

Introduzione a Vivado HLx

Gianluca Brilli, Paolo Burgio

CALCOLO PARALLELO, 2019/2020

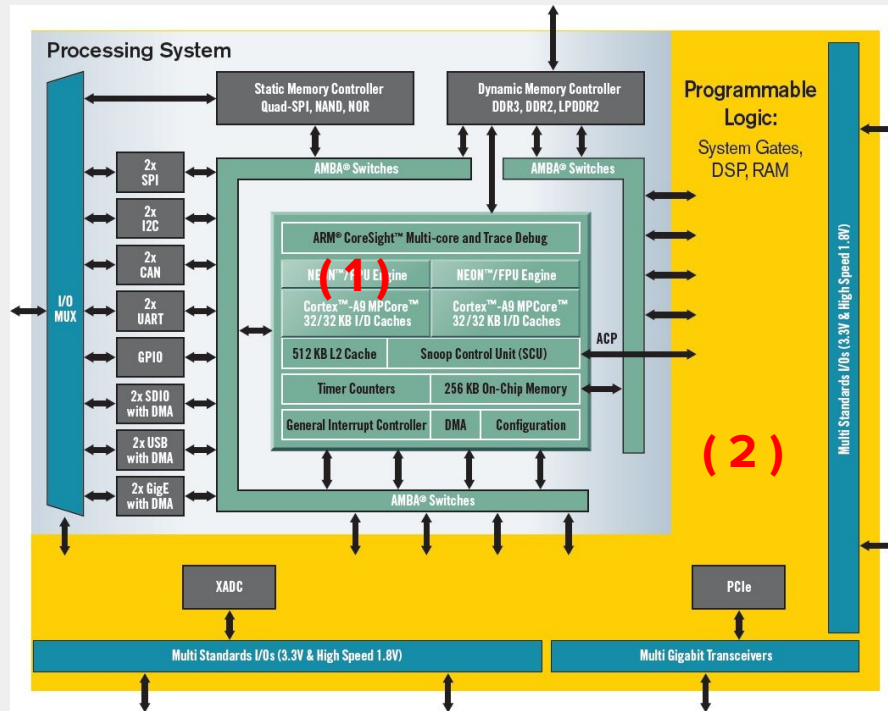
gianluca.brilli@unimore.it, paolo.burgio@unimore.it



Introduzione

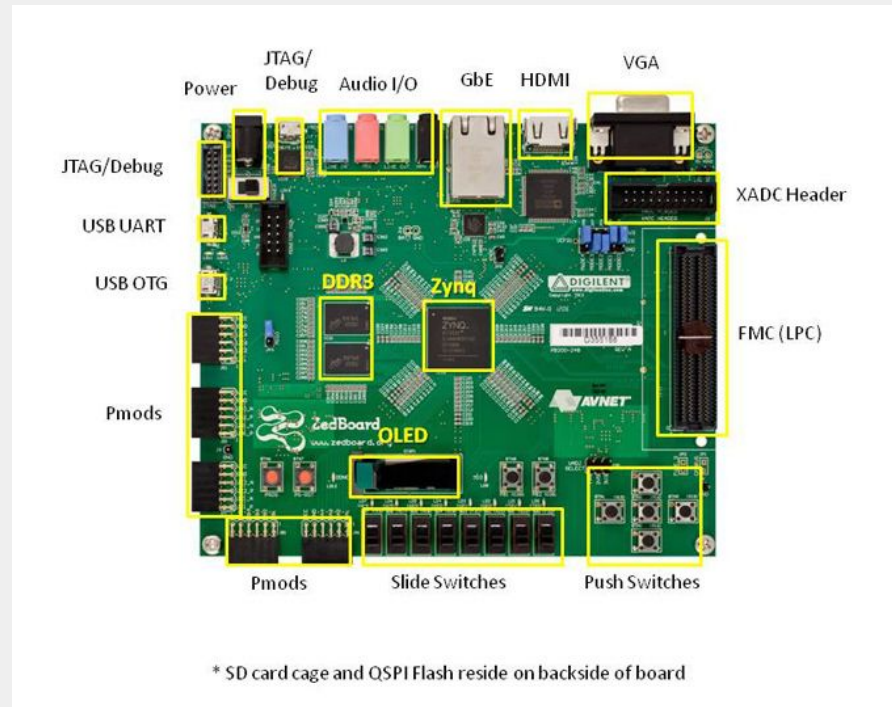


System-on-Chip Zynq-7000



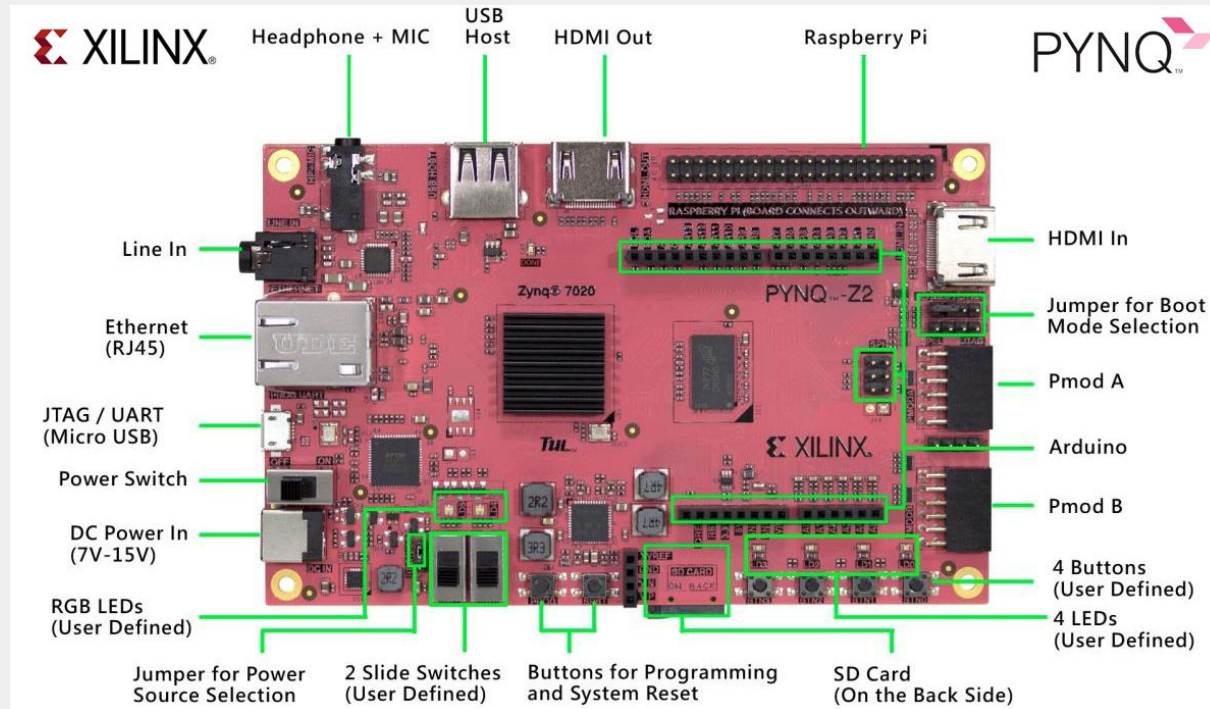
1. CPU avente due cores ARM Cortex A9 (600 MHz);
2. Logica programmabile (FPGA).

Avnet ZedBoard (SoC Zynq-7000)



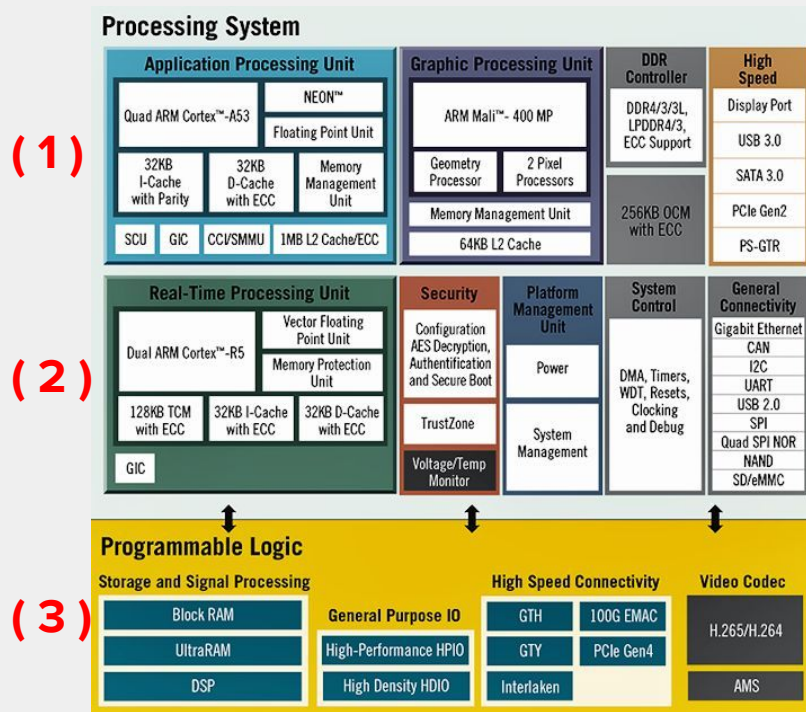
- Molte porte di ingresso e uscita, che la rendono versatile per diversi utilizzi;
- Relativamente compatta come dimensioni.

