Tarea 2

IPD-432 Diseño Avanzado de Sistemas Digitales

24 de octubre del 2025

1st Sven Klein Plarre
Departamento de Electrónica
Universidad Técnica Federico Santa María
Valparaíso, Chile
sven.klein@sansano.usm.cl

2nd Bastián Rivas Cordero
Departamento de Electrónica
Universidad Técnica Federico Santa María
Valparaíso, Chile
bastian.rivas@usm.cl

***Resumen*—Este documento corresponde a un acelerador de procesos implementado en FPGA. Contiene uso de UART, interfaces e IP. Este trabajo es parte de la Tarea 2 del ramo IPD-432 del año 2025.**

***Index Terms*—FPGA, clock, FSM, UART, coprocessing, accelerator**

I. Introducción

Un coprocesador es un dispositivo utilizado para realizar operaciones predefinidas por FPGA o hardware de manera más rápida que un microcontrolador o procesador. Su función es complementar a estos últimos, recibiendo de estos información a procesar y devolviéndola a la unidad de procesamiento central en un tiempo más rápido del que le tomaría a este procesar. El presente coprocesador se comunica por interfaz UART a un computador, recibiendo dos arreglos de 1024 valores de 10 bits, guardándolos en memoria BRAM local y pudiendo luego realizar operaciones entre los valores de las dos memorias y reenviar los resultados al computador. El coprocesador será activado por dos funciones de Matlab: write2dev para escribir un arreglo de datos de 10 bits de ancho y 1024 valores guardados en un documento de texto a una de las 2 memorias y command2dev, el cual indica al coprocesador que operación realizar de las siguientes: lectura, suma vectorial, promedio vectorial, distancia euclidiana, distancia de Manhattan y producto punto. Los resultados de estas operaciones son retornados y procesados en Matlab.

II. Requerimientos

Se requiere en primer lugar diseñar 2 funciones de Matlab. La primera es write2dev('file', 'BRAMX', COM). Esta función toma un archivo 'file.txt' con una columna de 1024 valores de 10 bits de tamaño (entre 0 y 1023). Se envía una instrucción que dicta que se escriban en la memoria BRAMX, que puede ser BRAMA o BRAMB. Y se envían estas instrucciones y datos al puerto COM dado.

El coprocesador debe escribir todos los datos enviados a la memoria indicada por la instrucción de manera correcta.

Luego, se tiene la función command2dev('xxx', COM). Esta función indica una operación 'xxx' al coprocesador que debe realizar. Estas pueden ser "readVec", "sumVec", "avgVec", "eucDist", "manDist" y "dotProd". Cada una de estas operaciones envían un mensaje codificado al puerto COM con la tarea a realizar. Estas son: leer todos los datos de una memoria y enviarlos por UART (en este caso, se inserta un parámetro extra opcional llamado "BRAM" que puede ser "BRAMA" o "BRAMB" que indica de que memoria se debe leer), realizar una suma vectorial de los datos en ambas memorias y enviarlos por UART, realizar un promedio vectorial de los datos en cada memoria y enviar estos valores por UART, obtener la distancia euclidiana entre los vectores almacenados en cada memoria, mostrarlos por el display de 7 segmentos y enviar el resultado por UART, obtener la distancia de Manhattan entre los vectores de cada memoria, mostrarla por el display de 7 segmentos y enviarla por UART y finalmente, obtener el producto punto entre ambos valores, mostrarlo por el display de 7 segmentos y enviarlo por UART.

III. Diseño

El diseño del dispositivo está dividido en 3 dominios: dominio de entrada, dominio de procesos y dominio de salida. Cada uno de estos dominios tienen su propio reloj y señal de reinicio sincronizada con este. Los buses de datos están conectados directamente entre los 3 dominios, siendo las señales y pulsos de comunicación manejados por sincronizadores para ser correctamente interpretados en los diferentes dominios. Debido a que el módulo conectado a la interfaz física UART es el mismo para la entrada y salida de datos y ambas deben operar bajo los mismos parámetros, se utiliza el reloj de entrada para este bloque, dos relojes iguales para ambos dominios y la misma señal de reinicio para estos dos. En la Figura [1] se encuentra un diagrama de bloques del diseño.

III-A. Dominio de entrada

El diseño del coprocesador se divide en 3 secciones o dominios, cada uno con su propio reloj. El primero es el dominio de entrada. En este dominio se toman los datos de la interfaz UART de entrada, se guardan en bytes y estos se usan para entregar las instrucciones de operación a la unidad de control y los datos de los vectores a la memoria.

El primer módulo de este dominio es UART_RX. Este módulo se encarga de recibir la señal de entrada desde la

