**Corso FPGA SoC.** prima sessione, gennaio 2017.

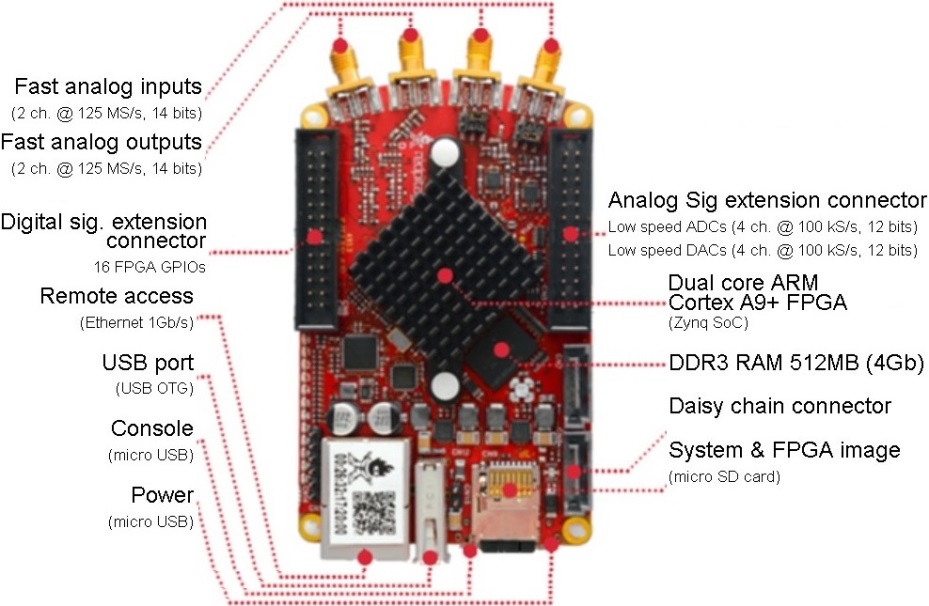
**La scheda RedPitaya.**

**RedPitaya**.

La scheda RedPitaya monta un chip ARM Cortex A9 dual core con una potente sezione FPGA basata su Zynq 7000 SoC, capace di controllare 4GB di Ram DDR3.

I circuiti della sezione FPGA controllano due canali analogici di input in grado di campionare a 125Msp con i quali si possono realizzare performanti strumenti di misura e di diagnostica anche nel campo professionale e della ricerca.

La scheda risulta simile ad un altro prodotto commerciale chiamato “parallela” che mostra più potenza di calcolo ma meno efficacia per la creazione di strumentazione perché non dispone della sezione analogia ottimizzata.



RedPitaya (anno 2016)

Entrambe RedPitaya e Parallela montano lo stesso circuito integrato Zynq processor, ma il frontend analogico è specifico della Pitaya.

Oggi, queste piattaforme costituisco il TOP dell’hardware disponibile per fare schede di sviluppo e strumentazione customizzabile.

Entrambe sono in grado di caricare un Sistema operativo, tipicamente Linux ma potrebbe trattarsi di Android in funzione dei gusti e delle capacità del programmatore.

I sistemi sono residenti in SD card e nella fase di boot vengono caricati in un’area protetta della memoria RAM allo scopo di minimizzare gli accessi ai segmenti SD che si deteriorerebbero velocemente.

Quando si accede in modifica a file o parti del sistema va tenuto presente che esistono cartelle che vanno caricate in maniera “cross” e che i programmi sono spesso sviluppati tramite dei “cros compiler” ovvero la compilazione e l’editazione avviene in una postazione PC diversa, ad esempio la workstation o il PC dell’ufficio/laboratorio e poi i file vanno inviati, in ambiente linux, nella cartella di destinazione tramite un trasferimento SCP.

Questo significa che queste schede hanno sicuramente una porta ethernet e gestiscono un servizio web (webserver).

L’accesso avviene puntando all’indirizzo IP statico configurato nella scheda.

**Programmare la sezione FPGA di RedPitaya.**

La scheda RedPitaya, nata come dispositivo di misura polifunzionale, ovvero in grado di sostituire un oscilloscopio, un generatore di funzione, un analizzatore di spettro e molto altro, con caratteristiche analoghe ai singoli strumenti “veri e professionali”, è customizzabile dall’utente finale che sia in grado di programmare in ambiente HDL.

I fabbricanti hanno allocato gli strumenti canonici, quindi disponibili senza modificare la scheda, agli indirizzi di memoria mostrati nella tabella “FPGA memory map”.

**FPGA Memory Map.**

La tabella descrive la suddivisione della sezione FPGA con i relativi indirizzi di inizio e fine in cui sono allocati gli strumenti standard accessibili tramite l’interfaccia AXI GP0 interface. Tutti i registri utilizzati per l’accesso hanno un offset di 4 byte e quindi un’ampiezza di 32 bit. Questo comporta che i dati in trasferimento da e verso le aree indicate avranno una dimensione di transfert size paria a 32 bit.

L’organizzazione del movimento del dato è di tipo **little-endian**.

L’area di memoria disponibile (specifico per RedPitaya) è divisa in 8 parti ognuna occupata da un IP core (architettura dell’applicativo utente).

Relativamente allo specifico prodotto RedPitaya, ma concettualmente applicabile a qualunque nuovo progetto che volessimo sviluppare in tecnologia FPGA (ad esempio con chip della famiglia ZYNQ) la dimensione di ogni IP core è stata assegnata di 4MByte.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Indirizzo iniziale** | **Indirizzo finale** | **Nome del modulo** |
| CS[0] | 0x40000000 | 0x400FFFFF | Riservato alla casa costruttrice |
| CS[1] | 0x40100000 | 0x401FFFFF | Oscilloscopio a due canali. |
| CS[2] | 0x40200000 | 0x402FFFFF | Arbitrary signal generator (ASG) |
| CS[3] | 0x40300000 | 0x403FFFFF | PID controller |
| CS[4] | 0x40400000 | 0x404FFFFF | Analog mixed signals (AMS) |
| CS[5] | 0x40500000 | 0x405FFFFF | Daisy chain |
| CS[6] | 0x40600000 | 0x406FFFFF | FREE |
| CS[7] | 0x40700000 | 0x407FFFFF | Power test |

Osserviamo lo slot CS[6] che inizia all’indirizzo 0x40600000 e si estende per 4Mb. Lo spazio è lasciato libero affinchè gli utenti in grado di farlo possano allocare un loro applicativo programmando in HDL.

Gli indirizzi mostrati in tabella sono utilizzabili dall’applicativo sezione ARM, tipicamente sviluppato in “C”, ad esempio tramite “Linaro tool chain” seguendo le indicazioni dello sviluppatore originale, ad esempio la sorgente di trigger dell’oscilloscopio (quello originale fornito con la scheda RedPitaya) sarà accessibile con un offset pari a 0x4 a partire dall’indirizzo base 0x40100000.

In definitiva è necessario seguire le indicazioni dello sviluppatore originale per poter utilizzare il prodotto tramite un’interfaccia customizzabile di qualsiasi tipo realizzata in qualsivoglia linguaggio.

**Loggarsi alla scheda RedPitaya.**

La scheda è gestita da un sistema operativo, disponibile online nel bazar della RedPitaya, chiamato ecosystem, di fatto una release di Linux.

Se non si è cambiato l’indirizzo di default questo è: 192.168.0.1

Nella scheda in uso è 192.168.62.13 perché è stata inserita come device autorizzato in una delle sotto reti dell’istituto RFX del cnr.

L’utente di default è: root

Password: root

Gli applicativi vanno in esecuzione grazie al programma **axi\_write** che è stato inserito nella cartella /root/prescaler in cui possiamo anche trovare il sorgente C con nome axi\_write.c

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/mman.h>

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define RP\_OK 0;

#define RP\_EMMD 1;

#define RP\_EUMD 1;

#define RP\_ECMD 1;

#define RP\_EOMD 1;

static int fd = 0;

int Init()

{

if (!fd) {

if((fd = open("/dev/mem", O\_RDWR | O\_SYNC)) == -1) { return RP\_EOMD; }

}

return RP\_OK;

}

int Release()

{

if (fd) {

if(close(fd) < 0) {

return RP\_ECMD;

}

}

return RP\_OK;

}

void \* Map(size\_t size, size\_t offset)

{

static void\* mapaddr = NULL;

if(fd == -1) { return NULL; }

if(!mapaddr)

mapaddr = mmap(NULL, size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, offset);

if(!mapaddr) { return NULL; }

return mapaddr;

}

int main(int argc, char \*\*argv) {

printf("Axi write test \n");

if(argc<3) {

printf("usage: %s address size [d1 d2 ... ]\n",argv[0]);

return 1;

}

size\_t map\_addr = strtol(argv[1],NULL,16);

size\_t map\_size = atoi(argv[2]);

int i;

printf("writing on address %d with size %d\n",map\_addr,map\_size);

// int data[20];

// for(i=3; i<argc; ++i) {

// data[i-3] = atoi(argv[i]);

// }

Init();

int \*addr = Map(map\_size, map\_addr);

for(i=0; i<argc-3; ++i) {

\*(addr+i) = atoi(argv[i+3]);

}

for(i=0; i<argc-3; ++i) {

printf("data[%d] = %d\n",i,\*(addr+i));

}

Release();

return 0;

}

Per accedere alla console è bene utilizzare i software:

* WinSCP
* PuTTY

WinSCP ci permette agevolmente di trasferire gli applicativi da manda in esecuzione mentre putti li lancia con i relativi parametri.

I programmi che possono essere eseguiti dallo ZYNQ7000 della Pitaya avranno formato “.bit” ovvero bitstream che sono l’output della compilazione dell’ambiente Vivado.

Se non si cambiano le impostazioni di base, è bene porre l’applicazione da eseguire nella cartella:

/dev/xdevcfg

Nota: Il riavvio della Pitaya potrebbe comportare la perdita del programma e la necessità di ritrasferirlo.

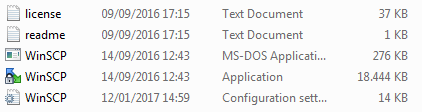
**Prima prova.**

Inserire un programma precompilato nella Pitaya e mandarlo in esecuzione.

