# Diseño de Alta Velocidad. Temporización y Segmentación (*Pipelining*)

## Objetivo

El objetivo de esta práctica es el familiarizarse con la problemática del diseño de alta velocidad y utilizar diferentes estrategias de mejora de los resultados de "*timing*". Como ejemplo, se usará un circuito que computa un *checksum* (suma de verificación) como el utilizado en las cabeceras de los paquetes IP (*Internet Protocol*), UDP o TCP. Durante el desarrollo de la práctica se verá cómo interpretar los reportes de "*timing*" proporcionados por las herramientas; también se prestará atención a los reportes relacionados con el área ocupada por el diseño.

## Descripción del Diseño

El *checksum* calculado, de 16 bits, es el complemento a uno, de la suma en complemento a uno, de todas las palabras de 16 bits de las cabeceras y carga útil de las tramas del protocolo. En este circuito simplificaremos el problema al calcular el *checksum* de 8 palabras de entrada de 16 bits.

*Complemento a uno y la operación de Suma.*

Un sistema de **complemento a uno** es un sistema donde los números negativos están representados por el inverso de las representaciones binarias de sus correspondientes números positivos. En tal sistema, un número es negado (convertido de positivo a negativo o viceversa) calculando el complemento de cada bit. Un número de N bits en complemento a uno sólo puede representar enteros en el rango $-(2^{N-1}-1)$ a $2^{N-1}-1$ (el *complemento a dos* puede expresar $-2^{N-1}$ a $2^{N-1}-1$). En complemento a uno existe doble representación del cero (000…000 y 111…111).

La regla empírica para la suma (o resta) es que se debe recircular el bit de acarreo. Por ejemplo, en un sistema de representación de 5 bits (números decimales entre -15 y 15), la suma 10dec + -5dec (10 dec = 01010 bin; -5 dec = 11010 bin) será:

```
  0 1 0 1 0      10 dec
  1 1 0 1 0      -5 dec
-------------
(1) 0 0 1 0 0
+         1
-------------
  0 0 1 0 1       5 dec
```

NOTA: Una posibilidad para sumar en complemento a uno dos números es hacer dos sumas en complemento a dos, una con una entrada de acarreo (*carry in*) a 0 y la otra con esta entrada a 1. Dependiendo de si hay acarreo de salida (*carry out*), se seleccionaría el resultado de una u otra suma en complemento a dos como el resultado de la suma en complemento a uno. Esto puede verse implementado en una función VHDL en el código entregado.

*Suma en complemento a uno usando sumadores en complemento a dos.*

Una interesante propiedad es que la suma de múltiples números complemento a uno puede hacerse con una suma en complemento a dos (suma de enteros), con una posterior reducción complemento a uno (recirculando los bits de acarreo).

Ejemplos de suma de 4 valores en 5 bits. Podemos sumar los valores en complemento a 2 y realizar la reducción complemento a uno.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 1 0 1 1 | 11 | | 1 1 1 1 1 | -0 | | 1 1 0 1 1 | -4 |
| 1 0 0 0 1 | -14 | | 1 1 1 1 1 | -0 | | 1 1 1 0 1 | -2 |
| 0 1 1 0 1 | 13 | | 1 1 1 1 0 | -1 | | 0 1 0 1 0 | 10 |
| 1 1 0 1 1 | -4 | | 1 1 1 1 1 | -0 | | 1 1 1 0 1 | -2 |
| (1 0) 0 0 1 0 0 | | | (1 1) 1 1 0 1 1 | | | (10) 1 1 1 1 1 | |
| + 1 0 | | | + 1 1 | | | + 1 0 | |
| ( 0) 0 0 1 1 0 | 6 | | ( 0) 1 1 1 1 0 | -1 | | ( 1) 0 0 0 0 1 | |
| + 0 | | | + 0 | | | + 1 | |
| 0 0 1 1 0 | 6 | | 1 1 1 1 0 | -1 | | * 0 0 0 1 0 | 2 |