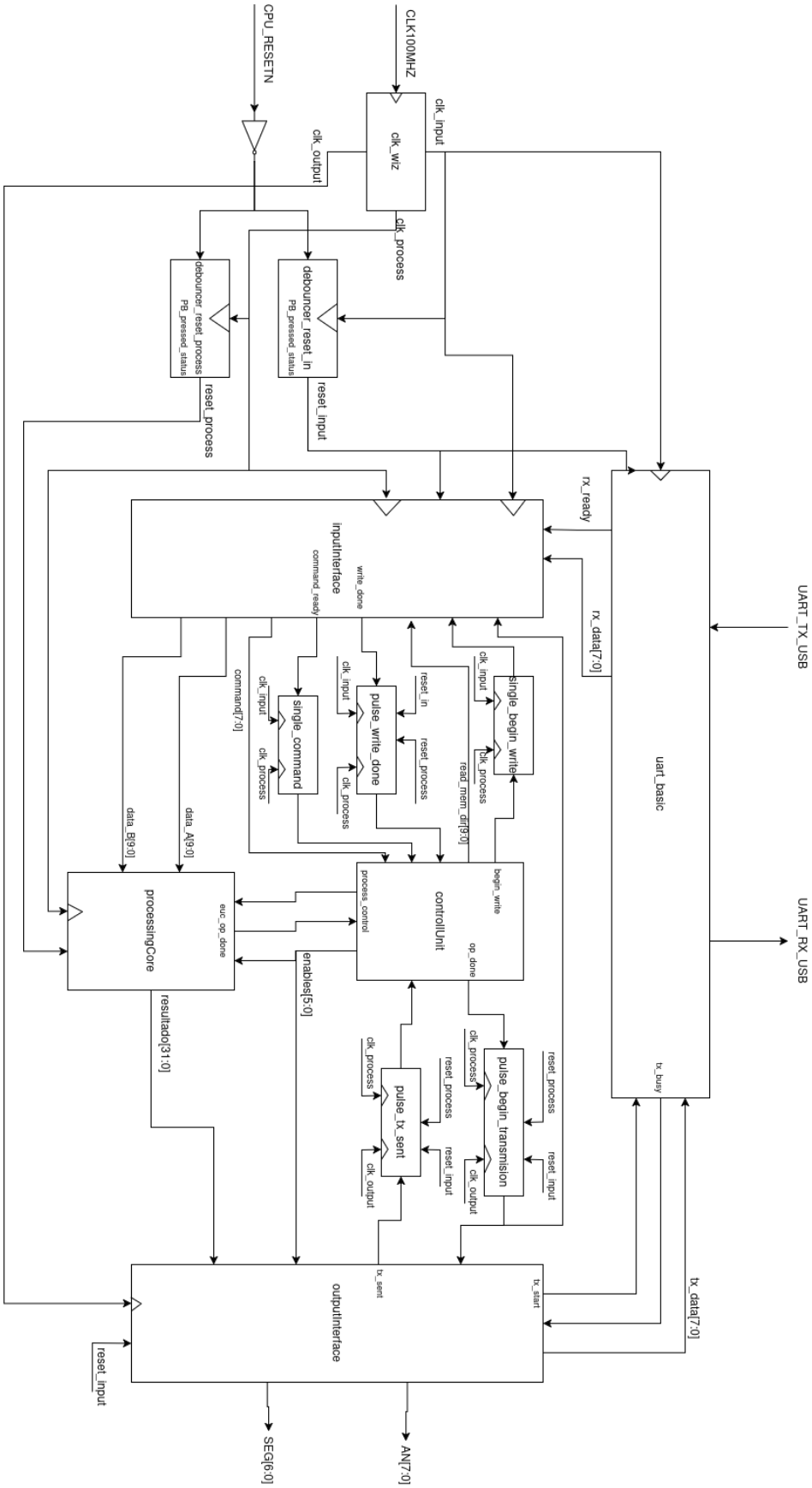

Figura 1. Diagrama de bloques del coprocesador.

interfaz serial y convertirla en buses de un byte. Este módulo recibe como entrada la señal rx de la interfaz serial, la señal reset y el reloj de 100[Mhz] que corresponde a este dominio. Las salidas son un bus de 8 bits con la información recopilada por la UART y una señal "rx_ready" que indica que se termina de recibir el byte completo. Vale decir que este módulo forma parte del módulo "uart_basic", el cual maneja operaciones de escritura y lectura a través de la interfaz.

Luego se encuentra el módulo "inputInterface", el cual contiene varios módulos encargados de procesar los datos recuperados de UART y enviarlos a memoria de ser necesario. Este módulo contiene también las dos unidades de memoria utilizadas para almacenar los datos del proceso, por lo que también posee una entrada para el reloj del dominio de procesos. Las conexiones IO de este módulo se pueden ver en el siguiente listado:

- "input_domain_clk": Reloj del dominio de entrada.
- "processor_domain_clk": Reloj del dominio de procesos.
- "reset": Señal de reinicio.
- "rx_ready": Señal de entrada recibida de la UART. indica que se terminó de recibir un byte y este se puede leer.
- "begin_write": Señal de control de entrada que indica que se deben escribir los datos recibidos en memoria.
- "op_done": Señal de control proveniente de la unidad de control que indica que se completo una operación.
- "rx_data": Bus de 8 bits recibido de UART.
- "read_mem_dir": Bus de 10 bits proveniente de la unidad de control que contiene la dirección de memoria a leer.
- "write_done": Pulso de salida que indica que se terminó la operación de escritura.

El diagrama con los bloques y componentes de la interfaz de entrada se encuentra en la Figura [2], donde se muestra que incluye un decodificador de comandos, una unidad que controla la secuencia de escritura, un contador con la dirección de memoria de escritura y las dos memorias.

III-A1. Decodificación de comandos: El decodificador de comandos es un bloque que recibe un dato ya registrado desde la UART y lo traduce a un conjunto de señales de activación para distintos módulos y a un selector de BRAMs. Una vez que determina qué comando se recibe, entra en un estado de bloqueo, envía las señales de activación hacia el controlador principal a través de `command_out` y reporta que su salida es válida con la señal `command_ready`. Al bloquearse, permanece en dicho estado y mantiene sus salidas estables hasta que se le indique que la operación fue concluida con la señal `op_done`. En el caso particular del comando de escritura, se utiliza la señal `bram_sel` para indicarle al controlador en qué BRAM escribir.

Como decisión de diseño, la codificación de los comandos se presenta en la Tabla [I], donde BRAM Sel es 0 o 1 para escribir en la BRAM A o B, respectivamente. Por otra parte, los bits de `op_code` entregan el código de operación a realizar, viniendo representados por los valores de la Tabla [II]. Los nombres de las operaciones siguen la nomenclatura presentada en el encabezado del ejercicio.

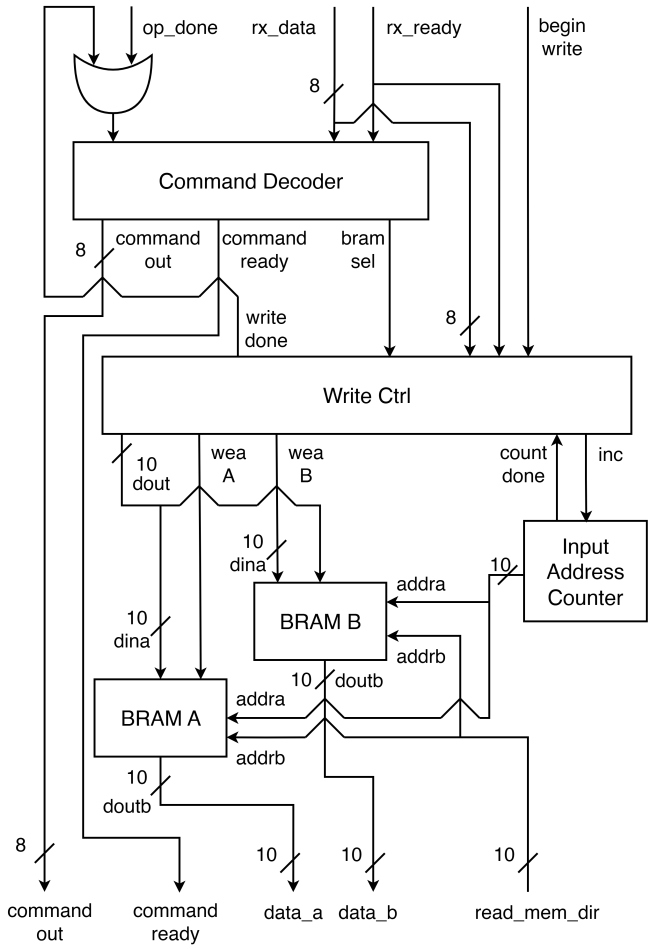

Figura 2. Diagrama de la interfaz de entrada. Se omiten señales clk y reset para mejorar la legibilidad

Tabla I
ESTRUCTURA DE LOS MENSAJES DE COMANDOS RECIBIDOS POR EL DECODIFICADOR.