Xilinx Pynq (SoC Zynq-7000)



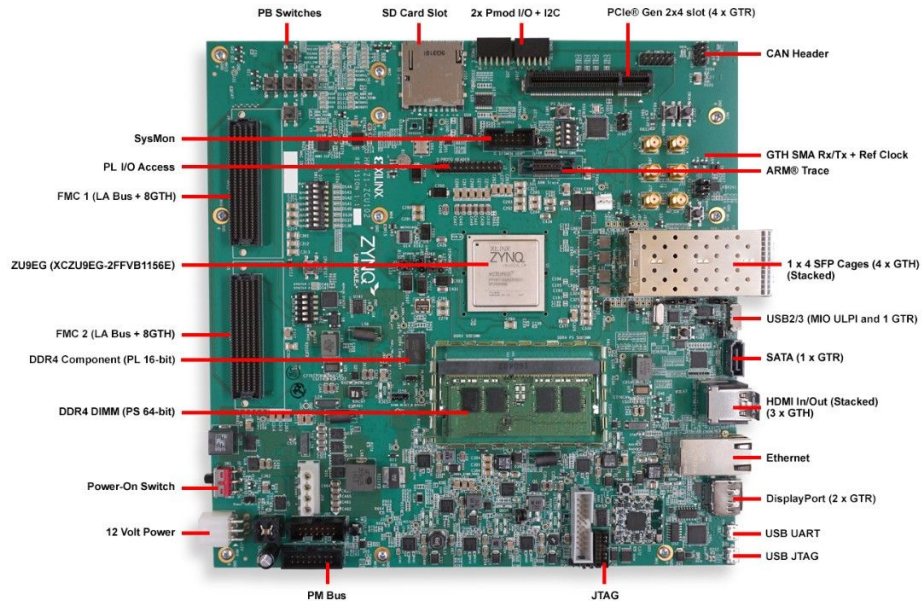
- Più compatta della precedente;
- Framework Pynq a disposizione.

System-on-Chip UltraScale+



1. APU: Application Processing Unit, composta da quattro cores ARM Cortex A53;
2. RPU: Real-Time Processing Unit, composta da due cores ARM Cortex R5;
3. Logica Programmabile (FPGA).

Xilinx ZCU102 (SoC UltraScale+)



Avnet Ultra96 (SoC UltraScale+)



- Molto piccola e compatta;
- Buona connettività, dispone di un modulo WIFI e Bluetooth;
- E' inoltre possibile espandere le funzionalità tramite delle schede aggiuntive.

FPGA Design Overview



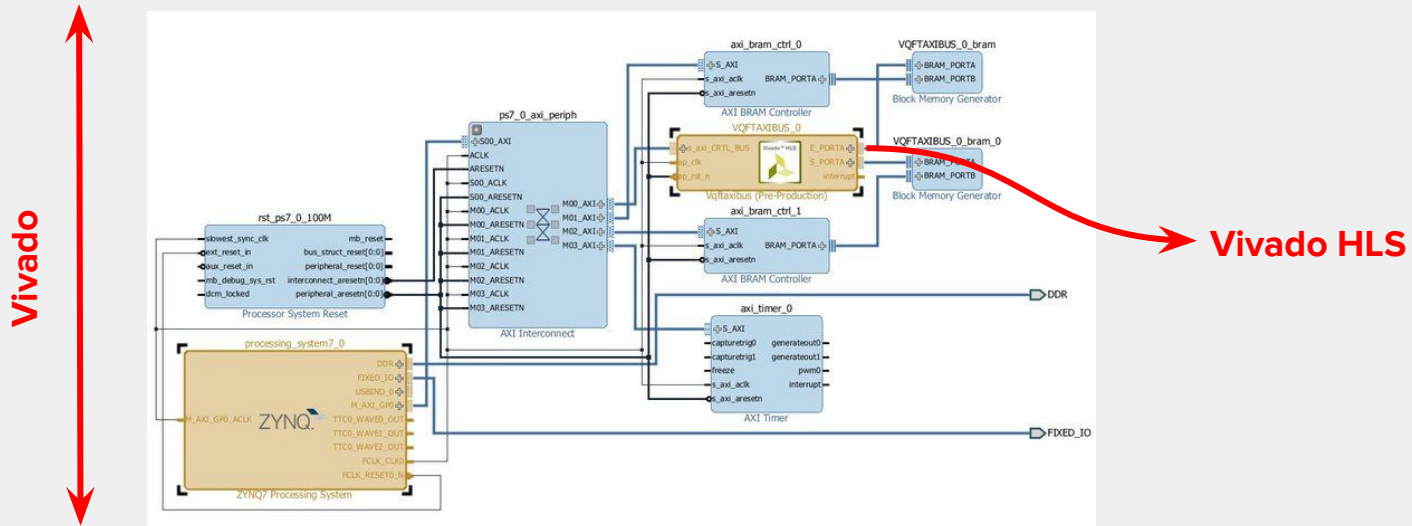
- Strumento principale sviluppato da Xilinx e utilizzato per la programmazione FPGA.
- Suite composta da tre tools:

- Vivado HLS
- Vivado HLx
- Xilinx SDK



FPGA Design Overview: Hardware

- La progettazione FPGA con questi strumenti consiste nella creazione di uno **schema a blocchi**, come quello in figura:

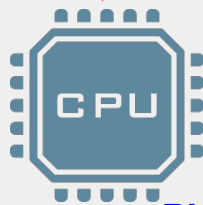


FPGA Design Overview: Software

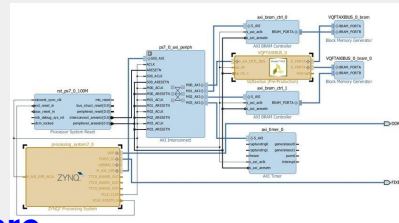


Software di controllo per
la periferica HW, scritto
con Xilinx SDK

- CPU presente
nel chip Zynq.



BUS Hardware
(AXI4)



- Hardware
design
realizzato con
Vivado

High Level Synthesis

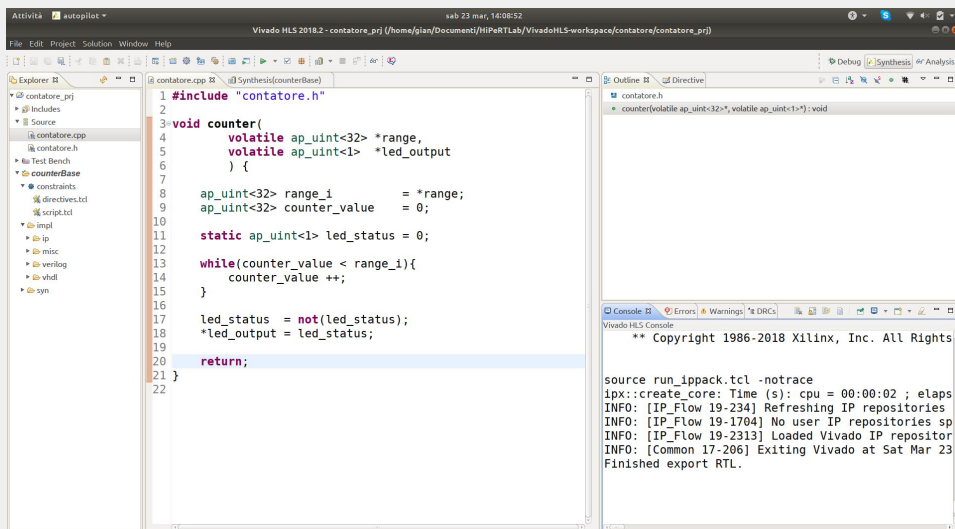
*HiPeRT
Lab*

High-Performance Real-Time Lab



Vivado HLS: Interfaccia

- IDE basato su Eclipse, che permette la traduzione da codice C/C++ in codice HDL.



The screenshot displays the Vivado HLS 2018.2 IDE interface. On the left, the 'Project Explorer' shows a project named 'contatore_prj' with files like 'contatore.cpp', 'contatore.h', and 'counterBase'. The main editor window shows the C++ code in 'contatore.cpp':

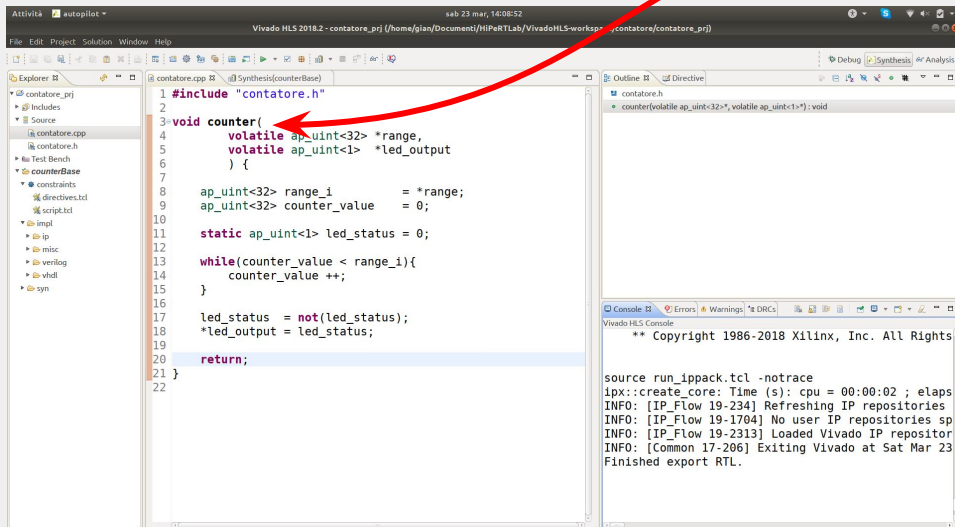
```
1 #include "contatore.h"
2
3 void counter(
4     volatile ap_uint<32> *range,
5     volatile ap_uint<1> *led_output
6 ) {
7
8     ap_uint<32> range_i    = *range;
9     ap_uint<32> counter_value = 0;
10
11     static ap_uint<1> led_status = 0;
12
13     while(counter_value < range_i){
14         counter_value ++;
15     }
16
17     led_status = not(led_status);
18     *led_output = led_status;
19
20     return;
21 }
22
```

On the right, the 'Outline' pane shows the function 'counter'. Below it, the 'Console' pane displays the Vivado HLS console output:

```
** Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
source run_ippack.tcl -notrace
ipx::create_core: Time (s): cpu = 00:00:02 ; elaps
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories sp
INFO: [IP_Flow 19-2313] Loaded Vivado IP repositor
INFO: [Common 17-206] Exiting Vivado at Sat Mar 23
Finished export RTL.
```

- Svolge la funzione di Sintesi per IP customizzati.

