**ADAV**

**Memoria Práctica\_A2**

1.- Introducción

En esta práctica se implementará un sistema de filtrado utilizando dos operadores, un multiplicador y un sumador/restador implementado mediante una estructura RippleCarry. Este sistema contará con dos interfaces para la comunicación externa, **Interfaz de entrada** e **Interfaz de salida**, con una unidad de **Control** y un **Datapath** que hará uso de los operadores. Estos están implementados en dos unidades independientes **RippleCarry** y **Multiplicador**, los cuales importaremos al datapath para su uso. Todos estos módulos serán controlados por la unidad **Top**, el cual se encargará de unir todos estos módulos y obtener el sistema final.

## 2.- Interfaces de entrada y salida

Para adaptar el sistema a un control externo hemos incluido la señal ack\_in. Esta señal transmite un pulso al control exterior cuando el sistema esta validado y los datos cargados en la entrada.

Haciendo algo similar en el interfaz de salida hemos creado la señal ack\_out. Esta señal recibe el estado del sistema externo, ‘0’ significa libre y ‘1’ significa ocupado. Mediante esta señal, al confirmar que el sistema externo esta libre, procedemos cargar los valores de las salidas en el bus data\_out y ponemos la señal valid\_out a ‘1’.

Forma

Descripción generada automáticamente con confianza media

## 3.- Planificación tabla de reserva

El sistema base contaba con 18 operaciones para implementar el sistema filtrado, teniendo en cuenta que cada operación ocupaba 1 ciclo de reloj, el número total de ciclos que tardaba en realizar un filtrado eran 18 ciclos de reloj.

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Con las operaciones simplificadas ordenadas procedemos a realizar la tabla de reserva.

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Viendo la tabla de reserva observamos que podemos realizar la paralización de varias operaciones, pudiendo usar un multiplicador y un sumador en el mismo ciclo. A continuación, realizaremos la paralelización de dichas operaciones.

Tabla

Descripción generada automáticamente con confianza baja

Con esta paralelización obtenemos la siguiente tabla de reserva (reduciendo de 18 a 11 ciclos necesarios):

Imagen que contiene luz, foto

Descripción generada automáticamente

Con esta información, podemos saber que registros podemos agrupar, quedándonos la tabla de reserva con los registros agrupados, de la siguiente manera:

Una captura de pantalla de un videojuego

Descripción generada automáticamente con confianza media

De las posibles agrupaciones que hemos tenido en cuenta, esta es la óptima ya que solamente hace un cambio de registro, siendo este el de temp18 a sv1:

|  |  |  |  |
| --- | --- | --- | --- |
| Carga de datos finales en operaciones iniciales | tmp15 -> sv4 | **r6** | **r6** |
| tmp16 -> sv3 | **r5** | **r5** |
| tmp17 -> sv2 | **r4** | **r4** |
| tmp18 -> sv1 | r1 | r3 |

