## 1. Objetivos

El objetivo principal de este proyecto ha sido la implementación de un procesador tolerante a fallos transitorios. Para ello se ha dividido en 3 tareas: Diseñar el procesador y su implementación, dotar al diseño de tolerancia a fallos transitorios y comprobar el funcionamiento tanto de los microprocesadores antes como después de modificarlo.

## 2. Procesador

El microprocesador utiliza instrucciones codificadas en 32 bits

El juego de instrucciones elegido ha sido extraído del repertorio de instrucciones ARM, conocido como THUMB-2. Estas instrucciones buscan reducir la densidad del código, es decir, utilizar menos instrucciones para realizar el mismo trabajo que en otros repertorios. Se utiliza principalmente en los microprocesadores ARM de la gama CORTEX.

La arquitectura Harvard permite acceder a la memoria de instrucciones al mismo tiempo que se accede a la memoria de datos.

Posee 16 registros de propósito general y está segmentado en 5 etapas para aprovechar al máximo el hardware.

## 3. Instrucciones

Las instrucciones que ejecuta el procesador se divide en tres tipos:

Instrucciones de acceso a memoria, lectura y escritura.

Procesamiento de datos, incluye operaciones aritméticas (sumar, restar, mover y comparar) y lógicas (and, or y or exclusiva). Los operandos de estas instrucciones tener origen en dos registros o un registro y un inmediato extraído de la instrucción.

Las operaciones de control, o instrucciones de bifurcación, controlan el flujo del programa, es decir modifican la dirección de la instrucción que se debe cargar a continuación.

## 4. Segmentación

El procesador se ha segmentado en 5 etapas, en cada una de ella se ejecuta una parte de la operación de la instrucción.

Las etapas son:

1) Búsqueda de instrucción (IF): Se carga la instrucción de memoria.

2) Decodificación de la instrucción (ID): Se decodifica la instrucción y se leen los operandos del banco de registros.

3) Ejecución (EX): Se realizan las operaciones aritméticas o lógicas.

4) Memoria (MEM): Se accede a la memoria de datos.

5) Escritura (WB): Se almacenan los resultados en el banco de registros.

La segmentación permite tener varias instrucciones de forma simultánea en el procesador, acelerando la ejecución del programa.

Como podemos ver en la imagen, en cada ciclo se accede a una instrucción y esta va avanzando por las etapas del procesador permitiendo que se carguen las siguientes instrucciones.

## 5. Segmentación

El procesador se ha segmentado en 5 etapas, en cada una de ella se ejecuta una parte de la operación de la instrucción.

Las etapas son:

1) Búsqueda de instrucción (IF): Se carga la instrucción de memoria.

2) Decodificación de la instrucción (ID): Se decodifica la instrucción y se leen los operandos del banco de registros.

3) Ejecución (EX): Se realizan las operaciones aritméticas o lógicas.

4) Memoria (MEM): Se accede a la memoria de datos.

5) Escritura (WB): Se almacenan los resultados en el banco de registros.

La segmentación permite tener varias instrucciones de forma simultánea en el procesador, acelerando la ejecución del programa.

Como podemos ver en la imagen, en cada ciclo se accede a una instrucción y esta va avanzando por las etapas del procesador permitiendo que se carguen las siguientes instrucciones.

## 6. Procesador

En esta imagen se muestra el diseño del procesador. Las líneas continuas (o negras) muestran la ruta de datos del procesador mientras que las líneas discontinuas (o azules) muestran las señales de control.

Las líneas verticales (o rojas) son los registros de control, estos son los elementos más importantes de la segmentación. En cada ciclo de reloj almacenan los datos calculados por los módulos que se la proporcionan, para que estos ejecuten la siguiente instrucción.

En la etapa de búsqueda se accede a la memoria de instrucciones y se calcula la dirección de la siguiente instrucción. En la etapa de decodificación se extrae de la instrucción la información que permite ejecutar la instrucción y se cargan los valores de los operandos. En la etapa de ejecución se realizan los cálculos aritmético-lógicos y se calcula la dirección de salto. En la etapa de memoria se cargan datos o se almacenan en la memoria de datos. Para terminar, en la etapa de escritura se escriben los resultados en el banco de registros.

## 7A. Fallos

Los fallos se pueden clasificar en fallos permanentes y fallos transitorios. Nos hemos centrado en tolerar los fallos de esta última categoría.

## 7B. Fallos

A su vez, dentro de esta categoría existen varios tipos, pero nosotros nos centraremos en los “Single-Event Upset”

Estos afectan a los biestable inviertiendo su valor. Si el valor cargado en el biestable fuese 0, en este se cargaría un valor 1, provocando un error en la ejecución del sistema.

## 8.Tolerancia

El método de tolerancia a fallos transitorios aplicado es la redundancia modular.

Este consiste en replicar un módulo un número impar de veces y conectar un nuevo elemento conocido como “votador de mayoría” a sus salidas. Este nuevo elemento.

En las imágenes vemos el resultado de aplicar la redundancia modular triple a un módulo.

## 9A. Votador

El implementación debe cumplir la tabla de verdad que vemos. Su salida debe coincidir con la mayoría de sus entradas, así se tolera el fallo de uno de los módulos.

El diseño del votador utilizado es uno de los más comunes, consiste en 3 puertas AND comparando las posibles combinaciones de sus entradas y una puerta OR conectada a las puertas AND.

## 9B. Votador (ejemplo)

En este ejemplo vemos el funcionamiento del votador.

El módulo C falla, invirtiendo su valor. Y se obtiene el valor correcto en la salida del votador gracias a que éste enmascara el fallo.

Para el proyecto se ha aplicado la TMR en los registros de control, triplicando los registros e insertando estos votadores.

## 10. Ejecución 1

El programa ejecutado realiza una multiplicación mediante sumas acumulativas en un búcle.

Primero se accede a memoria y se lee un operando de la operación y se inicializan las demás variables. A continuación se ejecuta el bucle. Y para finalizar se almacena el resultado en memoria.

## 11. Ejecución 2

Se vuelve a ejecutar el mismo programa en el procesador sin tolerancia. Esta vez se insertan tres fallos.

1. El primer fallo se genera en el bus de lectura de la memoria. Como consecuencia se carga un 4 en vez de un 5 en el registro R2.
2. El segundo fallo se inserta en las señales de control de la etapa de memoria, esto provoca que se escriba en memoria un dato.
3. El tercer fallo se genera en el bus del operando B de la etapa de ejecución. Esto provoca que la comparación dé un resultado positivo y se termine la ejecución del bucle antes de tiempo.

El resultado final es un programa que ha obtenido un resultado erróneo y además ha modificado datos en memoria que puede estar utilizando otro programa.

## 12. Ejecución 3

Se ha ejecutado una última vez el programa sobre el procesador tolerante a fallos y se insertan los mismos fallos. Estos fallos se introducen en uno de los registros triplicados que almacenan la misma información.

Como podemos ver, el programa se ha ejecutado de forma correcta y sin modificar otros elementos en memoria.