IPD432: Diseño Avanzado de Sistemas Digitales
Semestre II-2021
Tarea 2

Indicaciones generales

- La tarea se puede realizar en forma individual o en grupos de dos personas.
- Debe entregar un informe escrito en formato IEEE conference de doble columna, utilizando los templates disponibles en el siguiente link: `https://www.ieee.org/conferences_events/conferences/publishing/templates.html` (se recomienda el uso de LaTeX).
- Si lo desea, puede entregar su informe en inglés (opcional).
- Habrá penalización en la nota por detalles de formato y presentación (faltas de ortografía, errores gramaticales graves, texto desalineado, figuras en mala calidad, etc.). Esto quedará a criterio del profesor. Sea prolijo, ya que la evaluación del informe se realizará con el mismo criterio que se evalúa un paper.
- La fecha límite de entrega para la tarea (codigos e informe) es el 24 de Noviembre de 2021 a las 23:59 hrs. al mail *gcarvajalb@gmail.com*.

1. Investigación y discusión (40 puntos)

Utilizando como base los documentos *"Can FPGAs beat GPUs?"*, *"FPGAs versus GPUs in Datacenters"*, y *"Single Chip Heterogeneous Computing"*, disponibles en la carpeta Reference Readings de Piazza[1], busque literatura adicional y mas reciente relacionada a estos temas e investigue sobre los principios fundamentales del funcionamiento de FPGAs y GPUs. Escriba un reporte autocontenido que resuma su investigación, presentando una tabla comparativa de las principales ventajas y desventajas de cada arquitectura, identificando y discutiendo tipos de aplicaciones donde se enfatizan sus fortalezas y limitaciones (por ejemplo, aplicaciones orientadas a maximizar throughput o minimizar latencia, determinismo, resolución aritmética, etc.). En su análisis considere no solamente los aspectos técnicos de la arquitectura para procesamiento de datos, sino que también aspectos como usabilidad y penetración en el mercado actual. Ponga especial énfasis en la mención a herramientas de

[1]Estos nombres son de los archivos. Los títulos oficiales y autores los verán cuando descarguen los documentos

High-Level Synthesis y programación a alto nivel, que apuntan a aumentar la productividad en el diseño con FPGAs.

En su discusión indique en que consiste el concepto de *latency hiding* utilizado en la descripción de GPUs, como este principio limita el uso de GPUs en aplicaciones de tiempo real, y que ventajas ofrecen las FPGAs en este aspecto. Además, reporte como encajan las arquitecturas del tipo Many Integrated Cores (MIC) en este ecosistema (por ejemplo, tarjetas tipo Xeon Phi).

Soporte su análisis, discusión, y opiniones con referencias adicionales y actualizadas. Ponga cuidado en la presentación de su reporte. Utilice el template indicado mas abajo para formato de paper. Máximo 5 páginas (incluyendo referencias).

2. Diseño e implementación utilizando SystemVerilog. (60 puntos)

2.1. Coprocesador para operaciones sobre vectores.

El objetivo de esta actividad es la implementación de un coprocesador especializado en la FPGA que entregue soporte a un procesador principal para realizar operaciones sobre vectores. Siguiendo la nomenclatura utilizada y aceptada en el contexto de GPUs (que estudió en la primera parte de esta tarea), nos referiremos al procesador principal como *host* y al coprocesador como *device*[2].

Para el diseño del device considere el diagrama de alto nivel de la Figura 1. La arquitectura contiene dos bloques de memoria del tipo BRAM, cada uno capaz de almacenar 1024 palabras de 8 bits por palabra. Estos bloques de memoria almacenan dos vectores A y B, cuyos elementos son cargados desde el host por medio de la interfaz UART. Los bloques de memoria utilizan una configuración de doble puerta para separar los dominios de lógica de escritura y lectura. El bloque *Processing Core* permite realizar distintas operaciones sobre los vectores almacenados en los bloques de memoria, enviando el resultado de estas operaciones al host y también a puertos de entrada/salida de la tarjeta de desarrollo. Las operaciones a realizar sobre los vectores son configuradas e iniciadas desde el host por medio de comandos enviados a través de la UART, los que son decodificados en el bloque *Command Decoder* para iniciar las acciones correspondientes.

Considere los siguientes requerimientos para la implementación del device:

- Apuntando a la usabilidad de su coprocesador, debe diseñar la arquitectura del dispositivo para que pueda ser usado utilizando el template provisto en el script de Matlab `coprocessorTesting.m`, el cual automaticamente genera vectores de prueba aleatorios y verifica el resultado con la version de software [3].

[2] Se sugiere revisar los conceptos de coprocesadores y/o aceleradores. Encontrarán mucha más literatura asociada al contexto de GPUs, pero los conceptos fundamentales son generales y aplican a cualquier tipo de acelerador, incluyendo los basados en FPGA.

[3] ¿Es esto un analisis estático o dinámico? ¿Cuales son los pros y contras de este tipo de verificacion?