N° Bit	Uso
7	BRAM Sel
6	x
5	x
4	x
3	x
2	op_code[2]
1	op_code[1]
0	op_code[0]

III-A2. Escritura en memoria: Considerando que, como requerimiento de diseño las memorias tienen palabras de 10 bits, se agrega un módulo controlador "writeCtrl" con el que se acumulan los bytes correspondientes a cada dato. El módulo funciona en conjunto al decodificador, bancos de registros y el contador de dirección de escritura por medio de señales de control activadas en distintos estados. Los estados implementados se presentan en la Figura [3], donde se presentan estados para registrar en memoria la dirección, luego registrar el byte menos significativo, después el más significativo y, en base al

Tabla II
CÓDIGOS DE OPERACIÓN Y SU INTERPRETACIÓN.

Valor de op_code	Operación
001	Write2dev
010	ReadVec
011	SumVec
100	AvgVec
101	EucDis
110	ManDis
111	DotProd
Otro	Inválido

estado del contador de memoria, decidir si esperar a un nuevo dato o reportar a los otros módulos de control que la escritura fue concluida porque el contador alcanza su límite.

El contador de direcciones de entrada o "Input Address Counter" en la Figura [2] es un bloque dedicado a llevar el registro de la dirección donde se escribe en las BRAM. Posee una señal `inc` con la que se le indica que debe incrementar su valor y una señal `count_done` con la que indica que alcanza su valor límite y su vuelta a cero.

Las memorias en sí son módulos BRAM generados con una IP disponible dentro de Vivado. Como requerimiento del ejercicio, estos bloques se definen en configuración *True Dual Port* para tener dos puertos A y B, cada uno con una señal de datos entrantes, datos salientes, dirección, reloj, `enable`, y `write enable`. Para el diseño implementado en este trabajo, se toma la decisión de utilizar el puerto A como uno dedicado al dominio de las entradas y escritura de memoria, mientras que el puerto B se utiliza para leer desde la memoria y se dedica al dominio del procesamiento de datos. Considerando también los bloques diseñados, se decide mantener los puertos `enable` en alto, y para evitar comportamientos indeseados, se decide definir las señales `write enable` y entrada de datos en el puerto B como siempre cero.

III-B. Dominio de procesamiento

En este dominio se encuentran la unidad de control y la de procesos. La unidad de procesos se encarga de recibir datos de memoria y procesarlos de acuerdo a la operación entregada por el usuario. Las entradas de este módulo son los datos de memoria A y B, un bus de 7 enables que indica que operación se realiza en formato one-hot, una señal de control que se encarga de varios procesos dependiendo de la operación a realizar, el reloj del dominio y un reset. Las salidas son un bus de 32 bits con el resultado y una señal de op_done que anuncia cuando la operación de distancia euclidiana se ha completado.

Dentro de este módulo se encuentra un módulo para cada operación a realizar, cada uno con su propio enable. La señal de control indica que memoria leer en la función "ReadVec", se encuentra desconectada de la suma y el promedio y le indica al resto de operaciones cuando tomar y procesar información de los buses de entrada para agregarlos al resultado final. Las salidas de cada módulo con el resultado de la operación realizada se conectan al bus "result" de 32 bits de salida de acuerdo a la entrada de los enables de operación.

La salida de las operaciones se debe tomar en cuenta el peor caso de los datos obtenidos para obtener el tamaño del bus de salida, es decir, el valor de mayor tamaño posible que esta operación puede entregar. En el caso de la lectura, este valor es de 10 bits, igual que los datos. En la suma, el peor caso es de 1023 + 1023, este resultado es de 11 bits. En el caso del promedio el peor caso es el promedio de 1023 y 1023, también pudiéndose guardar en 10 bits. Al calcular la distancia euclidiana, el resultado de mayor valor posible corresponde a:

$$\sqrt{1023^2 \cdot 1024} = 32736$$

este valor puede ser guardado en un registro de 15 bits. Al calcular la distancia de Manhattan, el mayor resultado posible es de 1023 * 1024, resultado que entra en 20 bits. Finalmente, el resultado máximo de la operación de producto punto es de 1023 * 1023 * 1024, el cual necesita 30 bits.

Debido a que el envio de información por uart se realiza en bytes de 8 bits, todas las señales de las operaciones se colocan en buses múltiplos de 8 bits. Por esto las operaciones de lectura, suma, promedio y distancia euclidiana se transmiten en buses de 16 bits, la distancia de Manhattan se transmite en un bus de 24 bits y el producto punto en 32 bits.

Se puede observar un diagrama de los módulos internos de la unidad de procesos en la Figura [4].

La unidad de control se encarga de procesar las señales de control y coordinar los procesos de todos los periféricos. Esta consiste en una máquina de estados de 9 estados para las operaciones necesarias. El módulo se conecta por medio de las siguientes señales de entrada y salida:

- clk: señal de reloj del dominio de procesos.
- reset: señal de reinicio.
- "command_ready": señal de entrada indica que se recibió una instrucción desde UART.
- "write_done": señal de entrada que indica que se terminó de escribir en memoria todos los datos enviados.
- "tx_sent": señal de entrada que indica que un resultado completo se ha enviado por UART.
- "op_ready": señal de entrada que indica que se termino de procesar el resultado de la distancia euclidiana.
- "command": bus de 8 bits de entrada con las instrucciones para el cambio de estados.
- "process_ctrl": señal de salida que controla varios procesos de la unidad de procesos.
- "read_enable": señal que habilita la memoria para lectura.
- "begin_transmision": señal que le indica al dominio de salida que hay un dato procesado listo para ser transmitido.
- "begin_write": señal que le indica al control de escritura que debe empezar a escribir en memoria con los datos recibidos.
- "enables": bus de datos para indicarle a la unidad de control la operación a realizar.
- "mem_dir": bus de datos donde se le indica a la memoria BRAM que datos leer.