Vivado HLS: Interfaccia

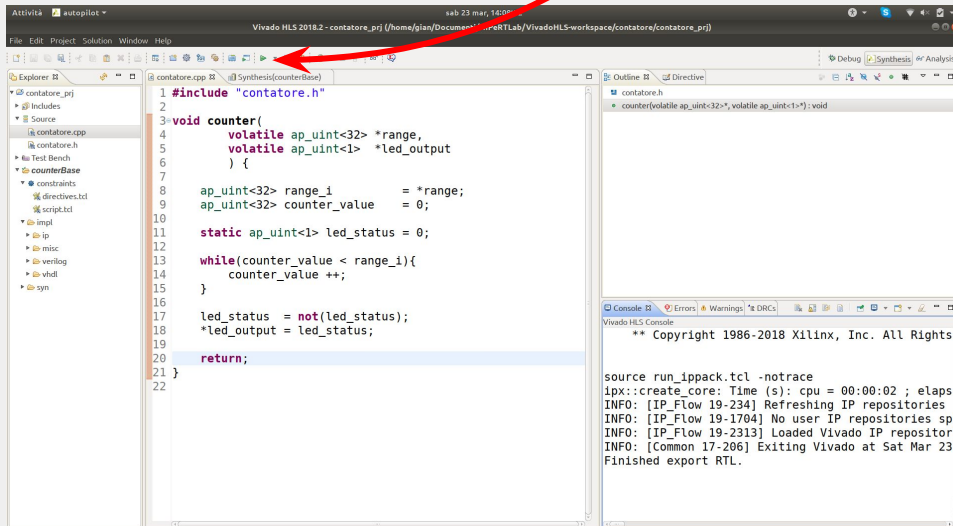


```
1 #include "contatore.h"
2
3 void counter(
4     volatile ap_uint<32> *range,
5     volatile ap_uint<1> *led_output
6 ) {
7
8     ap_uint<32> range_i    = *range;
9     ap_uint<32> counter_value = 0;
10
11     static ap_uint<1> led_status = 0;
12
13     while(counter_value < range_i){
14         counter_value ++;
15     }
16
17     led_status = not(led_status);
18     *led_output = led_status;
19
20     return;
21 }
22
```

Top Function, è la funzione che sarà convertita in un modulo hardware, i parametri della funzione saranno i segnali di ingresso e uscita che ritroveremo nell'IP sintetizzato.

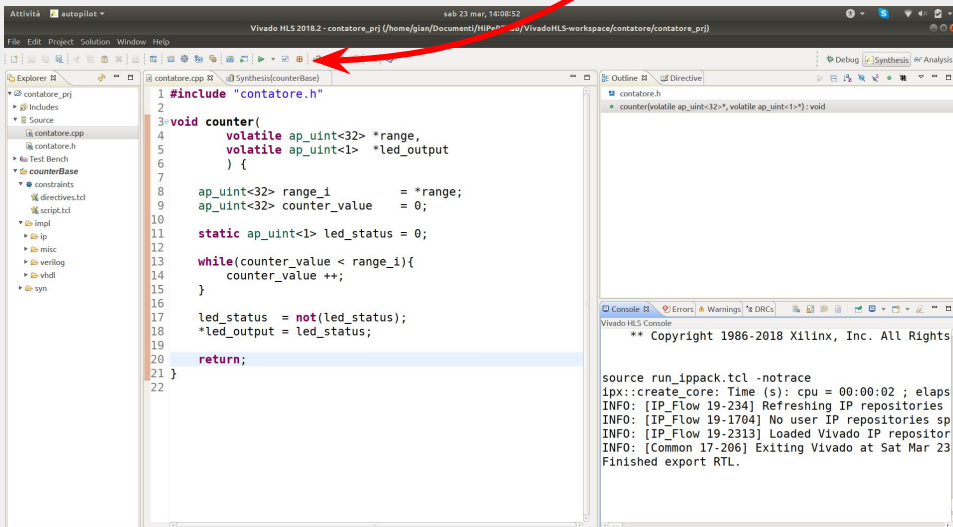
Vivado HLS: Interfaccia

Sintesi, esegue la traduzione da codice HLS (C/C++) in codice HDL.



Vivado HLS: Interfaccia

Export RTL, permette di esportare il codice Verilog / VHDL generato e creare un pacchetto importabile da Vivado.

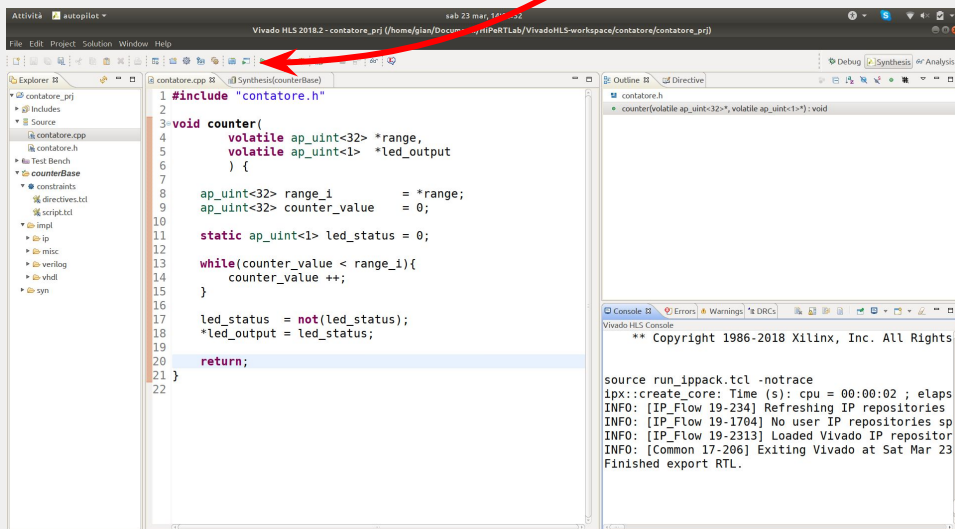


- RTL = Register Transfer Level;
- Ciò che abbiamo sintetizzato al passo precedente.



Vivado HLS: Interfaccia

Permette di eseguire un TestBench, ovvero una funzione main che va a richiamare la top function che abbiamo scritto.



Vivado HLS: Report Finale

General Information

Date: Thu Mar 7 21:50:46 2019

Version: 2018.2 (Build 2258646 on Thu Jun 14 20:25:20 MDT 2018)

Project: Convolution_prj

Solution: solution1

Product family: zynq

Target device: xc7z020clg484-1

Performance Estimates

Timing (ns)

Summary

Clock Target Estimated Uncertainty

ap_clk 10.00 9.634 1.25

Latency (clock cycles)

Summary

Latency Interval

min max min max Type

844870 844870 844870 844870 none

Detail

Instance

Loop

Utilization Estimates

Summary

Name BRAM_18K DSP48E FF LUT

DSP - 9 - -

Expression - - 0 784

FIFO - - - -

Instance 6 - 1680 2140

Memory 128 - 0 0

Multiplexer - - - 623

Register 0 - 1059 192

Total 134 9 2739 3739

Available 280 220 106400 53200

Utilization (%) 47 4 2 7

Periodo stimato: supponendo un periodo di 10 ns (clock a 100 MHz).

Numero esatto di cicli di clock, affinchè la Top Function termini l'esecuzione.

Numero di risorse utilizzate per l'implementazione della funzione, in termini di Flip-Flops, LUT, BRAM e DSP.

Vivado HLS: Report Finale

General Information

Date: Thu Mar 7 21:50:46 2019

Version: 2018.2 (Build 2258646 on Thu Jun 14 20:25:20 MDT 2018)

Project: Convolution_prj

Solution: solution1

Product family: zynq

Target device: xc7z020clg484-1

Performance Estimates

Timing (ns)

Summary

Clock Target	Estimated	Uncertainty
ap_clk 10.00	9.634	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
844870	844870	844870	844870	none

Details

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	9	-	-
Expression	-	-	0	784
FIFO	-	-	-	-
Instance	6	-	1680	2140
Memory	128	-	0	0
Multiplexer	-	-	-	623
Register	0	-	1059	192
Total	134	9	2739	3739
Available	280	220	106400	53200
Utilization (%)	47	4	2	7

- Stima del tempo di esecuzione della nostra Top Function:

- $9.634 * 844870 = 8.13947758 \times 10^6$ ns



- *Worst-Case-Execution-Time (WCET)*



Vivado HLS: Codice di esempio

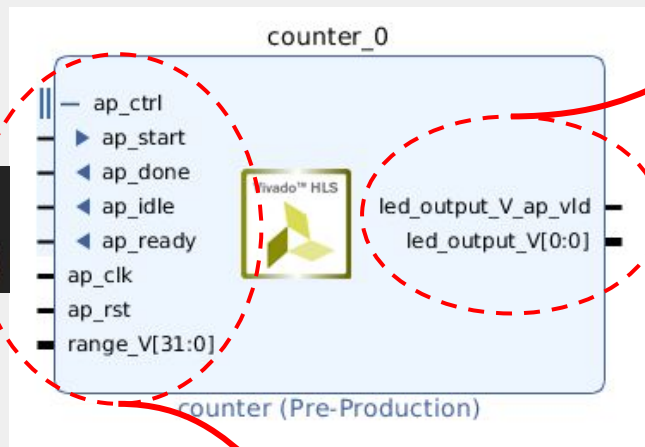
```
1
2 void counter(
3     volatile int range,
4     volatile bool *led_out) {
5
6     static bool led_status = 0;
7     volatile int temp_count = 0;
8
9     while (temp_count < range) {
10         temp_count = temp_count + 1;
11     }
12
13     led_status = not(led_status);
14
15     *led_out = led_status;
16 }
```

- Semplice contatore che accende e spegne un led.

