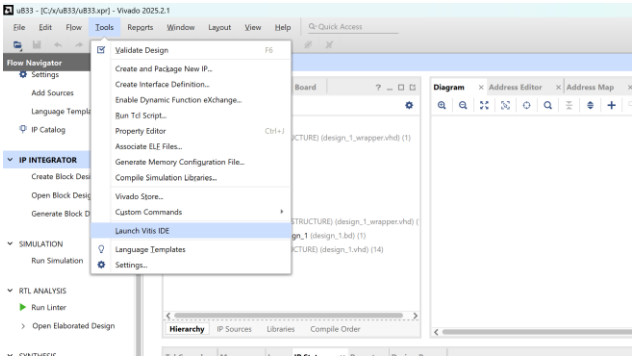
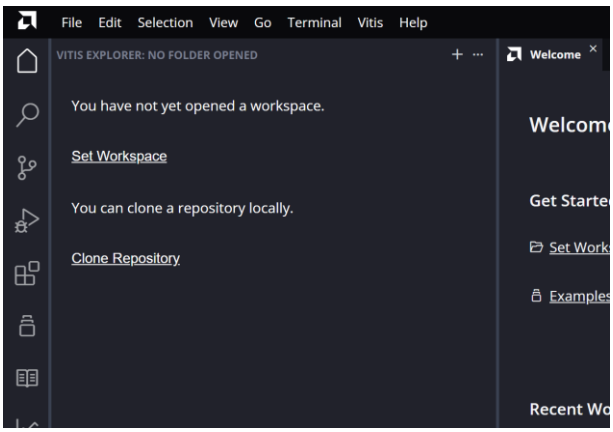
Guida alla modifica del SoC UB33 (realizzato con versioni di Vivado/Vitis precedenti) in Vivado/Vitis 2025.2.1, con indicazioni per superare alcuni problemi nell'aggiornamento del BSP da parte di Vitis. (seconda parte)

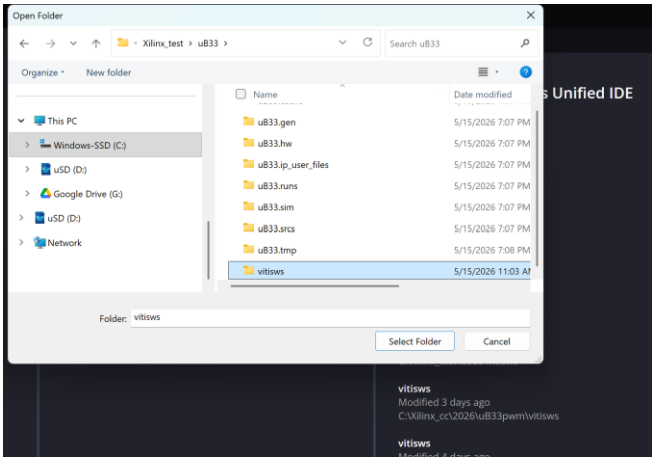
Apriamo Vitis da windows, o dall'interno di Vivado come mostrato in figura:



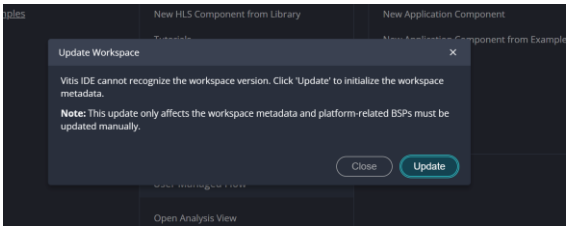
Una volta aperto:



per prima cosa selezioniamo "set workspace". Scegliamo la directory vitisws all'interno della directory uB33:

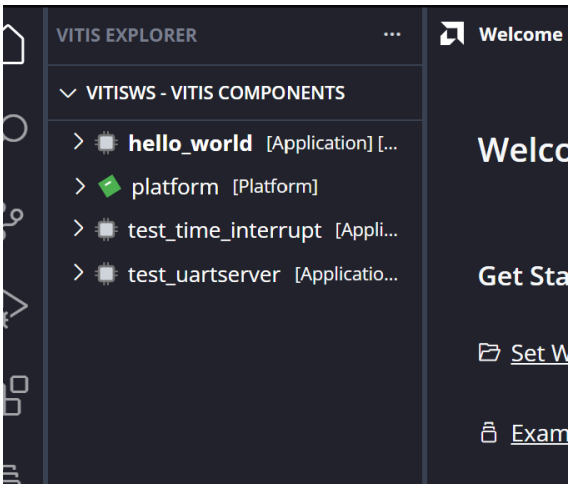


Quando compare la finestra:

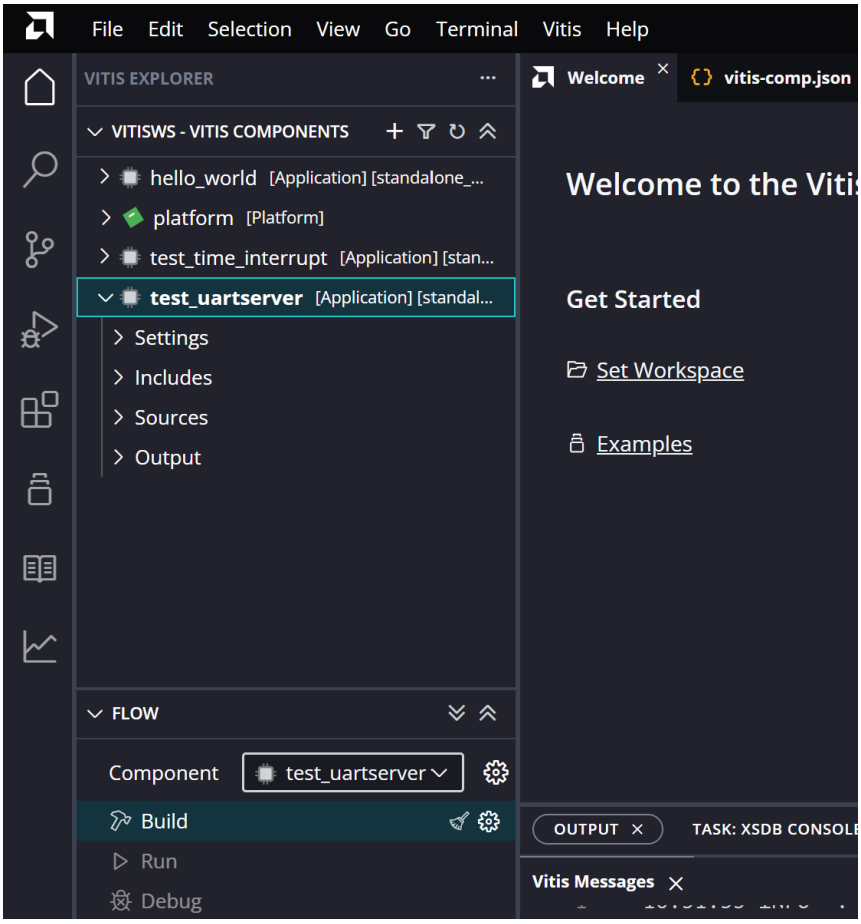


fare click su update.

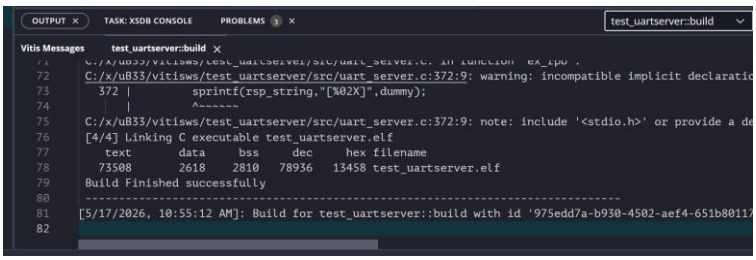
Al termine dell'upgrade nella finestra a sinistra troviamo "platform" che è il BSP per l'hardware dal quale siamo partiti prima delle modifiche e tre progetti software di complessità crescente che fanno riferimento a questo hardware.