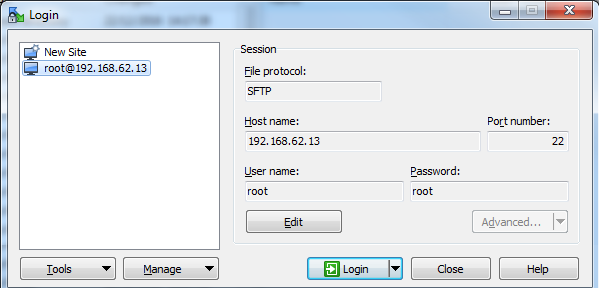
**Soluzione:** identificare la cartella dove Vivado ha generato il file .bit. Questo potrebbe figurare con il nome system\_wrapper .it

Loggarsi alla scheda usando sia WinSCP che PuTTY.

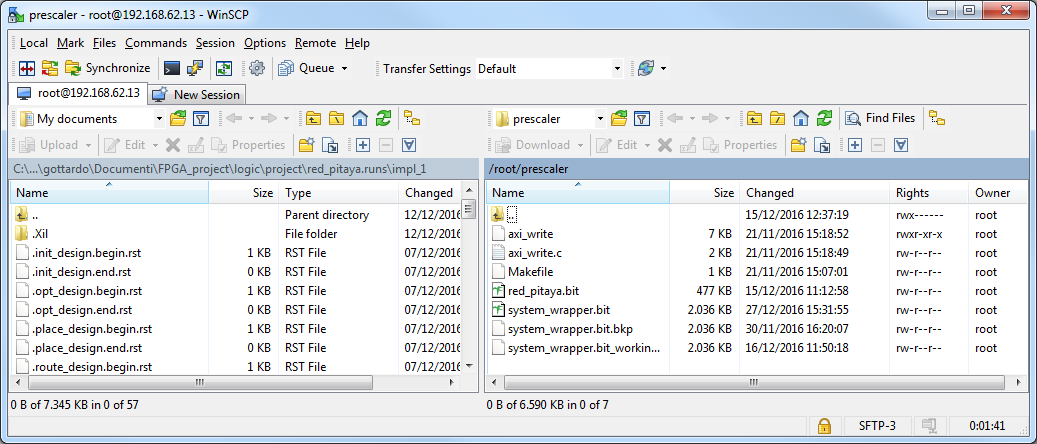
La cartella di WinSCP versione portable contiene:



Dopo avere verificato l’IP della scheda disponibile fornire queste impostazioni di accesso:



Eseguire il login, se tutto è OK allora vengono mostrati due pannelli, uno sulla macchina locale e uno sul filesystem della RedPitaya.



Nello specifico caso, che almeno nella parte iniziale dell’indirizzo dipende la PC in cui si eseguiranno le prove si avrà:

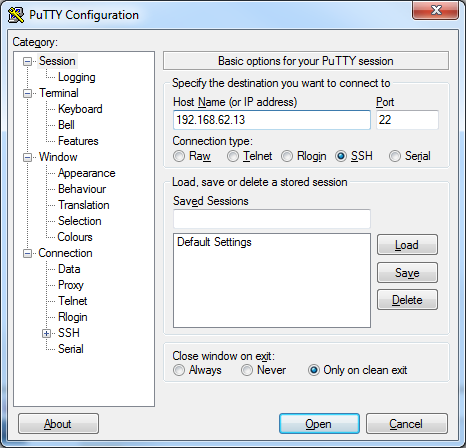
C:\NOBACKUP\gottardo\Documenti\FPGA\_project\logic\project\red\_pitaya.runs\impl\_1

Ovvero dove Vivado ha creato e manipolato il progetto, mentre all’interno della Pitaya saremo in /root/prescaler dove prescaler è la nostra applicazione che esegue un lampeggio e una divisione interna di frequenza.

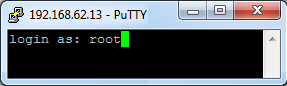
Usando WinSCP possiamo navigare agevolmente tra le cartelle del sistema operativo della scheda.

**Uso di PuTTY.**

Lanciamo Putty, allo scopo di inviare anche i parametri necessari all’applicativo quando questo viene lanciato.

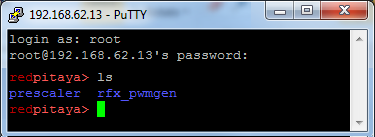


Si apre una console che attende l’inserimento della password per fare il login:



Log:root pass:root

La console acetterà i comandi linux. Come prima prova digitiamo “ls” per vedere in che direttorio ci troviamo.



In blu sono indicate le cartelle.

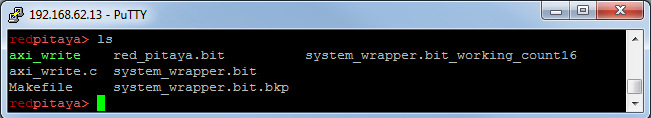
Con “CD” possiamo entrare ad esempio in “prescaler”.

rescaler rfx\_pwmgen

redpitaya> cd prescaler/

redpitaya>

digitiamo nuovamente “ls” per vedere il contenuto del direttorio.



Notiamo la presenza del file “system\_wrapper.bit” che fa le veci dell’eseguibile.

Mandiamo in esecuzione l’applicazione usando il programma axi\_write.

Da console digitiamo il comando:

./axi\_write 0x43c00000 1 300000

I parametri hanno il seguente significato, atteso dal programma axi\_write:

1. 0x43c00000 = Access point ovvero punto di allocazione dell’applicativo nella memoria dello ZYNQ
2. 1 = numero dei registri impegnati per la parametrizzazione
3. 300000 = parametro specifico per l’applicazione. In questo caso lo utilizzerà per generare uan frequenza di lampeggio su uno dei LED della Pitaya.

Proviamo ad aumentare questo valore e a osservare l’effetto nel LED della scheda.

Prima prova -> aggiungiamo 2 zeri.

Seconda prova -> aggiungiamo 3 zeri.

3/10

FPGA ready to go !!!!  
E' finalmente disponibile la prima edizione del libro "First step on FPGA", basato sull'hardware ZYNQ 7020, prodotto da Xilinx.  
Questi potentissimi chip integrano una estesa sezione FPGA con una potente sezione ARM dual core della generazione Cortex A9.  
Praticamente un mostro di potenza in pochi millimetri quadrati di silicio.  
Per focalizzare l'argomento il processor ARM multicore sono quelli con cui sono costruiti gli smart phone mentre la sezione FPGA costituisce il frontend dei più potenti ed evoluti sistemi elettromedicali o strumenti di diagnostica, acquisizione dati, strumenti di misura in genere e non solo.  
Normalmente lo ZYNQ trova impego nei sistemi SoC (System on Chip,) ovvero che al boot carica il sistema operativo.  
Vedremo come è possibile interfacciarsi alle schede SD in cui sia presente un kernel Linux e di conseguenza come iteragire con le RAM DDR3 di ultima generazione.   
Il libro è destinato a crescere in base alle esperienze che verranno accumulate durante le prime edizioni del corso, di certo unico nel suo genere.   
La prima edizione del corso potrà essere attivata a novembre, e saranno ammessi solo coloro che come prerequisiti risultino in possesso della laurea in ingegneria elettronica oppure informatica. Come propedeuticità è bene possedere l'attestato rilasciato al mio corso di programmazione dei microcontrollori PIC.  
Gli interessati posso già prenotarsi. Il corso accetterà al massimo 7 allievi indifferentemente italiani o stranieri. In caso di necessità il corso potrà essere impartito in lingua inglese.

11/11

what is a IP?  
cos'è un IP? Sono i componneti principali di cui si compone un programma, nella sessione block design, dell'ambiente Vivado.  
Letteralmente tradotto in "Intellectual Property" abbreviato "IP".  
Esistono IP di libreria e ovviamente quelli programmati dall'utente.  
La visualizzazione grafica, maschera un linguaggio più a basso livello, di tipo classico a riga di comando, composto di costrutti interconessi secondo una sintassi che può essere:  
1) HDL (hardware description language), lo standard.  
2) VHDL (VHSIC Hardware Description Language, dove VHSIC è la sigla di Very High Speed Integrated Circuits).  
3) Verilog, un po datato ma più intuitivo e di rapida applicazione.  
La struttura dell'hardware design ha origine nel TOP model, che il programmatore dovrà scegliere come passo iniziale, probabilmente dalle librerie standard. Nel nostro caso il processore ZYNQ 7000, che concettualmente diventerà il top model a cui agganciare gli altri IP.   
Fortunatamente, l'ambiente Vivado, mette a disposizione il tool "Run Connection Automation" che implementa automaticamente l'interconessione degli IP, anche via interfaccia AXI una volta che questa sia stata definita nel TOP model.   
Benchè la struttura a blocchi possa sembrare un'insieme di circuiti separati interconessi, si tratat in realtà di implementazioni interne al Chip allo stesso livello di astrazione e le uniche conessioni verso il mondo estreno verranno rappresentate come delle "frecce" di rinvio, verso la DDR o i PORT.   
Osserviamo attentamente l'immangine postata per poi discuterne assieme fissando i concetti base e acquisendo i successivi.