* En este último ejemplo fue necesario el segundo ajuste (recirculación final del bit)

El problema del cálculo de nuestro *checksum* se simplifica por tanto en realizar 8 sumas de 16 bits y la reducción complemento a 1. Una construcción rápida será realizar un árbol de suma de los 8 elementos como el sugerido en la Figura 1 de abajo, más una reducción complemento a uno final.
El circuito adicionalmente dispone de una señal SoP (*Start of Packet*) que señala cuándo capturar datos de entrada. La señal *ChksumValid* señala cuándo es válido el dato a la salida.

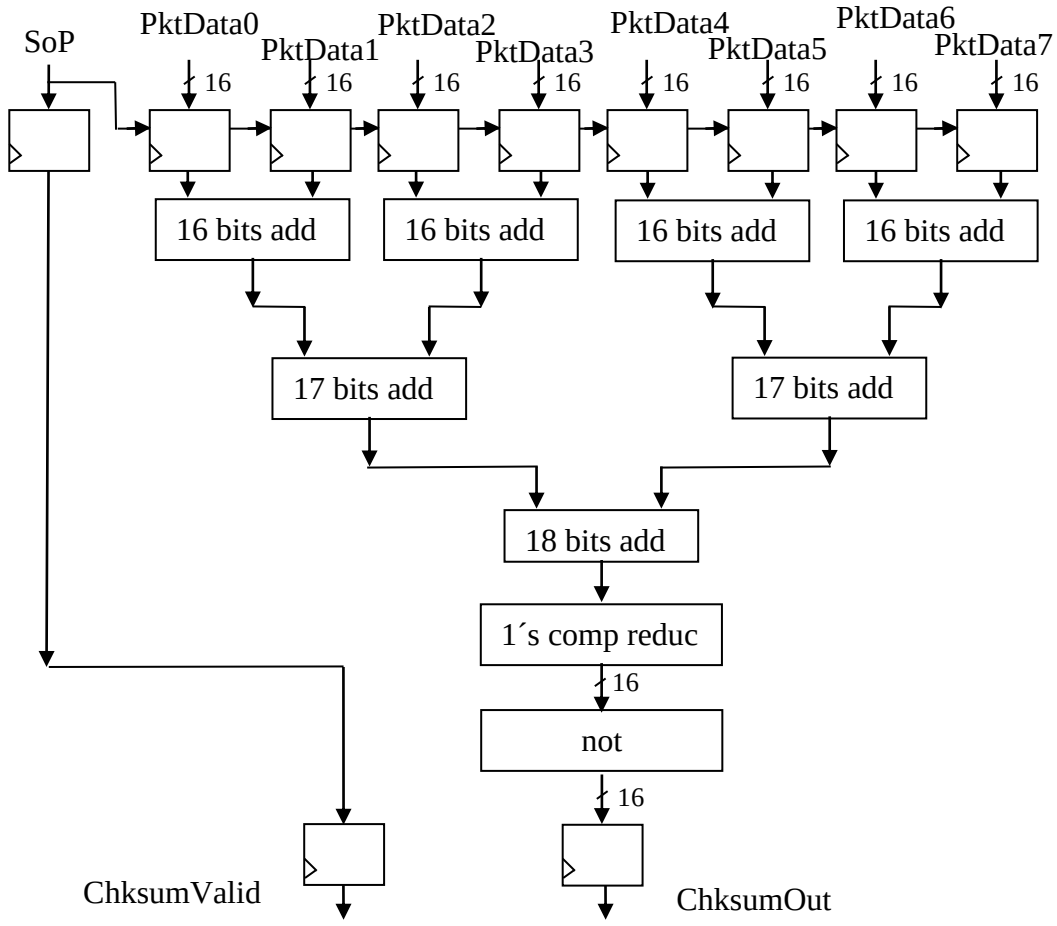



Figura 1. Circuito que calcula el checksum, con registros en entrada y salida.

