Di seguito è riportata la struttura di massima del FW attuale. La legenda dei colori utilizzati per I blocchi è la seguente:

top_gcu1f3 => BLOCCO che richiederà (probabilmente) una revisione/adattamento
CDR clock input section => BLOCCO che può essere tolto (ma può servire in futuro)
mcp_block => BLOCCO che verrà rimosso definitivamente
picoblaze_top => BLOCCO che (probabilmente) non richiederà una revisione/adattamento

*top_gcu1f3 (src/user/top_gcu1f3.vhd)
*clk_manager (src/user/clk_manager.vhd)
NOTA: può usare il clock interno o quello ricavato dal CDR (clock esterno sincrono).
*clk_wiz (src/user/clk_wiz.vhd)
-PLL (MMCME2_ADV)
(https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf)
clk_0 -> come CLKIN (125MHz)
clk_1 -> 250MHz
clk_2 -> 200MHz
clk_3 -> 100MHz
clk_4 -> 125MHz, sfasato di 90°
clk_5 -> 31.25MHz
*delay_manager (src/user/delay_manager.vhd)
NOTA: incapsula un IDELAYCTRL ipcore
(https://www.xilinx.com/support/documentation/ip_documentation/util_idelay_ctrl/v1_0/pb044-util-idelay-ctrl.pdf)
*CDR clock input section
*process: reset_generator
NOTA: genera un segnale di reset 3 secondi dopo il boot
*process: local_time_generator:
NOTA: conta il main clock (da quarzo esterno) locale, non utilizza BEC
*network_interface (src/user/network_interface.vhd)
NOTA: interfaccia principale di rete e IPBUS core (vedi dopo), contiene tutti gli SLAVE IPBUS
*mcp_block (x3) (src/user/mcp_block.vhd)
NOTA: serve per risincronizzare il segnale pulse_valid tra il dominio di clock interno e quello dell'ADU, instanziato 3 volte (una per ADU)
*shaper (src/user/shaper.vhd)
NOTA: window generator -> prende un segnale in ingresso e genera impulsi lunghi 10 clock
*synchronizer (src/user/synchronizer.vhd)
*edge_detector (src/user/edge_detector.vhd)
*top_level (src/user/synch/top_level_cont.vhd)
NOTA: evito il dettaglio interno, gestisce la sincronizzazione tramite il link dedicato
*adu_top_wrapper (x3) (src/tsinghua/adu_top/src/adu_top_wrapper.vhd)
NOTA: incapsula tutta la logica di interfaccia con le ADU, evito il dettaglio tanto andrà tolta
*dac_config_wrapper (src/tsinghua/adu_top/src/dac_pll_config/dac_config_wrapper.vhd)
NOTA: relativo al funzionamento delle ADU
*picoblaze_top (src/user/picoblaze_top.vhd)
NOTA: incapsula l'interfaccia con la porta UART

STRUTTURA INTERNA DI network_interface

*network_interface (src/user/network_interface.vhd)

- *tri_mod_ethernet_mac_0
  - NOTA: interfaccia Ethernet
- *mac_fifo_axi4
- *process: ipbus_reset
  - NOTA: genera un segnale di ipbus_reset all'avvio (dopo un ritardo di mezzo secondo)
- *ipbus_ctrl (externals/ipbus-firmware/components/ipbus_core/firmware/hdl/ipbus_ctrl.vhd)
  - NOTA: interfaccia principale IPBUS, non andrà modificata.
- *USER_ACCESSE2 (IPCORE)
  - NOTA: contiene il MAC-Address della scheda, settato in create_bitstream.tcl
- *ipbus_shim (externals/ipbus-firmware/components/ipbus_core/firmware/hdl/ipbus_shim.vhd)
  - NOTA: fa da interfaccia tra IPBUS_CTRL e PAYLOAD (lavora sui bus dati IPBUS)
- *payload (src/user/ipbus_utils/payload.vhd)
  - *conversione time_stamp in codifica gray (usata da ADU)
  - *ipbus_subsys (src/user/ipbus_utils/ipbus_subsys.vhd)
    - NOTA: contiene tutti gli slave IPBUS
    - *ipbus_fabric_sel (externals/ipbus-firmware/components/ipbus_core/firmware/hdl/ipbus_fabric_sel.vhd)
      - NOTA: address decoder per IPBUS
    - *tutti gli ipbus_slaves (alcuni verranno rimossi)

NOTA GENERALE:

Tutti gli IPBUS slave sono difatti delle interfacce di Memoria con cui si comunica tramite IPBUS. Il modo in cui queste memorie si interfacciano con i blocchi di codice VHDL interni può essere scelto in base alla convenienza.

Al momento, il codice che implementa la memoria è direttamente inserito nei vari slave, quindi in pratica il blocco network_interface contiene sia l'interfaccia di rete che tutto il codice che utilizza tale interfaccia. In futuro si può pensare di svincolare le due parti.

File di configurazione della memoria IPBUS

Per gestire la memoria IPBUS si deve editare il file xml in uhal_py/gcu1f3.xml. Questo file contiene una descrizione della mappa di memoria e viene utilizzato per generare il pacchetto vhdl situato in src/user/ipbus_utils/ipbus_decode_gcu1f3.vhd

La memoria è organizzata in nodi, ciascun nodo viene caratterizzato dalle sue proprietà principali:

-id : stringa di riferimento per l'accesso al nodo (da software), **NECESSARIA**

-address : indirizzo del nodo di memoria (assoluto oppure relativo al nodo di livello superiore)

-description e tags: stringhe utilizzate per la documentazione e l'assegnazione di tag a livello SW (**totalmente opzionali**)

-fwinfo : informazioni utilizzate per la generazione del codice vhdl **(necessaria per I nodi con sottostruttura);**

-mask : maschera del registro (**NECESSARIA per I nodi che assegnano un alias ai bit di un nodo di ordine superiore)**

-permission : direzionalità del registro ("w", "r", "rw")(**NECESSARIA per tutti I nodi senza struttura interna)**

-mode : modalità di accesso ("single", "non-incremental/port", "incremental/block") **(NECESSARIA per nodi senza struttura interna)**

-size : dimensione del blocco di memoria **(NECESSARIA per nodi di tipo "incremental", OPZIONALE er nodi di tipo "port")**

Ovviamente non tutti i nodi necessitano di tutte le proprietà.