Vivado HLS: Struttura di un IP

- Ovvero cosa otteniamo dopo aver sintetizzato il codice HLS e dopo aver esportato l'RTL.

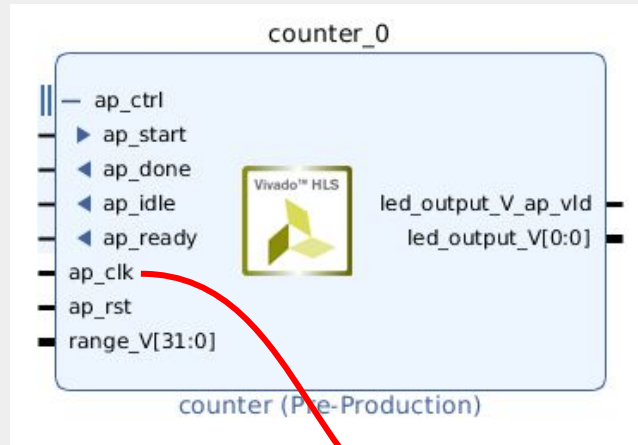
```
void counter(  
    volatile int    range,  
    volatile bool *led_out)
```



Porte di uscita

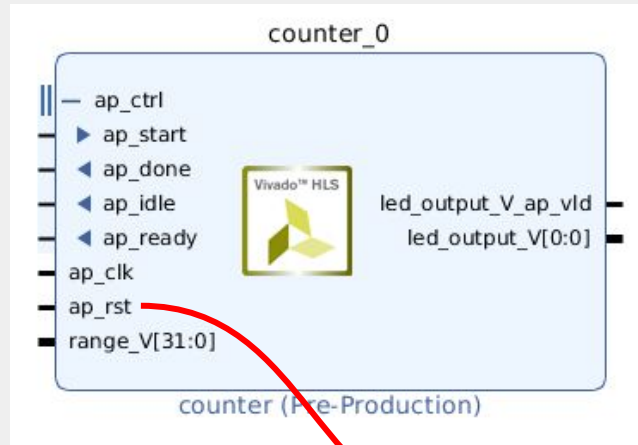
Porte di ingresso

Vivado HLS: Struttura di un IP



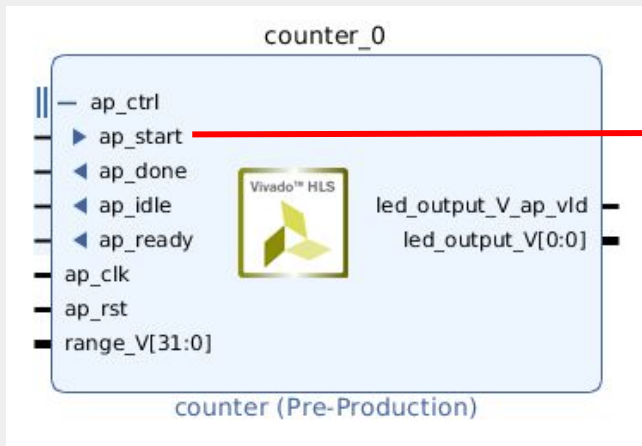
Clock in ingresso,
collegato ad una sorgente
esterna.

Vivado HLS: Struttura di un IP



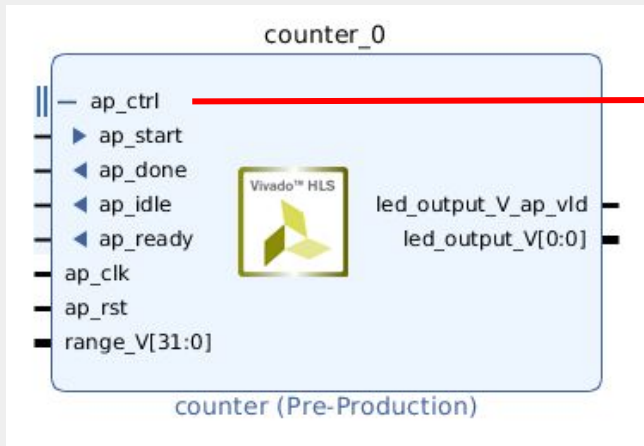
Reset attivo alto,
necessario in ogni logica
sequenziale.

Vivado HLS: Struttura di un IP



Bit di start: quando settato la Top Function parte.

Vivado HLS: Struttura di un IP



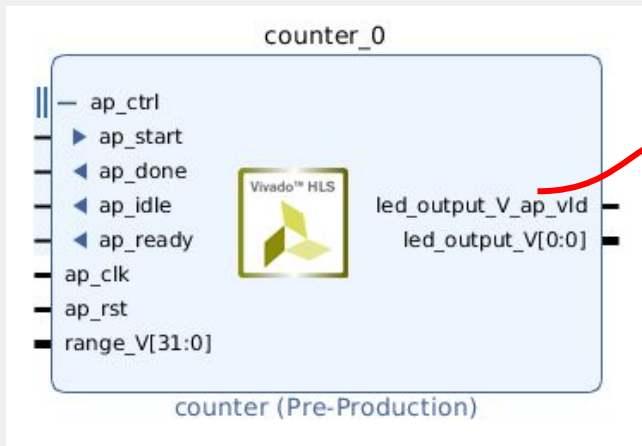
Bit di stato, indicano lo stato interno della Top Function e possono essere usati come interrupts

Done: La Top Function ha terminato;

Idle: La Top Function non sta eseguendo al momento;

Ready: Il modulo è pronto ad eseguire.

Vivado HLS: Struttura di un IP



Bit di validità, indica se nella rispettiva porta di uscita è presente un dato valido. Non sempre indispensabile, può essere disabilitato tramite direttiva pragma.

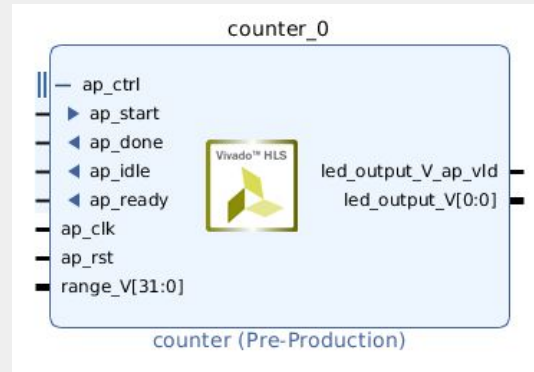
Design Globale

*HiPeRT
Lab*

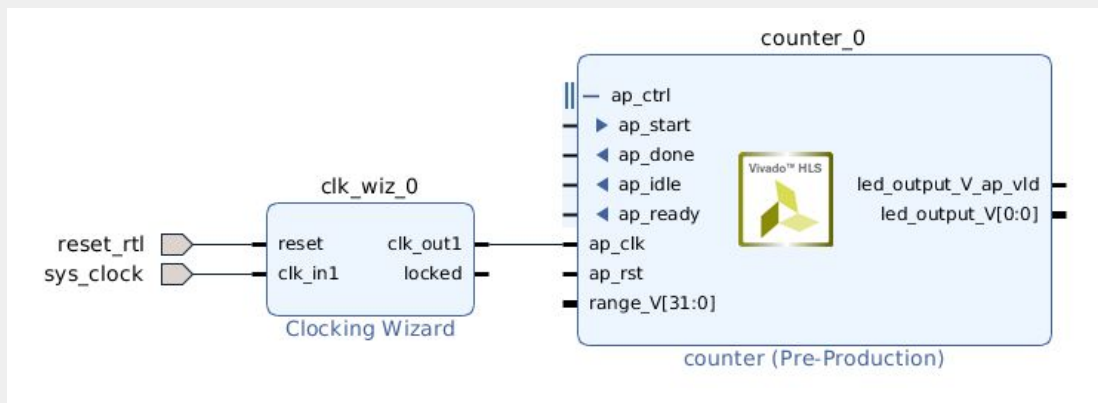
High-Performance Real-Time Lab



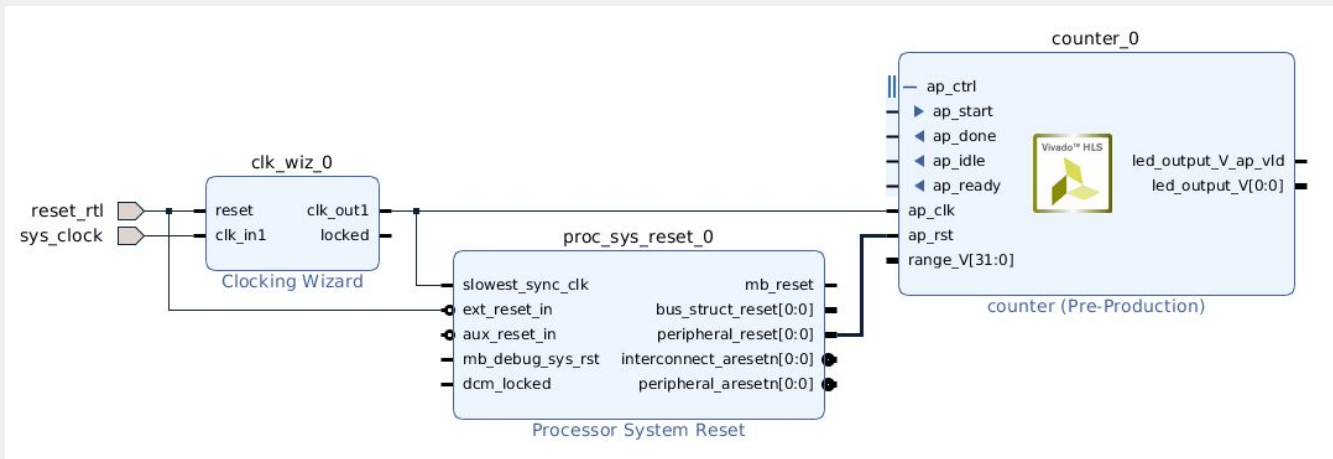
Vivado: Creazione di un semplice Block Design