## *Realización de la práctica*

La práctica consta de 4 ejercicios en los que se analiza el circuito a diferentes frecuencias de operación y finalmente se expone la estrategia de *pipelining* para aumentar su máxima frecuencia.

***Preparación del diseño***
Con objeto de favorecer la limpieza y comprensión del diseño se entregan directorios con la estructura siguiente, que habrá de mantenerse:

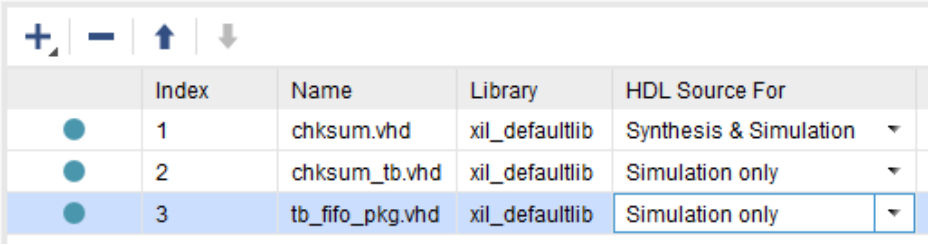
```
P3_checksum/
```

- `rtl/` - Código fuente VHDL para el diseño
- `sim/` - Directorio de simulación, que contiene los ficheros del banco de pruebas (*testbench*).
- `vivado/` - Directorio que contendrá el proyecto Vivado; se entrega con un fichero *.xdc* con las restricciones (*constraints*) de posicionamiento de pines y una restricción inicial para el reloj.

## *Ejercicio 1: Puesta en Marcha*

En este primer ejercicio se busca comprender el diseño, crear un proyecto Vivado para el mismo y realizar un primer análisis, tanto de área ocupada como de temporización:

1. Crear un nuevo proyecto Vivado "chksum", con su propio subdirectorio en el directorio `vivado/`. Añadir al proyecto el fichero ***chksum.vhd*** (con la casilla *Copy sources into project* **desactivada**) y, como ficheros de simulación (seleccionar "**Simulation only**", ver imagen debajo), los archivos de testbench ***chksum_tb.vhd*** y package ***tb_fifo_pkg.vhd***.




   Agregar también el **fichero de constraints** (*.xdc*) proporcionado. Utilizar un dispositivo **Zynq 7z020 con package clg484 y speed grade -1**. Se trata de un dispositivo relativamente pequeño pero con suficientes pines para las entradas-salidas del diseño

2. Lo primero es entender el funcionamiento del circuito de cómputo de *checksum.*
3. Realizar una simulación funcional del diseño.¿Funciona correctamente? Interpretar los resultados. Obsérvese que la simulación tiene una primera zona con pulsos de la entrada *SoP* espaciados, seguida de otra zona con *SoP* a '1' de forma continua. Comprobar la relación temporal entre *SoP* y *ChksumValid* (comparar para esto las ondas con el código de *chksum.vhd*).

   ⇨ En la tabla de resultados al principio de la Hoja de Respuestas, rellenar la primera celda (*Latencia* para el *Ej.1*). Se puede obtener la respuesta razonando sobre el código o viendo las ondas de la simulación.

4. Observar el contenido del fichero de restricciones ¿Cuál es la frecuencia objetivo? ¿Qué se restringe adicionalmente en este fichero?

