Cómo estructurar un proyecto FW

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2024/11/30/como-estructurar-un-proyecto-fw/

Blog: https://soceame.wordpress.com/

GitHub: https://github.com/DRubioG

Fecha última modificación: 23/02/2025

No espero que esta entrada os haga cambiar cómo estructuráis vuestros proyectos, pero sí que os replanteéis si podéis añadir mejoras a vuestras estructuras que a lo mejor no os habíais planteado.

Esta entrada es el resultado de un motón de experiencias y de motón de fallos y problemas después de desarrollar un motón de proyectos FW tanto propios como proyectos que han desarrollado otros y han recaído en mi para terminarlos o para mantenerlos. Y muchos de los problemas residen en el caos en el que la gente trabaja, muchas veces por desconocimiento. La solución es muy simple, **estructuración de carpetas**, parece una tontería pero cuando tienes que buscar ficheros en VHDL o en Verilog que son funcionales en un proyecto te aseguro que facilita mucho la vida.

Ahora te voy a exponer los diferentes problemas que hay a la hora de estructurar los proyectos firmware. Y al final te expongo mi forma de estructurar los proyectos, que me ha salvado a mi y a más gente de darme contra una pared.

Documentación

Uno de los problemas típicos al recoger el trabajo de otros es que han juntado la documentación con el proyecto funcional, entonces, tienes que buscar entre un motón de ficheros la documentación.

Por eso la documentación siempre tiene que estar separada de la parte técnica, así todo lo que no es proyecto firmware está en otro sitio distinto y es más fácil de localizar.

Meter el código fuente con el proyecto de la herramienta

Es muy típico de la gente con poco nivel meter el proyecto firmware que crea la herramienta (Vivado, Quartus, Libero,..) en la carpeta de código fuente. Bien pues esto es un error de bulto, porque estás creando una carpeta donde se supone que metes todo tu desarrollo junto con todos los ficheros que crean las herramientas. Entonces, creas carpetas de las que una pequeña parte de los ficheros son útiles y el resto son ficheros prescindibles o que genera la herramienta.

Además, de que si la versión de la herramienta se vuelve una versión obsoleta ese proyecto no va a valer para nada. Por lo que tienes que adaptarte a poder regenerar tu proyecto en una nueva versión de la herramienta, esto también **soluciona la deuda tecnológica creada.**

Esto se soluciona de dos formas, **separando el código o los ficheros útiles o de reconstrucción** de tu proyecto de dónde creas el proyecto (además, las herramientas te permiten referenciar a esa carpeta, por lo que cualquier modificación del Firmware se actualiza en la carpeta donde está el código fuente) y con un **pequeño documento comentando la estructura de tu proyecto** (también se puede utilizar un TCL que permita regenerar tu proyecto, esto lo generan todas las herramientas).

El código y su cajón de sastre

Otra muy típica es meter el código fuente, con los testbenchs, con los bitstreams y con el código de test en una misma carpeta, este es otro error de bulto.

A la hora de buscar en tu código el que lo tenga que hacer, incluido tú mismo, lo vais a pasar bien jodido tratando de localizar aquellos ficheros que son de código fuente, de los de test.

Esto tiene una solución muy simple, **separar los ficheros por tipo de funcionalidad**, teniendo una carpeta para cada tipo de fichero. Una carpeta para todo el código fuente (esta carpeta tiene que estar impoluta, sin subcarpetas ni nada, solo código funcional), una carpeta para los testbenchs, otra carpeta para los ficheros de test (esta carpeta la podéis llenar de «mierda» hasta los topes) y una carpeta para meter los bitstreams (todos los bitstreams tienen que estar numerados). También puede ser necesario si se utiliza los diagramas de bloques de las herramientas, meter el fichero que permita regenerar esos diagramas.

Control de versiones

Esto es fundamental para evitar el caos, es necesario hacer un control de versiones de todo el código que se desarrolla, ya sea utilizando una herramienta o mediante una carpeta separada. Esta carpeta debe contener versiones comprimidas del proyecto y también del código funcional, los bitstreams. Que estén comprimidas es para garantizar su inmutabilidad y que también ocupen menos.

También es importante en esta carpeta tener un documento en el que se pueda ver el número de versión, fecha en la que se creo y que actualizaciones incorpora, detallando todo, y si se añade o elimina algo, añadir también en qué versión se añade o en que versión se elimina o comenta.

Versión	Fecha	**Comentario**
1.0	24-5-2024	Versión inicial
1.1	24-11-2024	Versión en la que se ha cambiado el valor de la frecuencia de la UART de 115200 (0x4FD) a 9600 baudios (0x343).

Control de versiones de los testbenchs

Los testbenchs son muchas veces los grandes infravalorados del mundo del FW porque nos centramos más en el desarrollo del FW funcional, *en una entrada futura os comentaré los poderes de los testbenchs además de solo simular.*

Puede parecer una tontería pero los testbenchs son ficheros muy descriptivos del funcionamiento real del sistema exterior, por lo que es recomendable hacer un control de versiones específico de ellos, apuntando diferentes detalles sobre su trabajo con ellos. Sobretodo porque al principio se hacen unos testbenchs que según va aumentando el desarrollo del proyecto estos testbenchs se quedan obsoletos y ya no valen, y toca rehacerlos, entonces, es bueno tenerlos controlados.

Un ejemplo de esquema a seguir es el siguiente.

Versión	**Nombre**	**Módulos simulados**	**Estado**	**Fecha de última simulación**	**Comentarios**

2.00	top_tb	top – ADC – DAC – PWM – Watchdog	Simulado con fallo	24-11-2024	Esta versión comprueba que todo el sistema funciona correctamente añadiendo unos ADCs de otro modelo.
1.15	top_tb	top – ADC – DAC – PWM – Watchdog	Simulado	15-06-2024	Esta versión actualiza el watchdog a los requisitos de la versión 1.15 del FW

Numerar los bitstreams

Es necesario que los bitstreams estén numerados, con la fecha y/o con la versión del FW debido a que muchas veces la cosas funcionan en una versión pero no en otra, entonces tener el bitstream en una versión separada hace que se pueda hacer una trazabilidad.

Cómo yo estructuro los proyectos

Esta es la forma en la que yo estructuro los proyectos, sé que se puede mejorar la forma en la que lo hago y que puede ser susceptible de cambiar con el tiempo, pero de momento cumple con los requisitos anteriormente expuestos.

```
Nombre proyecto
- docs
    - documento (aquí se describe todo el FW y su funcionamiento)
    - imágenes (la imágenes del documento)
    - esquemático de la placa
    - test (carpeta para todo lo documental no oficial)
    - (_old, esta carpeta es donde todo los documentos antiguos
que pueden ser útiles)
    - (control de versiones de la documentación, si fuera
necesario)
        - documentos
        - esquemáticos
        - imágenes
- proyecto
    - firmware
        - src (esta carpeta tiene que estar impoluta y sin
subcarpetas, todas las subcarpetas o pruebas van a la carpeta
test)
        - bitstream
        - testbenchs
        - test (en esta carpeta va todo lo _no oficial_ que me
ayuda a desarrollar)
        - (diagrama de bloques, con una foto o un TCL)_
```

```
    - control de versiones
        - src (con su documento de control de versiones)
        - testbench (con su documento de control de versiones)
        - bitstream (el documento de control de versiones puede
ser el mismo que el del src)
```

Tener el código fuente limpio y en una carpeta separada permite decirle a la herramienta de firmware que se direccione a esa carpeta, haciendo que todos los cambios se hagan en la carpeta del código fuente.