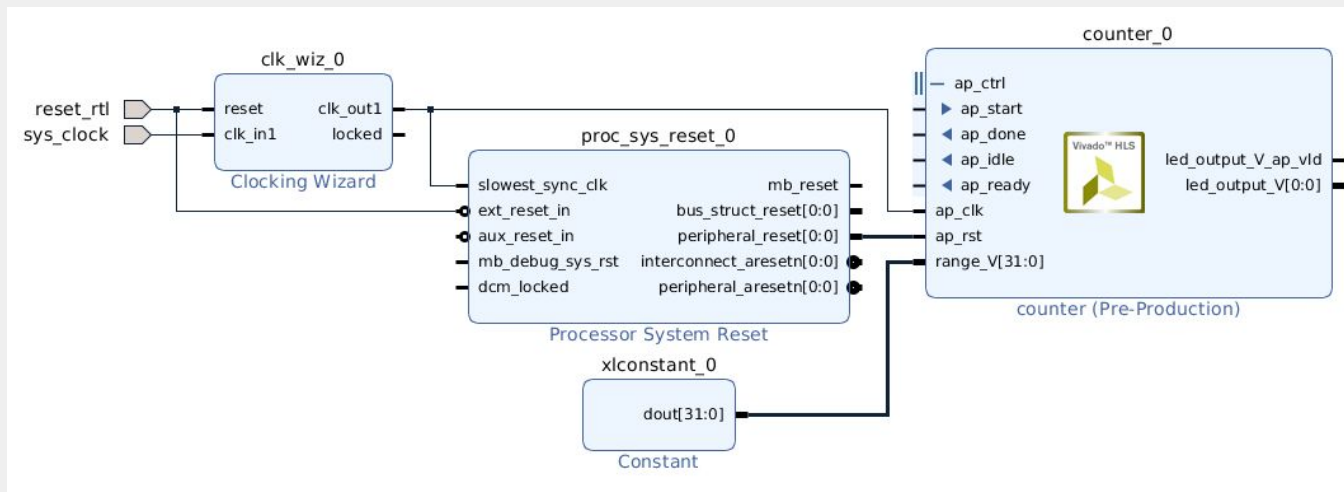
Vivado: Creazione di un semplice Block Design



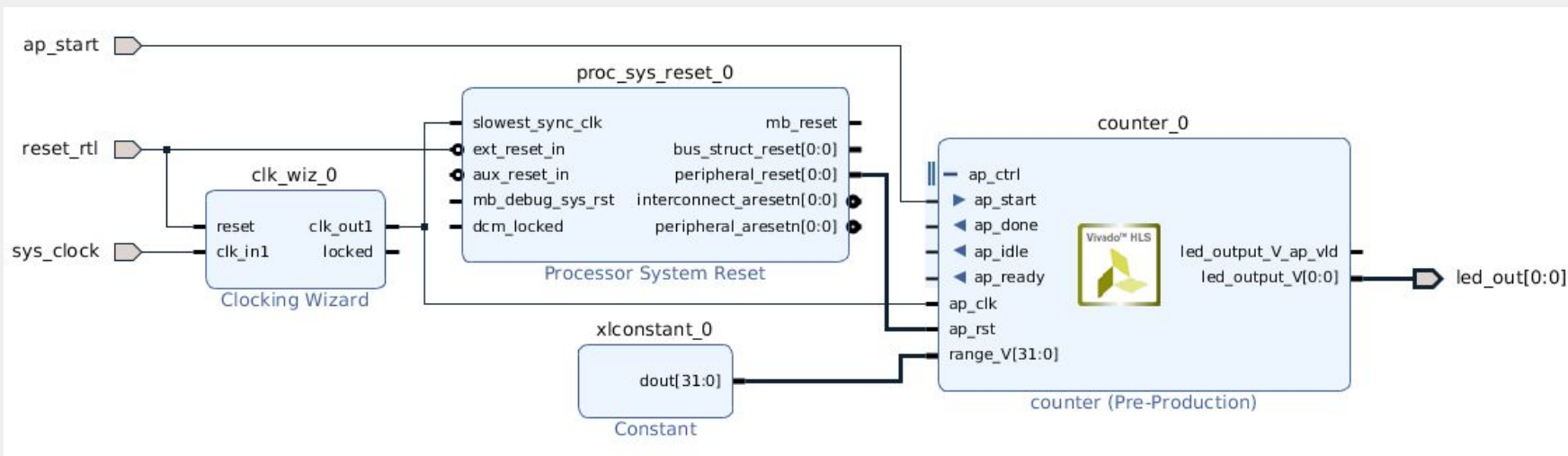
Vivado: Creazione di un semplice Block Design



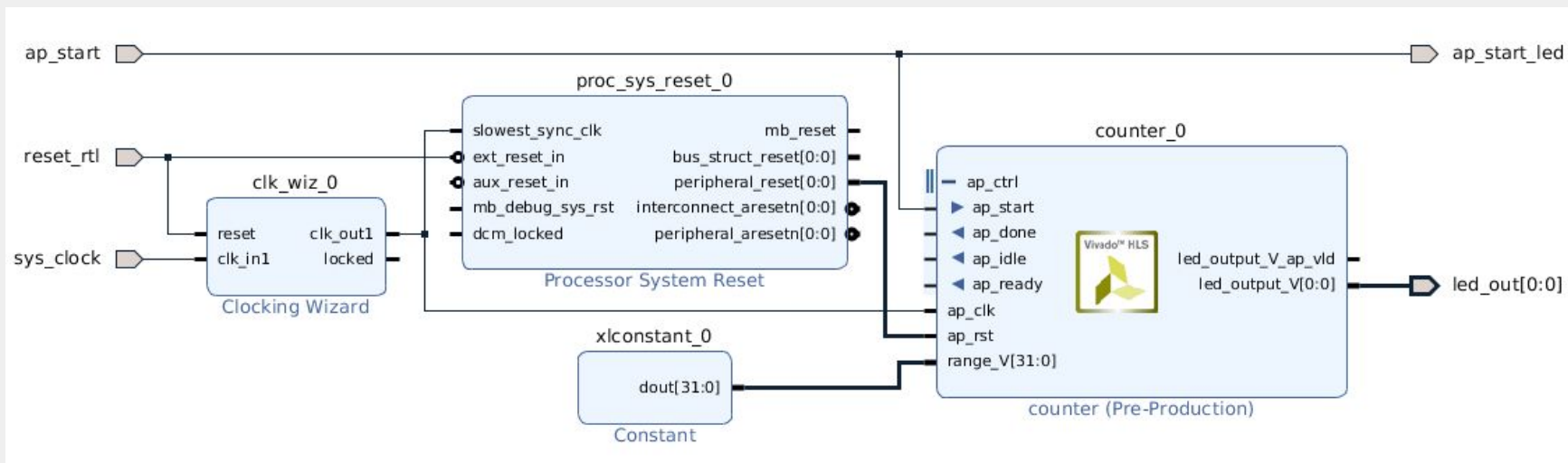
Vivado: Creazione di un semplice Block Design



Vivado: Creazione di un semplice Block Design



Vivado: Creazione di un semplice Block Design



Interazione HW-SW

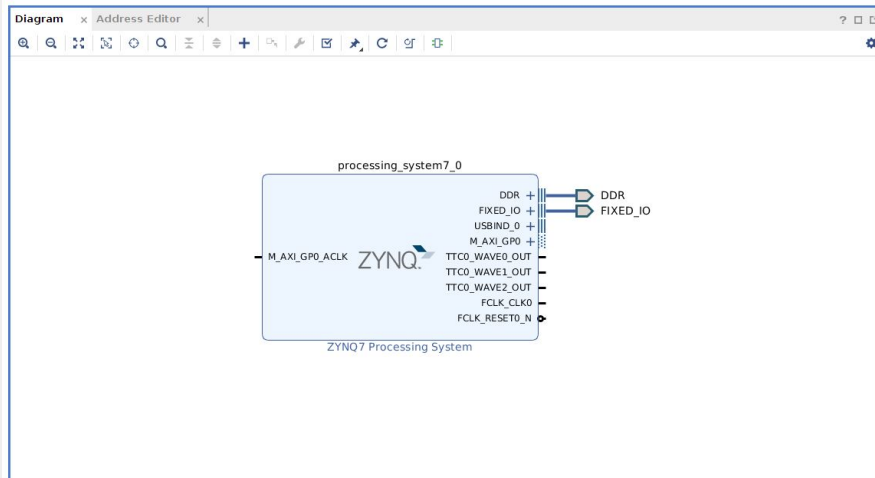
*HiPeRT
Lab*

High-Performance Real-Time Lab



Interazione HW-SW

- Supponiamo di voler controllare l'IP realizzato tramite software;
- Utilizzare l'IP Zynq PS7;

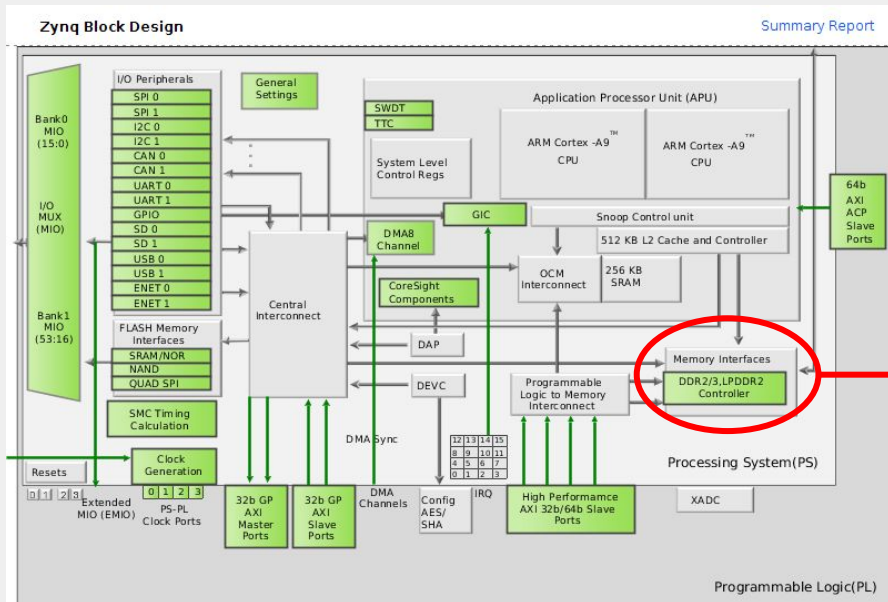


- Le connessioni tra CPU e FPGA, avvengono tramite BUS AXI4;
- Piccole modifiche al codice HLS.