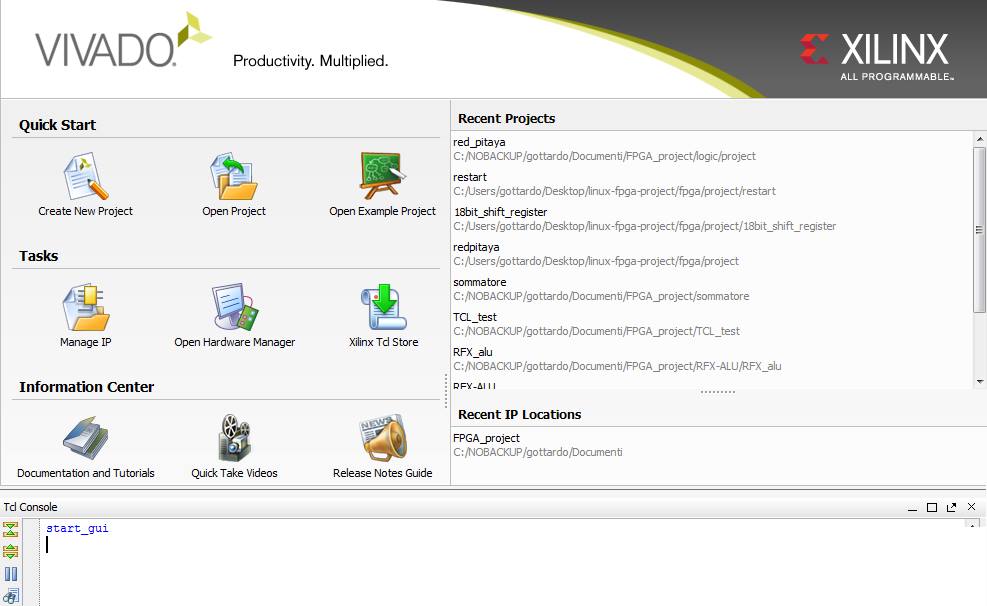


11/11

Eccoci giunti al primo passo da fare assieme nell'ambito della programmazione delle FPGA. Presto faremo la prima edizione del corso, presumibilmente a gennaio 2017.  
1) Create un account personale sul sito della Xilinx. Questo passaggio è fondamentale perchè vi permette di scaricare non solo il software ma anche le licenze gratuite (e non) che vorrete utilizzare.  
2) La versione che useremo nel gruppo è la 2015.4, molto completa, molto pesante, molto performante. Occhio...esistono molte versioni, ad esempio la 2016, ma avrete problemi di compatibilità quindi restate sulla 2015.4, dato che ci ho anche scritto il libro.  
3) Tra i pacchetti, selezionate anche il HLS (High Level Sinthesis) serve per implementare algoritmi complessi, ad esempio in C, come ad esempio le OpenCV, all'interno della FPGA. Praticamemte impossibili da implementare tramite HDL (Hardware Description Lenguage).  
4) Ricordatevi di installare il pacchetto SoC (System On Chip), che sarà riferito alla serie 7000 dei processori ZYNQ. Questi integrano una potente sezione ARM Dual Core, 700MHz, Cortex A9 con una estesa sezione FPGA (Field Gate Array). La comuniczione tra le due sezioni avviene internamente tramite un bus dedicato di derivazione AMBA di cui parleremo al corso e nei post successivi. Solo questo protocollo AXI, può accedere via FPGA alla memoria DDR3.

Cliccando sull'icona nel desktop vi comparità il control panel , visibile nell'immagine postata. Questo è diviso in tre sezioni, la prima per il Quick start (crea e manipola un progetto FPGA), la seconda che gestisce i task, ad esempio per la manipolazione degli "IP" che cono gli elementi fonfamentali della sintesi della rete FPGA e la comunicazione con le schede demoboard per il deploy del progetto, e l'information center in cui c'è un'ampia documentazione che risulta esere dispersiva se non si è accompagnati nei primi passi.  
Come visibile nell'immagine io lavoro in una carteòòa all'interno di documenti che ho chimato FPGA\_project, e consiglio a tutti di fare altrettanto.

Vorrei fondare un gruppo di ricerca e sviluppo privato...sotto forma di associazione di ingegneri, con sede a Padova, nei locali della G-Tronic Robotics Didactis division, Via Austria 19b, con possibilità di aderire anche in teleconferenza e telelavoro.  
Chi sarebbe interessato?  
Presto definirò le date della prima sessione del corso, e i requisiti per aderire e tesserarsi a questa nuova associazione.  
Il libro di testo "First step on FPGA Xilinx, introduzione alla progettazione dei sistemi SoC" è già online, e disponibile al link <http://www.lulu.com/>…/first…/paperback/product-22886982.html



15/11

Gli struementi di simulazione di Vivado.  
All'interno della piattaforma Vivado, la Xilinx ha integrato un potente tool di simulazione del codice VHDL che si realizzerà nella sezione block design.  
Le variabili visibili sono quelle definite nello specifico IP (intellectual property, vedi post precedenti), ovvero i pin di input output come anche le variabili interne.  
Il concetto di base è che i blocchi IP vengono simulati uno alla volta quindi nella sezione "Simulation" bisognerà fare in modo che l'IP sotto test venga considerato dal simulatore (e solo dal simulatore, non dal block design) come TOP level.  
Noteremo che l'asse temporale potrà essere tarato fino ai pico secondi, questo grazie all'estremamente alta velocità in cui sono in grado di operare le FPGA.  
Nell'immagine postata l'asse è stato rallentato all'ordine dei nano secondi (che può apparire comunque molto rapido).  
Le variabili che compariranno saranno suddivise in PORT e signals. A queste si aggiungeranno le eventuali variabili interne.  
La sintassi VHDL, dell'IP realizzato, che implementa un prescaler, ovvero un divisore continuo di frequenza, ha la seguente implementazione:

-- Company: RFX cnr (istituto gas ionizzati Padova)  
-- Engineer: Marco Gottardo   
-- Create Date: 11/09/2016 02:40:31 PM

library IEEE;  
use IEEE.STD\_LOGIC\_1164.ALL;  
use IEEE.numeric\_std.ALL;

entity Prescaler\_code is  
Port ( clk : in STD\_LOGIC;   
prescaler\_out : out STD\_LOGIC;  
divider : in STD\_LOGIC\_VECTOR (15 downto 0) := x"FFFF";  
prescaler\_output : out STD\_LOGIC  
);  
end Prescaler\_code;

architecture Behavioral of Prescaler\_code is  
signal counter : unsigned (15 downto 0) := (others => '0');  
signal internal\_counter : unsigned (31 downto 0) := (others => '0');  
signal set\_togle : unsigned (31 downto 0) := (others => '0');

begin  
set\_togle <= to\_unsigned(5000, 32);

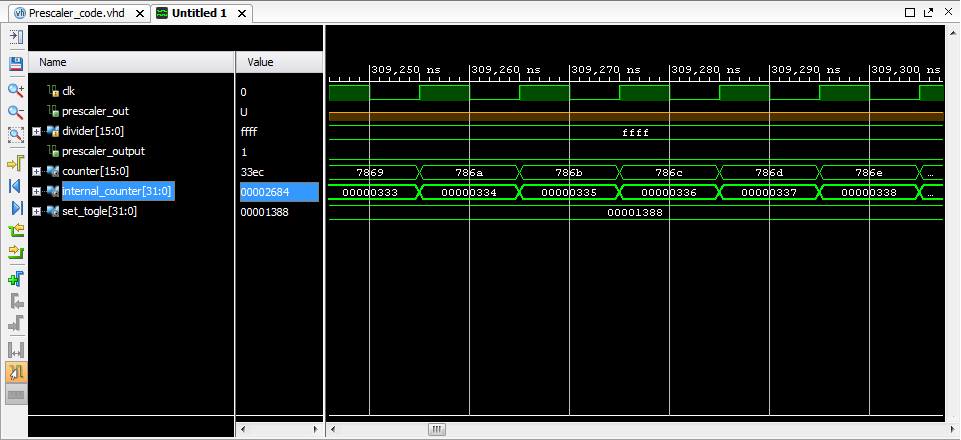
process(clk,internal\_counter)  
begin  
if rising\_edge(clk) then  
counter <= counter + 1;  
end if;

if rising\_edge(clk) then  
internal\_counter <= internal\_counter +1;  
if internal\_counter > set\_togle then  
prescaler\_output := '1';   
end if;  
if internal\_counter > set\_togle \* 2 then  
prescaler\_output <= '0';   
internal\_counter <= to\_unsigned(0, 32);  
end if;  
end if;

end process;

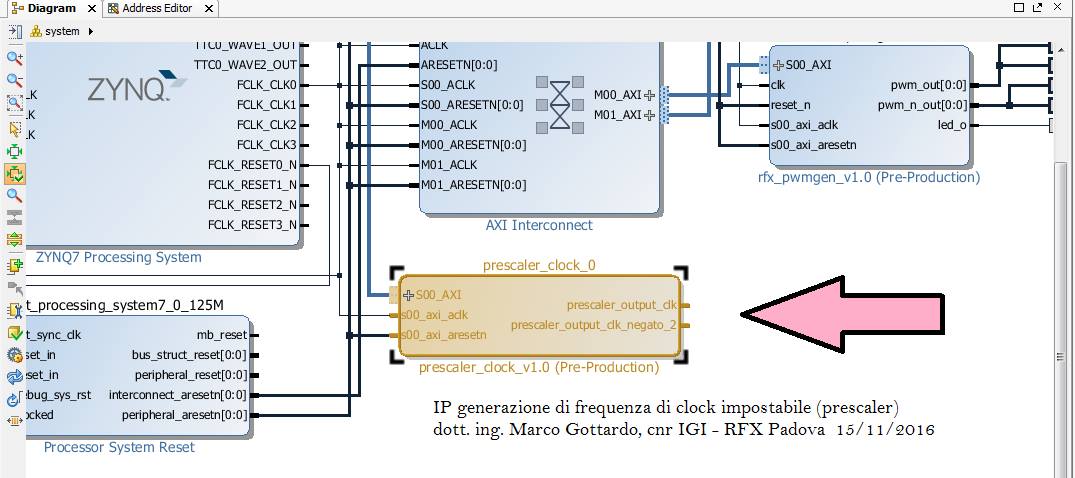
15/11

Strumenti di simulazione: IP prescaler che genera un clock di frequenza desiderata.