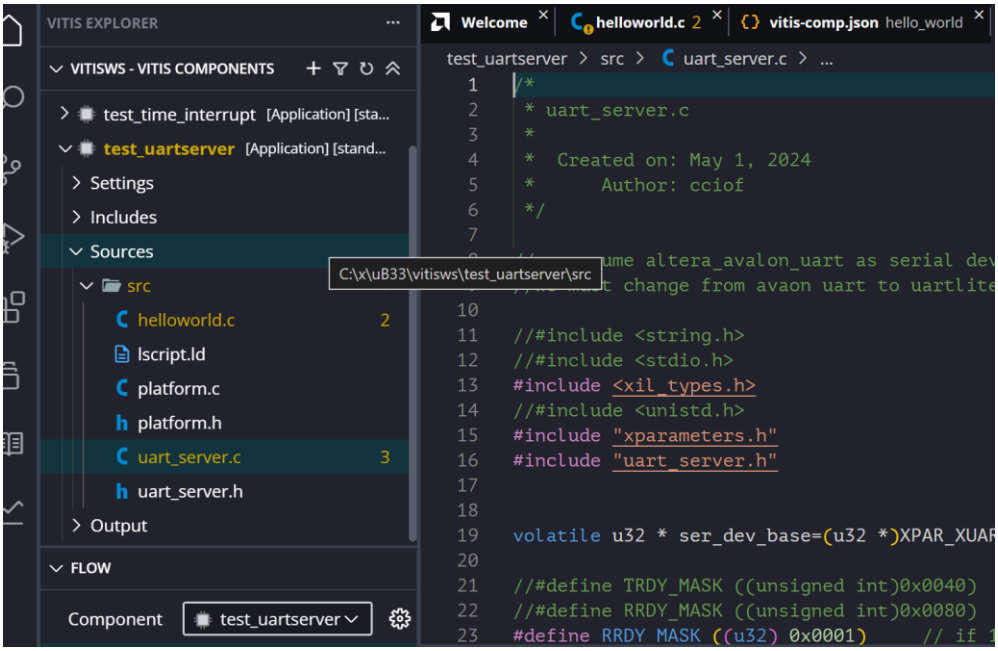
Noi lavoreremo su test_uartserver. Per prima cosa verifichiamo che, con il BSP precedente, test_uarserver possa essere compilata correttamente.



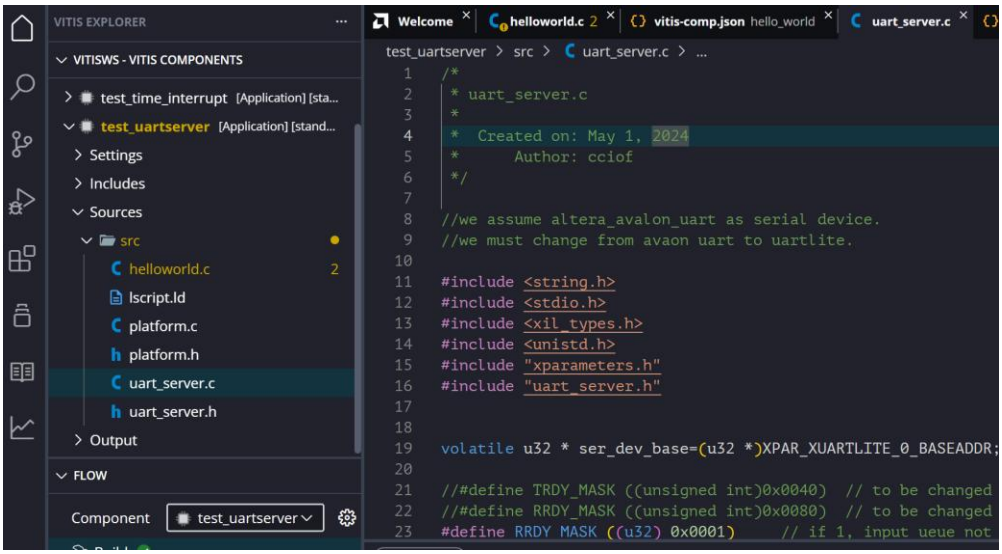
Selezioniamo il componente e facciamo click su build nella finestra Flow. Nella finestra ouput ci viene dato il messaggio di compilazione avvenuta ma ci sono anche errori relativi al fatto che sembrano essere assenti alcune librerie. Questo è il risultato di una svista nel preparare il progetto prima di caricarlo sul server e può essere corretto facilmente.



Nella gerarchia del componente test_uartserver cercare ed aprire il file uart_server.c:



A questo punto eliminiamo il segno di commento agli include relativi a string.h, stdio.h e unistd.h, ottenendo così:



Salviamo tutti i file e ripetiamo il built come in precedenza. A questo punto non dovrebbero esserci errori segnalati nella finestra di output:

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
[5/17/2026, 11:01:29 AM]: Build for test_uartserver::build with id 'dcd2844c-7908-4c55-8b0f-76be0beb
--------------------------------------------------------------------------------
Command: cmd.exe /C set CC= && set CXX= && empyro.bat build_app -s c:\x\uB33\vitisws\test_uartserver
[INFO]: Starting Application build process
[INFO]: Building Application
[1/2] Building C object CMakeFiles/test_uartserver.elf.dir/uart_server.c.obj
[2/2] Linking C executable test_uartserver.elf
   text       data     bss     dec     hex filename
  73508       2618    2810   78936   13458 test_uartserver.elf
Build Finished successfully
--------------------------------------------------------------------------------
[5/17/2026, 11:01:32 AM]: Build for test_uartserver::build with id 'dcd2844c-7908-4c55-8b0f-76be0beb9
```

Il prossimo passaggio è creare il nuovo bsp per il nuovo hardare e modificare test_uartserve per adattarlo ad esso, inserendo anche blocchi di codice per il test dei pwm e per pilotare gli stessi mediante comandi attraverso un terminale secondo il protodollo descritto nelle lezioni/video precedenti.

ATTENZIONE: Il processo di associazione di un nuovo bsp a un progetto già esistente può essere condizionato da un bug nel software:

VITIS > VITIS EMBEDDED DEVELOPMENT ...

digitalone (Member) asked a question.
April 2, 2024 at 7:12 PM

Workaround: Updating hardware specifications in a Vitis Unified project using XSA doesn't work.

Vitis Unified 2023.2 lwIP/FreeRTOS

Issue:
Updating hardware specifications in the Vitis Unified project using XSA doesn't work.

Reason:
Updating the platform using XSA doesn't update the bitstream file in the application.

Solution 1 (recommended):
After updating the platform using XSA, build the platform. Replace **{workspaceFolder}\app__ide\bitstream*_wrapper.bit** file with **{workspaceFolder}/platform/hw/sdt/*_wrapper.bit**.
