Laboratorio N°4:

Actividad 1:

¿Qué hace el código?

assign LEDR[0] = SW[0] & SW[1];

assign LEDR[1] = SW[2] | SW[3];

assign LEDR[2] = ~(SW[0] & SW[1]);

Asigna operaciones que puedo efectuar con los switches para que se prendan o se apaguen los LEDs.

Luego, el bus usado es como una estructura de datos, voy a tener 32 bits adentro de este, el cual el clock por cada ciclo, va a ir aumentándolo como un contador.

Sucede que este trabaja con los bits mas significativos del bus, y tiene dos modos, uno con los 4 anteultimos y otro con los 4 últimos, con los 4 ultimos bits el contador por obviedad, será mas lento, osea va a tener que aumentar todos los bits anteriores para poder aumentar uno de los 4 ultimos mas significativos, con los 4 anteultimos, será mas rápido. Y a medida que aumenta cada uno de estos, se los asigno cada uno de estos bits del bus a un led, en este caso a los leds de 6 a 9, por lo tanto, con los cambios entre 0 y 1 de estos bits, a su vez cambian los leds y se prenden.

Luego con el multiplexor, se agrego la posibilidad de usar el switch 4 para ir cambiando entre uno de los modos, osea tomar los anteúltimos 4 bits o cambiarlo a tomar los últimos 4 bits y que vaya mas lento. Por esto pone result 9 a 6.

Con el código de temporizador puedo ver que es sencillo, básicamente aumenta la salida en uno, donde salida es el bus, osea va a ir aumentando con un for y el ciclo del reloj (por eso posedge reloj) el bus en 1, luego cuando llega al final todo en 1111, hace overflow porque suma 1 y se sale del arreglo el primer uno y queda todo 0 y se reinicia así.

El multiplexor, toma depende lo que haya en select, los datos 1 (últimos) o datos 0 (anteúltimos) y con el select en 0 o 1 elije uno de los dos, y envía a resultado el elejido, donde resultado tiene los leds, y los activa depende esto.

Además, declara 4 clocks, donde solo utiliza uno de los clocks.

1. Periféricos de la Placa Altera DE1-SoC utilizados:

Clock (marca 4 pero usa uno solo)

LEDs

Switches

5. La asignación de pines es la generada en el código.

The DE-series board has fixed pin assignments. Having finished one design, the user will want to use the same pin assignment for subsequent designs. Going through the procedure described above becomes tedious if there are many pins used in the design. A useful Quartus II feature allows the user to both export and import the pin assignments from a special file format, rather than creating them manually using the Assignment Editor. A simple file format that can be used for this purpose is the Quartus II Settings File (QSF) format. The format for the file for our simple project (on a DE2 board) is:

set\_location\_assignment PIN\_N25 -to x1

set\_location\_assignment PIN\_N26 -to x2

set\_location\_assignment PIN\_AE22 -to f

By adding lines to the file, any number of pin assignments can be created. Such qsf files can be imported into any design project.

6. Con sintetizar genera el esquemático dado en el RTL

8.

Los LCAs están compuestos de una gran cantidad de celdas lógicas cuya función es programable, dichas celdas pueden ser interconectadas mediante conexiones programables de varios tipos. Estos dispositivos y sus desarrollos posteriores dan origen a las hoy llamadas FPGAs. Las FPGAs están basados en una estructura regular de bloques de procesamiento e interconexiones programables, rodeados de bloques dedicados a entrada salida. Los elementos de procesamiento pueden realizar funciones simples de un bajo número de entradas para dar como resultado una única salida. Esos bloques o celdas internamente están compuestos por una Look-Up Table (LUT) más algún elemento de memoria o flip-flop.

El objetivo de los simuladores se centraba en la captura y la verificación funcional del diseño (Computer Aided Engineering - CAE). Adicionalmente, surgieron herramientas para resolver el layout de los circuitos: determinar dónde ubicar las compuertas lógicas en el silicio y cómo conectarlas o rutearlas (Computer Aided Design – CAD). El término EDA (Electronic Design Automation) engloba las dos definiciones anteriores. Para diseñar con dispositivos lógicos programables es absolutamente necesario el uso de un conjunto de herramientas software de buena calidad, confiables y de fácil utilización.

La síntesis de alto nivel es el proceso de compilar una descripción comportamental dada en un lenguaje de alto nivel en una descripción estructural dada a nivel de transferencia de registros (RTL register-transfer level).

