Presentazione generale / intro :

* Obiettivi (risultati attesi)
* Problemi da risolvere “da progetto/stato dell’arte” / soluzioni : perché facciamo sta cosa? Perché non c’è un debug come il nostro : non ci sono debugger così semplici per guardare i segnali che il processore scambia con la memoria. Non ci sono metodi semplici per osservare l’interfaccia uP / memoria (system bus , code bus, AHB, PPB)

Metodi: come l’abbiamo implementato: testbench, diverse memorie con relativi problemi (memory address space scelto dal stm32…), axi4stream, diversi bus per diversi indirizzi, il uP chiede 1 address e noi (memoria) rispondiamo col contenuto di 4 celle, salvataggio dati in 4 celle, dig endian / little endian

Debug: la nostra idea è quella di scrivere “1” in uno specifico indirizzo (come per accendere un led), prima di un blocco di codice C, e poi scrivere “0”. In questo modo abbiamo una “finestra” nei segnali su vivado che ci permette di identificare con più facilità quali sono le operazioni/gli scambi di info che il processore fa con la memoria.

È un modo per vedere con facilità le operazioni di “basso livello” che vengono tradotte da C/assembly a AXI / bus sys / bus code.

Tenere in considerazione che c’è il bit-banding. Se vogliamo più informazioni riguardo i dati/indirizzi scambiati, si potrebbe utilizzare un SW che semplicemente decripti il protocollo.

Problema che abbiamo avuto : scrivendo nel registro corretto della periferica GPIO, questa non rispondeva. Probabilmente perché avremmo dovuto emulare/scrivere il codice di tutta la periferica HW per avere una risposta in uscita. Forse il registro A scrive direttamente sul B senza l’aiuto del uP… o forse abbiamo sbagliato noi… molto probabile! Comunque abbiamo risolto scrivendo in un indirizzo e, ipotizzando che lì ci sia la periferica, abbiamo osservato quello specifico indirizzo su vivado per osservare la “finestra” (abbiamo considerato questa la risposta del GPIO)

Abbiamo messo file.mem in entrambe le flash perché il PC parte dall’indirizzo 0x0000.., dove dovrebbe esserci il bootloader che fa il jump all’indirizzo 0x08000…, ma noi il jump lo facciamo direttamente con il jump della reset\_handler (quindi facciamo 0x000.., 0x0004 -> 0x08000…) -> stack init + jump to reset handler (startup code)

Conclusione : è stata dura ma l’antico vaso è stato portato in salvo