Build the platform and project.

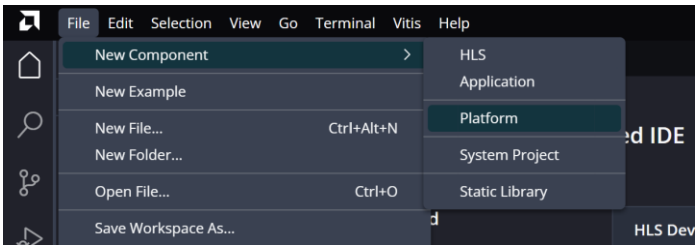
Solution 2:
Go to the "Settings" option from the application and open **launch.json**. Set the Bitstream File option to the following file.
{workspaceFolder}/platform/hw/sdt/*_wrapper.bit.

VITIS EMBEDDED DEVELOPMENT & SDK

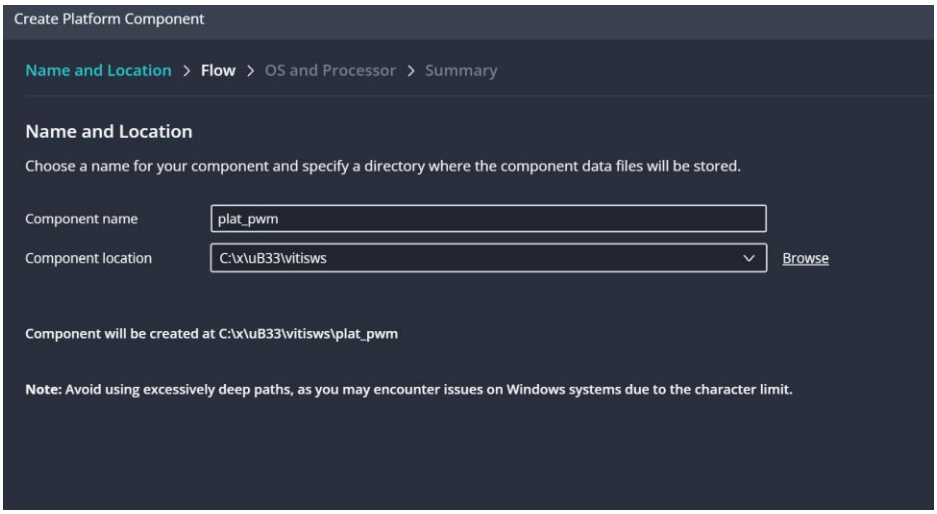
Like Share 6 answers • 2.48K views

In queste note seguiamo il procedimento che sembra evitare di incorrere in problemi.

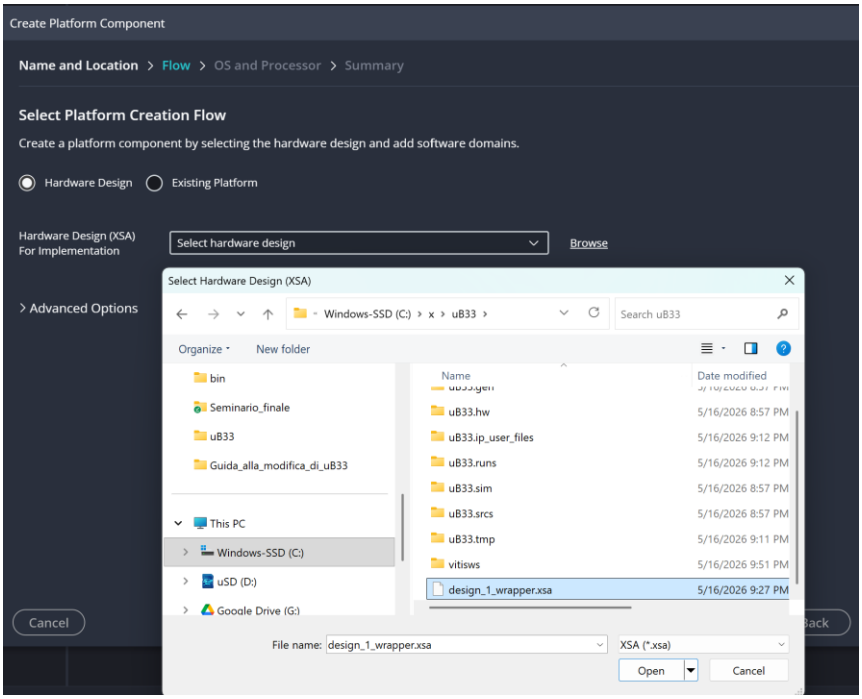
Iniziamo creando il nuovo BSP



Partiamo da File->New Component ->Platform

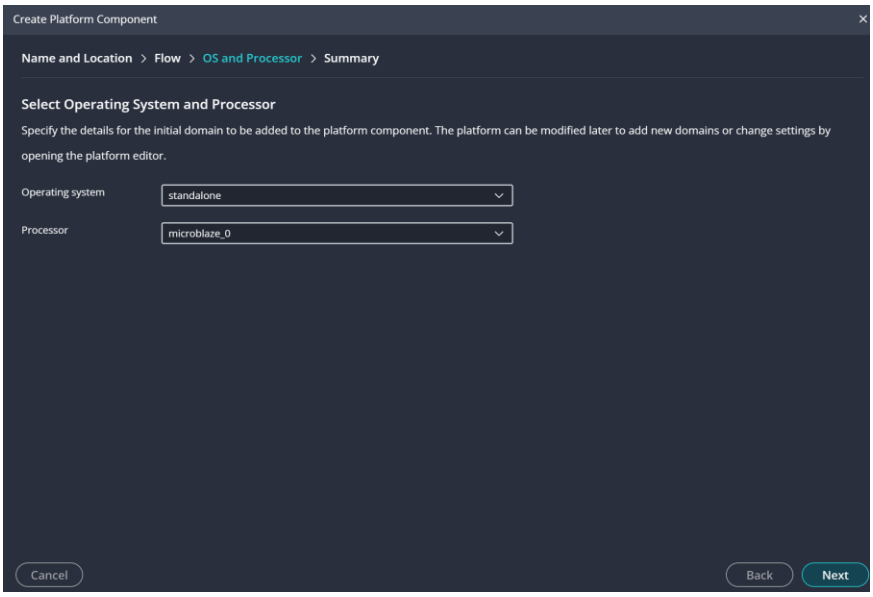


Inseriamo il nome del componente e lasciamo inalterata la component location. Quindi selezioniamo next.

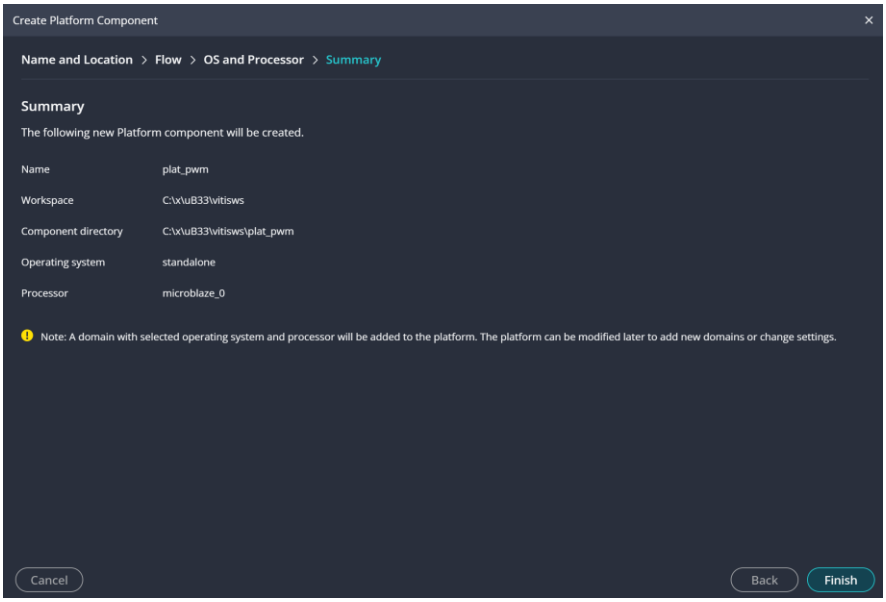


Selezioniamo “Hardware Design” e con browse cerchiamo l’hardware esportato (desig_1_wrapper.xsa nella directory uB33).

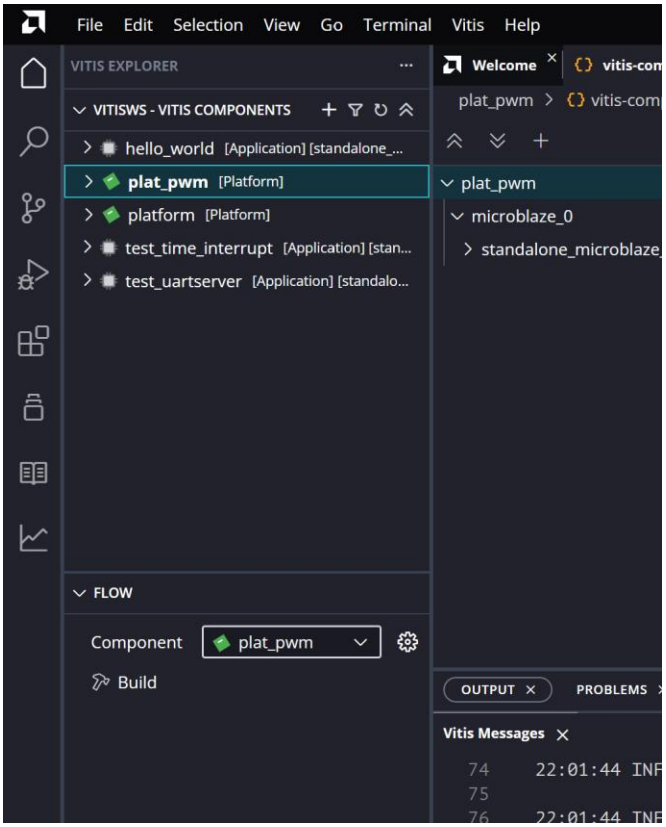
Quindi facciamo click su “next”. Varrà mostrata questa schermata:



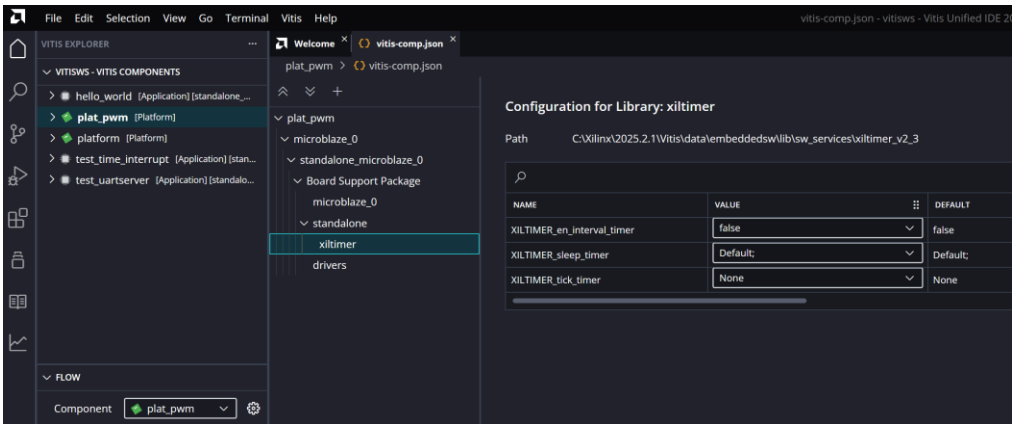
Selezioniamo next.



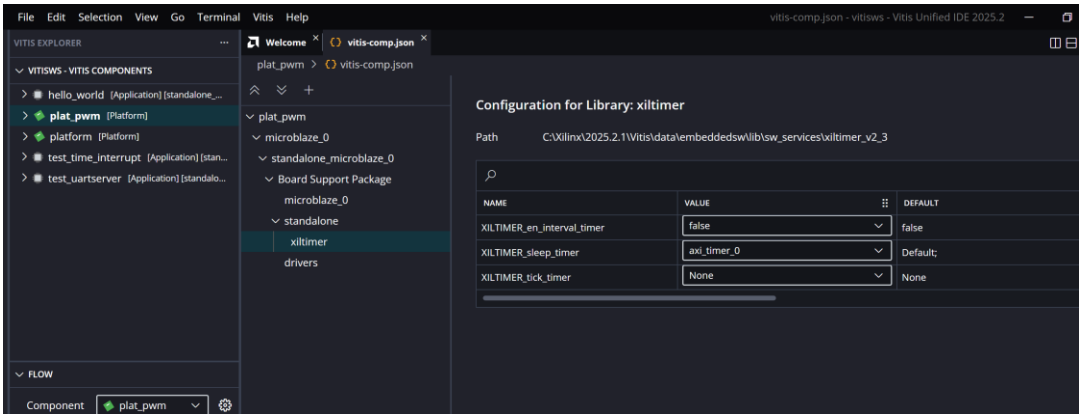
Ignoriamo il punto esclamativo e selezioniamo finish. Dopo un po’ di tempo e di elaborazioni, nell’elenco dei componenti a sinistra comparirà il nome del nuovo bsp:



Selezioniamo plat_pwm in modo che il nome compaia nel riquadro in basso nella sezione “Flow”. Quindi facciamo click sulla icona di impostazioni.

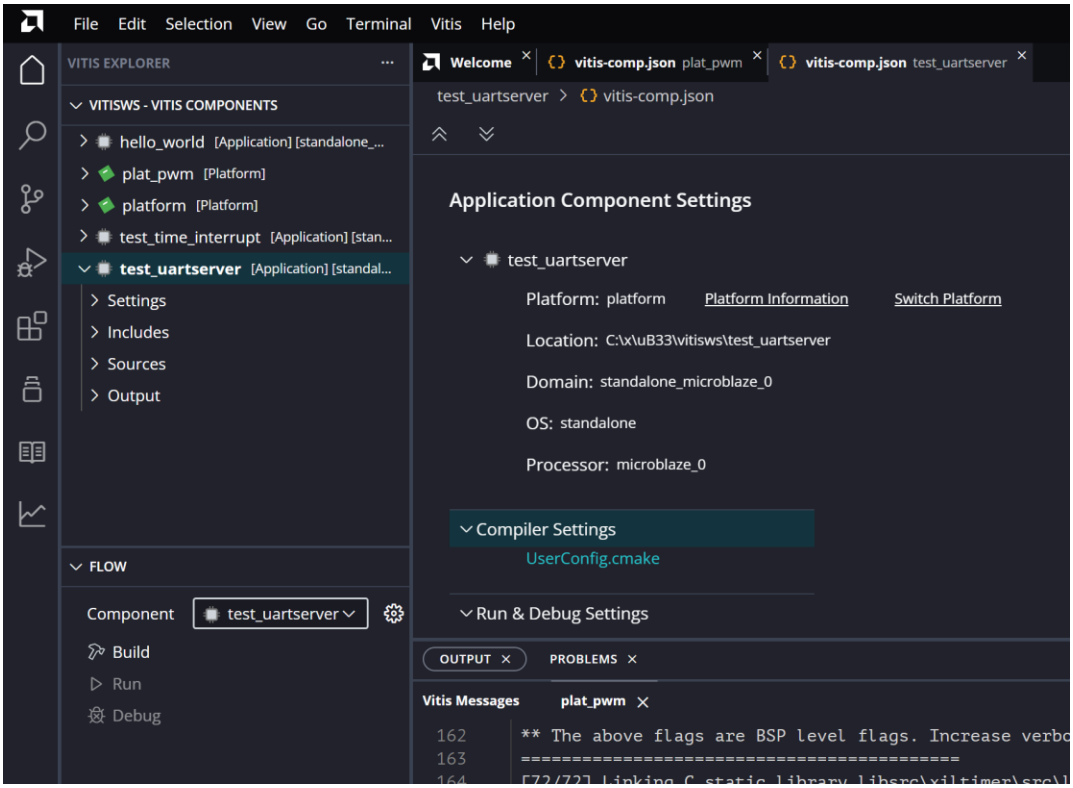


Percorrere la gerarchia come in figura selezionando xiltimer e cambiare l’impostazione XILTIME_sleep_timer da Default a “axi_timer_0” ottenendo:



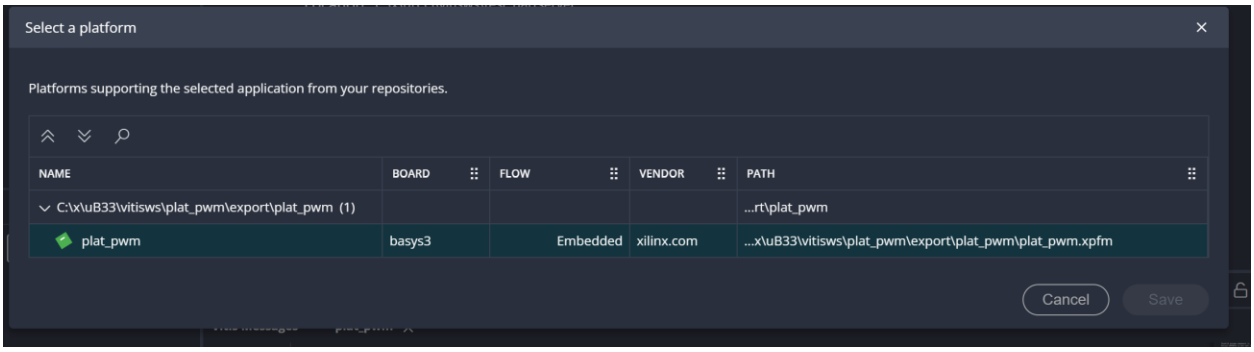
Possiamo passare ora all'applicazione test_uartserver. Dobbiamo intanto specificare il nuovo BSP.

Selezioniamo "test_uartserver" e il simbolo impostazioni nel riquadro Flow. Otteniamo:



Selezioniamo "Switch Platform" nel riquadro "Application Component Settings".