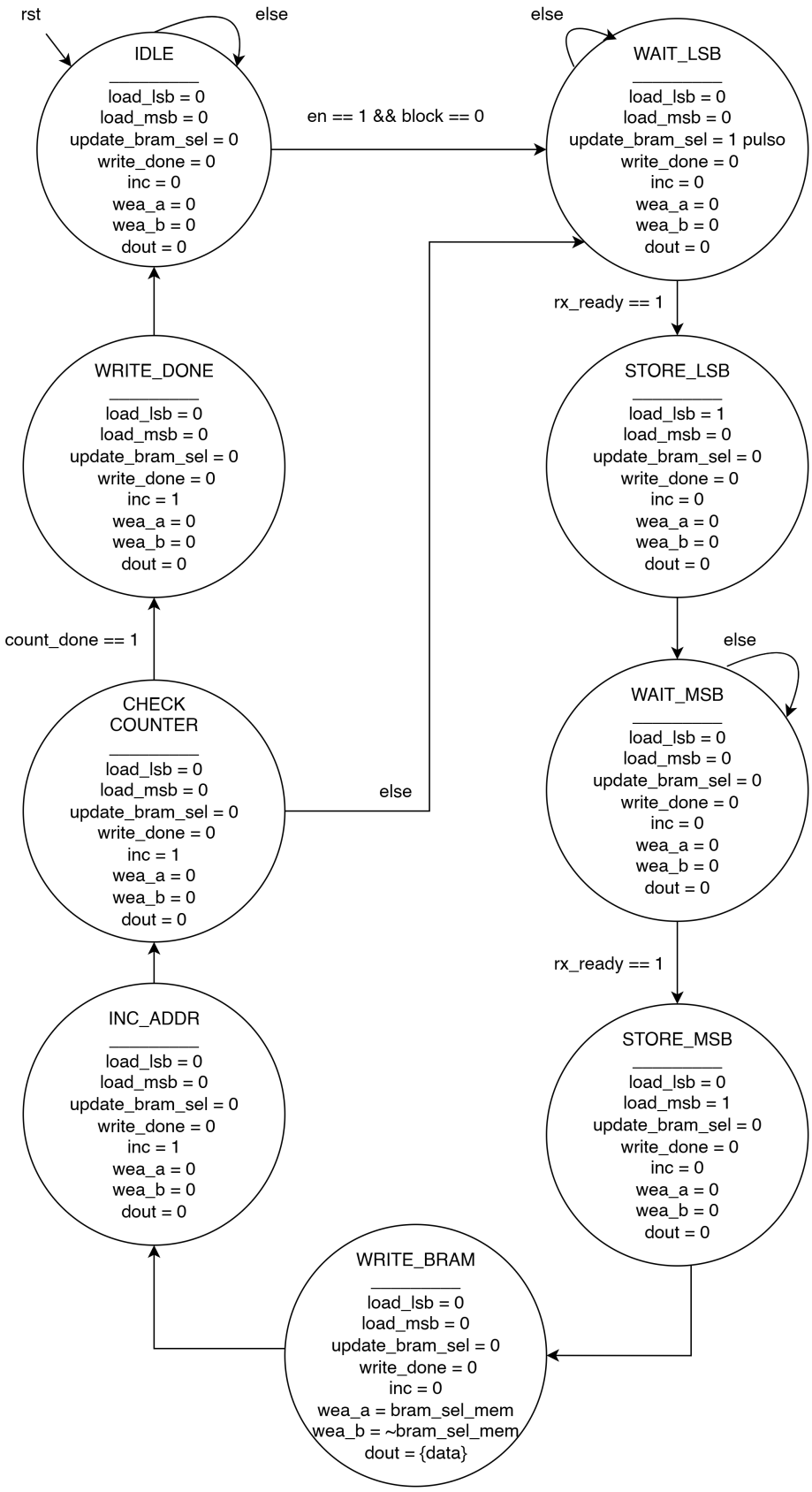

Figura 3. Diagrama de estados de writeCtrl.

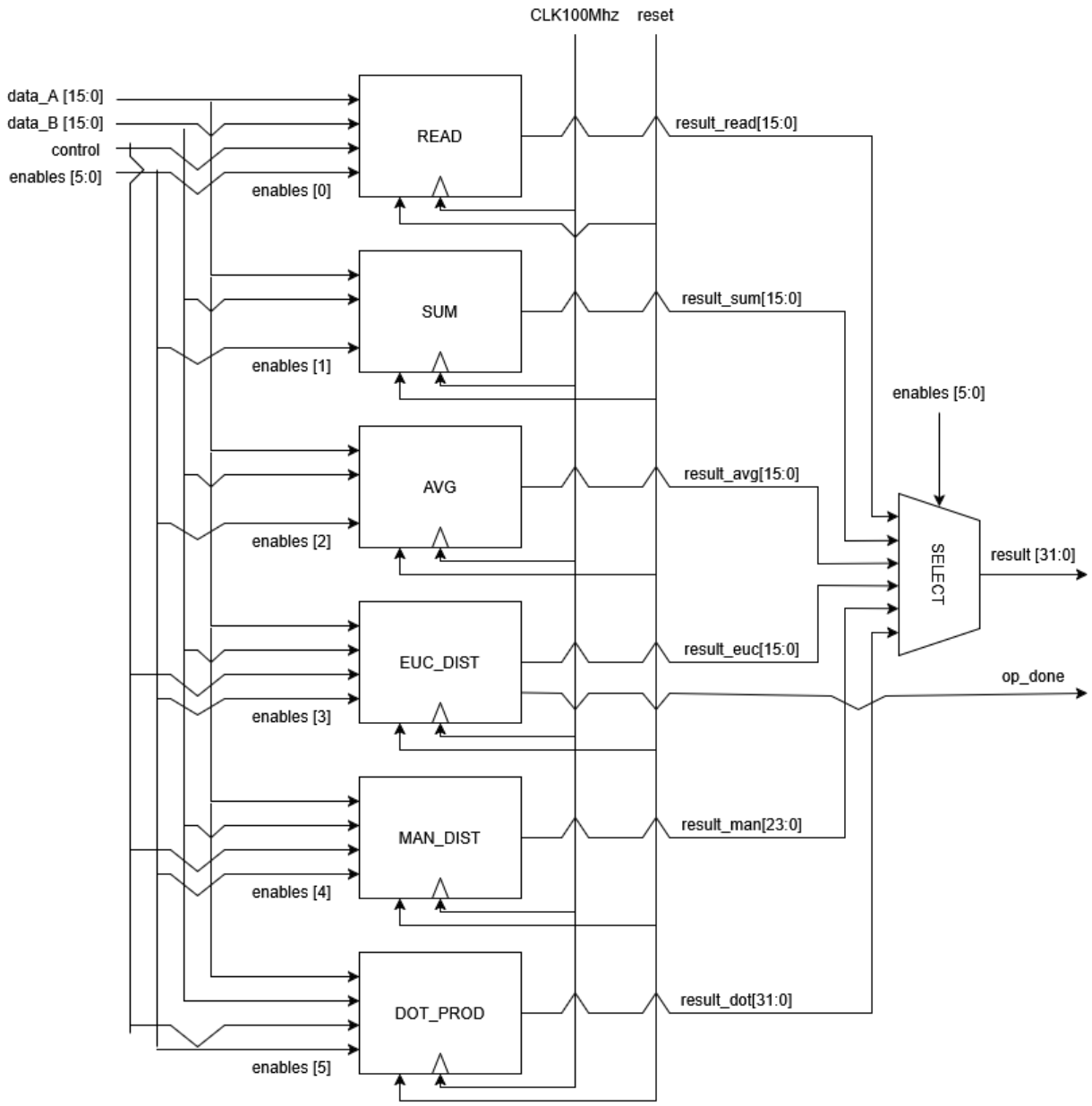

Figura 4. Diagrama de la unidad de procesos.

