EDU-FPGA y su programación en `apio`

1. Introducción

La EDU-CIAA-FPGA o EDU-FPGA es una placa diseñada por el Grupo de Investigación y Desarrollo de Sistemas Embebidos (ASE) de la UTN FR Haedo. La documentación oficial del proyecto puede ser encontrada en su Wiki [2]. Esta placa fue diseñada con tres objetivos fundamentales en mente: bajo costo, flexibilidad y facilidad de uso. Está orientada a la enseñanza en nivel secundario, terciario y universitario, y también para cualquier persona interesada en adentrarse en el mundo FPGA.

1.1. Bloques funcionales

La placa se basa en una FPGA Lattice iCE40 HX4K a la cual se agregaron los mínimos componentes necesarios para su funcionamiento y fácil utilización:

1. Memoria flash para almacenar la configuración de la placa
2. Interfaz FTDI para comunicar la placa con la PC mediante USB
3. Pulsadores y LEDs incorporados para probar ejemplos sencillos sin necesidad de componentes externos

La Figura 1 muestra un diagrama en bloques simplificado y la forma en que se interconectan varios de los componentes de la placa de desarrollo.

1.2. Componentes de la placa

Como se mencionó antes, la FPGA elegida como núcleo de la plataforma es la Lattice ICE40 HX4K TQ144, la cual posee las siguientes especificaciones:

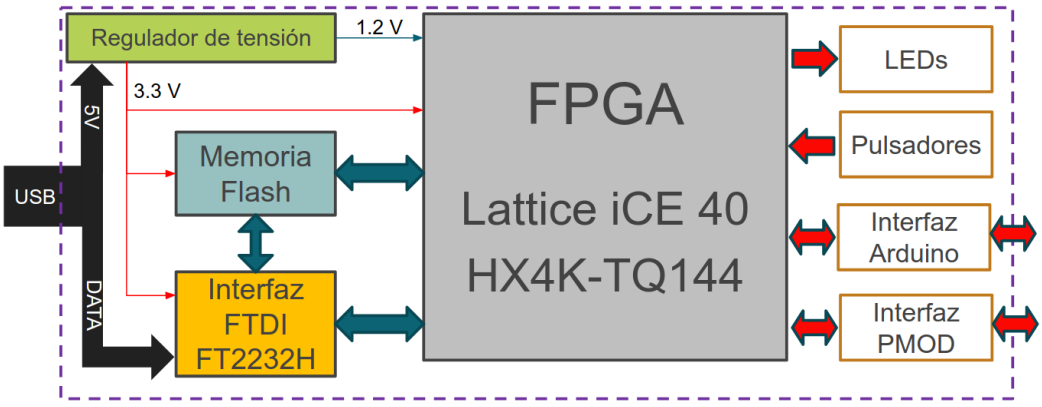

Figura 1: Diagrama en bloques de la plataforma

Lattice ICE40 HX4K TQ144	Datos
Celdas Lógicas	3520
Bloques de Memoria RAM de 4Kbits	20
Memoria Total	80 Kbit
Phase Locking Loops (PLL)	2
Bloques DSP	No
Pines E/S	95
Pares Diferenciales E/S	12
Encapsulado	TQ144

Al igual que la mayor parte de las FGPA, la Lattice ICE40 HX4K carece de memoria no volátil interna para almacenar su configuración. Por lo tanto, es necesario agregar una memoria flash que cumpla esta función. A tal fin, la EDU-FPGA v1.2 posee una memoria flash SPI WinBond W25X40CL con las siguientes características:

WinBond W25X40CL	Datos
Tensión de Alimentación	2.3 a 3.6 V
Interfaz de Comunicación	SPI
Frecuencia de Trabajo	104 MHz
Capacidad	4 MBit

La interfaz de configuración entre el conector USB que utiliza la placa para conectarse a la PC y la conexión SPI de la memoria flash y la FPGA se implementa a través de un chip FTDI FT2232H.