De la siguiente manera será como quedaran los registros y las correspondientes operaciones asociadas a estos:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ciclo | Operación con registros antiguos | | Operación con registros nuevos | |
| 0 (idle) | tmp0 = entradas | | r1 = entradas | |
| 1 | tmp1 = m\_tmp1 (39 downto 16) = tmp0 \* b1 | | r2 = r1 \* b1 | |
| 2 | tmp2 = m\_tmp2 (39 downto 16) = tmp0 \* b2 | tmp9 = tmp1 + sv1 | r2 = r1 \* b2 | r3 = r2 + r3 |
| 3 | tmp3 = m\_tmp3 (39 downto 16) = tmp0 \* b3 | tmp8 = tmp2 + sv2 | r2 = r1 \* b3 | r4 = r2 + r4 |
| 4 | tmp4 = m\_tmp4 (39 downto 16) = tmp0 \* b4 | tmp7 = tmp3 + sv3 | r2 = r1 \* b4 | r5 = r2 + r5 |
| 5 | tmp5 = m\_tmp5 (39 downto 16) = tmp0 \* b5 | tmp6 = tmp4 + sv4 | r2 = r1 \* b5 | r1 = r2 + r6 |
| 6 | tmp10 = m\_tmp10 (39 downto 16) = tmp9 \* inv\_a1 | | salidas = r3 = r3 \* inv\_a1 | |
| 7 | tmp11 = m\_tmp11 (39 downto 16) = tmp10 \* neg\_a2 | | r6 = r3 \* neg\_a2 | |
| 8 | tmp12 = m\_tmp12 (39 downto 16) = tmp11 \* neg\_a3 | tmp15 = tmp8 + tmp11 | r4 = r6 \* neg\_a3 | r6 = r4 + r6 |
| 9 | tmp13 = m\_tmp13 (39 downto 16) = tmp12 \* neg\_a4 | tmp16 = tmp7 + tmp12 | r4 = r4 \* neg\_a4 | r5 = r5 + r4 |
| 10 | tmp14 = m\_tmp14 (39 downto 16) = tmp13 \* inv\_a5 | tmp17 = tmp6 + tmp13 | r1 = r4 \* inv\_a5 | r4 = r1 +r4 |
| 11 | tmp18 = tmp5 + tmp14 | | r1 = r2 + r1 | |

## 4.- Implementación RippleCarry y Multiplicador

Para la implementación del RippleCarry hemos utilizado 24 FullAdder, poniéndolos en serie y obteniendo la estructura Ripple Carry. Mediante la señal de control r\_mode seremos capaces de seleccionar el modo de funcionamiento del RippleCarry, ‘0’ suma y ‘1’ resta, en este caso al interesarnos el operando suma lo dejaremos a ‘0’.

Para la implementación del módulo hemos realizado la operación de multiplicación de dos vectores de 24 bits que se proporcionan por las entradas de este. Esta operación produce un vector de 48 bits el cual hemos decidido recortar en este propio módulo, para no tener que gestionar vectores de 48 bits en el datapath, a un vector de 24 bits el cual es el que se devuelve. Hemos usado la misma reducción de bits que la proporcionada.

## 5.- Diseño del datapath y planificación de las señales en módulo de control

Tras hacer la planificación del sistema, junto con la pertinente reducción de registros, procedemos a realizar el diseño del datapath:

Diagrama

Descripción generada automáticamente

Como se puede observar en la imagen anterior de 4 multiplexores para proporcionar los datos a los operadores.

Tabla

Descripción generada automáticamente Gráfico

Descripción generada automáticamente

Tabla

Descripción generada automáticamenteTabla

Descripción generada automáticamente con confianza media

Disponemos de 1 multiplexor por cada registro, todos son controlados por la señal comando. Esta señal es la generada por el *process* del módulo control proporcionado, incrementando los estados desde clk0 hasta clk11.

Pantalla de computadora

Descripción generada automáticamente con confianza media

Como se puede ver en la tabla anteriro hemos decidido cargar los valores de entrada en el registro r1 en ciclo 0 (idle). Esto se debe a que necesitamos que esten los datos en el registro r1 antes del ciclo 1 que es cuando comenzamos a realizar operaciones.

El último multiplexador es el que se encarga de cargar los valores del r3 (tmp10) en la salida, mediante la señal de control c\_s. Esta señal se pone a ‘1’ desde el ciclo 6 hasta el ciclo 11, cargando el valor de r3 a la salida.

Todas las señales de control se implementan en el módulo de **Control** y mediante el módulo **Top** se interconecta con el datapath.

Tabla

Descripción generada automáticamente

Teniendo en cuenta estas tablas hemos implementado el **Datapath** mediante un *process* combinacional en el cual implementamos todas las estructuras de *case* necesarias para implementar los multiplexores. Para implementar los registros hemos utilizado un *process* secuencial, cargando así los valores al inicio del ciclo de reloj. El módulo de control lo hemos implementado mediante un *process* secuencial el cual se encarga de generar el estado del sistema y mediante un *process* combinacional, el cual depende de clk y de la señal estado, hemos determinado el valor de todas las señales de control dependiendo del estado en el que se encuentre el sistema.