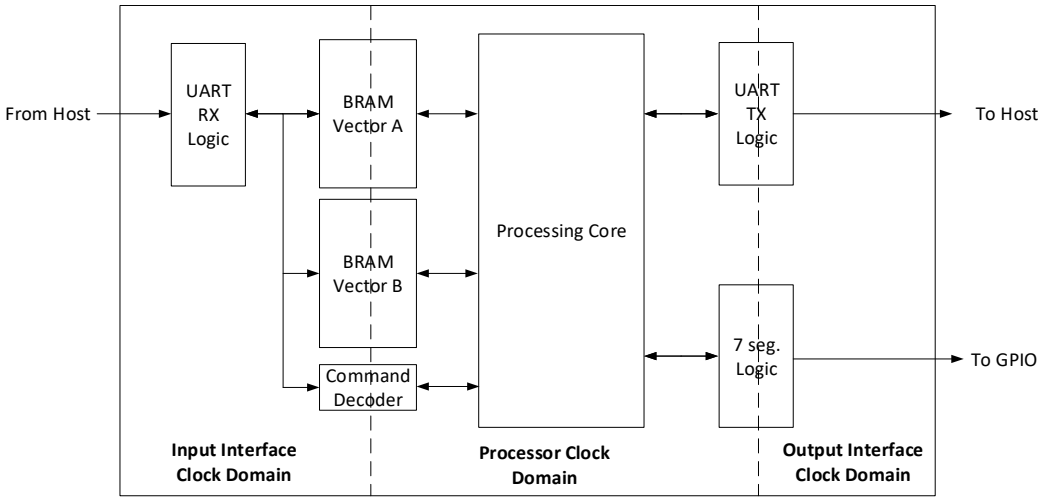

Figura 1: Diagrama de alto nivel de procesamiento en el coprocesador (device).

- Para el envío de vectores desde el host al device, considere que los vectores están almacenados en un archivo de texto con el siguiente formato:

```
Element0
Element1
.
.
.
Element1023
```

- La interfaz de usuario para el uso del corprocesador debe proveer la función `write2dev('file', 'BRAMX', COM)`, donde `file` es el archivo que contiene el vector a almacenar, `BRAMX` es el bloque de memoria donde se quiere almacenar el vector (BRAMA o BRAMB), y `COM` corresponde al puerto a utilizar para comunicarse por la UART. Esta función debe automatizar el llenado de las memorias, leyendo los valores desde el archivo de texto indicado y enviándolos como un solo stream. Por simplicidad, asuma que el usuario proveerá archivos de texto que contengan como máximo 1024 puntos y sin errores [4]. El coprocesador debe incluir lógica para setear las señales de control internas de lectura/escritura en los bloques cuando se ejecute esta función.

- La interfaz también debe proveer la función `command2dev('xxx', COM)`, donde `xxx` especifica una operación a realizar sobre los vectores previamente almacenados. Esta función envía secuencias de bits por la UART que son interpretados por el bloque *Command Decoder*, el cual setea el *Processing Core*, gatilla el cálculo de la operación especificada, y envía el resultado hacia el host por medio de la UART. Se debe soportar como mínimo las siguientes operaciones (puede agregar más si lo desea):

[4] Esto es el escenario ideal para simplificar la tarea. En la práctica, deberían considerar el caso en que el archivo este incorrecto, y decidir entre hacer un check en software verificando que el archivo esta correcto antes de enviar los datos o bien incroporar un chequeo en hardware para abortar y evitar estados inesperados en el caso de que algun dato este corrupto. ¿Cuales serían los pros y contra de cada caso?

- `readVec`: lee el vector almacenado en el bloque de memoria correspondiente (indicado en otro argumento).
- `sumVec`: calcula la suma elemento a elemento de los vectores A y B previamente almacenados.
- `avgVec`: calcula el promedio elemento a elemento de los vectores A y B previamente almacenados.
- `eucDist`: calcula la distancia euclideana entre los vectores A y B previamente almacenados.
- `manDist`: calcula la distancia de Manhattan entre los vectores A y B previamente almacenados.

- En el caso de que las operaciones realizadas sean la distancia Euclideana o la de Manhattan, debe dimensionar el número de bits del resultado para que alcance todo el rango de operación, y el resultado de la operación debe quedar también registrado en los displays de 7 segmentos de la tarjeta de desarrollo en formato decimal, manteniéndose visibles hasta que se realice otra operación. En el caso de realizarse operaciones que no sean de distancia, los displays deben quedar apagados o bien mostrar alguna información útil asociada a la última operación realizada.
- El device implementado en la FPGA debe ser completamente controlado desde el host, no debe requerir manipulación física del usuario (exceptuando el encendido), y una vez encendido debe estar continuamente disponible para recibir comandos (por ejemplo, no debería requerir reiniciarse o apretar un botón luego de cada operación). Por simplicidad, asuma que el usuario no eviará un nuevo comando si no se ha terminado de ejecutar el anterior. Por ejemplo, puede dejar bloqueada la UART si hay una operación en curso para evitar conflictos. Esto simplifica el diseño al omitir el manejo de colas, arbitraje, y/o interrupciones[5].