La síntesis lógica es el proceso de convertir una descripción RTL en un netlist optimizado. El netlist es una descripción del circuito a nivel de compuertas y registros y las interconexiones entre esos elementos. Las tareas realizadas en el proceso de síntesis incluyen la optimización de la lógica combinatoria y la optimización de la lógica secuencial.

El mapeo tecnológico se refiere a la mapeo de un determinado netlist genérico en las primitivas o bloques disponibles en una determinada tecnología. En el caso de FPGAs, por ejemplo, se deberán mapear las funciones lógicas en las LUTs disponibles en la familia utilizada. Debe hacerse algo similar para los registros, pasando de un registro genérico a los tipos de registro disponibles en la tecnología.

Las primeras etapas pueden hacerse con herramientas genéricas independientes de los fabricantes de dispositivos programables, pero las etapas de placement y routing requieren la utilización de herramientas propietarias.

It is well established that the quality of an FPGA-based implementation is largely determined by the effectiveness of accompanying suite of CAD tools. Benefits of an otherwise well designed, feature rich FPGA architecture might be impaired if the CAD tools cannot take advantage of the features that the FPGA provides.

The software flow (CAD flow) takes an application design description in a Hardware Description Language (HDL) and converts it to a stream of bits that is eventually programmed on the FPGA. The process of converting a circuit description into a format that can be loaded into an FPGA can be roughly divided into five distinct steps, namely: synthesis, technology mapping, mapping, placement and routing. The final output of FPGA CAD tools is a bitstream that configures the state of the memory bits in an FPGA. The state of these bits determines the logical function that the FPGA implements.

* Logic synthesis: The flow of FPGA starts with the logic synthesis of the netlist being mapped on it. Logic synthesis [26, 27] transforms an HDL description (VHDL or Verilog) into a set of boolean gates and Flip-Flops. The synthesis tools transform the register-transfer-level (RTL) description of a design into a hierarchical boolean network. Various technology-independent techniques are applied to optimize the boolean network. The typical cost function of technology-independent optimizations is the total literal count of the factored representation of the logic function. The literal count correlates very well with the circuit area. Further details of logic synthesis are beyond the scope of this book.
* Technology mapping: The output from synthesis tools is a circuit description of Boolean logic gates, flip- flops and wiring connections between these elements. The circuit can also be represented by a Directed Acyclic Graph (DAG). Each node in the graph represents a gate, flip-flop, primary input or primary output. Each edge in the graph represents a connection between two circuit elements. Given a library of cells, the technology mapping problem can be expressed as finding a network of cells that implements the Boolean network. In the FPGA technology mapping problem, the library of cells is composed of k-input LUTs and flip-flops. Therefore, FPGA technology mapping involves transforming the Boolean network into k-bounded cells. Each cell can then be implemented as an independent k-LUT. Technology mapping algorithms can optimize a design for a set of objectives including depth, area or power. The FlowMap algorithm [64] is the most widely used academic tool for FPGA technology mapping. FlowMap is a breakthrough in FPGA technology mapping because it is able to find a depth-optimal solution in polynomial time. FlowMap guarantees depth optimality at the expense of logic duplication. Since the introduction of FlowMap, numerous technology mappers have been designed that optimize for area and run-time while still maintaining the depth-optimality of the circuit [65–67]. The result of the technology mapping step generates a network of k-bounded LUTs and flip-flops.
* Clustering: The logic elements in a Mesh-based FPGA are typically arranged in two levels of hierarchy. The first level consists of logic blocks (LBs) which are k-input LUT and flip-flop pairs. The second level hierarchy groups k LBs together to form logic blocks clusters. The clustering phase of the FPGA CAD flow is the process of forming groups of k LBs. These clusters can then be mapped directly to a logic element on an FPGA. Clustering algorithms can be broadly categorized into three general approaches, namely top-down [39, 78], depth-optimal [84, 100] and bottom-up [14, 17, 43]. Top-down approaches partition the LBs into clusters by successively subdividing the network or by iteratively moving LBs between parts. Depth-optimal solutions attempt to minimize delay at the expense of logic duplication. Bottom-up approaches are generally preferred for FPGA CAD tools due to their fast run times and reasonable timing delays. They only consider local connectivity information and can easily satisfy clusters pin constraints. Top-down approaches offer the best solutions; however, their computational complexity can be prohibitive.
* Placement: Placement algorithms determine which logic block within an FPGA should implement the corresponding logic block (instance) required by the circuit. The optimization goals consist in placing connected logic blocks close together to minimize the required wiring (wire length-driven placement), and sometimes to place blocks to balance the wiring density across the FPGA (routability-driven placement) or to maximize circuit speed (timing-driven placement). The 3 major classes of placers in use today are min-cut (Partitioning-based) [6, 40], analytic [32, 53] which are often followed by local iterative improvement, and simulated annealing based placers [37, 105].
* Routing: The FPGA routing problem consists in assigning nets to routing resources such that no routing resource is shared by more than one net. Path f inder [80] is the current, state-of-the-art FPGA routing algorithm. Path f inder operates on a directed graph abstraction G(V, E) of the routing resources in an FPGA. The set of vertices V in the graph represents the IO terminals of logic blocks and the routing wires in the interconnect structure. An edge between two vertices represents a potential connection between them.
* Timing analysis: Timing analysis [99] is used for two basic purposes:

• To determine the speed of circuits which have been completely placed and routed,

• To estimate the slack [68] of each source-sink connection during routing (placement and other parts of the CAD flow) in order to decide which connections must be made via fast paths to avoid slowing down the circuit.

* Bistream generation: Once a netlist is placed and routed on an FPGA, bitstream information is generated for the netlist. This bitstream is programmed on the FPGA using a bitstream loader. The bitstream of a netlist contains information as to which SRAM bit of an FPGA be programmed to 0 or to 1. The bitstream generator reads the technology mapping, packing and placement information to program the SRAM bits of Look-Up Tables. The routing information of a netlist is used to correctly program the SRAM bits of connection boxes and switch boxes.

The CAD flow involves the following steps:

• Design Entry – the desired circuit is specified either by means of a schematic diagram, or by using a hardware description language, such as Verilog or VHDL.

• Synthesis – the entered design is synthesized into a circuit that consists of the logic elements (LEs) provided in the FPGA chip.

• Functional Simulation – the synthesized circuit is tested to verify its functional correctness; this simulation does not take into account any timing issues.

• Fitting – the CAD Fitter tool determines the placement of the LEs defined in the netlist into the LEs in an actual FPGA chip; it also chooses routing wires in the chip to make the required connections between specific Les.

• Timing Analysis – propagation delays along the various paths in the fitted circuit are analyzed to provide an indication of the expected performance of the circuit.

• Timing Simulation – the fitted circuit is tested to verify both its functional correctness and timing.

• Programming and Configuration – the designed circuit is implemented in a physical FPGA chip by programming the configuration switches that configure the LEs and establish the required wiring connections.

The FPGA device must be programmed and configured to implement the designed circuit. The required configuration file is generated by the Quartus II Compiler’s Assembler module.

Cuando quiero correr el programa y me tira la ventanita: the configuration file “nombre”.sof is listed in the window. If the file is not already listed, then click Add File and select it. This is a binary file produced by the Compiler’s Assembler module, which contains the data needed to configure the FPGA device. The extension .sof stands for SRAM Object File. Note also that the device selected is EP2C35F672, which is the FPGA device used on the DE2 board.

9. bitstream es el código ya armado. El circuito lo que hace es correr un contador.

¿Qué diferencias encuentra al programar el FPGA de esta manera en lugar de hacerlo en modo Active (quad) Serial usando el dispositivo de configuración EPCS128? ¿En qué escenarios es más adecuado el uso de cada una de estas alternativas?

Altera’s DE-series board allows the configuration to be done in two different ways, known as JTAG and AS modes. The configuration data is transferred from the host computer (which runs the Quartus II software) to the board by means of a cable that connects a USB port on the host computer to the leftmost USB connector on the board.

In the JTAG mode, the configuration data is loaded directly into the FPGA device. The acronym JTAG stands for Joint Test Action Group. If the FPGA is configured in this manner, it will retain its configuration as long as the power remains turned on. The configuration information is lost when the power is turned off.

The second possibility is to use the Active Serial (AS) mode. In this case, a configuration device that includes some flash memory is used to store the configuration data. Quartus II software places the configuration data into the configuration device on the DE-series board. Then, this data is loaded into the FPGA upon power-up or reconfiguration. Thus, the FPGA need not be configured by the Quartus II software if the power is turned off and on. The choice between the two modes is made by the RUN/PROG switch on the DE-series board. The RUN position selects the JTAG mode, while the PROG position selects the AS mode.

La diferencia es que puedo mantener lo que grabe en el FPGA con un modo y con el otro no.