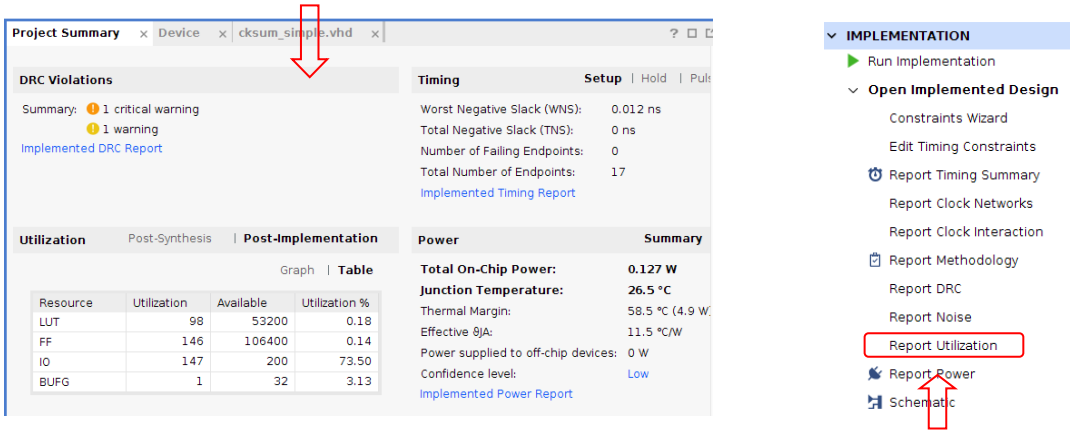
   ⇨ Anotar frecuencia y periodo objetivo en la hoja de resultados de la práctica.

5. Implementar el diseño y observar los resultados post-implementación.

   5.1. Área.

   Podemos obtener datos de área en varios sitios, por ejemplo:

   - En el panel de resumen del proyecto (*Project Summary*). Si se hubiese cerrado esta ventana, puede volver a abrirse en *Window-> Project Summary* (icono sumatorio Σ de la barra). Aquí aparece una visión resumida de la información.




   - Una información más completa puede obtenerse abriendo el diseño implementado (*Open Implemented Design*) y allí accediendo al informe de uso (*Report Utilization*)

   ⇨ En la tabla de la Hoja de Respuestas, completar las filas en rojo para la columna “Ej.1.”. Han de ser valores tras implementación, no tras síntesis.

   5.2. Timing.

   Podemos obtener un informe de tiempos ejecutando *Report Timing Summary* (también en el panel izquierdo). De nuevo, ha de tenerse cuidado de obtener el informe tras implementación, no tras síntesis. En el resumen podemos ver el peor caso para el slack (WNS, *Worst Negative Slack*). Podemos ver los 10 peores caminos en cuanto a margen para cumplir el *setup* en el apartado *Intra-Clock Paths/clk/Setup*. El camino crítico (**el de menor *slack***, ojo, **no** el de menor retardo) es el primero de la lista. En la tabla podemos ver varios valores al respecto, pero haciendo doble clic sobre la línea correspondiente podremos acceder a una ventana con más detalles:

   - Resumen de datos del camino: origen, destino, *slack*, reloj utilizado, etc. (haciendo clic sobre los valores en azul se muestra la fórmula utilizada para su cálculo).
   - *Source Clock Path*: camino que sigue el reloj desde la entrada a la FPGA hasta el pin de reloj del flip-flop origen.
   - *Data Path*: camino de datos entre el flip-flop origen y el flip-flop destino.
   - *Destination Clock Path*: camino que sigue el reloj desde la entrada a la FPGA hasta el pin de reloj del flip-flop destino.

   Estudiar este camino crítico. En él veremos diferentes recursos de la FPGA, tanto recursos combinacionales como flip-flops, separados por recursos de interconexión (*net*). Entre los elementos combinacionales encontraremos recursos de las *slices* (p.ej. *LUT3*, *CARRY4*), buffers de entrada/salida (p.ej. *IBUF*, *OBUF*) y buffers de reloj (*BUFG*). Los flip-flops reciben distintos nombres según la configuración utilizada (p.ej. *FDRE* para un FF tipo D configurado para usar patas de Reset síncrono y Enable).

Dos columnas nos muestran el retardo de cada elemento (*Incr*) y el retardo acumulado en el camino (*Path*). Realmente en la primera columna, además de retardos, aparecen la contribución de otros valores temporales asociados al elemento en cuestión (p.ej. el *setup* de un FF), el instante de tiempo relativo en que ocurren los flancos de reloj implicados, y valores de ajuste del cálculo (p.ej. *clock pessimism*, *clock uncertainty*).