Finalmente, se incluyeron dos reguladores lineales de la serie LD1117 para proveer las tensiones de alimentación necesarias para los componentes:

Regulador	Tensión	Alimenta a
LD1117S12	1.2 V	Núcleo de la FPGA
LD1117S33	3.3 V	E/S de la FPGA, memoria flash, interfaz FTDI

1.3. Periféricos y pines de E/S

La placa cuenta con un total de 49 pines disponibles de entrada/salida de propósito general (GPIO)conectados a los 4 bancos de E/S de la FPGA iCE40. Esta gran cantidad de conexiones disponibles le ofrece al usuario innumerables posibilidades, contribuyendo a la flexibilidad y escalabilidad de la plataforma.

Los pines de la FPGA trabajan a 3.3 V y pueden manejar corrientes de hasta 6mA. No todos los pines de la FPGA están conectados; aquellos pines accesibles en cada banco son:

Banco E/S de la FPGA	Pines accesibles	Cantidad total
Banco 0	122, 124, 125, 128-130, 134-139, 141-144	16
Banco 1	79-85, 95-99, 104-107	16
Banco 2	37	1
Banco 3	7-12, 15-24	16
	TOTAL	**49**

Por otro lado, la placa cuenta con un oscilador de cristal a 12 MHz, 4 pulsadores y 4 luces LEDs que sirven para para agilizar el desarrollo y ensayo de prototipos sin necesidad de conectar componentes adicionales. Estos elementos están conectados a los siguientes pines de la FPGA:

Componente	Pin de la FPGA
Cristal 12 MHz	94
LED Verde	1
LED Rojo	2
LED Amarillo	3
LED Azul	4
Pulsadores	31, 32, 33 y 34

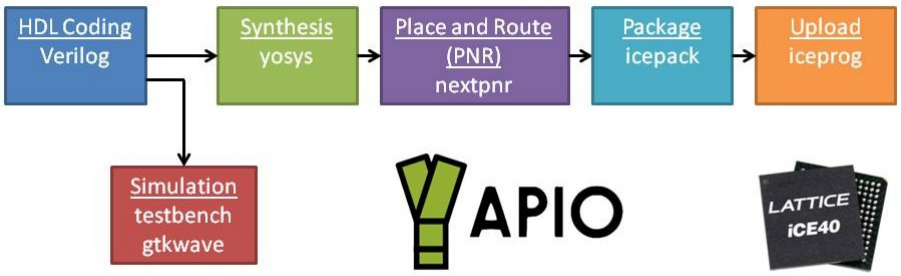

Figura 2: Flujo de diseño para FPGA

2. Flujo de Diseño

El diagrama de la Figura 2 muestra los pasos del proceso típico para crear un diseño para un FPGA [1]. Si está acostumbrado a programar para computadoras o microcontroladores, notará que los nombres de los pasos son diferentes de lo que podría estar acostumbrado (por ejemplo, no hay un paso de "compilación"). Tenga en cuenta que cada paso enumera la herramienta que planeamos usar a lo largo de este curso.

- **Codificación HDL (HDL Coding):** en lugar de conectar componentes físicos en una placa de prueba o mandar a construir una PCB personalizada para piezas lógicas digitales discretas, describirá su diseño en un HDL. Los tres lenguajes de descripción de hardware más populares son VHDL, Verilog y SystemVerilog. Usaremos Verilog ya que es el único lenguaje totalmente compatible con el resto del conjunto de herramientas open-source en este momento.
- **Simulación (Simulation):** antes de construir y cargar el diseño en un FPGA real, a menudo querrá simular el diseño para asegurarse de que todo funcione. Este paso implica escribir más código HDL para alternar las entradas y las señales de reloj para que pueda observar el comportamiento de salida en un programa de simulación. El código HDL de prueba se conoce como "banco de prueba" (*testbench*). Escribiremos bancos de prueba en Verilog y simularemos el diseño en `gtkwave`.
- **Síntesis (Synthesis):** como HDL no es un lenguaje de procedimiento (es decir, su código no se ejecuta secuencialmente en un procesador), no se puede "compilar". Más bien, el proceso de "síntesis" toma el código HDL y lo convierte en representaciones a nivel de compuertas que la FPGA puede entender. En esencia, convierte su código en un circuito digital. Usaremos la herramienta `yosys` para la síntesis.
- **Place and Route (PNR):** la salida de la síntesis es solo la estructura del circuito necesaria para realizar su diseño (como se detalla en HDL). La herramienta PNR determina exactamente cómo conectar las diversas partes dentro de su FPGA particular para que esto suceda. El proceso es similar al uso de un enrutador automático en un programa de diseño de PCB. Usaremos `nextpnr` como nuestra herramienta PNR.
- **Empaquetado (Package):** El resultado de la herramienta PNR es un archivo ASCII legible por humanos que detalla qué conexiones deben realizarse entre los componentes dentro de la FPGA. La herramienta de empaquetado convierte el archivo ASCII en un archivo binario que puede ser leído por el proceso de configuración en el chip de la FPGA. Para este proceso, utilizaremos `icepack`, que forma parte del conjunto de herramientas del Proyecto IceStorm.
- **Impactar en la FPGA (Upload):** finalmente, el archivo binario debe cargarse en la FPGA. Tenga en cuenta que la mayoría de los FPGA modernos no contienen memoria flash no volátil integrada. Como resultado, el archivo binario generalmente se carga en un chip de memoria flash en el mismo PCB que el FPGA. En el arranque, la FPGA lee los datos binarios almacenados para configurar sus celdas lógicas. Usaremos `iceprog` (también parte del Proyecto IceStorm) para cargar nuestro diseño sintetizado y enrutado a la FPGA.

