100 Gbps/40 Gbps PCS/PMA + MAC IP Core

Getting started guide:

1. Prerequisites:

In order to simulate and implement Aitia's 100 Gbps / 40 Gbps Ethernet PCS/PMA + MAC IP core you must meet the following requirements:

- PC with at least 8 GB RAM is needed (simulation usually takes 4-5 GB)
- Xilinx ISE logic edition with support for the targeted FPGA
- implement: compatible Virtex-6 or 7 series FPGA device (refer to Xilinx Coregen for CAUI-10 compatible devices; in default GTH serdeses are supported)

2. Installing the core:

Directory structure of the IP core bundles:

– SRC → CG_PCSPMA (100G) → GTH : Virtex specific transciever instantiation example
→ XLG_PCSPMA (40G) → GTH : Virtex specific transciever instantiation example
→ IP core sources, NGC, or HDL model for simulation depending on the license type selected
→ Common : contains common design components
→ sim : contains simulation testbenches
– DOC : contains documentation
– PRJ : contains example ISE projects for simulation or implementation

The included serdes (pcs_phy_xxG) module wrapper is for XC6VHX255T-2FF1155 Virtex-6 FPGA, implemented and field tested in C-GEP 100 devices.

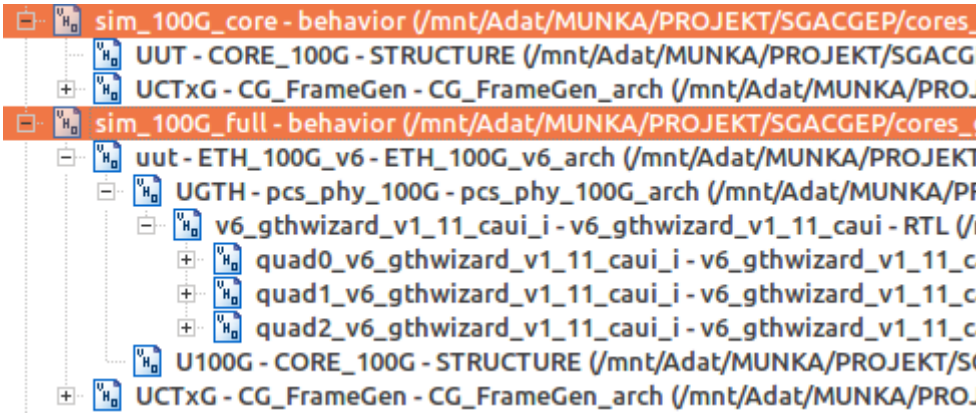
In order to instantiate the IP cores you have three possibilities according to the licensing conditions:

a) simulation only model:

After opening the ISE project, you can simulate the behavioural mode of the free to use IP core samples in two testbenches as shown below:

- sim_xxG_core: simulate the 100G / 40G core only.
- sim_xxG_full: simulate the 100G / 40G core attached to a Virtex device specific transciever (GTH)

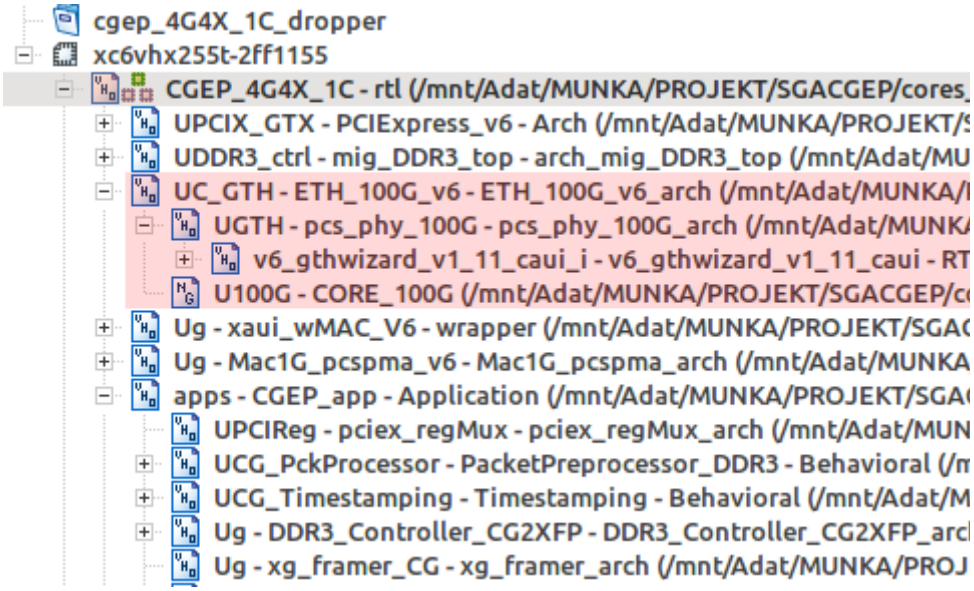
In both cases the core is feeded by a simple frame generator. The generated frames contain the number of the transmitted frame, and a running word-counter for debug purposes.