La transmisión y comportamiento de los estados se puede observar en la Figura [5]. En este diagrama “operation” es un registro que guarda la entrada “command” cuando se recibe la señal “command_recived” en el estado IDLE. “counter” es un contador utilizado para la dirección de memoria que se actualiza cada vez que se procesa un dato. “t” es un contador de tiempo que se reinicia cada vez que se cambia de estado. Se puede, entonces observar del diagrama que la unidad de control corresponde a una máquina de Mealy con medición de tiempo. A continuación se puede observar una descripción de los estados:

- IDLE: estado de espera para la recepción de comandos.
- WRITE: estado de escritura en memoria.
- READ: estado de lectura de memoria seleccionada y transmisión por uart.
- SUM: estado donde se suman vectorialmente las memorias y se transmite el vector suma.
- AVG: estado donde se calcula el promedio de los valores del vector y se transmiten por UART.
- EUC_DIST: cálculo de distancia euclidiana. Se obtiene un resultado y se envía por uart.
- MAN_DIST: cálculo de la distancia de Manhattan. Se envía tras terminar.
- DOT_PROD: cálculo y envío del producto punto.
- SENDING: estado que inicia y coordina el envío de datos.

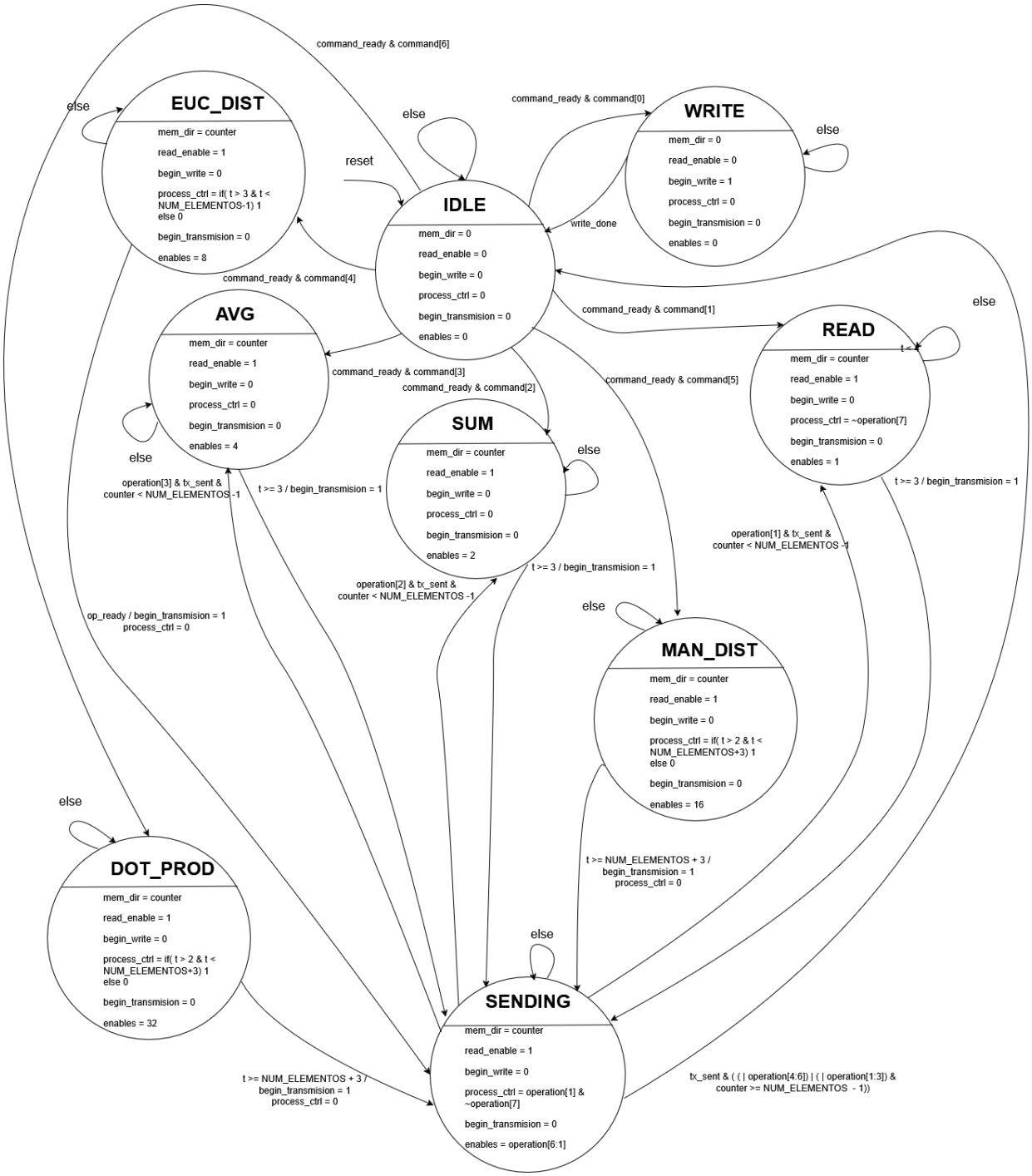

Figura 5. Diagrama de estados de la unidad de control.