Zynq: Spazio di Indirizzamento

- Lo spazio di indirizzamento della memoria principale, (nel caso del SoC Zynq-7000) è condiviso tra:

- Cortex A9 core_0;
- Cortex A9 core_1;
- Programmable Logic.

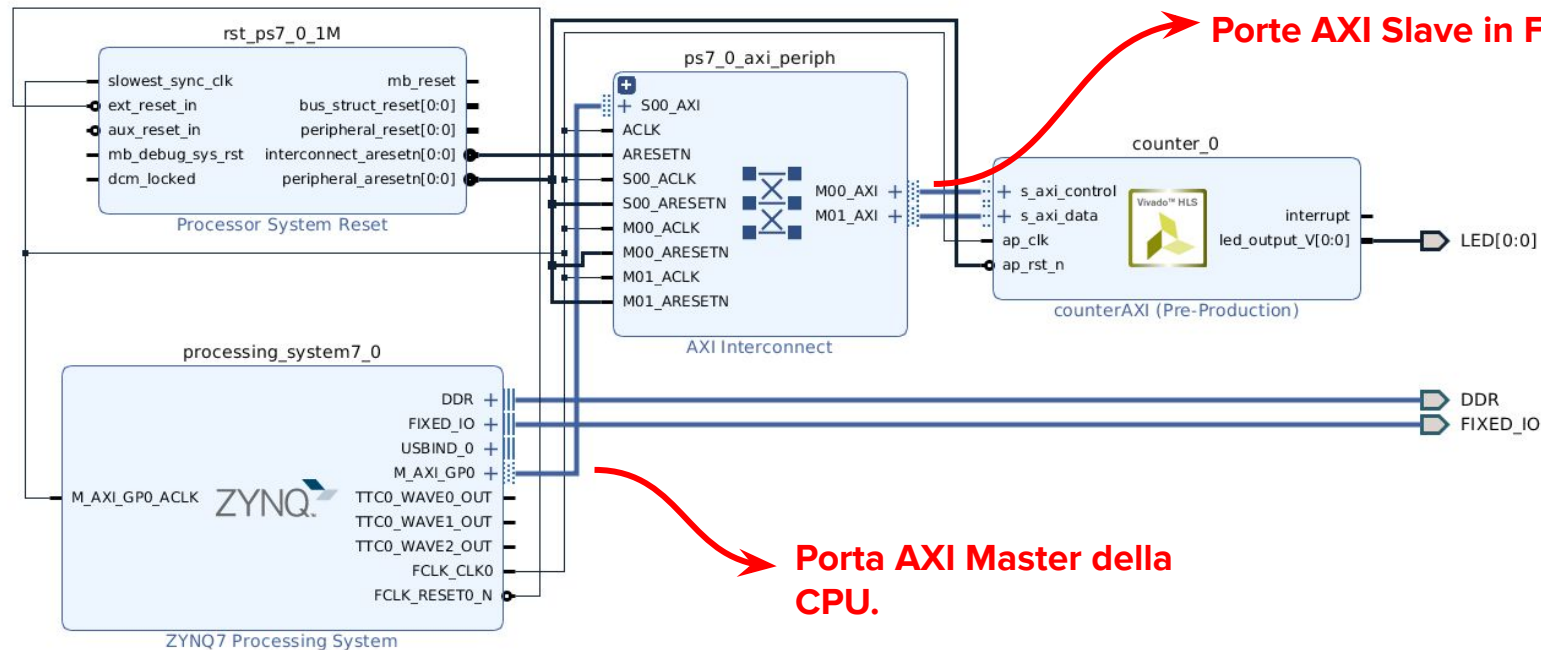


Modifiche al codice HLS

```
1 void counter(  
2     volatile int range,  
3     volatile bool *led_out) {  
4  
5     #pragma HLS INTERFACE s_axilite port=range bundle=control  
6     #pragma HLS INTERFACE s_axilite port=return bundle=control  
7  
8     static bool led_status = 0;  
9     volatile int temp_count = 0;  
10  
11     while (temp_count < range) {  
12         temp_count = temp_count + 1;  
13     }  
14  
15     led_status = not(led_status);  
16  
17     *led_out = led_status;  
18 }
```

- Interfacce AXI4 per la comunicazione PS/PL.

Interazione HW-SW



Zynq: Spazio di Indirizzamento

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
ps7_intc_dist_0	0xf8f01000	0xf8f01fff		REGISTER
ps7_gpio_0	0xe000a000	0xe000afff		REGISTER
ps7_scutimer_0	0xf8f00600	0xf8f0061f		REGISTER
ps7_slcr_0	0xf8000000	0xf8000fff		REGISTER
ps7_scuwdt_0	0xf8f00620	0xf8f006ff		REGISTER
ps7_l2cachec_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_scuc_0	0xf8f00000	0xf8f000fc		REGISTER
ps7_qspi_linear_0	0xfc000000	0xfcffffff		FLASH
ps7_pmu_0	0xf8893000	0xf8893fff		REGISTER
ps7_afi_1	0xf8009000	0xf8009fff		REGISTER
ps7_afi_0	0xf8008000	0xf8008fff		REGISTER
ps7_qspi_0	0xe000d000	0xe000dfff		REGISTER
ps7_usb_0	0xe0002000	0xe0002fff		REGISTER
ps7_afi_3	0xf800b000	0xf800bfff		REGISTER
ps7_afi_2	0xf800a000	0xf800afff		REGISTER
ps7_globaltimer_0	0xf8f00200	0xf8f002ff		REGISTER
counter_0	0x43c00000	0x43c0ffff	s_axi_control	REGISTER
ps7_dma_s	0xf8003000	0xf8003fff		REGISTER
ps7_iop_bus_config_0	0xe0200000	0xe0200fff		REGISTER
ps7_xadc_0	0xf8007100	0xf8007120		REGISTER
ps7_ddr_0	0x00100000	0x1ffffff		MEMORY
ps7_ddrc_0	0xf8006000	0xf8006fff		REGISTER
ps7_ocmc_0	0xf800c000	0xf800cfff		REGISTER
ps7_pl310_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_uart_1	0xe0001000	0xe0001fff		REGISTER
ps7_coresight_comp_0	0xf8800000	0xf880ffff		REGISTER
ps7_ttc_0	0xf8001000	0xf8001fff		REGISTER
ps7_scugic_0	0xf8f00100	0xf8f001ff		REGISTER
ps7_ethernet_0	0xe000b000	0xe000bfff		REGISTER
ps7_dev_cfg_0	0xf8007000	0xf80070ff		REGISTER
ps7_dma_ns	0xf8004000	0xf8004fff		REGISTER
ps7_sd_0	0xe0100000	0xe0100fff		REGISTER
ps7_gpv_0	0xf8900000	0xf890ffff		REGISTER
ps7_ram_1	0xffff0000	0xfffffdff		MEMORY
ps7_ram_0	0x00000000	0x0002ffff		MEMORY

Interfaccia s_axi_control dell'IP che abbiamo realizzato.



Mappata all'indirizzo 0x43C00000

Codice ARM

```
53 int main() {
54     init_platform();
55
56     XCounter counterInstance;
57
58     u32 range = 100000;
59
60     XCounter_Initialize(&counterInstance, 0);
61
62     printf("Counter initialized, range: %d\n\r", range);
63
64     XCounter_Set_range_V(&counterInstance, range);
65
66     printf("%x\n", Xil_In32(0x0));
67
68     while(1) {
69         XCounter_Start(&counterInstance);
70
71         while(!XCounter_IsDone(&counterInstance));
72         printf("Counter Done!\n\r");
73     }
74     cleanup_platform();
75
76     return 0;
77 }
78
79 }
```

Struttura dati che mantiene lo stato della periferica FPGA;

Setta i parametri della Top Function, il passaggio avviene attraverso AXI4-Lite;

Setta il bit di start;

Controlla il termine tramite Polling.