Behavioural simulation testbenches for the 100G core

b) Implementable IP Core:

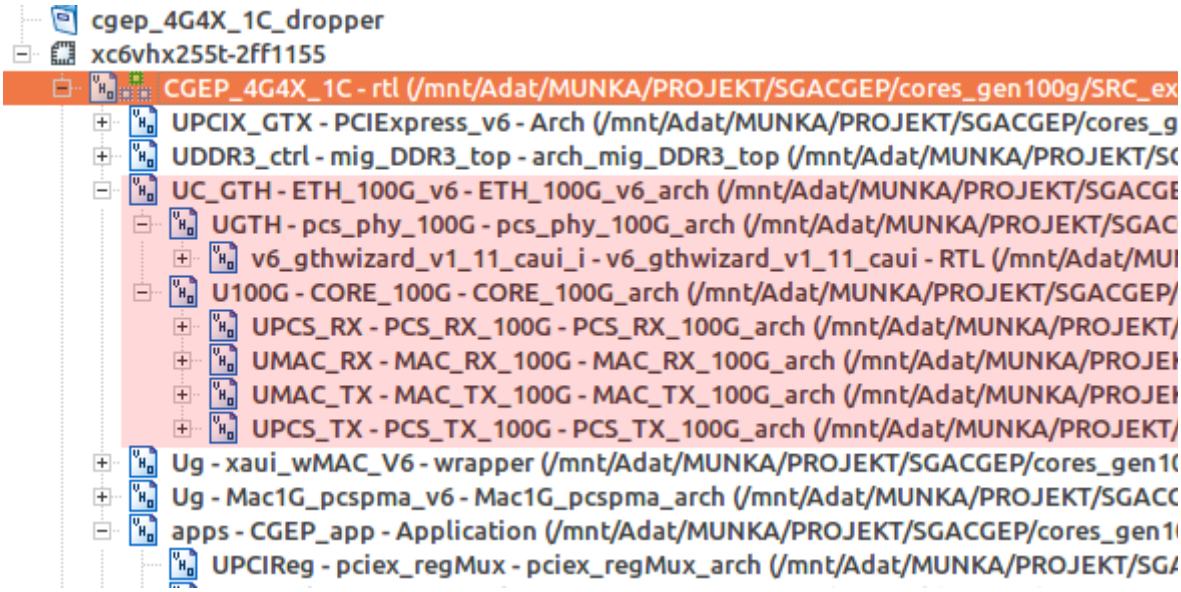
In this case we provide you a Xilinx specific pre-synthetised binary file (NGC) of the core. This contains the gate-level description of the IP which will be used in the implement phase of the ISE project flow. We provide you basic UCF implementation constraints for Virtex-6 FPGA-s.



Binary instantiation example of the 100G core

c) Source code format IP core:

This contains all the modules of the IP core including the Receive, Transmit PCS/PMA and MAC layers and CRC check and generator components. You are free to use, modify or port the source code to different FPGA-s as you like. We provide you basic UCF implementation constraints for Virtex-6 FPGA-s.



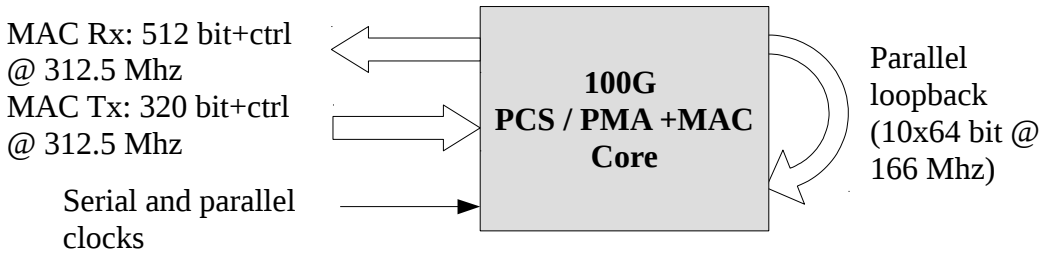
Source code instantiation of the 100G IP core

3. Simulating the 100G IP core

We provide you a basic simulation enviroment for the 100G PCS/PMA and MAC core.