La forma de comprobar si el *path* cumple con las restricciones de tiempo es ver si el dato llega al FF de destino antes que el reloj. En relación con esto puede observarse lo siguiente:

- El *Data Path* sigue acumulando retardos a partir del punto en que termina el *Source Clock Path*, encontrándose como primer elemento del *Data Path* el tiempo de respuesta del FF (entre su reloj y su salida).
- El *Destination Clock Path* tiene como punto de partida el siguiente flanco de reloj respecto al *Source Clock Path*.
- El *setup* del FF destino contribuye como un valor que restamos del camino del reloj para el FF destino (como tenemos que tener el dato listo X tiempo antes del flanco de reloj, a la hora de realizar la comparación mencionada arriba es como si el flanco llegase X tiempo antes). Aquí se da una circunstancia que despista bastante: el setup del FF en los dispositivos de la *Serie 7* de Xilinx es negativo, con lo que parece que se suma en vez de restarse (y además en la columna *Incr* aparece como positivo, porque muestra el incremento en el *path*, más que el valor correspondiente al elemento).
- El pesimismo en el cálculo de los caminos del reloj (*clock pessimism*) suma al camino del reloj hacia el FF destino, es decir, se compensa el pesimismo poniendo más fácil cumplir la comparación mencionada arriba.
- La incertidumbre en los tiempos a considerar para los flancos de reloj (*clock uncertainty*) resta al camino del reloj hacia el FF destino, es decir, pone más difícil cumplir el objetivo.

Tras entender el reporte:

⇨ En la tabla de la Hoja de Respuestas, completar las filas en azul para la columna “Ej.1.”.

⇨ Rellenar los valores de la figura de camino crítico de flip-flop a flip-flop que aparece en la Hoja de Respuestas.

⇨ Sobre la ventana de **reporte de este peor path**, hacer clic con el botón derecho y ejecutar “***Export Entire Path to Spreadsheet…***”; usar el nombre de fichero **TableEj1.xls** (en el directorio *vivado/*).

**RECOMENDACIÓN:**

Crear un archivo comprimido en este momento y al terminar cada uno de los siguientes ejercicios (ejercicio 2, 3a, 3b y 4). De esta forma, si se desea, se podrá volver a abrir el proyecto tal como estaba en estos puntos.

## *Ejercicio 2: Tiempos en entradas y salidas*

Para la realización de este ejercicio se tendrán en cuenta las siguientes características del sistema del que la FPGA formará parte:

- Los retardos de la placa de circuito impreso (PCB) se considerarán despreciables (los chips están muy cercanos entre sí).
- La FPGA está situada entre dos chips, el primero entrega los datos sobre los que realizar el checksum y el segundo utiliza el valor de checksum que la FPGA calcula.
- Los tres chips funcionan síncronamente con el flanco positivo de un mismo reloj, el que la FPGA recibe en su puerto *Clk*. Este reloj tiene una frecuencia de 50 MHz.
- En el datasheet del primer chip puede verse que éste entrega sus salidas 3 ns tras el flanco de reloj. Esto es, la FPGA recibe sus datos de entrada (PktData* y SoP) 3 ns tras el flanco de reloj.
- En el datasheet del segundo chip puede verse que éste requiere respetar un tiempo de setup de 0.4 ns para los datos que recibe, respecto al flanco positivo del reloj. Esto es, la FPGA debe entregar sus salidas (*ChksumOut* y *ChksumValid*) con tiempo suficiente para que el chip que las recoge las vea estables desde 0.4 ns antes del flanco de reloj.