15/11

Oggi realizziamo un IP (intellectual property), in linguaggio VHDL nella piattaforma Vivado di Xilinx, per i potenti device della famiglia ZYNQ7000. Ricordo che questi chip integrano una sezione PL (programmable logic, ovvero un'enorme area FPGA) con una PS (processor system, che nella versione anche base del chip, ovvero il ZYNQ7000 a 400 pin BGA, è costituita da due core ARM di cernerazione CORTEX A9 csacuono a 700MHz, in grado di interfacciarsi alla DDR3 di ultima generazione e di caricare da una SD un kernel di derivazione ad esempio Linux o Android.  
Questi Chip sono il futuro (in realtà attuale) della robotica applicata.  
Il primo corso di programmazione di FPGA sarà attivato a gennaio 2017, gli interessati devono verificare di avere i requisiti necessari per accedervi.  
Vi ricordo che le nozioni di queste post le trovate approfondite nel mio libro Primi passi con le FPGA, in vendita solo su [http://www.lulu.com/…/first…/paperback/product-22886982.html](http://www.lulu.com/shop/marco-gottardo/first-step-on-fpga-xilinx-introduzione-alla-progettazione-dei-sistemi-soc/paperback/product-22886982.html)



16/11

Let's see some basic concepts about IP in the FPGA programming.  
As seen in previous posts IP are represented as graphic blocks in the block design section. The state on pin gate or internal behaviors are modifiable in the VHDL code.  
The code structure is divided into standard sections.  
1) The entity (la sua struttura hardware)  
2) The behavior (comportamento)  
The behavior is decribe by the process.  
The processes are called as instances of the IP block, and constitute the current result of the internal described network.  
suppose we want to implement a simple flip-flop within the FPGA network.

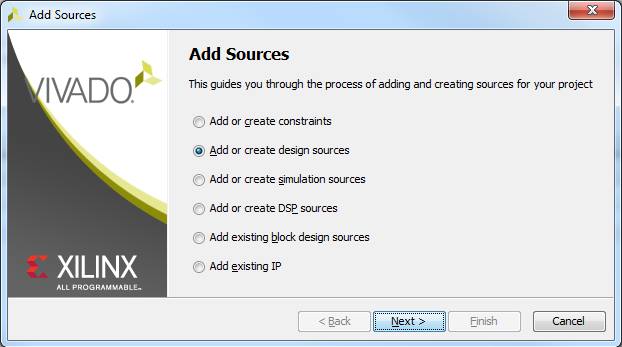
entity Flip\_Flop is  
port (S,R,clk :in bit; q, q\_not : out bit);  
end Flip\_Flop;

architecture basic of Flip\_Flop is  
begin  
flipflop\_behavior : process is  
begin  
waite until clk ='1';  
q <= S after 2ns;  
if rising\_edge(R) then  
q <= '0';  
end if;   
end process flipflop\_behavior  
end architecture basic;

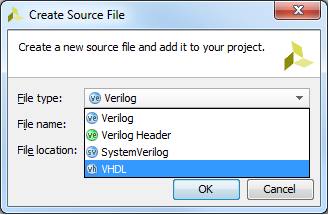
To add a new IP block (per inserire il nuovo blocco IP), click on "add surce" in the Project manager section of your Vivado istallation. Be scure to had select the top model IP, before act.   
It appear a box, like show in posted picture. So insertin the editor the VHDL here over. so repackage IP...and run automatic connection on the block design editor.  
You have implemented your first FPGA program.

Commenti:

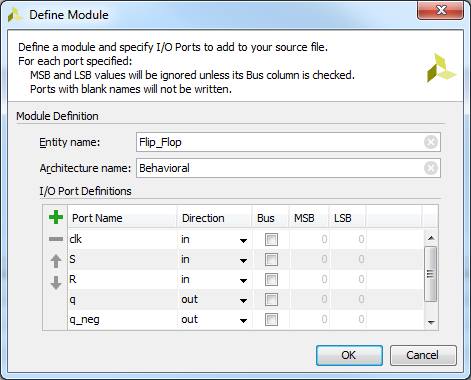
* Add source -> permette di introdurre il codice VHDL per un nuovo IP in Vivado.



* Attenzione. Warning. There are three main languages in which implement IP blocking. 1) Verilog (il più vecchio, ma ben documentato e con materiale disponibile in bibliografia. 2) HDL (generico) 3) VHDL il più potente e performante. Noi useremo questo.

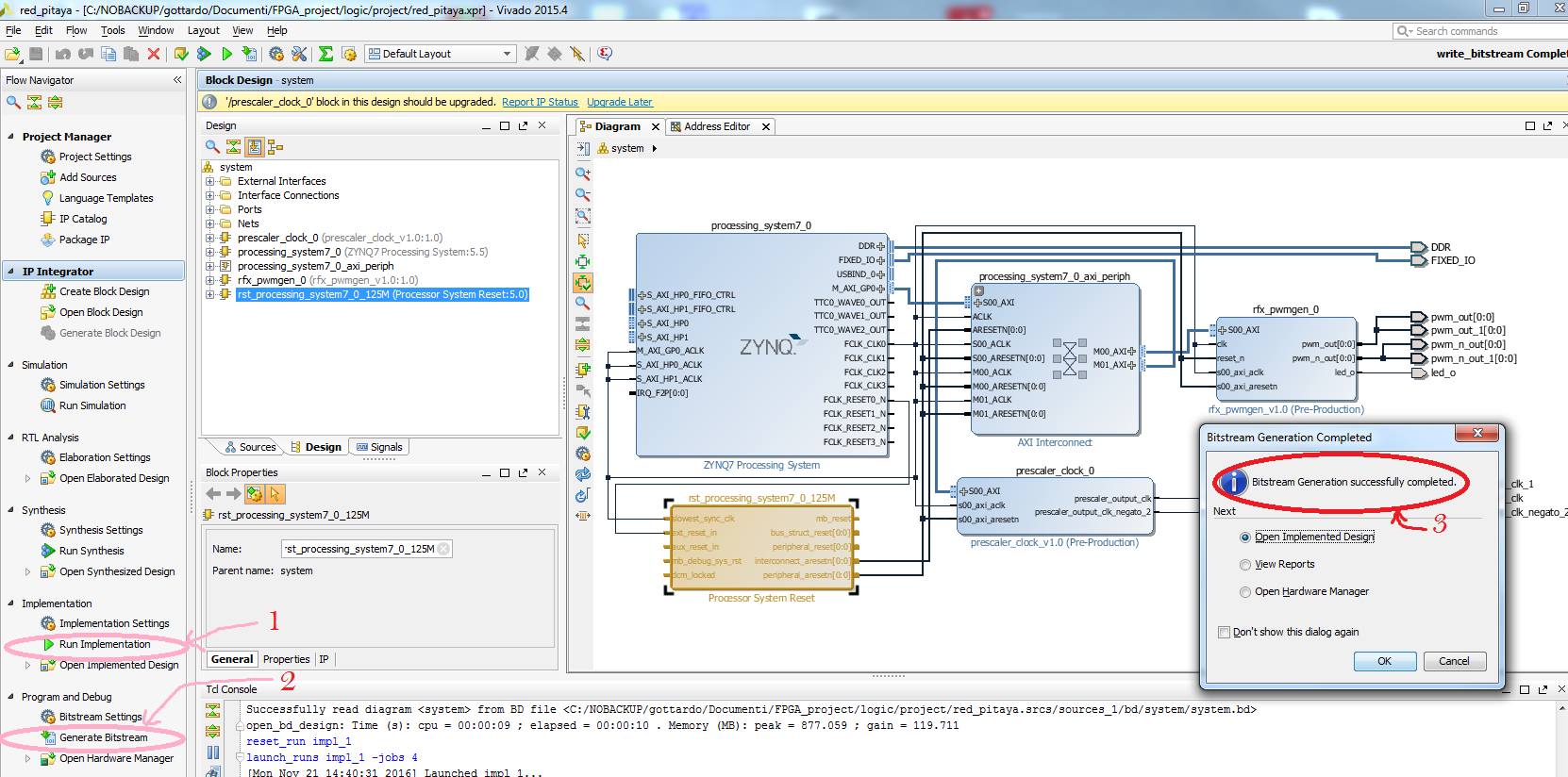


* Nulla vieta di usare un IP svritto in Verilog all'interno di un design scritto in VHDL perchè gli IP una volta chiusi, mostrano alla rete il solo behavior (comportamento) tramite le porte. Ciò che accomuna i vari linguaggi saranno le compatibilità tra le tensioni hardware e i protocolli di comunicazione. Userme AXI4 oppure la versione light di AXI 4.
* La creazione guidata del blocco IP tramite il menù "add surce" ci permette di definire graficamente i pin di ingresso e uscita del nuovo IP, ovvero crea la sezione entity del programma VHDL: entity Flip\_Flop is  
  port (S,R,clk :in bit; q, q\_not : out bit);  
  end Flip\_Flop;