The two provided test benches simulate:

a) the standalone 100G IP core in loopback on the parallel PMA interfaces without the serdes modules:



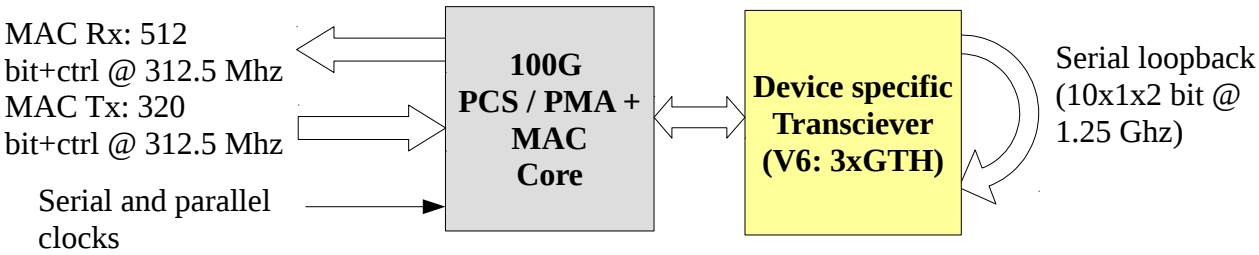
Standalon IP core simulation model

In this case the clock signals are generated from the testbench by previously defined constrains:

- GTH parallel clock has a period of 6.2 ns [156.25x(66/64) MHz]
- Tx/Rx MAC clocks have a period of 3.2 ns [312.5 MHz]
- (bit clock for serial testing has a period of 96.97 ps [156.25x(66 bit) MHz]

The parallel 10x64 bit outputs of the core are in direct loopback.

b) the whole 100G IP core including device specific (Virtex 6) GTH serdeses in serial loopback



IP core simulation model including device specific transcievers

In this case the only generated clock signal is the reference clock for the GTH serdeses, the Rx/Tx MAC clock is derived from the GTH parallel clock:

- GTH parallel clock has a period of 6.2 ns [156.25x(66/64) MHz]
- Tx/Rx MAC clocks have a period of 3.2 ns [312.5 MHz]

The serial 10x2 bit differential outputs of the serdeses are in direct loopback.

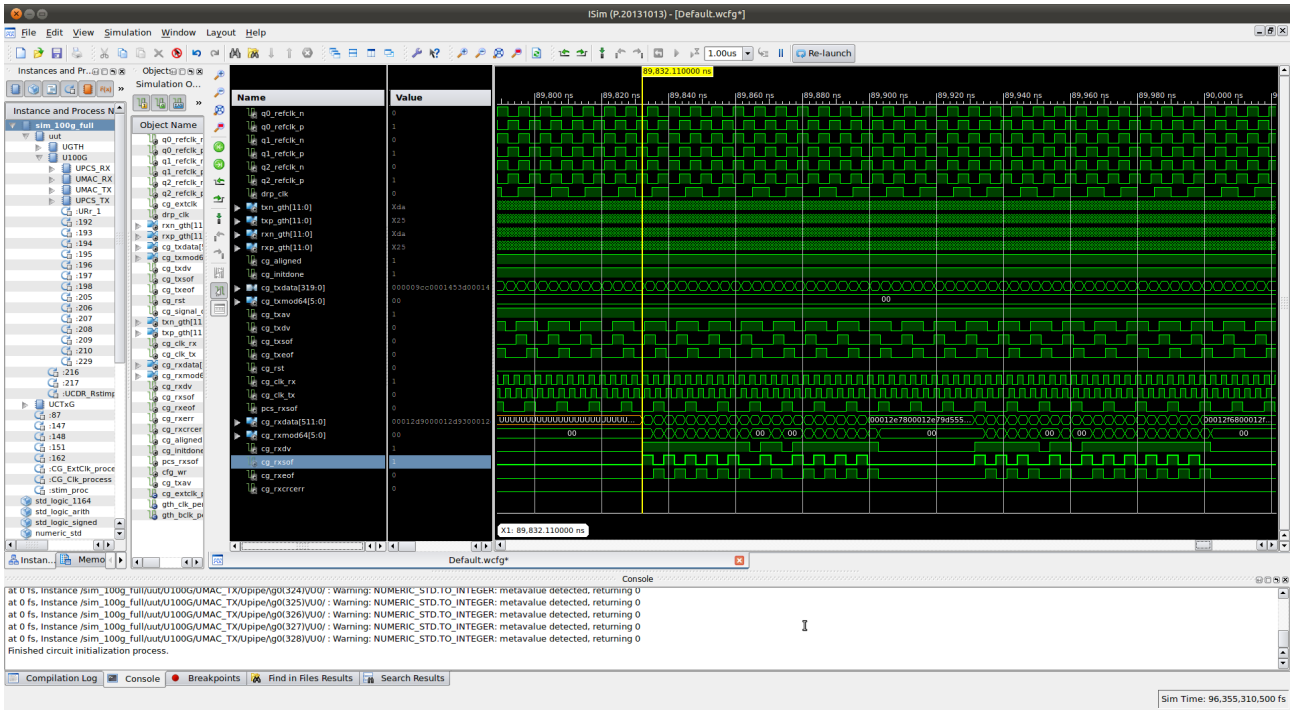
(c) A third simulation model is also available upon request for validation purposes, where we generate serialized and encoded vectors from software which can be directly fed to the GTH receiver.