Con active serial precisas un dispositivo de memoria flash para poder almacenar dicho programa.

En que escenarios es conveniente? No se me ocurre.

2) El always y el assing tienen que estar siempre incluidos en los modulos, uno es para secuenciales y el otro para combinacionales.

El always sirve para identificar cuando algo de adentro del código del modulo cambia.

Si pongo always@(\*) se fija si cambia cualquier cosa adentro ejecuta, sino pongo n1 o n2, esto es para el modulo A.

El bloque always va a ejecutar cuando cualquiera de los bloques de la lista sensitiva cambia.

Para el modulo B no necesito el always, ya que cambia solo con el modulo A. Y aquí no tengo que poner wire para input ni para output debajo de su definición, ya que no está el always.

Si hay un always si o si siempre hay que poner output como reg.

Actividad 2:

Aquí básicamente lo que se hace es armar hardware, armar el circuito vos mismo a bajo nivel, y poder ver los resultados de esto, que es lo que da, de modo tal de poder hacerlo lo más eficiente posible.

Pongo los switches según el numero que quiero.

Con el key 3 lo aprieto varias veces y lo mando a alguno de los 8 registros latch

Con el key 2 luego lo presiono una vez y lo envio a los 8 registro

Pones un numero con el switch, key 3 y key 2.

Pones otro numero key 3 y key 2.

Y asi, luego con key 1 voy viendo los números.

Actividad 3:

The Qsys tool allows a designer to choose the components that are desired in the system by selecting these components in a graphical user interface. It then automatically generates the hardware system that connects all of the components together.

The Nios II processor is connected to the memory and I/O interfaces by means of an interconnection network called the Avalon switch fabric. This interconnection network is automatically generated by the Qsys tool.

A special JTAG UART interface is used to connect to the circuitry that provides a USB link to the host computer to which the DE-series board is connected. This circuitry and the associated software is called the USB-Blaster. Another module, called the JTAG Debug module, is provided to allow the host computer to control the Nios II system. It makes it possible to perform operations such as downloading Nios II programs into memory, starting and stopping the execution of these programs, setting breakpoints, and examining the contents of memory and Nios II registers.

Eight slider switches on the DE1-SoC board, SW 7−0, are used to turn on or off eight LEDs, LEDR7−0. To achieve the desired operation, the eight-bit pattern corresponding to the state of the switches has to be sent to the output port to activate the LEDs. This will be done by having the Nios II processor execute a program stored in the on-chip memory. Continuous operation is required, such that as the switches are toggled the lights change accordingly.

After completing the New Project Wizard to create the project, in the main Quartus Prime window select Tools > Qsys, which leads to the window in Figure 4. This is the System Contents tab of the Qsys tool, which is used to add components to the system and configure the selected components to meet the design requirements. The available components are listed on the left side of the window.

The hardware system that will be generated using the Qsys tool runs under the control of a clock. For this tutorial we will make use of the 50-MHz clock that is provided on the DE1-SoC board. Your hardware system should contain a clock source called clk\_0, whose frequency is 50-MHz. You can check that its frequency is indeed 50-MHz by double clicking the component, and checking the Clock frequency parameter of the component. If your system does not already contain clk\_0, it is possible to add a clock source by selecting Basic Functions > Clocks; PLLs and Resets > Clock Source in the IP Catalog tab, then clicking Add....

4. Next, specify the processor as follows: • On the left side of the Qsys window expand Processors and Peripherals, select Embedded Processors > Nios II (Classic) Processor and click Add..., which leads to the window in Figure 5.

• Choose Nios II/e which is the economy version of the processor. This version is available for use without a paid license. The Nios II processor has reset and interrupt inputs. When one of these inputs is activated, the processor starts executing the instructions stored at memory addresses known as reset vector and interrupt vector, respectively. Since we have not yet included any memory components in our design, the Qsys tool will display corresponding error messages. Ignore these messages as we will provide the necessary information later. You must disable the Include reset\_req signal for OCI RAM and MultiCycle Custom Instructions setting in the Advanced Features tab to ensure proper operation. It is important that the checkbox is unchecked, otherwise you will experience issues when loading programs onto the system using the Altera Monitor Program. Click Finish to return to the main Qsys window, which now shows the Nios II processor specified as indicated in Figure 6.

Y veo las siguientes instrucciones viendo las figuras.