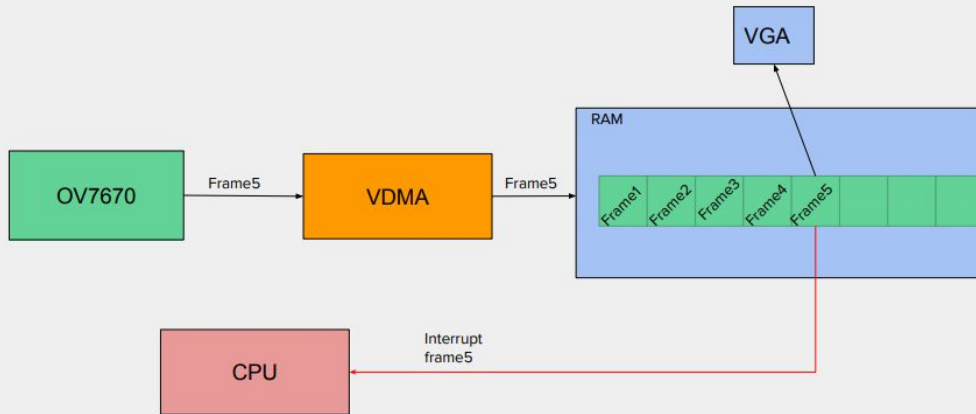
Applicazioni

HiPeRT
Lab

High-Performance Real-Time Lab

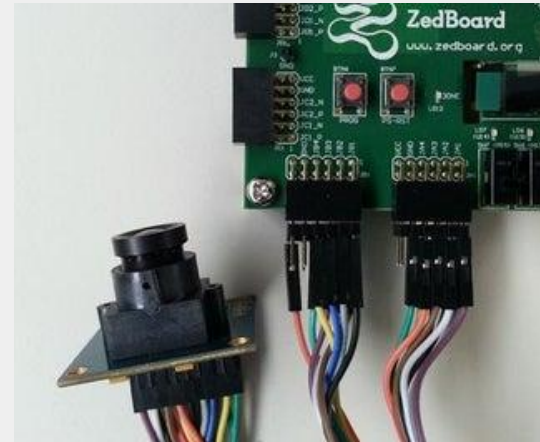


Periferica di acquisizione di frames video in DMA

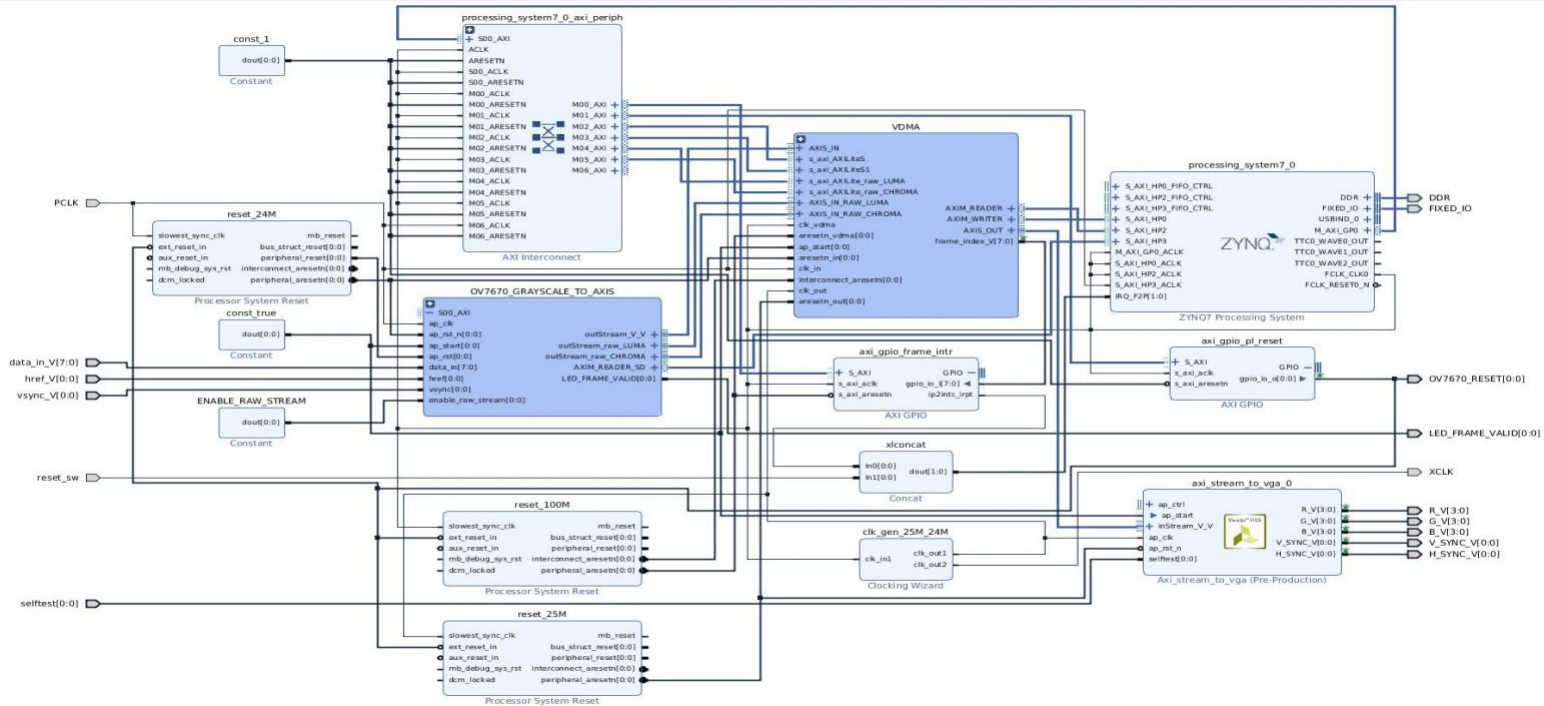


- Design Hardware realizzato da colleghi di Bologna (prof. Stefano Mattocchia)
 - *Tesi di Laurea.. (Andrea Bernardi)*

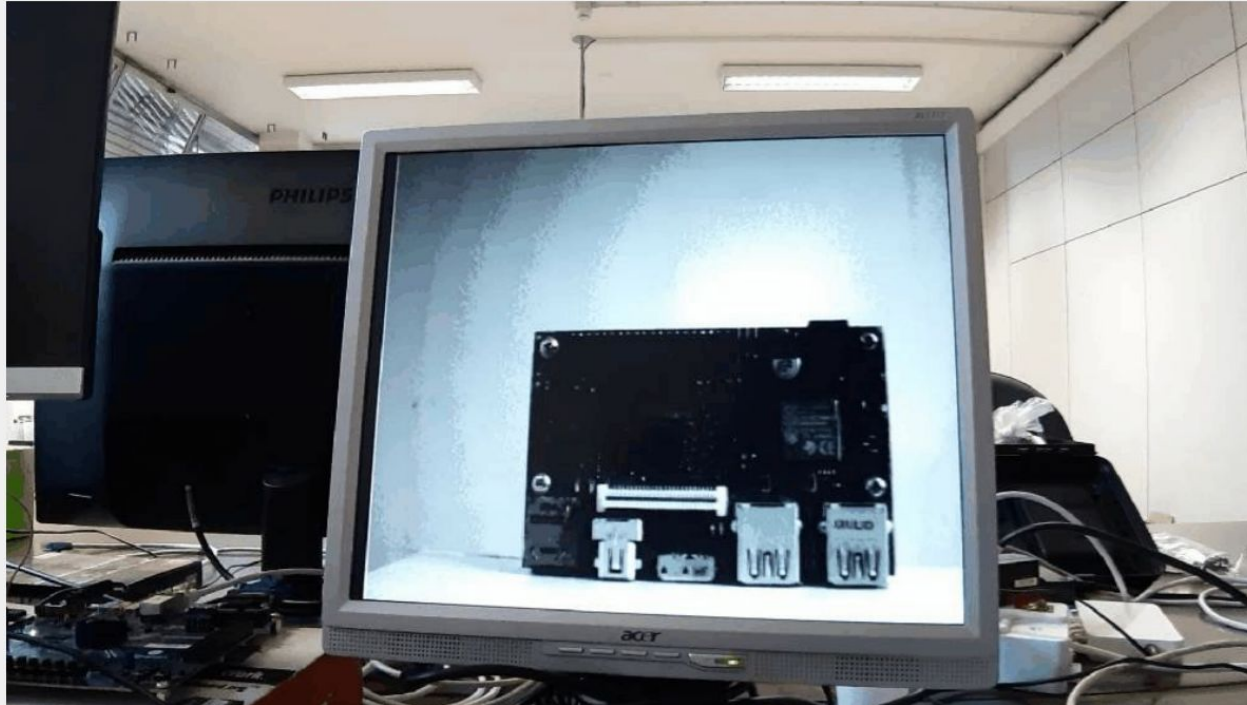
- Tramite sensore video connesso ad una ZedBoard.



Periferica di acquisizione di frames video in DMA



Periferica di acquisizione di frames video in DMA



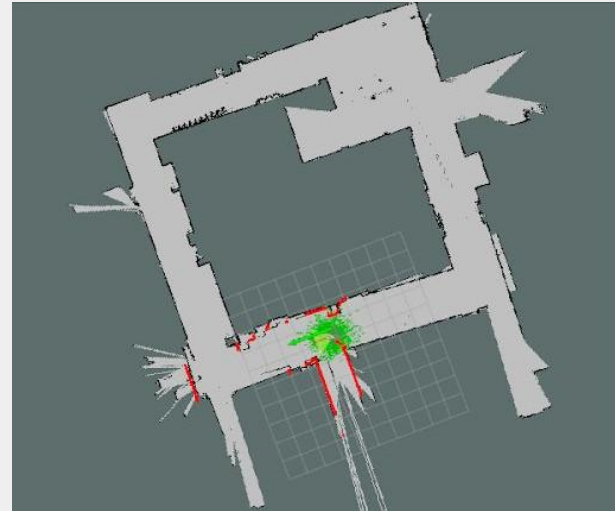
- Scrittura di drivers Linux per l'accesso alla periferica FPGA;
- Integrazione in OpenCV.

Localizzazione di un robot in FPGA

- Implementazione di un algoritmo di localizzazione (Particle Filter), utilizzato su una macchina autonoma in scala 1/10.