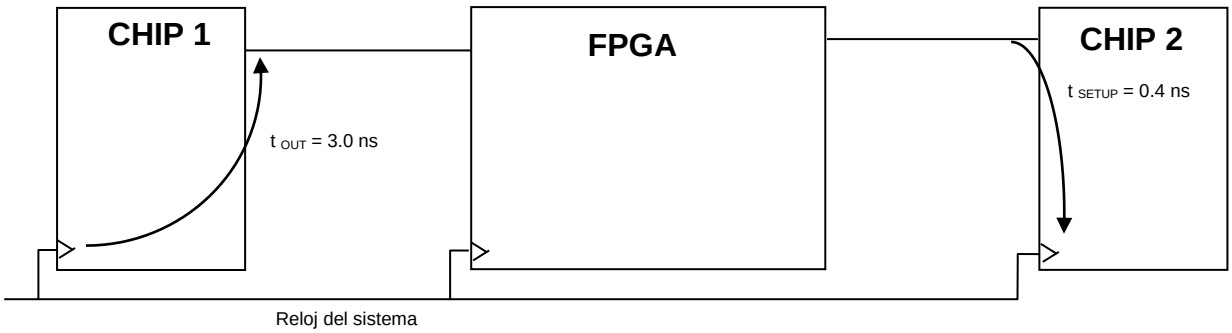



Figura 2. Condiciones temporales de las entradas y salidas de la FPGA en el sistema

Teniendo en cuenta todo esto:

- Modificar las restricciones (*constraints*) de tiempos de entrada/salida que aparecen al final del fichero de constraints, *set_input_delay* y *set_output_delay*, para que la herramienta intente cumplir los requisitos de la FPGA en el sistema. Descomentar las líneas borrando el primer carácter (#) y modificar los valores. Entender los argumentos que forman parte de estos comandos de constraints (sin necesidad de investigar la sintaxis en detalle).
- Re-implementar y generar reportes de timing específicos para las entradas y para las salidas. Para ello, ejecutar dos veces *Reports* > *Timing* > *Report Timing…*; en un caso especificar como *Start Points* "[all_inputs]" (sin las comillas) y en el otro como *End Points* "[all_outputs]", dejando en cada caso la casilla contraria en blanco.
  - ⇨ Estudiar el reporte del camino peor resultante en cada uno de los dos reportes de timing. Sobre la ventana de reporte **de este peor path**, hacer clic con el botón derecho y ejecutar "***Export Entire Path to Spreadsheet…***" Usar el nombre de fichero **TableEj2in.xls** para las entradas y **TableEj2out.xls** para las salidas (ambos ficheros deben quedar en el directorio *vivado/*).
  - ⇨ ¿Se cumplen los requisitos? ¿Cuál es el *slack* para la constraint de las entradas? ¿Y para la de las salidas? Responder a estas preguntas en la Hoja de Respuestas.

## *Ejercicio 3: Cambios en la frecuencia de reloj*

En este ejercicio se introducen requisitos más exigentes para la velocidad del reloj. Los procesos de *Synthesis, Mapping* y *Place and Route* tratarán de cumplir los requisitos impuestos, tras lo cual realizaremos un análisis de tiempos.

a)

1. Partiendo del fichero de constraints tal y como lo hemos dejado en el ejercicio anterior, asignar ahora un objetivo de periodo de reloj para el diseño de 12 ns (83.333 MHz), editando el fichero XDC. Tener en cuenta que en la constraint el reloj debe mantener su forma ("-*waveform*") con duty cycle del 50%, esto es, primer flanco en t=0 y segundo en t=Periodo/2.
2. Reimplementar y mirar el reporte de *Report Timing Summary*.

⇨ ¿Se logra cumplir los requerimientos?¿Qué tipo de camino provoca el peor *slack* (entrada, salida o interno) y cuál es el valor de este *slack*?. Responder en la Hoja de Respuestas.

⇨ Anotar los resultados de área y *timing* en la tabla de la Hoja de Respuestas ("3a").