Para el inciso 3 tengo que exportar select clock y los dos pio, porque sino no van a andar los switches.

pio\_1 to LEDs

pio\_0 to switches

Detalle importante tuve que definir a los pio0 y pio 1 como LEDr y switches sino no iba a andar. Tuve que cambiar el tamaño de la memoria on chip a 65536

Tuve que abrir desde tolos el nios II de eclipse desde quartus prime, de ahí en el hello world, copio y pego el código provisto por la cátedra.

Inciso 10:

Puedo ver que nohal trabaja directamente con las direcciones de los leds, el otro de main.c se abstrae.

El problema que tiene el nohal, es que tiene definidas las direcciones de los leds, las cuales pueden cambiar en mi integración de qsys.

Cada vez que yo haga lo de base addres assign puede que cambie la dirección de los switches y los leds. Entonces el programa nohal siempre tiene la misma, y puede que no ande, por lo tanto, hay que cambiarlo en el código del nohal. No es para nada portable, tiene ese problema, si yo lo muevo a otro equipo, no es seguro que funcione el código. En cambio, con main si anda, ya que trabaja directamente con los nombres de las variables leds\_base, que están definidas en system.h, y este system.h te va a dar a partir del sopcinfo, las direcciones base asignadas a dichas componentes.

Lo bueno que tiene el main.c es que es mucho más portable, utiliza funciones predefinidas.

Lo mas importante es que si introducis una componente en el nohal, se te cambian todas las direcciones y las tenes que cambiar todas a mano.

En cambio en el main, al agregar una componente, se regenera y listo, tan solo agrego un pequeño cambio.

Inciso 11: Yo con el qsys lo que puedo hacer es definir las conexiones y solamente agarrar cosas que necesito, no necesariamente todas las funciones de cada componente, asi es mucho más minimalista el código y más pequeño, para cada propósito específico se realiza.

Proceso de desarrollo:

Definir el DSOC1 en el programita. Abrir herramienta de integración de qsys en el proyecto generado, agregar las componentes necesarias dentro de este, hacer el cableado, hacer la exportación de salidas.

Veo aca la ventaja de que al hacer el cableado, solo puedo utilizar lo que realmente preciso y no cosas redundantes, haciendo mucho más pequeño el programa y puede que ocupe menos memoria, y además puede llegar a ser más robusto al estar mas restringido.

Además, si quiero agregar algo, es muy sencillo, tan solamente tendría que entrar al sopcinfo y agregar una componente que precise y hacer el cableado.

Esta es la gran ventaja que veo en cuanto a agregar cosas a futuro, con el de1soc yo definia las componentes y directamente trabajaba sobre eso sin poder agregar otras.

Con este procesador soft core yo podría agregar otras componentes sin importar, y así hacer un desarrollo mas extenso.

Preguntar que onda que es el soft core, que otra diferencia tiene.

Agarrando el .qsys es de fácil modificación con respecto a cualquier otro hardware. Con otro hardware no puedo modificar nada, con este en cambio si, inclusive puedo agregar cosas, es super flexible

Actividad 4

Inciso 3. Puedo distribuirlo como quiera, puedo tener un pio solo para todo

Lo que te cambia es que ocupa un poco de mas un pio, osea un pio ocupa 8 bits, y el siguiente esperas que ocupe 8 bits también, pero ocupa un poco de mas, que es para resetear lectura, cosas de control, cada pio ocupa 15 o 16 bits.

Si vos definis un pio todo junto, tenes todos los bits de datos juntos, y los de control al final

Si pones los pios todos separados, entonces, la traslación a memoria se complica, ya que le tiene que ir sumando 15 por ejemplo para ir pasando a la siguiente dirección y obtener los datos, osea seria que tenes intercalado datos,bits de control, datos, bits de control, esto puede ser que te complique.

En nustro programa, nosotros pasábamos las direcciones del PIO, pero pudieramoss ahber hecho, conseguir la dirección del PIO 0 y después sumarle algo y pasar a obtener los datos del siguiente pio, pero anda a saber cuanto le tenes que sumar para obtenerlo.

7. El de harware es mas eficiente, es lo que tarda el retardo de las compuertas nomas, sin embargo lo que tiene de problema este es que a medida que yo aumento la cantidad de bits que quiero ordenar o componentes, se agranda muchísimo las componentes de hw que necesito. No es escalable

El de software es mucho menos eficiente, por ejemplo quicksort o mergeSort, pero lo bueno que tiene es que si tenes muchos mas números que ordenar, este es más genérico y no tenes que agregar nada, es escalable.