La revisión y validación de su diseño se hará en base al script `coprocessorTesting.m`. Se enfatiza que el usuario final debe ver la interfaz lo más simple posible y poder usar el hardware en forma intuitiva, siendo responsabilidad del desarrollador ocultar toda complejidad asociada al diseño y manejo del hardware a bajo nivel por medio de librerías que oculten todos los detalles en la codificación de los comandos (no pida al usuario que recuerde o configure bits). En otras palabras, el usuario deberá ser capaz de presionar *play* en el script, y los resultados deberán aparecer mágicamente, es decir, ni siquiera se necesita saber que el hardware externo existe. Una vez finalizado un test, el usuario debe ser capaz de correr otro inmediatamente, sin necesidad de manipular el hardware (cuide en abrir y cerrar puertos y archivos en el script de Matlab). Los argumentos propuestos para cada uno de los comandos son referenciales, y puede incluir argumentos adicionales si los considera necesarios. Sin embargo, debe reportar toda decisión de diseño eneste aspecto y dejar explícita cualquier instrucción para que el usuario pueda usar su código. La usabilidad de su sistema será considerada en la evaluación[6].

[5]Sin embargo, se recomienda como ejercicio pensar en hacer su sistema robusto ante posibles violaciones de estas condiciones o para maximizar la utilización del hardware

[6]No todo es desempeño. Pueden tener el sistema más eficiente del mundo, pero se vuelve inútil si no es usable. Debe investigar un poco más en profundidad sobre este tema en la parte I de esta tarea

Una vez implementado el diseño de hardware para el device, utilice el analizador lógico integrado (ILA) de Vivado para depurar su diseño, observar el estado y flujo de las señales internas de la FPGA, y medir la latencia para cada una de las operaciones, considerando el instante desde que llega el primer bit del comando correspondiente desde el host hasta que sale el último bit del resultado de vuelta al host. Determine la relación entre la latencia en número de ciclos de reloj y el largo de los vectores a procesar. **Recuerde que el core ILA utiliza lógica y agrega overhead en la utilización de recursos y análisis de timing de su sistema, por lo que debe utilizarse para depuración y removerse de la implementación final una vez realizado su análisis.**

En su informe debe incluir un diagrama de bloques de su diseño con el flujo de señales principales entre bloques, además de las codificaciones utilizadas para sus comandos. Incluya también diagramas de estado en caso de que haya usado FSMs. Cualquier funcionalidad o característica que considere necesaria y no este explícitamente especificada en el enunciado queda abierta a su decisión de diseño, lo cual debe ser reportado y justificado en su informe. Recuerde incluir cualquier indicación especial que el usuario deba saber para hacer funcionar el device. El informe debe incluir un reporte y análisis cuantitativo sobre el reporte de uso de recursos lógicos y la caracterización de latencia (en ciclos de reloj). Sea breve y conciso. No incluya código en el informe a menos que lo considere necesario para ilustrar algún punto relevante. El diseño entregado no debe incluir lógica asociada al ILA.

Debe entregarse una carpeta que incluya solo los archivos fuente necesarios para implementar su diseño. Esto incluye los archivos HDL (típicamente carpeta sources de su proyecto), constraints (típicamente carpeta constraint de su proyecto), y cualquier bloque de configuración de IP que haya instanciado. No incluya el archivo de programación .bit, ya que este se generará al momento de la revisión y se probará en la tarjeta Nexys4 DDR. Revise muy bien que esta enviando los archivos correctos y suficientes para implementar su sistema. Se penalizará la nota si no se incluyen todos los archivos necesarios y también si incluye archivos innecesarios derivados del proceso de síntesis e implementación en su computador personal. Recuerde incluir el script de Matlab con la interfaz de usuario que permite validar su diseño, con todas las indicaciones necesarias para hacerlo funcionar.

Actividades propuestas (no evaluadas en esta tarea.)

Explore las opciones que ofrece Vivado para cambiar las directivas de síntesis e implementación. Cambie los constraints de tiempo para determinar la máxima frecuencia de operación que puede alcanzar su diseño. Sintetice e implemente su diseño para distintos chips (no es necesario tener el hardware para analizar los diseños). Evalúe como estas variantes afectan el tiempo de síntesis, uso de recursos, y el desempeño del circuito.

Hay diversa información sobre el uso avanzado de Vivado en entornos profesionales que requieran hacer Design Space Exploration, incluyendo documentación oficial de Xilinx (aunque esta suele estar enterrada en diversos manuales). Para tener una primera aproximación, puede utilizar el siguiente link y hacer búsqueda de referencias adicionales: `https://hwjedi.wordpress.com/2017/01/04/vivado-non-project-mode-the-only-way-to-go-for-serious-fpga-designers/`