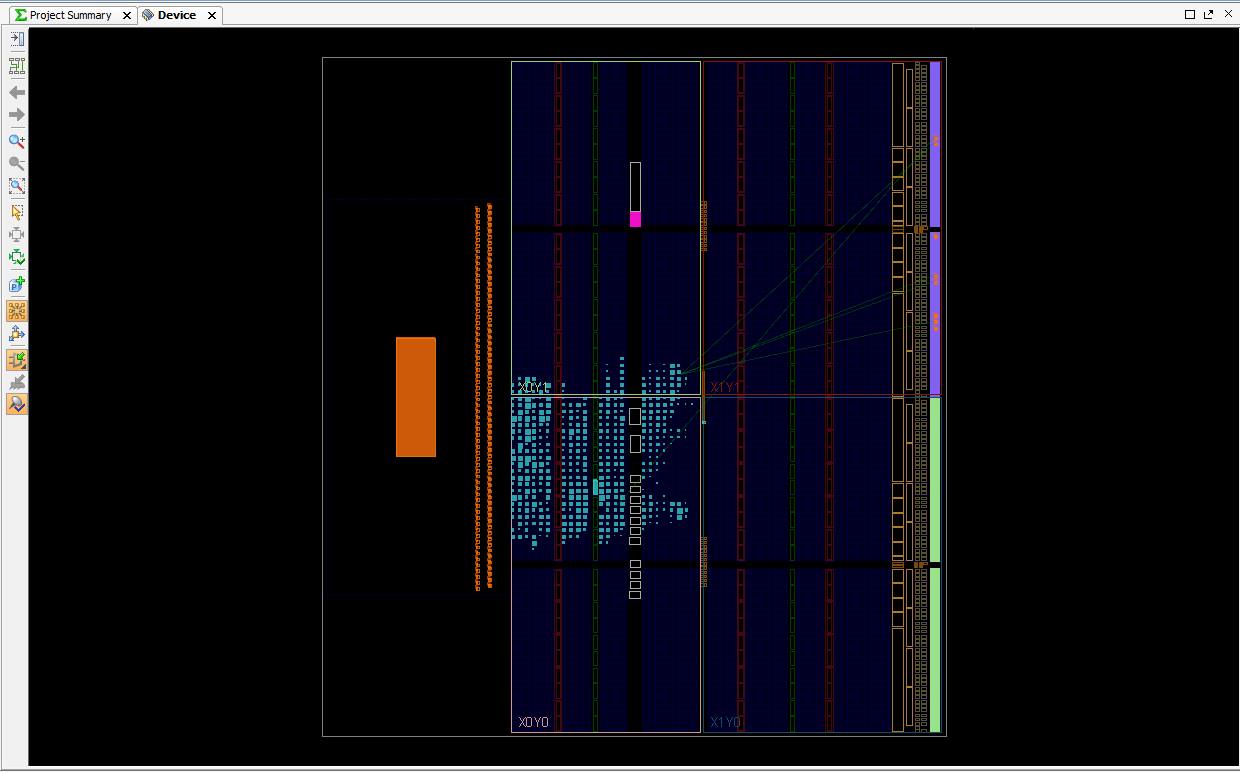
21/11

il formato riconosciuto dalla FPGA come caricabile è il .bit ovvero il bitstream. Per generare il bitstream, e quindio eseguire il deploy come sintesi della logica FPGA, bisogna che siaono stati eseguiti con successo tutti i passaggi precedenti.  
Creazione del progetto -> integrazione dei nuovi IP e relativo "run automation che si cura della conessione dell'AXI interno, -> simulation allo scopo di vedere se il behavior (comportamento della rete) è quello atteso, -> RTL analysis (mostra la superficie di silicio impegnata), -> Syntesis (crea i file finali da assemblare nel bistream) -> Generate bitstrean (passaggio finale).  
Ora siamo pronti per il deploy.  
Nell'immagine postata vediamo, sul lato sinistro, i 7 passaggi in cui gli ultimi due creano il bitstream. Sul lato destro il sistema risponde dicendo che il file di bitstream è stato generato con successo.

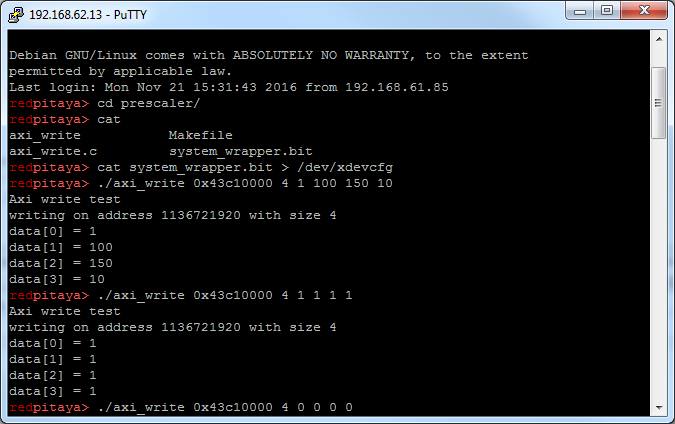


Commenti:

* Come viene mostrato l'implementation design, ovvero la nuova rete logica generata dal nostro programma sulla superficie interna dell'FPGA.

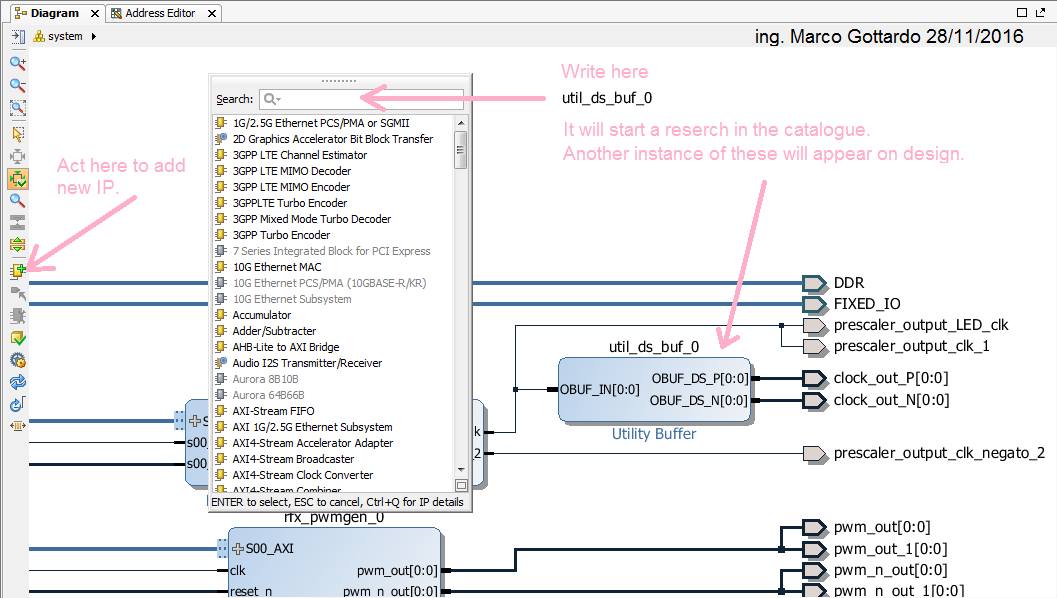


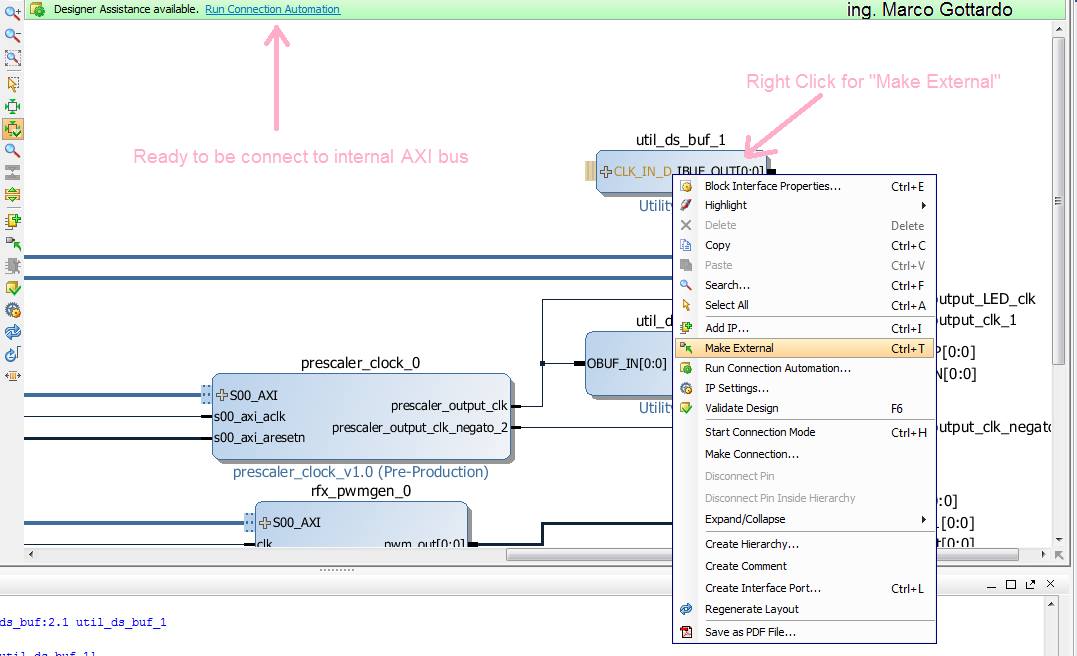
* è possibile zoomare a un livello molto spinto e visualizzare le singole LUT ovvero i quadratini azzurri. Trovate la teaoria sul libro first step on FPGA programming disponibile su [lulu.com](http://l.facebook.com/l.php?u=http%3A%2F%2Flulu.com%2F&h=TAQFkIC6M) ...immagino che per molti sial la prima volta che possono vedere una configurazione interna di chip a livello di silicio, customizzata su un vostro programma.
* Per caricare il programma nella FPGA (operazione di deploy) muniamoci di una shell SSH e logghiamoci come root al device SoC in cui sia in esecuzione un kernel Linux. Nel caso della Pitaya, in cui è montato uno Zynq7000, con 4 Giga di RAM DDR3 entreremo tramite interfaccia ethernet. L'indirizzo del mio esemplare è statico e accessibile dalla LAN come 192.168.62.13