I nodi possono anche essere nested uno dentro l'altro. Vediamo alcuni esempi.

```
<node id="trigger_manager" address="0x800" description="trigger logic configurator" fwinfo="endpoint;width=2" tags="slave0">
        <node id="auto_trigger_mode" address="0x0"  description="pre trigger duration">
                        <node id="force_trigger_ch0" mask="0x00000001" />
                        <node id="force_trigger_ch1" mask="0x00000002" />
                        <node id="force_trigger_ch2" mask="0x00000004" />
                        <node id="trigger_source_selection" mask="0x00000008" />
                        <node id="trigger_path_selection" mask="0x00000010" />
                        <node id="external_trigger_enable" mask="0x00000020" />
                        <node id="trigger_on_threshold" mask="0x00000040" />
                        <node id="global_trigger_enable" mask="0x00000080" />
                </node>
</node>
```

La prima riga definisce un nodo (trigger manager) con base address 0x800.
La proprietà **fwinfo="endpoint;width=2"** identifica il nodo come un endpoint nella struttura ipbus e definisce il numero di bit riservati per gli indirizzi interni al blocco stesso. Questo nodo ha una sua sottostruttura (cioè ha una mappa interna di memoria), dunque non richiede ne la caratteristica **permission (**in quanto non direttamente leggibile/scrivibile) ne la caratteristica **mode**.
La dichiarazione del nodo termina con l'ultima riga.

La seconda riga definisce una locazione di memoria ("**auto_trigger_mode**") interna al nodo, con indirizzo relativo 0x0. Poichè questo nodo non ha una sua mappa di memoria interna, può essere letto/scritto. Non essendo dichiarate le caratteristiche **permission** e **mode** , assumeranno il loro valore di default ("**rw**" e "**single**" rispettivamente). Il nodo è chiuso dalla penultima riga.

Tutte le righe in mezzo definiscono una suddivisione del registro **auto_trigger_mode**, assegnando ai vari bit (secondo la caratteristica **mask**) un ruolo diverso. Questa suddivisione viene implementata solo a livello SW, dando la possibilità di accedere in lettura/scrittura ai singoli bit che costituiscono il registro. Ciascuno di questi nodi è definito inline tramite la chiusura di linea "/>" ed eredita la caratteristica **permission** dal nodo padre (difatti la lettura/scrittura di questi nodi passa attraverso la lettura/scrittura del nodo padre).

```
<node id="ram" address="0x1000" mode="block" size="0x400" description="1kword RAM" fwinfo="endpoint;width=12"/>
```

Qui viene definito un nodo ("**ram**") con base address 0x1000 e modalità "**block**". Questo rappresenta un'area di memoria contigua (di dimensione definita dalla caratteristica **size**). La **width** degli indirizzi di memoria interni deve ovviamente essere compatibile con la dimensione dichiarata.
Il vantaggio di definire un blocco di memoria contiguo sta nel fatto che (lato SW) è possibile fare una lettura multipla con incremento automatico dell'indirizzo.

```
<node id="dsp_loader" address="0x500" fwinfo="endpoint;width=2" >
    <node id="fifo_reset" address="0x0" mode="single" permission="rw"/>
    <node id="fifo_wr" address="0x1" mode="non-incremental" permission="w"/>
    <node id="fifo_rd" address="0x2" mode="non-incremental" permission="r"/>
    <node id="fifo_count" address="0x3" mode="single" permission="r"/>
  </node>
```

Concentrandosi sulle righe 3 e 4, sono definiti due registri (read e write) per accedere ad una memoria di tipo non-incremental , ovvero ad una porta che legge I dati da una memoria. In questo caso non è necessario inserire la proprietà **size**, che difatti serve solo per limitare a livello software il numero di dati contigui che possono essere letti in una singola operazione. Da notare che in questo caso l'implementazione della memoria a livello di VHDL va gestita correttamente, ricordando che il protocollo IPBUS non prevede, per I registri di tipo **port** o **non-incremental** di generare da se gli strobe di lettura e scrittura della memoria.

NOTE IMPORTANTI
* lo script che interpreta il file xml è in grado di rilevare conflitti nell'assegnazione di memoria (e.g base address ripetuti o zone di sovrapposizione tra nodi diversi

*non è però in grado di accorgersi se la **width** dichiarata per un componente è insufficiente a gestire tutti I suoi indirizzi interni

*inoltre sta all'utente verificare che il base address di un blocco abbia a zero tuttti I bit che vengono utilizzati per l'indirizzamento interno

*il pacchetto VHDL che viene prodotto tiene difatti conto solo dei nodi di livello più alto, tutti gli altri vanno poi correttamente implementati nel vhdl nella forma di IPBUS slave

*per ogni nodo caratterizzato dalla caratteristica "**endpoint**", nel pacchetto VHDL verrà aggiunta una entry che definisce un numero intero progressivo che identifica lo slave e che viene utilizzato nel VHDL per indirizzarlo (vedere ad esempio il file src/user/ipbus_utils/ipbus_subsys.vhd).

*tutta la struttura di memoria è incapsulata dentro un nodo di livello 0 (al momento si chiama "GCU").