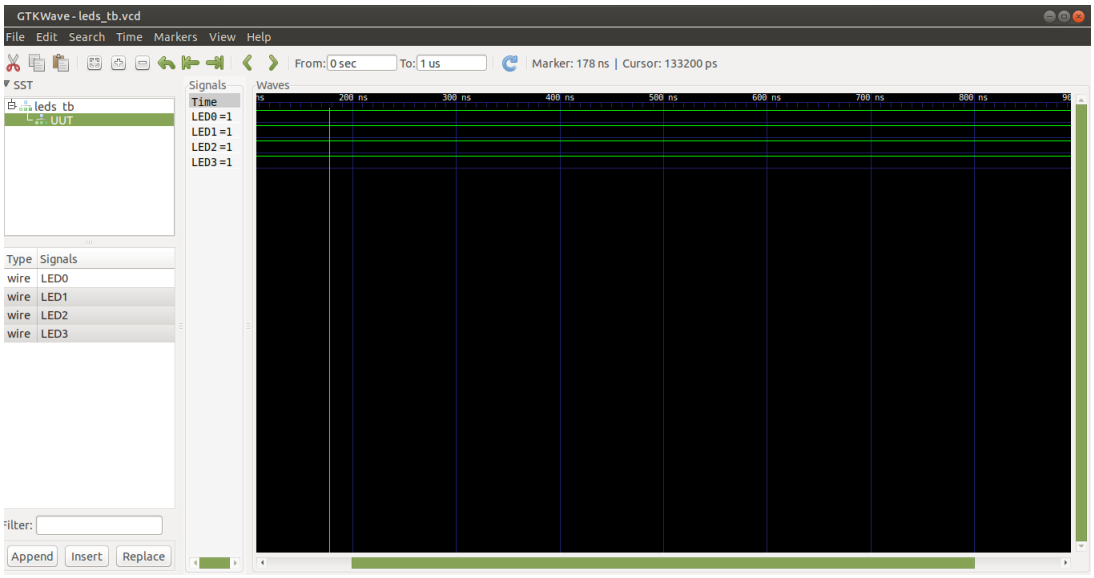

Figura 3: Simulación ejemplo encendido 4 leds en gtkwave

No necesitará instalar y usar estas herramientas individualmente. Confiaremos en la herramienta `apio` de código abierto y multiplataforma para ayudar a administrar estas herramientas. Como analogía, si `yosys` es como `gcc`, entonces `apio` es como el IDE de programación. `apio` está hecho específicamente para funcionar con las populares placas de desarrollo `iCE40` y hace que el proceso de construcción/carga sea mucho más fácil para que podamos concentrarnos en aprender los componentes básicos del diseño de FPGA.