III-C. Dominio de salida

La sección de salida del diseño contempla enviar datos por UART hacia el Host y mostrar valores decimales en los displays de 7 segmentos para las operaciones de distancia y producto punto. Tomando en cuenta las cantidades de bits recibidas por cada parte, se hace necesario dividir el resultado en cierta cantidad de bytes para poder ser enviado hacia el Host. Además, los valores se trabajan internamente como valores binarios puros, por lo que se debe hacer una conversión hacia valores decimales para ser desplegados en el display. Por lo tanto, se plantea un diseño que maneje los bytes, controle la secuencia en que se envían, y convierta resultados en formato BCD para ser leídos como números decimales. El detalle de las señales de entrada y salida se presenta en la siguiente lista:

- "clk" : Señal de reloj en el dominio de salida.
- "reset" : Señal de reinicio.
- "begin_transmission" : Orden de inicio de transmisión de datos.
- "tx_busy" : Señal que indica que el transmisor UART está ocupado.
- "enables_in" : Vector de señales de habilitación provenientes de la *Control Unit*, ordenadas como {dot, man, euc, avg, sum, read}.
- "result_data" : Dato de resultado de 32 bits desde el dominio de procesamiento.
- "tx_start" : Señal que activa el inicio de transmisión UART.
- "tx_sent" : Señal que indica que la transmisión ha finalizado correctamente.
- "segments" : Líneas de salida para controlar los segmentos del display de 7 segmentos.
- "tx_data]" : Dato de 8 bits a enviar mediante el transmisor UART.
- "AN" : Señal de control de ánodos para los dígitos del display.

El diseño de bloques del dominio de salida se presenta en la Figura 6, donde se presenta una máquina de estados controladora de la transmisión, una interfaz de salida con la que se hace el manejo de bytes a mostrar, un driver para el display de 7 segmentos que se activa para determinadas operaciones y el bloque correspondiente a la transmisión por UART. En la parte superior se muestran las señales provenientes del dominio de procesamiento y la unidad de control, mientras que en la parte inferior se concentran las señales que salen hacia los displays y la UART conectada al Host.

III-C1. Control de transmisión: La máquina de estados que controla la secuencia de envío de datos se encuentra en el bloque `Tx Ctrl`, y todos sus estados se presentan en la Figura 7. Desde el dominio de procesamiento se obtiene un dato de hasta 32 bits con el resultado de una operación, y dado que el módulo UART puede enviar hata 8 bits por mensajes, se divide el resultado en distintos bytes y se envía. También se agrega un delay para permitir que los datos se registren correctamente. Esto se hace usando un timer que, si supera el tiempo especificado por el parámetro

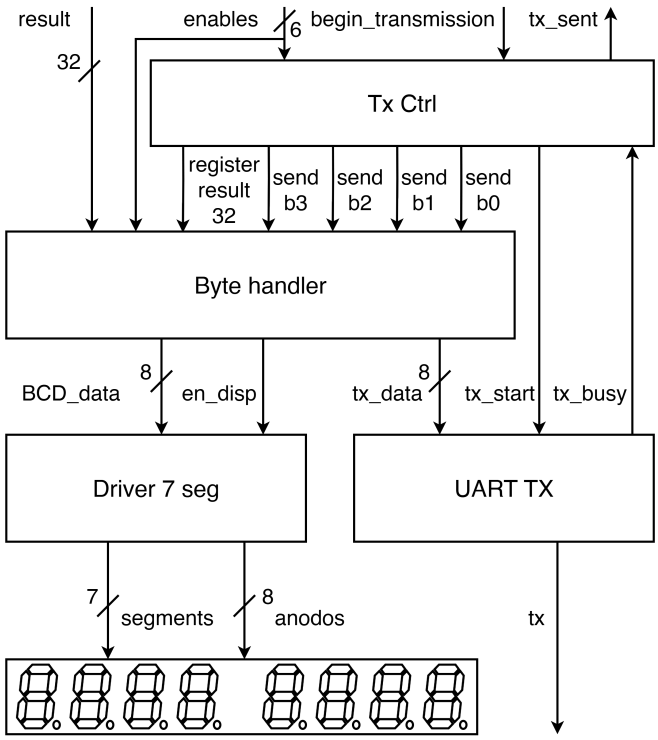

Figura 6. Diagrama de bloques del dominio de salida.

`WAIT_FOR_REGISTER_DELAY`, permite el paso del estado de registro hacia los estados de envío. Siguiendo un diseño similar a la entrada, se hace que la FSM envíe señales de control a un banco de registros dentro del bloque "Byte handler" que separa bytes y hace llegar solo uno a la vez al módulo transmisor junto a las señales de control de dicho módulo.

La cantidad de bytes enviados depende de la operación realizada, siendo esta indicada por la señal de `enables`. Considerando la cantidad máxima de bytes a los que puede llegar cada operación, se envían 2 bytes si se está en las operaciones de distancia euclideana, lectura, suma y promedio. Por otra parte, los cálculos de distancia de Manhattan y producto punto llegan a requerir hasta 4 bytes, por lo que se decide enviar por defecto 4 bytes a menos que se esté ejecutando una de las operaciones que utilizan menos.

III-C2. Despliegue de datos: Para convertir los datos desde un formato binario hacia BCD, se implementa dentro del mismo Byte handler un bloque conversor que aplica el algoritmo Double Dabble. La salida de este algoritmo se entrega por medio de la señal `BCD_data` hacia el driver de 7 segmentos. La habilitación de los displays se controla con la señal `en_disp`, que enciende los displays para los cálculos de distancia y producto punto. Se incluye también un registro que retiene el estado del enable de los displays entre operaciones. Como decisión de diseño, se apagan los displays para las operaciones que no sean las anteriormente mencionadas.