⇨ Exportar el reporte detallado **del camino crítico**, igual que se ha hecho en otros apartados. Utilizar el nombre **TableEj3a.xls** (de nuevo este fichero debe quedar en el directorio *vivado/*).

b)

3. A continuación imaginaremos que la comunicación de la FPGA con los otros chips no es un problema (hay técnicas para mejorar el *timing* en las interfaces que quedan fuera de los objetivos de esta práctica). Nos centraremos en la constraint de reloj. Quitar las restricciones de entrada-salida: Volver a comentar, en el fichero XDC resultante del apartado a), las lineas *set_input_delay*… y *set_output_delay*… (poniendo un carácter '#' al principio de la línea).
4. Modificar el requisito de periodo de reloj a 8 ns (125 MHz)
5. Reimplementar y mirar el reporte de *Report Timing Summary*.

⇨ ¿Se logra cumplir los requerimientos? ¿Qué *slack* tenemos? Responder en la Hoja de Respuestas.

⇨ Anotar los resultados de área y *timing* en la tabla de la Hoja de Respuestas ("3b").

⇨ Exportar el reporte detallado **del camino crítico**, igual que se ha hecho en otros apartados. Utilizar el nombre **TableEj3b.xls** (de nuevo este fichero debe quedar en el directorio *vivado/*).

## *Ejercicio 4: Pipeline*

En este ejercicio se realiza una segmentación (*pipeline*) del circuito, aplicando una cadena de flip-flops en mitad de la ruta de datos (ver Figura 3 más abajo). Esto aumenta la latencia (retardo de entrada a salida), pero produce una mejora significativa en la velocidad de reloj soportada.

Partir del proyecto tal y como queda al final del ejercicio 3. Esta vez no tocaremos el fichero de *constraints* sino el VHDL. Añadir una etapa de registros intermedios de pipeline en *chksum.vhd*, tal como se muestra en la Figura 3.

- **Simular** el circuito, **comprobando su corrección** tras los cambios.
- **Reimplementar** y mirar el reporte de *Report Timing Summary*.

⇨ ¿Se logran los requerimientos? ¿Qué *slack* tenemos? Responder en la Hoja de Respuestas.

⇨ Anotar los resultados de área y *timing* en la tabla de la Hoja de Respuestas (“4”).

⇨ Exportar el reporte detallado **del camino crítico**, igual que se ha hecho en otros apartados. Utilizar el nombre **TableEj4.xls** (de nuevo este fichero debe quedar en el directorio *vivado/*).