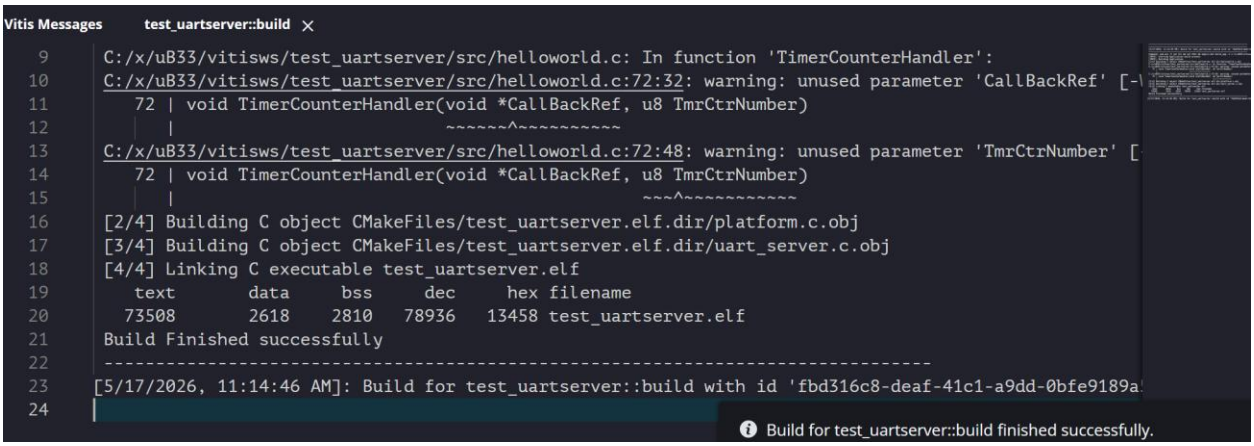
Nel pop up scegliamo il nuovo bsp:



Facciamo quindi click su plat_pwm e poi save.

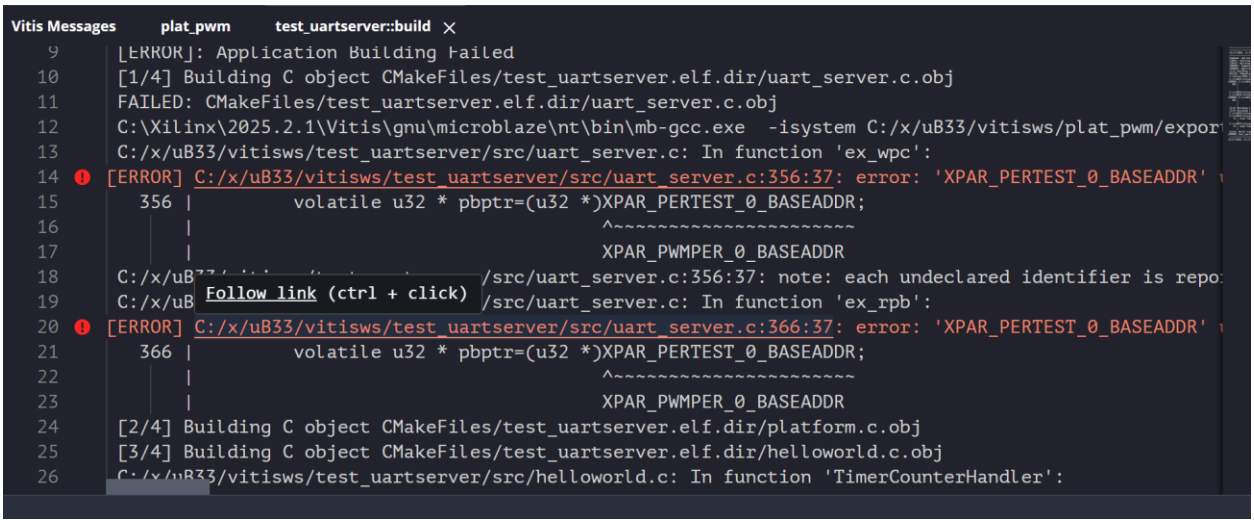
Ricordiamo che nell'hardware di partenza era presente una periferica che abbiamo eliminato prima di inserire il/i pwm. Il software test_uartserver contiene riferimenti alla periferica eliminata. Se quindi facciamo ora un build di testuartserver dovremmo ricevere degli errori perché i suddetti riferimenti non esistono più.

Purtroppo, a parte dei warning irrilevanti, otteniamo la compilazione senza errori, segno che siamo vittima del bug menzionato in precedenza.



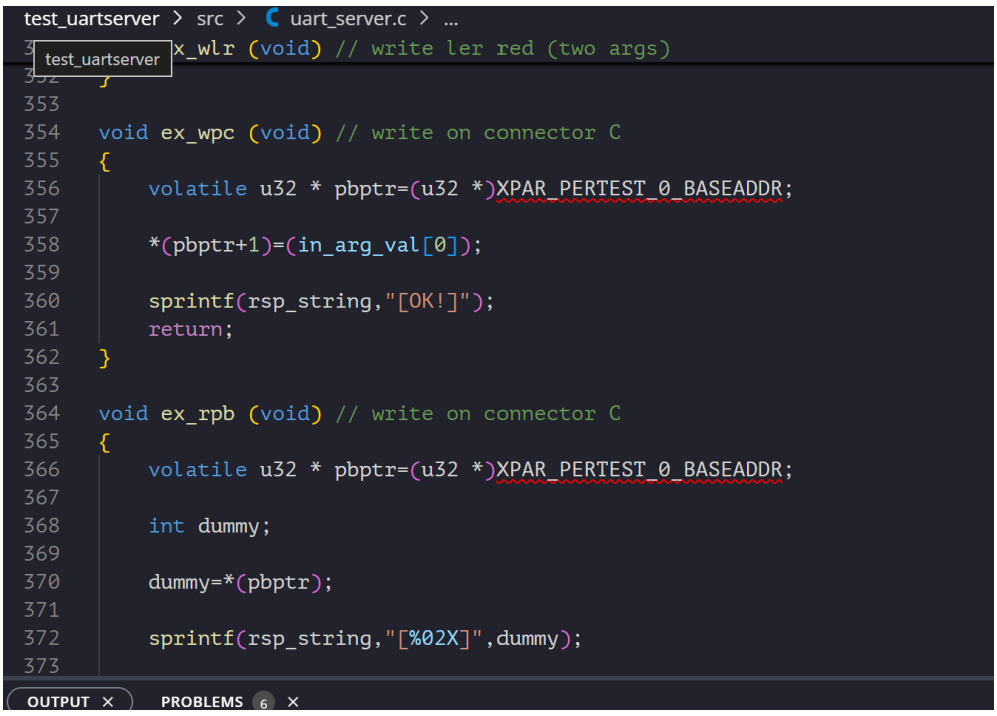
Il problema si può aggirare in due passi. Il primo passo consiste nel chiudere e riavviare vitis (vedi sotto); il secondo passo consiste nell'eseguire le correzioni riportate nel report di bug prima menzionato (vedi più avanti).

Come detto, procediamo a chiudere Vitis. Riavviamolo e riapriamo il workspace (che dovrebbe essere presente fra quelli recenti): questa operazione "forza" il riconoscimento del nuovo BSP e infatti, se a questo punto si fa nuovamente il build del progetto test_uartserver, compaiono gli errori che ci aspettavamo:



Gli errori consistono in riferimenti a una costante definita all'interno del BSP che rappresenta l'indirizzo del primo registro della periferica PERTEST che esisteva nell'hardware di partenza ma che non esiste più nel nuovo hardware.

Per eliminare gli errori, apriamo intanto il file uart_server.c:



Questo codice era scritto per la piattaforma precedente in cui compariva una periferica di nome PERTEST che noi abbiamo eleminato. Non esiste più un puntatore ai registri di questa periferica. Per eliminare l'errore, e posto che comunque non useremo le funzioni ex_wpc e ex_rpb che agivano sulla periferica (visto che la periferica non c'e' più), sostituiamo XPAR_PERTEST_0_BASEADDR con il valore 0:

```
353
354  void ex_wpc (void) // write on connector C
355  {
356      volatile u32 * pbptr=(u32 *)0;
357
358      *(pbptr+1)=(in_arg_val[0]);
359
360      sprintf(rsp_string,"[OK!]");
361      return;
362  }
363
364  void ex_rpb (void) // write on connector C
365  {
366      volatile u32 * pbptr=(u32 *)0;
367
368      int dummy;
369
370      dummy=*(pbptr);
```

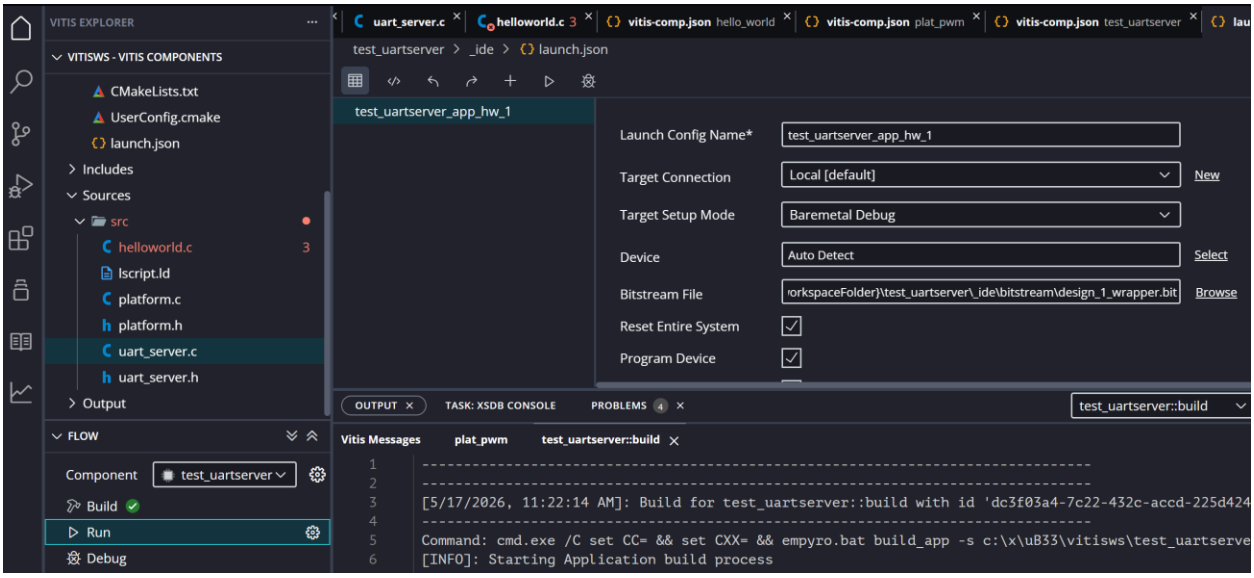
Salviamo tutto e ripetiamo il build.

A questo punto non dovrebbero esserci più errori. Se ci fossero ancora errori, assicurarsi di aver seguito “alla lettera” la procedura indicata in queste note. Si può provare a rifarla da zero cancellando la directory vitisws e sostituendola con quella di partenza che si ottiene dal file scaricabile dal sito. Se anche la seconda volta non funziona, rivolgersi al docente.

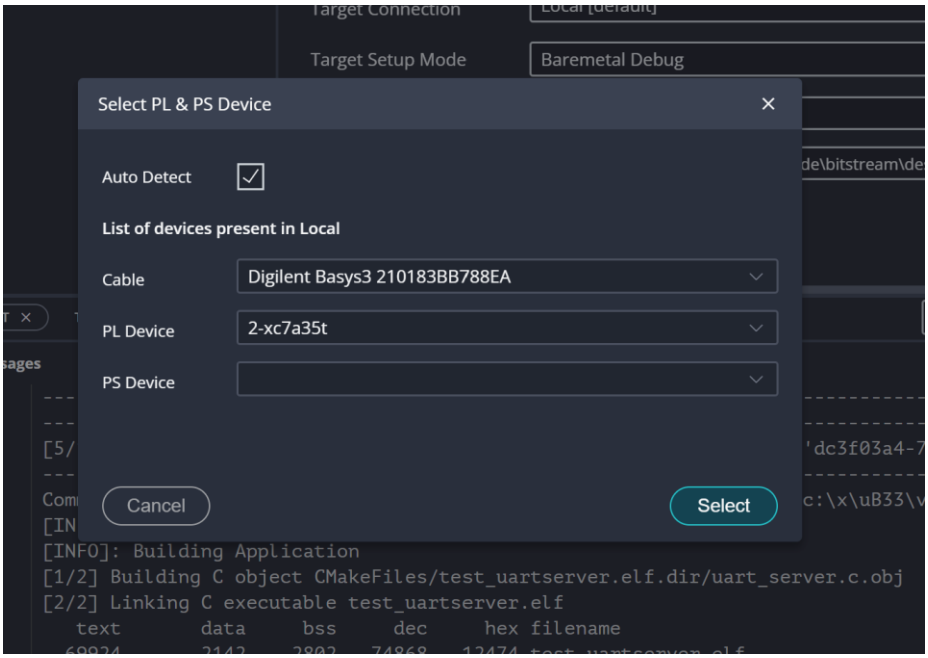
Se la compilazione è corretta, possiamo già verificare si poter configurare l’hardware e scaricare il software sulla scheda Basys3. Il Sofwtare test_uartserver contiele da definizione di comandi di test come [NOP] per verificare che la comunicazione con la scheda attraverso la seriale sia attiva.

Per procedere con la configurazione della scheda collegarla al PC e accenderla.

Stabiliamo il collegamento fra Vitis e la scheda per poter scaricare configurazione hardware e software. Fare click sull’icona di “configurazione” a fianco dell’opzione “run” nella finestra Flow come in figura:



Nel riquadro in alto a destra fare click su “select” nella riga relativa a Device. Dovrebbero comparire i dati della scheda collegata come in figura:



Fare click su select.

A questo punto dobbiamo intervenire per correggere il bug di cui si è parlato prima, con uno dei due metodi suggeriti:

VITIS > VITIS EMBEDDED DEVELOPMENT ...

digitalone (Member) asked a question.
April 2, 2024 at 7:12 PM

Workaround: Updating hardware specifications in a Vitis Unified project using XSA doesn't work.

Vitis Unified 2023.2 lwIP/FreeRTOS

Issue:
Updating hardware specifications in the Vitis Unified project using XSA doesn't work.

Reason:
Updating the platform using XSA doesn't update the bitstream file in the application.

Solution 1 (recommended):
After updating the platform using XSA, build the platform. Replace **{workspaceFolder}\app__ide\bitstream*_wrapper.bit** file with **{workspaceFolder}/platform/hw/sdt/*_wrapper.bit**.
Build the platform and project.

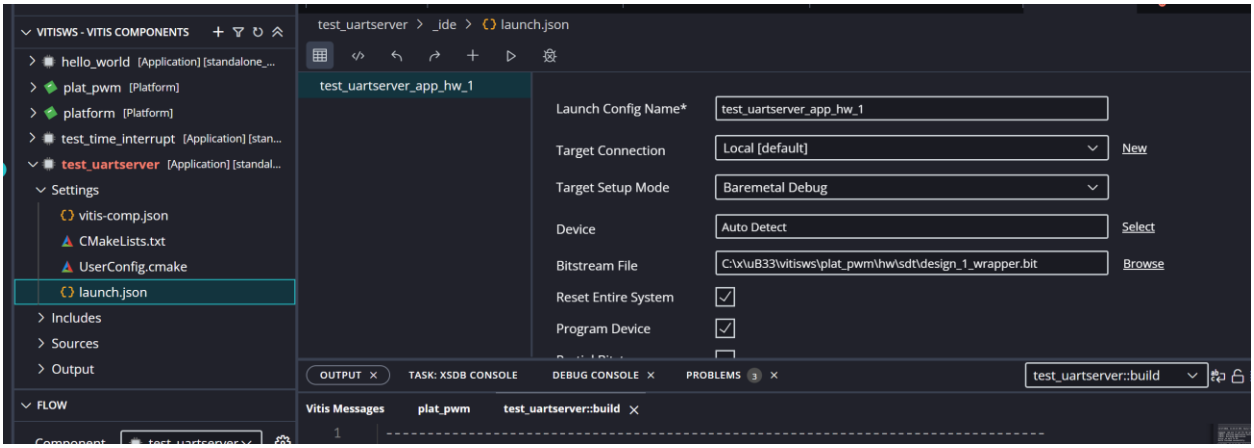
Solution 2:
Go to the "Settings" option from the application and open **launch.json**. Set the Bitstream File option to the following file.
{workspaceFolder}/platform/hw/sdt/*_wrapper.bit.

VITIS EMBEDDED DEVELOPMENT & SDK

Like Share 6 answers • 2.48K views

Usiamo la soluzione 2.

Nel riquadro in alto a sinistra espandiamo "Settings" nella gerarchia test_uartserver, facciamo click su "launch.json" e modifichiamo la riga relativa al "Bitstream file" come indicato nella "Solution2" usando ovviamente i nostri nomi.



A questo punto, facendo click su "run" nella sezione flow, la scheda viene configurata.

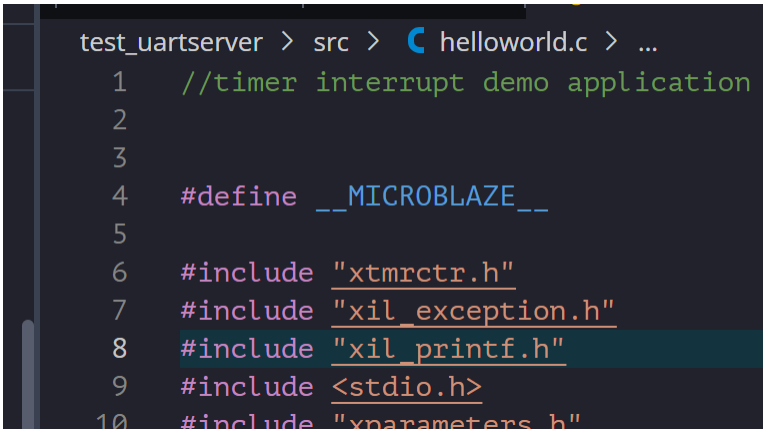
Può essere necessario fare il reset dell'intero sistema. Il reset è collegato al pulsante centrale del gruppo di 5 presenti sulla scheda.

Al reset, due display sono accesi e due spenti, consistentemente con il fatto che il pwm è inattivo e che le uscita pwm_out sono a '0' mentre le uscite pwm_out_n sono a '1'.

Nota: dopo aver configurato la scheda una prima volta ho notato un comportamento anomalo e sono risalito al fatto che la prima versione di periferica costruita risultava incorretta (mancavano i file vhdl aggiunti come parte della gerarchia). Ho dovuto ricreare da zero la periferica. Non so se il problema è stato dovuto a un errore nei vari copia e incolla che sono stati necessari per compilare queste note o a "iniziative inopportune" prese da vivado che potrebbe "essersi accorto" che stavamo generando una seconda periferica con gli stessi file della prima. In ogni caso conviene prestare attenzione a verificare gli step precedenti ogni volta che si nota che qualcosa non funziona come dovrebbe.

Ora facciamo un piccolo programma di test iniziale dei pwm e creiamo anche dei comandi per interagire con queste periferiche dall'esterno.

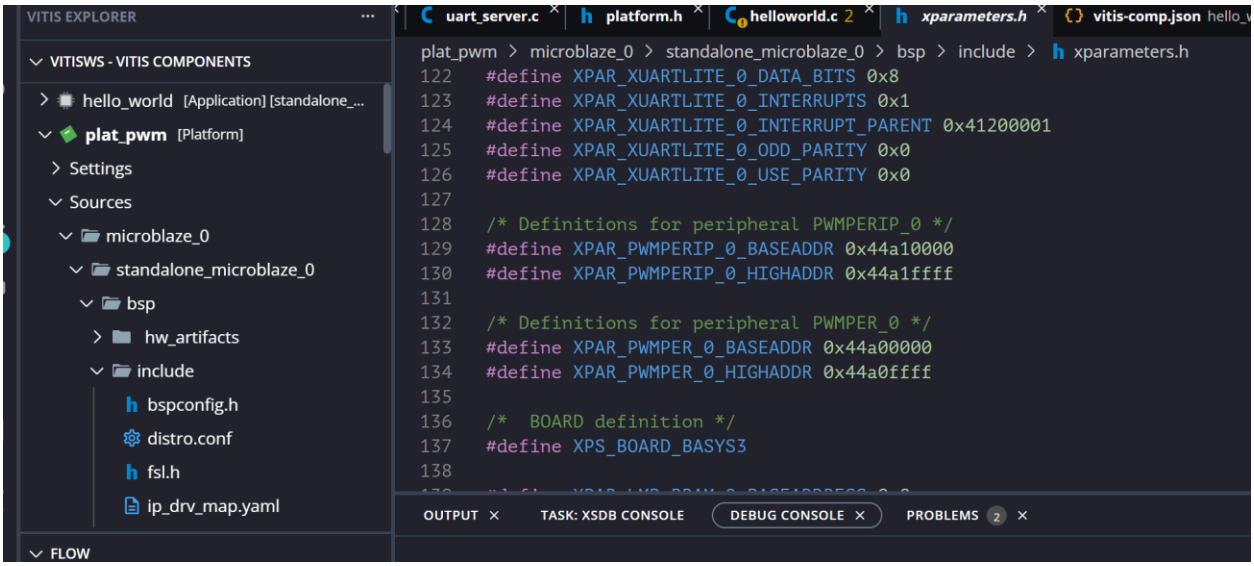
Prima però aggiungiamo la linea #define __MICROBLAZE__ all'inizio del file helloworld.c Questo consente di eliminare le segnalazioni di errore (non vere) dovute, ancora una volta, a un bug irrisolo del sistema. Nota che rimangono segnalazioni di warning (variabili dichiarate e non usate) che, volendo, possono essere facilmente eliminate esaminando il codice e intervenendo nei punti opportuni.



Per "parlare" con le periferiche dobbiamo procurarci l'indirizzo iniziale dello spazio di memoria da loro occupato.

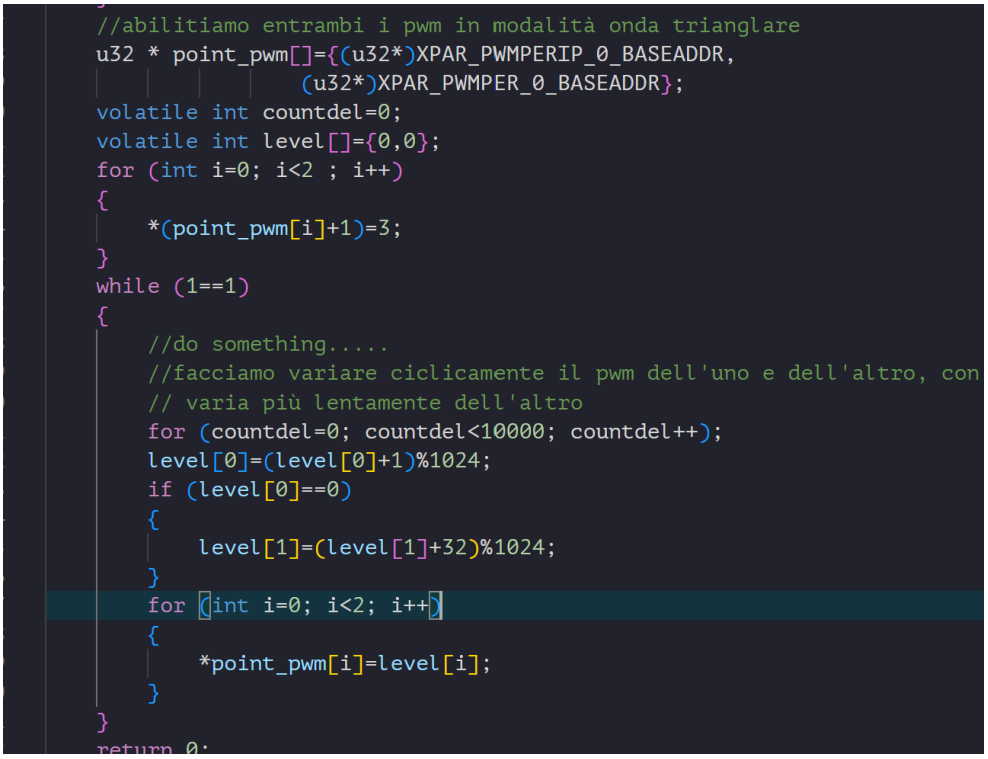
Si tratta di due periferiche distinte ma identiche dal punto di vista del funzionamento. Per questo motivo ci procuriamo i puntatori al primo registro di entrambe le periferiche e li memorizziamo in una array di puntatori nel main, così possiamo riferirci all'uno o all'altro con l'indice dell'array.

Apriamo intanto il file "xparameters.h" che troviamo seguendo la gerarchia: palt_pwm->Sources->microblaze_0->standalone_microblaze_0->bsp->include:



Come si vede ci sono due costanti che individuano gli indirizzi delle periferiche: XPAR_PWMPER_0_BASEADDR e XPAR_PWMPER_0_BASEADDR.

All'interno del main, dopo le inizializzazioni dello uart_server e della callback del timer, aggiungiamo:



```
//abilitiamo entrambi i pwm in modalità onda trianglare
u32 * point_pwm[]={(u32*)XPAR_PWMPERIP_0_BASEADDR,
                   (u32*)XPAR_PWMPER_0_BASEADDR};
volatile int countdel=0;
volatile int level[]={0,0};
for (int i=0; i<2 ; i++)
{
    *(point_pwm[i]+1)=3;
}
while (1==1)
{
    //do something.....
    //facciamo variare ciclicamente il pwm dell'uno e dell'altro, con
    // varia più lentamente dell'altro
    for (countdel=0; countdel<10000; countdel++);
    level[0]=(level[0]+1)%1024;
    if (level[0]==0)
    {
        level[1]=(level[1]+32)%1024;
    }
    for (int i=0; i<2; i++)
    {
        *point_pwm[i]=level[i];
    }
}
return 0;
```

Compiliamo il software e facciamo run. Eventualmente facciamo reset premendo il tatso centrale. Si dovrebbero osservare le intensità dei display a sette segmenti variare nel tempo.

Ora creiamo dei comandi per interagire con le periferiche. Ripristiniamo il main nella sua forma originale e spostiamo la dichiarazione di point_pwm nel file uartserver.c così che sia visibile alle funzioni all'interno di questo file. Definiamo due comandi

- Comando PWC di configurazione con due argomenti: il primo argomento è il numero di PWM (0 o 1) e il secondo argomento è il valore di configurazione di modo ed enable (valori in decimale da 0 a 3);
- Comando PWL che fissa il livello con 3 argomenti: il promo argomento è il numero di PWM (0 o 1); il secondo argomento è la parte alta e il terzo la parte bassa del valore della soglia su 10 bit (esadecimale valori da 00 00 a 03 FF).

Modifichiamo quindi il file uart_server come segue (le parti aggiunte sono cerchiate in giallo):