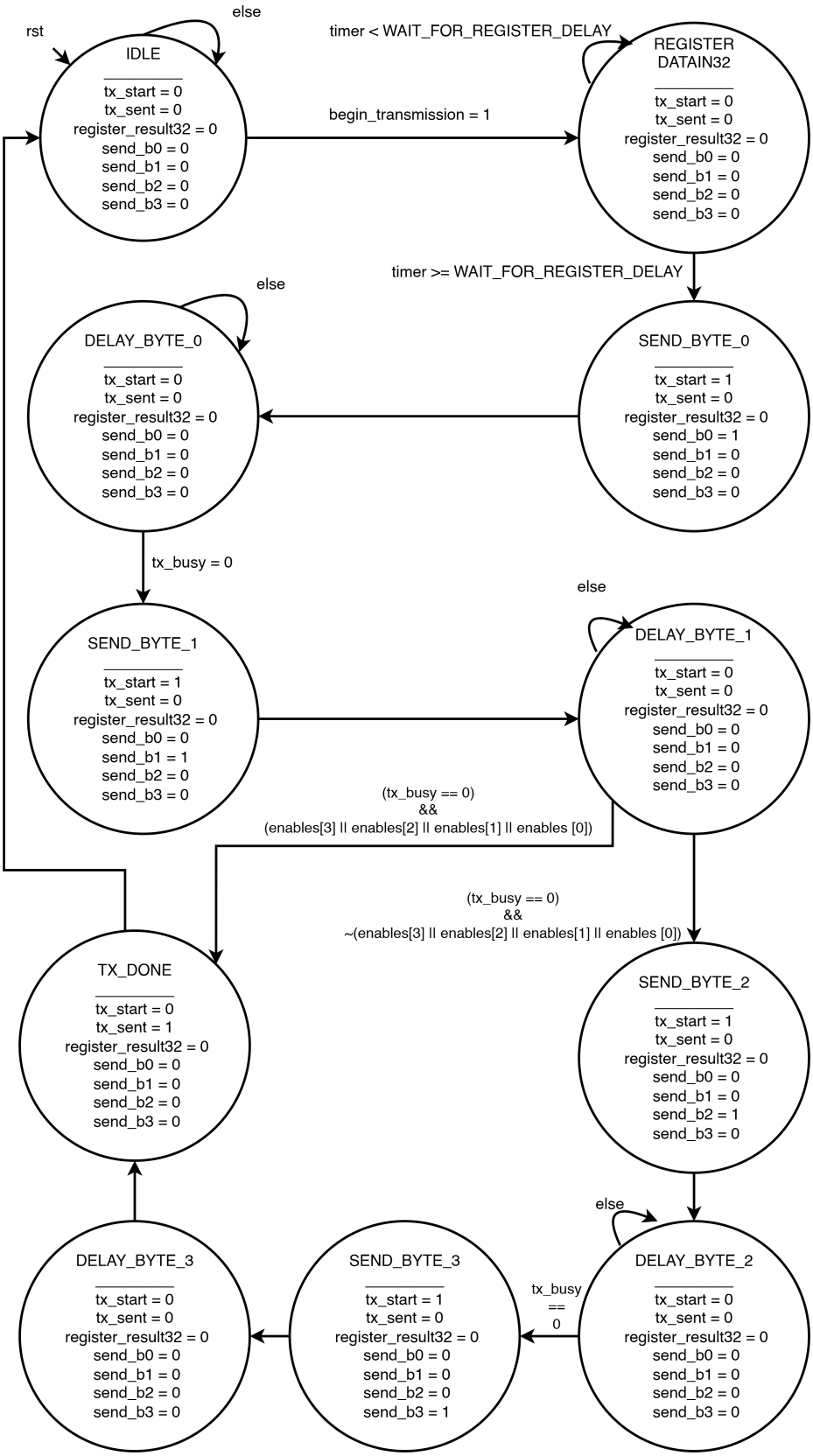

Figura 7. Diagrama de estados del controlador de salidas.

IV. Pruebas del Sistema

IV-A. Funcionalidad básica

En primer lugar se comprueba la funcionalidad básica del dispositivo sin acelerar el procesador utilizando el script "coprocessorTesting.m" incluido en el enunciado de la tarea. Este se ajustó hasta no tener fallas de tiempo y pasar todos las pruebas de tiempo de Vivado. Una vez hecho esto se ejecutó el script y se obtubieron buenos resultados, notando dos excepciones. La primera es que este script tiene la posibilidad de generar valores de escritura de 1024. Estos valores causan un overflow de memoria y hacen que en la BRAM se guarde un 0. Debido a que la función de matlab genera valores entre 0 y 1024, no se puede reparar este error desde el dispositivo, si no desde el propio archivo de pruebas de Matlab. Segundo. Se tomó la decisión de que el resultado de la distancia euclidiana se truncaría a un número entero por el manejo de datos en la FPGA y por no poder coordinar apropiadamente la precisión del resultado entre matlab y el dispositivo. Por esto, al calcular la diferencia entre el resultado obtenido entre matlab y el coprocesador, se debería obtener una diferencia entre -1 y 1.

IV-B. Análisis de frecuencia máxima

Para encontrar la frecuencia máxima de ejecución del dispositivo, se probará incrementalmente la frecuencia del reloj de procesamiento hasta que el dispositivo deje de funcionar correctamente. Se comentarán los hitos más importantes del incremento. Vale decir que un requerimiento para que la frecuencia del reloj de procesamiento sea válida es el no afectar a los relojes de los otros dominios, dado que esto causaría una falla en la comunicación del dispositivo y el computador.

Al aumentar a 110[Mhz] se empezaron a recibir alertas de algunos caminos del dispositivo que no cumplían con los requerimientos de tiempo de setup, sin embargo el dispositivo funcionaba correctamente. Después de esto se siguió aumentando gradualmente hasta llegar a los 366.6[Mhz], siendo este el último valor del reloj en el cual los procesos funcionan correctamente y el valor del reloj del dominio de salida no se ve afectado. A partir de entonces, al probar con una frecuencia de 400[Mhz], el resultado obtenido de de la distancia de Manhattan empieza a fallar y mostrar información errónea, manteniendose el funcionamiento del resto de las operaciones. Luego, se aumentó la frecuencia a los 450[Mhz], valor en el cual el propio "clock wizard" entregó una advertencia de que este valor podría producir errores. Al utilizar este reloj, todos los valores obtenidos mostraron información errónea. Sin embargo, no hubo problemas de transmisión, lo cual sugiere que el fallo se produjo en la unidad de procesos y no en la unidad de control.

IV-C. Cálculo de latencia para las operaciones

Para calcular cual es la latencia de las operaciones, se realizará un análisis de la síntesis del esquemático para el processing core a 366[Mhz]. Vale decir que al realizar el análisis en la síntesis de Vivado, es posible que el resultado no sea igual a la descripción del dispositivo programada en system verilog. Por esto, se considerarán solo las primitivas encontradas en el processingCore para el análisis y los resultados podrían no ser los esperados según la descripción o la operación completa.

En primer lugar, se define el periodo como 1/366.6[Mhz], o sea 3[ns]. Además cada flip flop tiene un setup time de 0.11[ns] y tiempo de propagación de 0.66[ns].

Se observa que las operaciones de suma, lectura y promedio han sido reducidas a un registro para cada bit de salida. Por lo que la latencia de cada dato es de 3.77[ns].

En el caso de la distancia euclidiana, la resta y multiplicación de los valores de memoria ahora se hace de forma externa, por lo que solo se realiza la operación de suma y raiz internamente. Al sumar los datos, se demora 1024 ciclos de reloj. Además la documentación de IP CORDIC dice que el tiempo de procesamiento es aproximadamente el ancho del bus de salida de 16 bits. Por esto la latencia de esta operación es de 3.121[us].

En la distancia de Manhattan hay un solo set de flip flops que realimentan carrys y LUT para ir sumando y guardando información previamente procesada. Se deben realizar 1024 sumas para completar el procesamiento, por lo que la latencia de esta operación es de 3.07277[us].

Finalmente, el producto punto es completado por una única primitiva que multiplica y acumula ambos valores. Esta tarea debería completarse en 1024 ciclos, de nuevo dando una latencia de 3.07277[us].