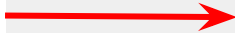
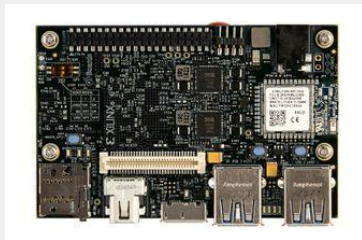


- *Work in progress.. (Andrea Bernardi)*



Implementazione del controllo di motori in FPGA

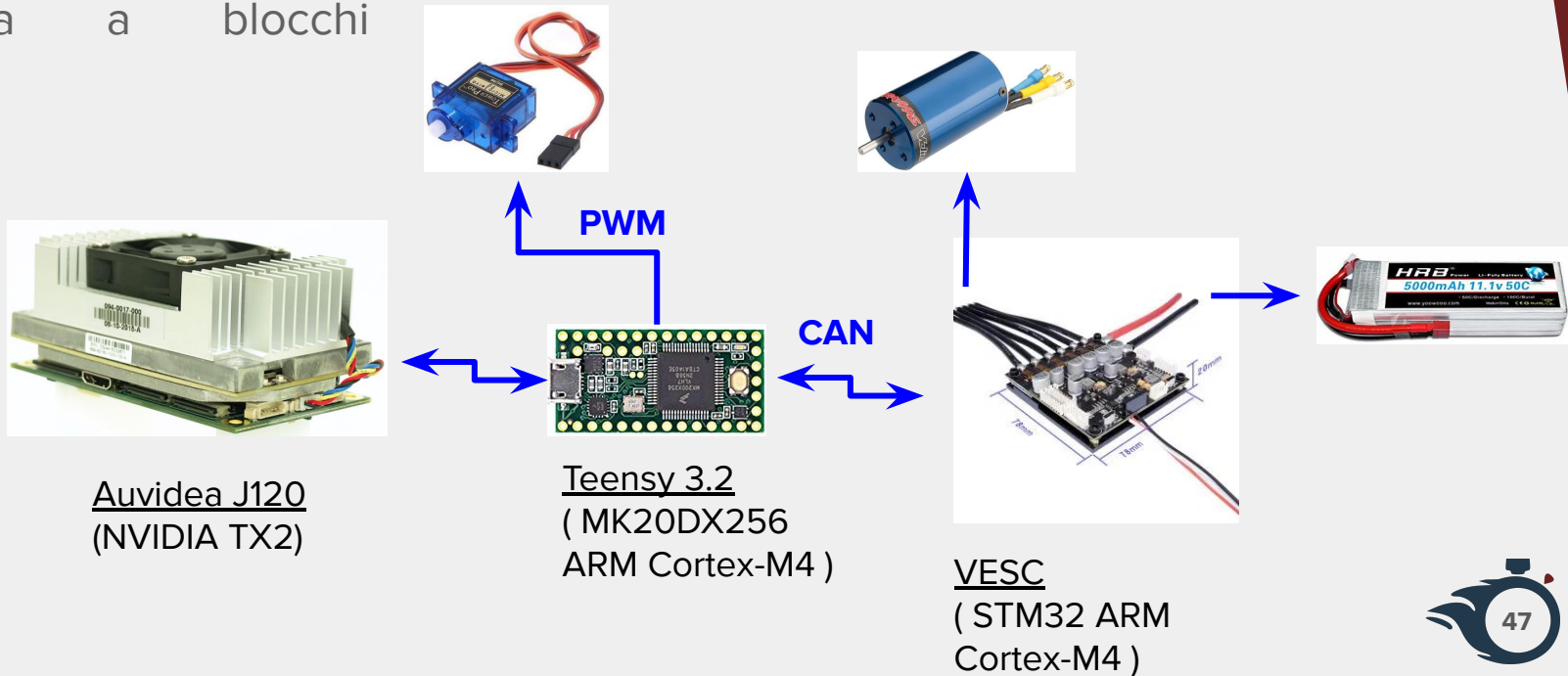
- Integrazione dello stack di guida autonoma (F1/10), su board Ultra96.



- *Work in progress.. (Federico Gavioli)*

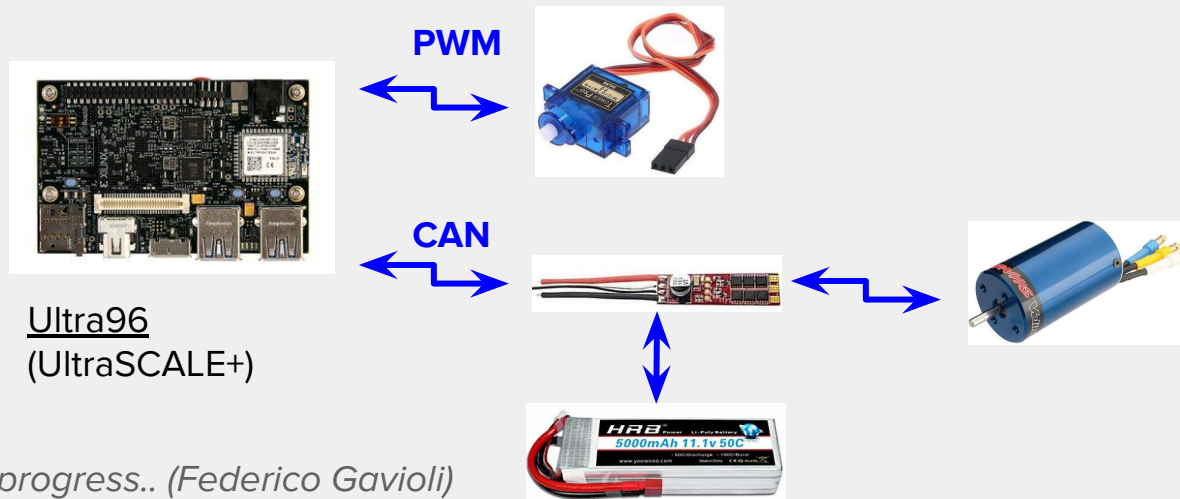
Implementazione del controllo di motori in FPGA

- Schema a blocchi attuale.



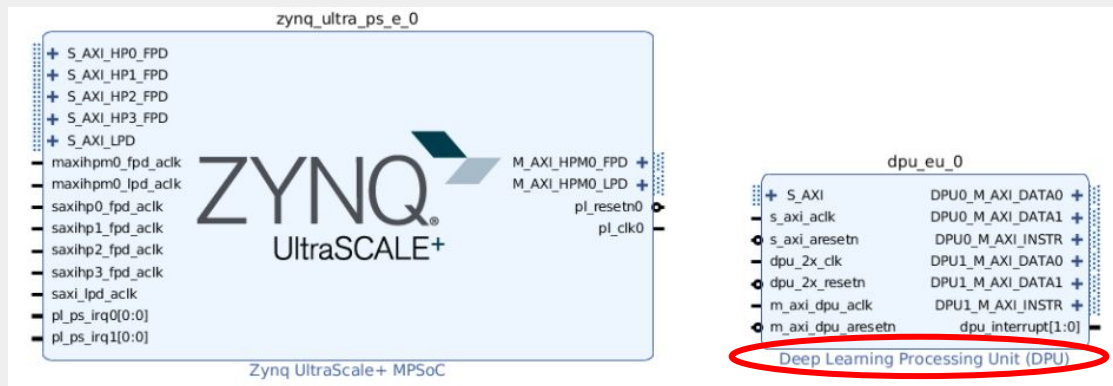
Implementazione del controllo di motori in FPGA

- Rimpiazzamento di tutti i microcontrollori che attualmente controllano i motori e successiva implementazione del controllo in FPGA (Ultra96).



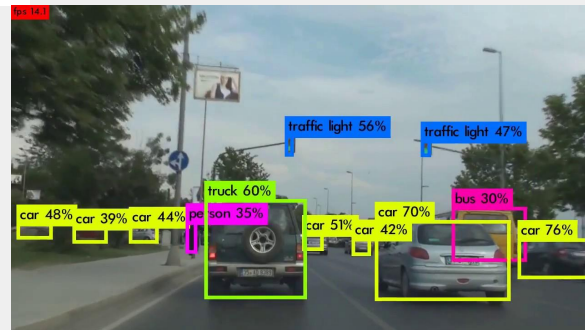
- *Work in progress.. (Federico Gavioli)*

Test di Acceleratori per Reti Neurali in FPGA

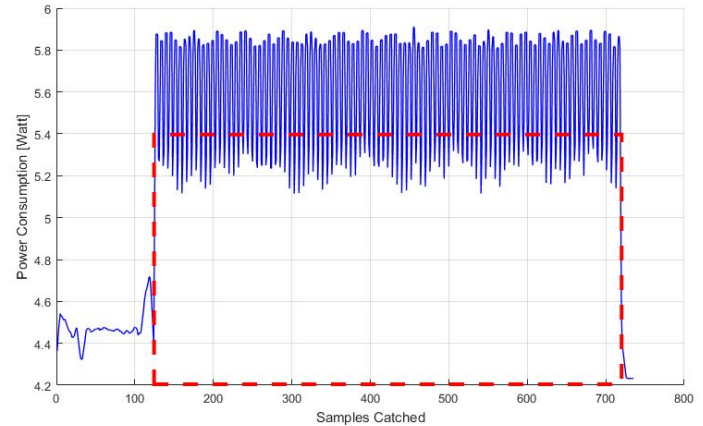


- Test di YOLO su architettura DPU;
- In termini di FPS, mAP e Power.

- Successiva comparazione rispetto alla controparte NVIDIA (NVIDIA Xavier / TX2).



Test di Acceleratori per Reti Neurali in FPGA



Thank you!

**Gianluca Brilli &
Paolo Burgio**

HiPER
Lab
High-Performance Real-Time Lab