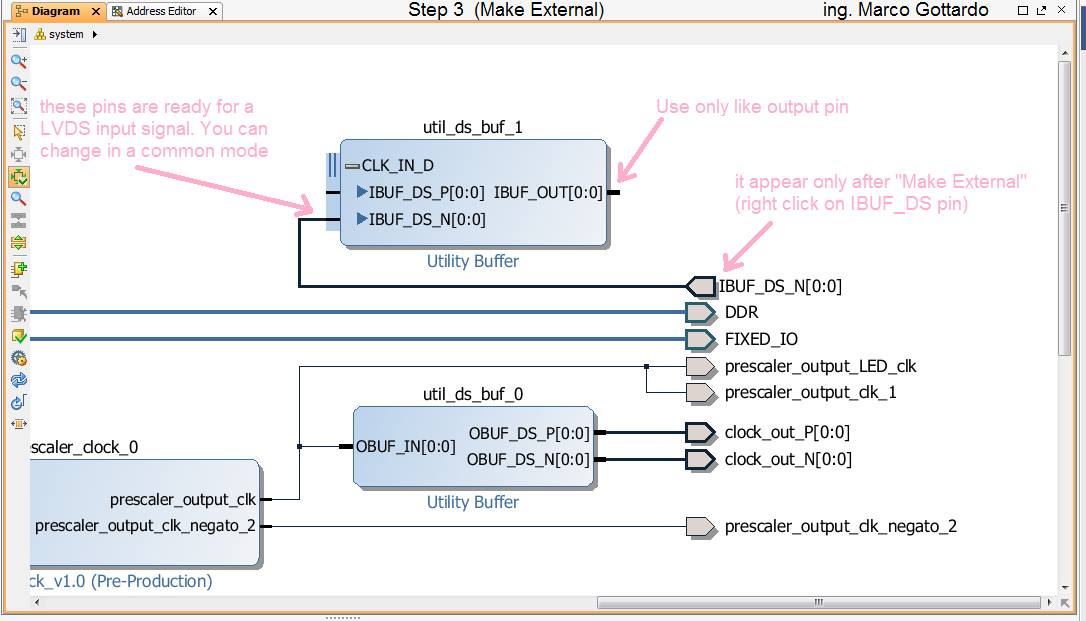


29/11

Creare una porta esterna per lo ZYNQ7000 series.  
Che il vostro design sia etremamente semplice o complesso prima o poi dovrete fare comunicare l'architettura interna con il mondo esterno.  
Ci sono due macrofamiglie di connessioni:  
1) Quelle connesse alla sezione PL (programmable logic, ovvero ai PORT della sezione FPGA.  
2) Quelle connesse a alla sezione PS (processor system, ovvero ai CORE ARM.  
Se si sta progettando un elettromedicale o uno strumento che deve acquisire i segnali ad alta velocità ci si dovrà collegare alla sezione PL.  
Laciamo la piattaforma Vivado ed apriamo il design a cui vogliamo aggiungere un collegamento esterno, in input oppur in output.  
Per semplicità ci riferiamo a un I/O digitale.  
Dovremmo indicare la tensione a cui opera, ad esempio 3v3 oppure 2v5, e la modalità ad esempio common mode oppure LVDS ovvero differenziale (con le eventuali varianti).  
Nella colonna del Flow Navigator agiamo su "Open Block Design".  
Agiamo sul tasto Add\_IP presente sul bordo dell'area di editor (vedi la prima immagine), il nuovo IP viene prelevato dalla libreria degli oggetti compatibili al design iniziato.  
Comparirà in una posizione riaggiustabile per trascinamento usando il mouse. Agiamo con il tasto destro -> Make External. Questo farà comparire, anche se scollegata, una nuova porta di tipo neutro, che andremo a configurare. (vedi la seconda immagine). Il pin compare solo se puntiamo con il mouse sul pin indicato del blocchetto.  
Ora collegheremo l'ingresso di clock e successivamente l'interfaccia AXI tramite run Automation.   
Si conclude modificando nei contraints il pin fisico di IO e sistemando nei surces il codice VHDL.



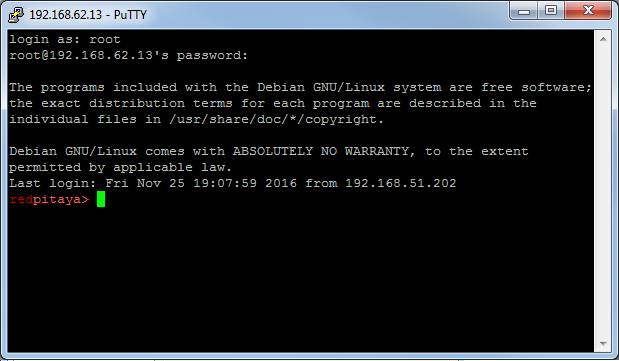


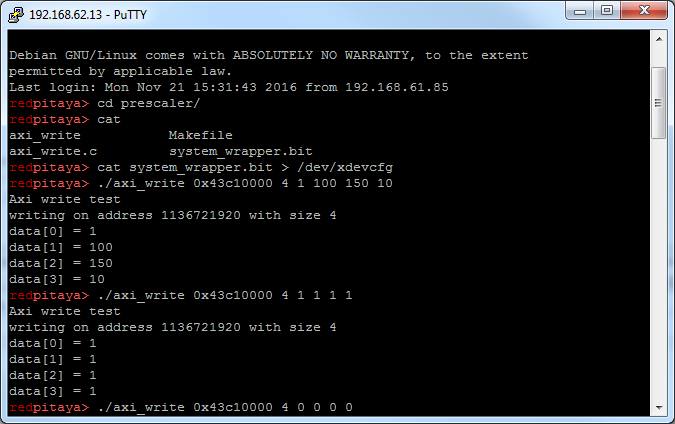


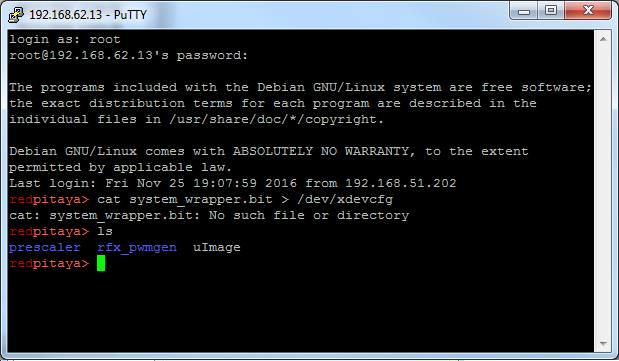
29/11

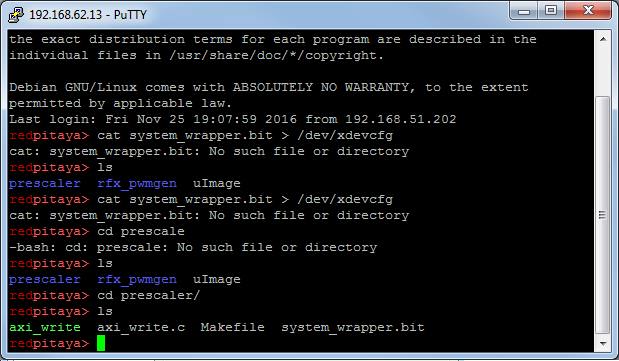
Caricare il programma su un sistema SoC (system on chip).  
Per eseguire un programma sviluppato sul nostro terminale all'interno di un sistema munito di OS, come ad esempio un raspberry o una redpitaya è necessario loggarsi a questo tramite una shell SSH e disporre delle credenziali dell'acount amministrativo. Supponiamo di avere sviluppato un programma in VHDL per per la sezione FPGA che implementa il frontend di un nuovo strumento di acquisizione dati realizzato per REDPITAYA. Il chip centrale è lo ZYNQ7000, munito di dual core ARM Cortex A9, entrambi i core a 700MHz, con a bordo da 1 a 4 GByte di DDR3, e un'ampia sezione FPGA di Xilinx, interconessa ai core tramite bus AXI (evoluzione dell'AMBA). Nella SSD da 8 giga, in dotazione, è residente un sistema operativo, di derivaione Linux, chiamato ecosystem. La bash accetta tutto il set standard Linux. Ci si può loggare alla Pittaya da windows usando il tool WinSCP, che integra anche una console PUTTY.  
Sul lato target, ovvero la folder corretta della Pittaya, è in esecuzione il software di gestione dell'AXI, di cui posto il sorgente.  
impostiamo un IP fisso sul lato target, nel mio caso 192.168.62.13 e riavviamo il sistema in modo che siano effettive le nuove impostazioni dell'eth0.  
Avviamo winscp e impostiamo nella schermata iniziale, protocollo SFTP, host name 192.168.62.13, porta 22, username root e passwod la vostra password (quella di fabbrica è root)   
Aperta la sessione WinScp, seguiamo i passaggi mostrati nelle immagini. comparirà la cartella progetto sul lato sinistro (lato PC) e la cartella di destinazione sul pannello detro (lato pitaya). Sovrascriviamo per trascinamento il nuovo bitstream nell'immagine compilato come systema\_wrapper.bit  
Ora il programma è dentro al sistema, ma deve essere mandato in esecuzione, per questo motivo ci serve un terminale putty con cui imputare i dati di istanza dell'applicativo, ma sopratutto l'offset di memoria (che ha l'aspetto di un registro), da cui lanciare l'esecuzione.  
La sessione di Putty potrebbe richiedere la connessione, (vedere immagine).  
Una volta stabilita la conessione richiede di loggarsi fornendo le credenziali (quelle di root).  
Siamo collegati quando il prompt risponde con "redpitaya>" e si mette in attesa di comandi.  
Spostiamoci nella cartella della scheda target in cui abbiamo trasferito l'applicativo. con "ls" vediamo dove siamo... con "cd nome cartella" ci spostiamo nei direttori, con "CD .." torniamo indietro di un livello tra i direttori.  
I direttori sono mostrati in blu mentre i file in bianco. linux mostra gli eventuali eseguibili in verde, qui vedremo dopo averlo trasferito solo axi\_write.  
Impostiamo l'allocazione del device con:  
cat system\_wrapper.bit > /dev/xdevcfg  
con "./axi\_write 0x43c0000 1 1" mandiamo in esecuzione il nostro applicativo.   
0x43c0000 è l'offset di allocazione del nostro applicativo, questo indirizzo ci viene dato in fase di sviluppo dall'ambiente Vivado.  
i due "1" sono altri registri settati nel programma.  
Per come è stato fatto lo specifico bitstream che stiamo caricando l'ultimo parametro corrisponde alla frequenza di oscillazione del led che faciamo lampeggiare.  
Dando invio il programma viene eseguito all'interno del nostro sistema SoC e l'area FPGA è stata rimappata.  
Nozioni piu approfondite si trovano sul libro "first step on FPGA programming" in vendita su lulu com, scritto da Marco Gottardo adottato sui corsi di FPGA previsti a gennaio a Padova. chi è interessato prend contatto anche tramite facebook.

