla para instanciar el clock de manera correcta dentro del código que usamos, la instancia para este caso se puede apreciar en la Figura_5

```
//----------- Begin Cut here for INSTANTIATION Template ---// INST_TAG

  clock_16MHz instance_name
   (
    // Clock out ports
    .clk_out1(clk_out1),     // output clk_out1
    // Status and control signals
    .reset(reset), // input reset
    .locked(locked),       // output locked
   // Clock in ports
    .clk_in1(clk_in1)      // input clk_in1
);

// INST_TAG_END ------ End INSTANTIATION Template ---------
```

Figura_5: Instancia para el Clock

Prueba de Funcionamiento

Una vez realizado las instancias de manera correcta se procede a realizar un Testbench apropiado para esta simulación, para ello se deben de conocer el funcionamiento de las entradas de la FIFO e interpretar sus salidas, ya que estas funcionan de acuerdo con el funcionamiento estándar de una FIFO, de la misma manera el Clock, los datos principales para una correcta inicialización se pueden apreciar en la Figura_6

```
parameter CLK_PERIOD = 62.5;
parameter integer PLL_LOCK_DELAY = 5;

always #(CLK_PERIOD/2) clk = ~clk;

initial begin
    // Reset
    srst = 1;
    repeat (PLL_LOCK_DELAY)#(CLK_PERIOD);
    srst = 0;
    #(2*CLK_PERIOD);
```

Figura_6: Inicialización del clock

Es importante mencionar la definición del parámetro CLK_PERIOD y PLL_LOCK_DELAY. Estos parámetros se defininen para simular en simulación la velocidad deseada de 16MHz del clock. Desde este punto se hace los intercambios correspondientes entre wr_en y rd_en para escritura y lectura de la FIFO a la velocidad del clock deseada y con las iteraciones correspondientes como en la Figura_7

```
wr_en = 1;
#(2*CLK_PERIOD);
// Escribe 5 valores
for (int i = 1; i < 5; i++) begin
    din = i;
    wr_en = 1;
    #(CLK_PERIOD);

end
wr_en = 0;
```

Figura_7: Procesos de escritura

Este proceso anterior, por ejemplo, sirve para escribir 10 valores en la FIFO

Por ultimo se corre el resto del código, y se busca apreciar en la ventana de comportamiento de la simulación el funcionamiento esperado. Dicho funcionamiento se puede apreciar finalmente en la Figura_8

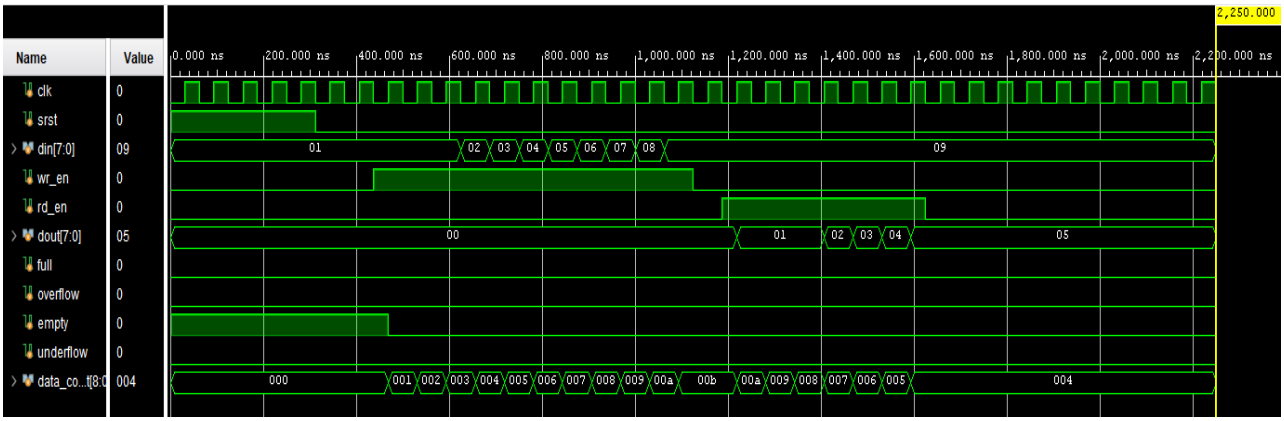

Figura_8: Funcionamiento final de la FIFO

En la Figura anterior se puede claramente apreciar que al iniciar el proceso de escritura se empiezan a guardar los valores en la FIFO según el data_count. En este caso se cuentan más ya que el valor 0 se guarda más de una vez, no es un problema para la demostración. Una vez baja wr_en se dejan de guardar números. Al activar el rd_en pasa el procedimiento opuesto, empiezan a desplegarse en la salida de datos dout los valores almacenados en la FIFO de manera que el ultimo que entró es el primero en salir, hasta que se desactive la bandera rd_en y la FIFO quede con los datos que no salieron. Cabe resaltar que la bandera empty está activa en el momento que no hay datos, y la bandera full se activará cuando la FIFO se llene. Este comportamiento se puede apreciar en la Figura_9 donde intencionalmente se busca llenar la FIFO y la bandera de full y overflow se activan

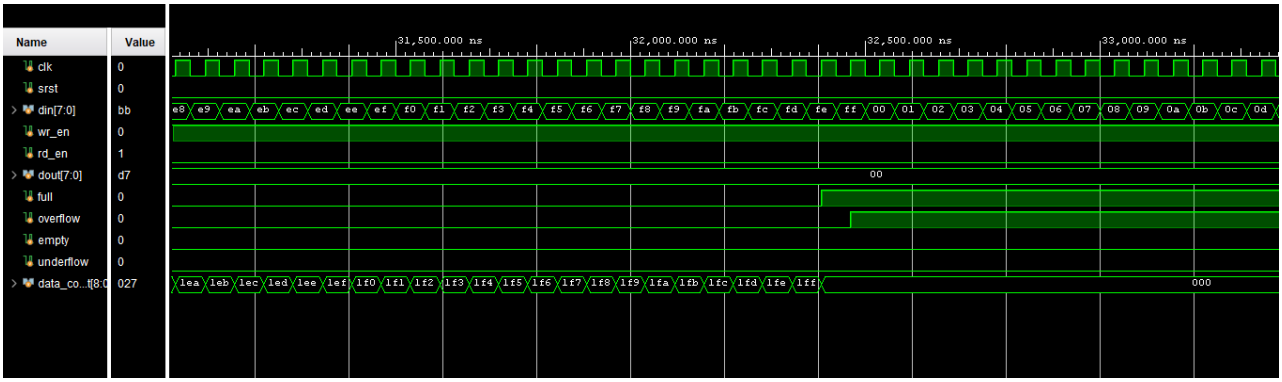

Figura_9: FIFO llena

Con esto se concluye la demostración del funcionamiento de la FIFO