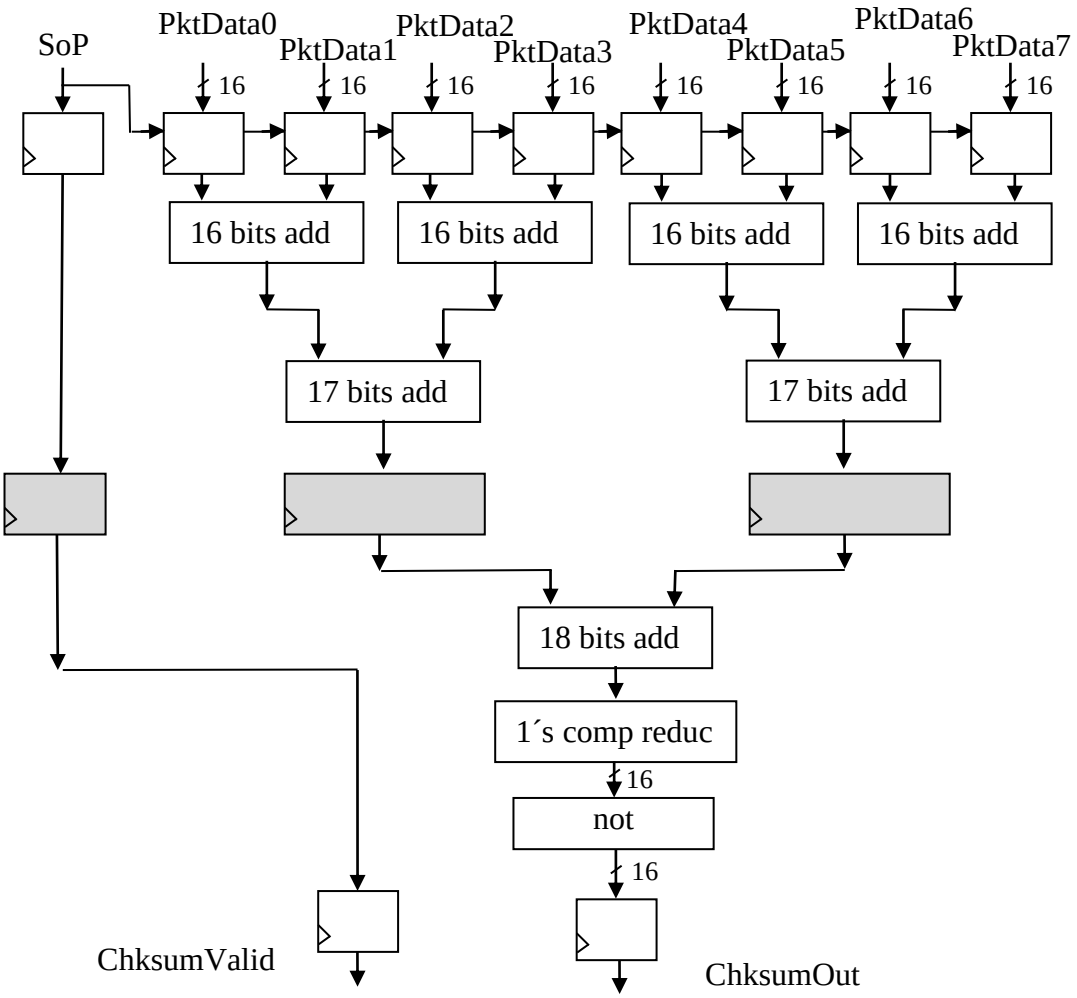



Figura 3. Checksum segmentado en dos etapas (una etapa de registros intermedia).

## *Preguntas adicionales para razonar (no se requiere entrega)*

a. Teniendo en cuenta solo la *constraint* de periodo de reloj, si obtenemos un determinado WNS (*Worst Negative Slack*), ¿cuál sería el mínimo periodo de reloj que podríamos utilizar?

b. ¿Cómo se relacionan para el camino crítico los conceptos: *WNS*, *Period Requirement*, *Data Path Delay*, *Clock Path Skew*, Tiempo de Setup del FF y *Clock Uncertainty*?

c. ¿Tiene sentido que al ser más restrictivo (reducir el período) cambie el uso de recursos (cantidad de registros y/o número de *slices* y *LUT*s)?

## *Entrega*

Antes de la fecha límite especificada en el calendario del laboratorio **deberá entregarse en Moodle un único archivo comprimido (zip o rar) con:**

- **El directorio *P3_checksum* completo**. Bajo el subdirectorio *vivado* deberá estar el **proyecto entero de Vivado** y los **ficheros Table*.xls** cuya generación se solicita en este enunciado.

- La **Hoja de Respuestas** que se proporciona en Moodle, con las respuestas rellenas y el nombre del alumno. Dejar esta hoja directamente bajo el directorio *P3_checksum*. La hoja puede entregarse en formato Word o PDF (en este último caso puede ser el resultado de escanear el documento relleno a bolígrafo, mientras se entienda bien).

El archivo comprimido deberá tener la siguiente **nomenclatura**:

```
<<nº de grupo>>_<<nº de alumno (dos dígitos)>>_<<nº de práctica>>.[zip|rar]
```

(Ej.: `3311_05_3.zip` para el alumno 5 del grupo de DIE de los lunes)

**Se recuerda que los alumnos deberán comprender el diseño completo y familiarizarse con él.**