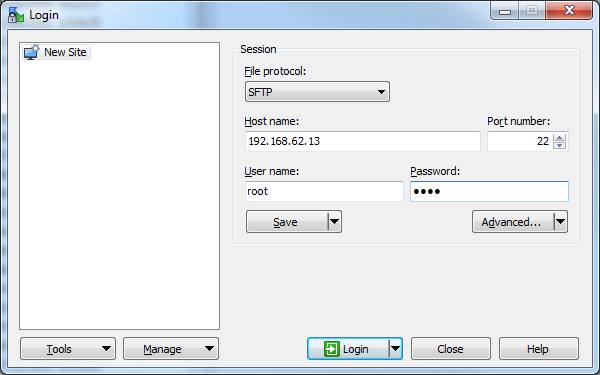


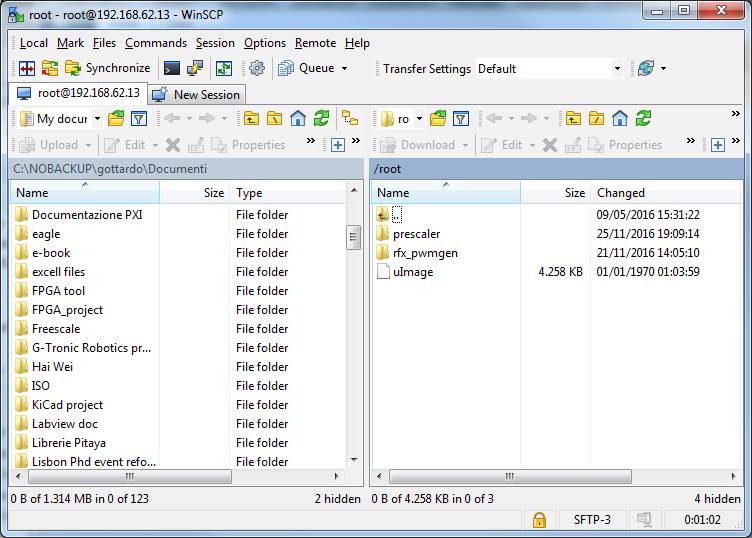


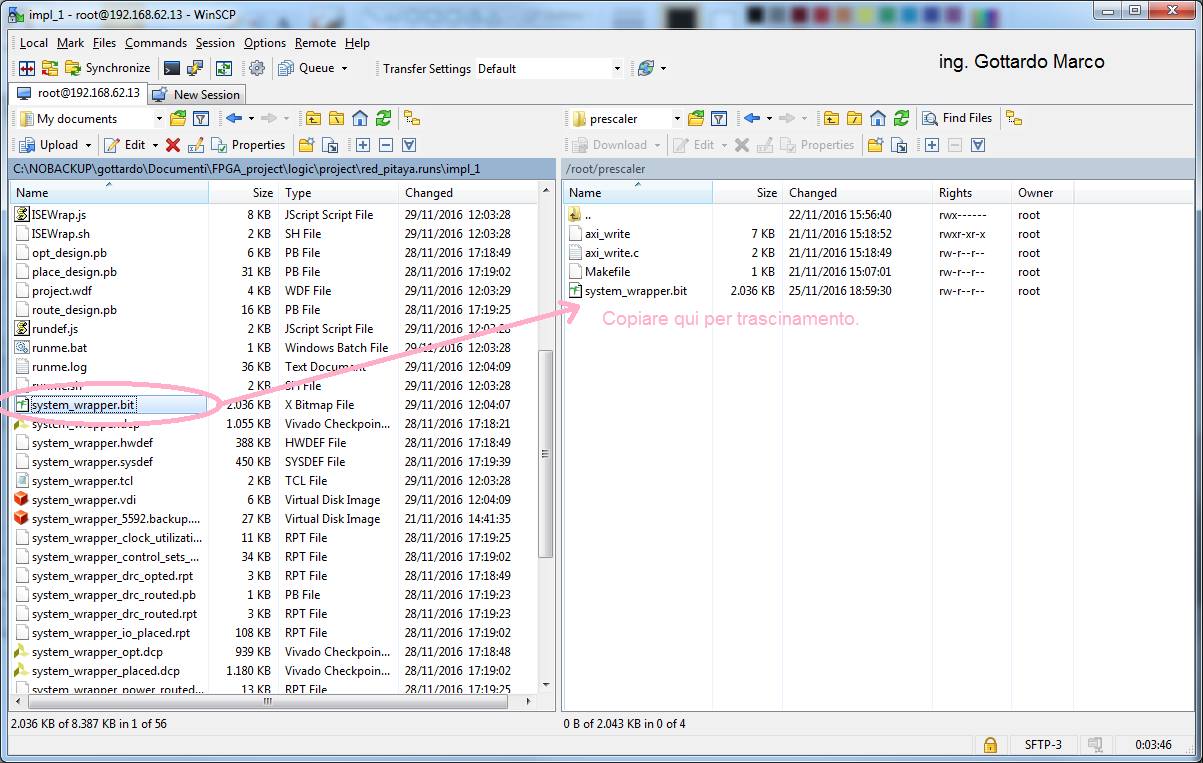


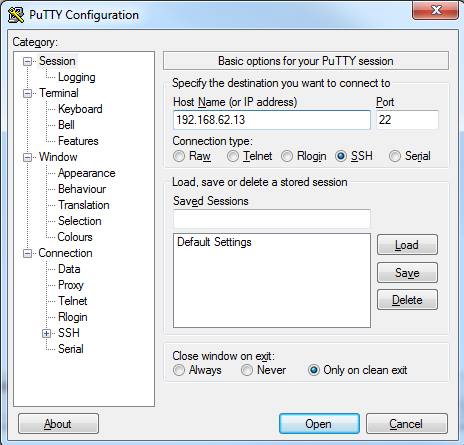












**Progressione lavori su FPGA/Redpitaya.**

Prima di affrontare lo sviluppo dell’obbiettivo della tesi che riguarda un sistema di acquisizione dati in streaming è necessario svolgere un insieme ben congeniato di test e esercizi sull’hardware disponibile.

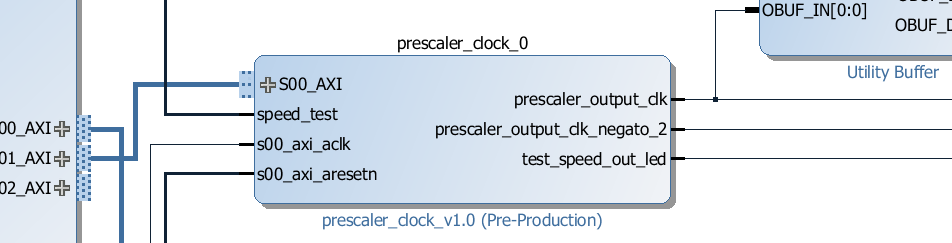
Nel periodo prima delle vacanze natalizie abbiamo svolto delle esperienze mirate a stimare la velocità e l’affidabilità della trasmissione differenziale sui GPIO dello ZYNQ7000 della RedPitaya.

Obbiettivo:

Generiamo internamente un segnale logico, lo trasformiamo in un segnale differenziale TMDS (dato che lo standard LVDS 25 non era utilizzabile a causa dei livelli di tensione vincolati nella RedPitaya a 3V3 anziché 2V5) con un utility buffer indicato con util\_ds\_buf\_0 che rende disponibile ai pin in output un differenziale di 400mV come mostrato in seguito.

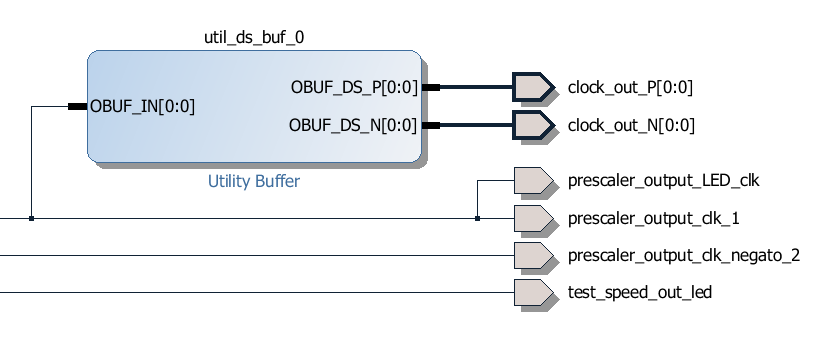
Tale segnale viene riletto da un’altra coppia differenziale allo scopo di testare la velocità con cui reagisce il sistema.

La prova avrà successo quando quanto generato viene acquisito correttamente ed un eventuale calcolo interno dei fronti di segnale risulti corretto.



Lo scopo dell’IP prescaler è quello di riferirsi a un clock interno e ricavarne una frequenza proporzionale da inviare, passando attraverso a un “utility buffer” a una coppia differenziale di pin di GPIO.

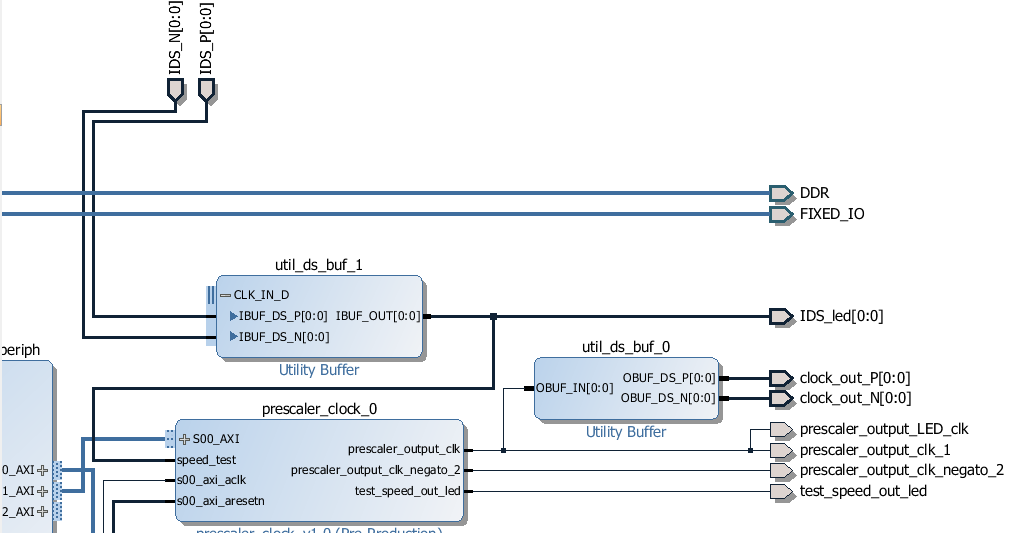
I segnali sono disponibili all’scita clock\_out\_P[0:0], clock\_out\_N[0:0].