IV-D. Método de raíz alternativo

En esta sección se busca una alternativa para obtener la raíz cuadrada a la IP CORDIC utilizada. Se seleccionó un algoritmo dígito a dígito desarrollado por projectf.io [1]. Este algoritmo produce una raiz cuadrada y un resto, en vez de truncarla como CORDIC. Al probarla en el dispositivo se tiene que reducir la frecuencia de reloj a 300[Mhz] por problemas de funcionamiento. Esto se toma en cuenta para el análisis temporal. El uso de recursos y diferencia de tiempos se puede observar en la Tabla [III]. Se puede apreciar que la IP CORDIC consume más recursos y tiene mejor latencia que el método dígito a dígito. Sin embargo la mayor diferencia se encuentra en el que hay que desacelerar el reloj de todo el sistema de procesamiento para que este funcione, lo cual es una grave desventaja. Vale también decir que ambos sistemas ofrecen métodos diferentes para lidiar con el resto de la raíz cuadrada. CORDIC la puede truncar, dejar el resultado como fracción o utilizar precisión limitada. Por otra parte, el algoritmo de raíz cuadrada retorna a demás un resto cuando finaliza la operación. Por esto ambos algoritmos requieren de implementación diferentes. Vale decir también que el algoritmo CORDIC es más difícil de usar dado su amplio espectro de usos no solo limitados a raíces cuadradas, junto con su versatilidad y abanico de opciones. La única opción del algoritmo bit a bit es el ancho de la entrada.

[1] https://projectf.io/posts/square-root-in-verilog/

	CORDIC	root
Slice LUT	354	87
Slice Registers	236	130
Slice	123	43
Frecuencia [Mhz]	366.7	300
Periodo [us]	3	3.3
Ciclos para completar	16	15
Latencia [ns]	47.77	50.77[ns]

Tabla III
COMPARACIÓN DE USO DE RECURSOS ENTRE AMBAS TÉCNICAS DE RAÍZ CUADRADA.

V. BITÁCORA

En esta sección se presenta la bitácora con los avances a lo largo de este trabajo. La bitácora se presenta en la Tabla

Tabla IV
BITÁCORA CON AVANCES POR FECHA Y HORA, BASTIÁN RIVAS

Día	**Hora Inicio**	**Hora Término**	**Actividades**
2 Oct	11:00	13:00	Lectura inicial del encabezado y revisión de requerimientos
4 Oct	17:00	18:00	Diseño inicial de núcleo de procesamiento
8 Oct	21:00	23:00	Revisión de literatura sobre BRAM y diseño de Ctrl unit. Analizar estructura genérica de bloques de operaciones
9 Oct	12:30	13-30	Diseño inicial procesador completo, reconocimiento de señales internas
	22:00	23:59	Estudio de cómo implementar los bloques de operaciones División de tareas: uno analiza la sección de operaciones y control mientras el otro ve entradas y salidas
11 Oct	18:00	20:00	Testeo de bloque BRAM. Instanciado y funcionalidad básica en 1 puerto
	22:00	23:59	Testeo en ambos puertos y diseño inicial de FSM de entrada
12 Oct	14:00	16:00	Análisis de alternativas: controladores por separado para escritura y operaciones o un monolito. Añadidas señales para el controlador
	23:00	02:00	Definición señales FSM entrada y conexiones. Elegida alternativa de controladores por separado
14 Oct	20:00	22:00	Reunión para ver estado de las distintas partes del diseño. Parte de operaciones ya implementada y discutidos unos temas con el diseño de la parte de recepción
16 Oct	13:00	15:00	Agregados detalles de FSMs y separación entre FSM y lógica combinacional
18 Oct	21:00	23:00	Testeo físico de BRAM
19 Oct	00:00	03:00	Implementación en código de decodificador y controlador de escritura de BRAMs
	14:00	16:00	Corrección de varios warnings arrojados por el Linter en el módulo de entrada. Había lógica y asignaciones que se tenían que arreglar
	17:00	20:00	Simulación por testbench y corregidas varias cosas relacionadas a la lógica de escritura en las BRAMs. Arreglados varios errores con la lógica que acompaña a las FSM que controlan la escritura
	21:00	22:00	Diseño de interfaz de salida
20 Oct	01:00	03:00	Prueba física de UART y escritura en BRAM. Hay algunos detalles con los bits que se van registrando, pero ya se confirma que es posible comunicarse por UART y grabar en la memoria. El problema proviene de unos pines de rx_data que no se conectaron a los bits más significativos, pero tras separar en dos registros en vez de usar uno grande se logra que funcione bien.
	12:00	14:00	Puesta al día con los avances. Se modifican las FSM de la parte de recepción y transmisión para que se comporten mejor con el controlador de la parte de operación
	19:00	00:00	Implementación de parte de transmisión y testeo de txCtrl. Se logra verificar la funcionalidad básica de la FSM Remodelación de lógica de control para hacer calzar ambas partes trabajadas
21 Oct	12:30	13:30	Solicitud de tiempo extra y arreglos en la lógica de entrada. Ahora CtrlUnit se encarga de activar el controlador de escritura y espera hasta que se le indique que terminó
22 Oct	17:00	22:00	Integración de todos los módulos y correcciones
23 Oct	15:00	16:00	Documentación y traspaso de diagramas a herramienta
	18:00	22:00	Validada parte de transmisión con simulación y test en la placa Unión de todas las partes y validación con script de Matlab.
24 Oct	00:00	04:00	Luego de algunos arreglos menores se logra recibir datos desde la placa y que retorne los resultados
	14:00	16:00	Diagramas y explicaciones de módulos de entrada
	18:00	19:00	Finalizada documentación de entrada
	20:00	21:00	Arreglo de algunos detalles de diagramas de estados. Finalizada documentación de salida

Tabla V
BITÁCORA SVEN KLEIN PLARRE

Fecha	Hora	Actividad
4-10	14:00-16:00	Lectura de actividad
6-10	14:00-18:00	Lectura referencias
9-10	12:30-17:30	Planeación de estructura
10-10	14:00-18:00 21:00-2:30	Módulo Procesos
12-10	14:00-17:30	Processing Core
13-10	20:00-3:45	Processing Core
16-10	16:00-17:00	Debug
18-10	19:00-22:30	Debug processing core
20-10	15:00-19:00 21:00-2:00	Modificaciones a core y procesing por cambios requisitos
21-10	15:30-19:00 21:00-2:00	Cambios interacción cu y memoria.
22-10	14:30-16:00 17:00-22:00 23:30-2:00	Integración y trabajo con input interface.
23-10	14:30-22:00 23:00-4:00	Primer prototipo funcional
24-10	9:00-12:00 13:00-23:59	Informe, pruebas relacionadas y actividades.