```
volatile u32 * point_pwm[]={(u32*)XPAR_PWMPERIP_0_BASEADDR,
                            (u32*)XPAR_PWMPER_0_BASEADDR};

void execute_cmd(void f());
void ex_nop(void);
void ex_wrp(void);
void ex_wlr(void);
void ex_rpb(void);
void ex_wpc(void);
void ex_pwc(void);
void ex_pwl(void);


struct command{
    char mnem[3];
    int num_args;
    void * fn_ptr;
};

struct command cmdlist[]={
        {"NOP",0,ex_nop}, //no operation
        {"WRP",1, ex_wrp}, //write a number of dots (".....") specifie
        {"WLR",2, ex_wlr}, //sets leds (Arg1:Arg2)&0x3FF;
        {"RPB",0,ex_rpb},
        {"WPC",1,ex_wpc},
        {"PWC",2,ex_pwc},
        {"PWL",3,ex_pwl}
```

e, per quanto riguarda la definizione delle funzioni aggiunte:

```
386
387  void ex_pwc (void) // set pwm status
388  {
389
390      *(point_pwm[in_arg_val[0]&1]+1)=in_arg_val[1];
391
392      sprintf(rsp_string,"[OK!]");
393      return;
394  }
395  void ex_pwl (void) // set pwm level
396  {
397
398      *(point_pwm[in_arg_val[0]&1])=(in_arg_val[1]&3)*256+in_arg_val[2];
399
400      sprintf(rsp_string,"[OK!]");
401      return;
402  }
403
```

Tutto il progetto uB33 modificato comprese le ultime modifiche a vitis sono scaricabili dal sito (uB33_finale).