L’obbiettivo è quello di leggere il segnale posto in output, eseguire una divisione per 10 e riproporlo in uscita dove viene rilevato da uno dei LED della Pitaya.

Sono operativi 2 led, il primo mostra la frequenza diretta, prima della divisione per 10 e il secondo un lampeggio a calcolo eseguito.

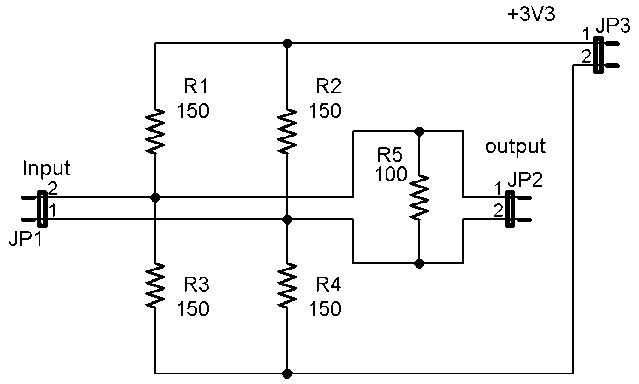
Impostando inizialmente via software una frequenza piuttosto bassa e stimabile ad occhio nudo, si stima se ogni 10 lampeggi del led direttamente connesso al segnale di clock avviene il cambio di stato del segnale suddiviso.



Osservando il percorso del segnale dagli ingressi differenziali IDS\_N[0,0], IDS\_P[0,0] all’uscita IDS\_led[0,0] si nota che risulta interamente implementato nella sezione FPGA ovvero non attraversa l’IP processiong\_system7\_0, quindi il segnale non è elaborato dalla sezione ARM dello ZYNQ, come è di nostra necessità in questa prova sperimentale.

L’uscita dell’utility buffer 1, viene diramata verso l’IP prescaler in cui è implementato il comparatore software in VHDL

L’interfacciamento avviene tramite il circuito sottostante.



JP3 viene collegato all’extension connector E1 della Pitaya, pin 1 al pin 26 mentre il pin 2 al pin 1 (ovvero massa) della Pitaya.

Ai capi di R5 che funge da carico terminatore, quindi al connettore JP2, misuriamo un segnale nelle due modalità:

1. Modo comune, ovvero riferito alla massa, sul reoforo superiore di R5 pari a 1V4, mentre sul reoforo inferiore pari a 1V.
2. Modo differenziale, ovvero tra i due reofori di R5, pari a 400mV.

Le misure sono corrette rispetto alle attese e rispetto allo standard LVDS25, che mostra un’ampiezza del segnale utile di 400mV, ma Il sistema è risultato particolarmente sensibile ai disturbi indotti e all’inadeguatezza dei cavi utilizzati.

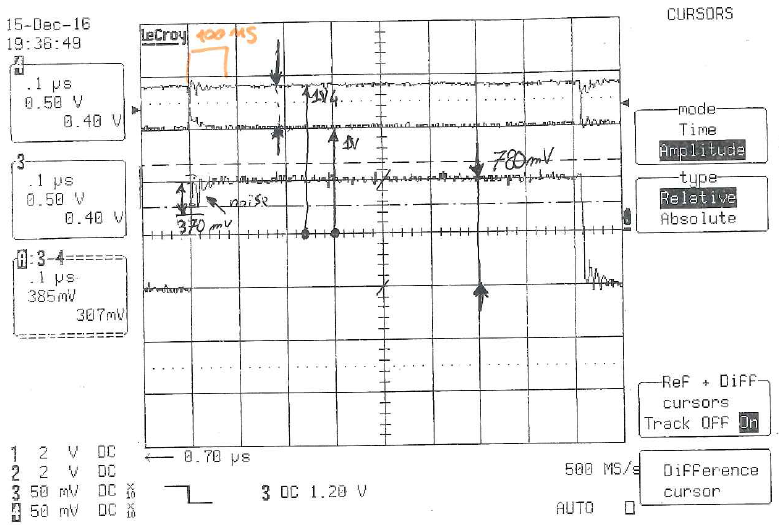
La prima interfaccia, realizzata con componentistica PTH, comportava errori di conteggio e comportamenti inattesi sia in lettura che in impostazione delle uscite anche a frequenze basse.

Osservando i segnali a monte e a valle dell’interfaccia resistiva con l’oscilloscopio si è rilevata la presenza di glitch di ampiezza sufficiente a confondere i comparatori software.

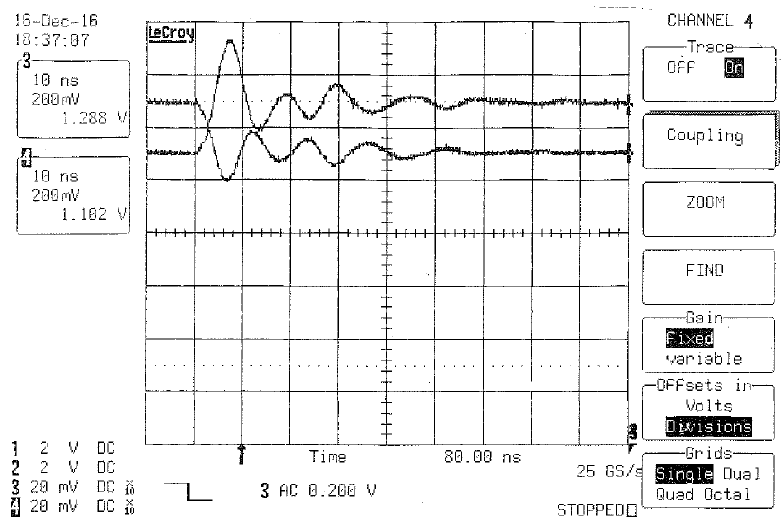
Sulla tracciata si osserva che l’ampiezza del disturbo, in prossimità dei fronti di commutazione, arriva a ben 370mV ed è quindi paragonabile al segnale utile.

Il risultato è che la commutazione del LED di rilevamento, invece che invertire lo stato ogni 10 fronti del segnale generato internamente -> inviato al GPIO -> letto dal GPIO e riportato all’interno avviene in maniera piuttosto costante ogni 3 o 4 fronti.

Verificata più volte l’esattezza del codice VHDL si è giunti alla conclusione che in pratica è l’interfaccia hardware a non funzionare.



Facendo uno zoom in prossimità del fronte:



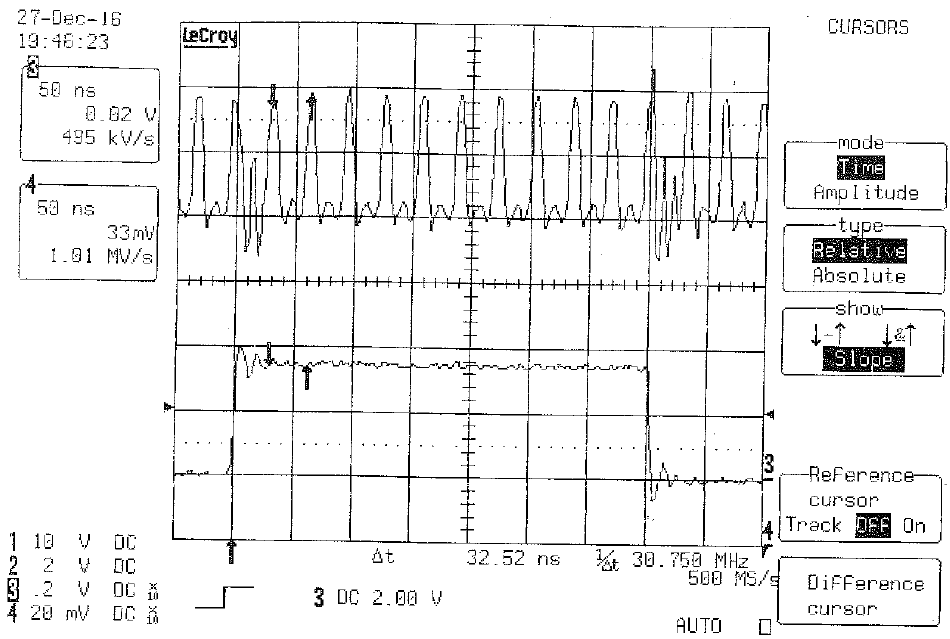
Abbiamo proceduto alla costruzione di una nuova interfaccia da innestare direttamente sul extension connector della Pitaya, eliminando i cavi, e realizzata in tecnologia SMD.

Nel circuito sono stati introdotti dei punti di test in cui inserire le sonde in maniera meno precaria.

La maggior compattezza e mancanza di punti di iniezione del disturbo a portato ai risultati attesi mostrati nel successivo tracciato oscilloscopico.

Alle basse frequenze, comunque oltre il MHz, la forma d’onda letta all’ingresso dell’interfaccia ovvero all’uscita non prescalata, risulta molto pulita e squadrata.

Nel tracciato del 27 dicembre, riportato in seguito, leggiamo un frequenza di 30,75MHz e rileviamo dei disturbi insufficienti a portare in errore i comparatori software.



**Step successivi:**

**Secondo esercizio:**

Per poterci avvicinare al protocollo che si userà presente nel modulo ADC portoghese si potrebbe procedere eseguendo una nuova prova per la lettura di un valore numerico subordinata alla sincronizzazione di un clock generato dalla Pitaya .

Fermo restando il clock di uscita già presente, aggiungiamo un canale differenziale TMDS con cui generare e successivamente rileggere un valore numerico a 18bit (come nel chip impiegato nel modulo).

Inviamo il dato fuori dalla Pitaya tramite un registro per rileggerlo in un secondo registro entrambi a 32 bit di cui i primi 18 conterranno il dato.

**Terzo esercizio:**

implementazione di un buffer circolare per contenere i valori del modulo ADC accessibile tramite opportuni registri.