3. Programación con `apio`

Para utilizar `apio` debemos tener instalados un intérprete de `Python` 3.5 (o superior) y `pip`.
Instale la última versión de `apio`:

```
pip install -U apio
```

Ejecute los siguientes comandos para instalar todas las herramientas y controladores necesarios:

```
apio install --all
apio install drivers
```

Muchas de las placas de desarrollo FPGA contienen un chip FTDI para traducir la comunicación USB a la comunicación UART (por ejemplo, para que pueda cargar su diseño FPGA sintetizado/empaquetado en la memoria flash integrada). Si su placa tiene un chip FTDI (la EDU-FPGA tiene uno), ingrese el siguiente comando:

```
apio drivers --ftdi-enable
```

3.1. Ejemplo: leds

Clonar el repositorio de Github `edu-ciaa-fpga-verilog` e ingresar a la carpeta `01-leds`:

```
git clone https://github.com/gmsanchez/edu-ciaa-fpga-verilog.git
cd edu-ciaa-fpga-verilog/01-leds
```

Verifique y simule el diseño:

```
apio verify
apio sim
```

Esto debería abrir gtkwave. Los cuatro pines LED (pines 1 a 4) deben tener una lógica ALTA (ver Fig.3).
Cuando esté satisfecho con la simulación, construya y cargue el diseño en su FPGA:

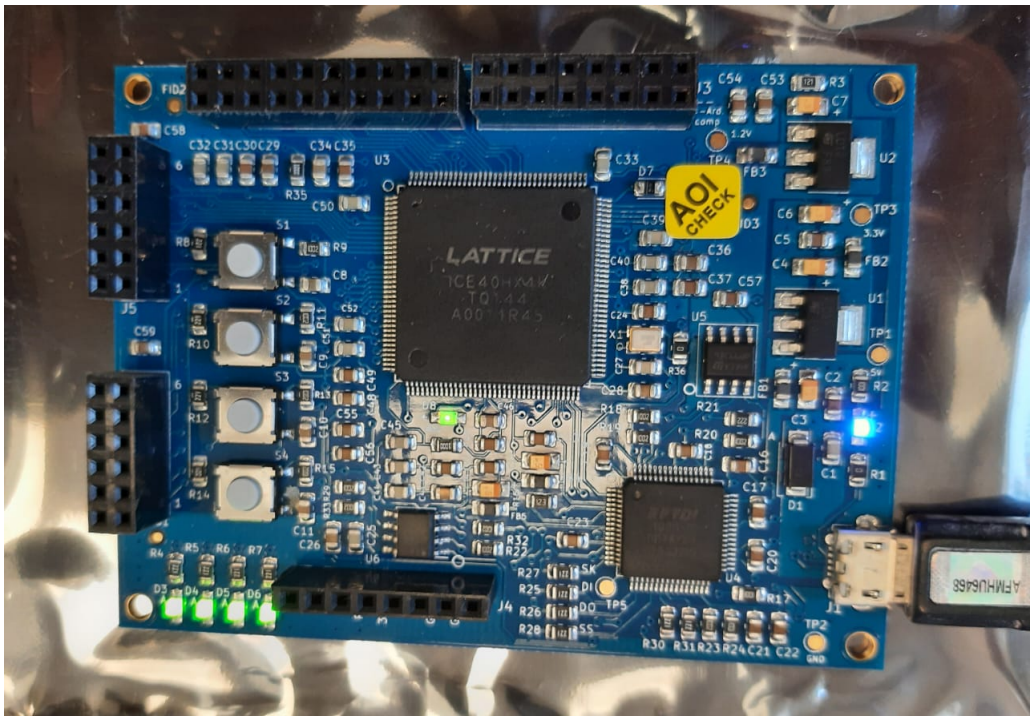

Figura 4: edu-ciaa-fpga con sus cuantro leds encendidos

```
apio build
apio upload
```

Debería ver que se encienden los LED en su placa FPGA (ver Fig.4).

Referencias

[1] Shawn Hymel. *Introduction to FPGA Part 1 - What is an FPGA?* `https://www.digikey.com/en/maker/projects/introduction-to-fpga-part-1-what-is-an-fpga/3ee5f6c8fa594161a655a9f960060893`. [Online; último acceso 28-Feb-2023]. 2023.

[2] Grupo de Investigación y Desarrollo de Sistemas Embebidos (ASE). *EDU-CIAA-FPGA*. `http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:edu-fpga`. [Online; último acceso 24-Feb-2023]. 2023.