Basic simulation states:

0. Reset: All modules and internal states are brought into initial state
1. Wait for serdes initialization: (CG_initdone) available only in full sim mode; indicates, that the GTH is initialized and operational
2. Wait for lane lock: (CG_locked66b) indicates, that all lanes have reached 64b66b code-locking
3. Wait for lane alignment: (CG_aligned) indicates, that all lanes are synchronized to each other by proper marker detection.
4. The core is ready to receive/send valid ethernet frames on the MAC interface

Traffic is injected on the Tx-MAC interface by a frame generator module (CG_Framegen) with static frame length and interframe-space values. After the core is done with initialization, the frames are presented on the receive MAC side, and checked for data reception and code errors (ethernet CRC checksum, and Rx Error indicator).
The testbench also has a MACRx_Loss signal which toggles high for one clock cycle when a frame loss is detected by comparing a frame counter to the one contained in the received frame, which is stored in the 10th dword in the first data nibble.

The following picture shows a fully initialized state with frame transmission and reception in progress in the Xilinx iSim program.
Note, that the receive data-valid signals frequency is different than the one on the transmit side, because the transmit interface is 5*64 bits wide, while the receive side is 8*64 bits wide. This is necessary to keep alignment of every frames first byte on the first MAC byte position.



Simulating the IP core in iSim

4. Interfacing to the IP core:

The following list contains the most important IP core signals:

```
    CG_Clk_RX           : OUT STD_LOGIC;
    CG_Clk_TX           : OUT STD_LOGIC;
```

Two independent MAC clocks are synthetized from the PCS clocks by the serdes wrapper for RX and TX. All user logic acts on the rising edge of those clocks.

```
    CG_RXData           : OUT STD_LOGIC_VECTOR(8*64-1 downto 0);
```

Receive Data from Rx-MAC. Ethernet frames always start on the first (LSB) byte, data is presented after the starting-frame-delimiter, from the ethernet destination MAC address.

- 100G: 8*64 bits
- 40G: 5*64 bits

```
    CG_RXMod64          : OUT STD_LOGIC_VECTOR(6-1 downto 0);
```

This shows the number of valid bytes at the end of the frame.
0 means all bytes are valid, b”111111” means 511 bytes are valid.

- 100G: 0 – 63
- 40G: 0 – 39

```
    CG_RXDv             : OUT STD_LOGIC;
```

Data valid signal: RXData, RXSof, RXEof is only valid when this signal is in high-state.
This signal can change into low state during frame reception. This can be prevented by a single frame-buffer module which is provided to you in every paid license.

```
    CG_RXSof            : OUT STD_LOGIC;
```

Start of frame signal: This indicates the first valid data-nibble in an ethernet frame.

```
    CG_RXEof            : OUT STD_LOGIC;
```

End of frame signal: This indicates the last valid data-nibble in an ethernet frame. (RXMod64 is only valid when this signal is in high-state)

```
    CG_RXCRCErr         : OUT STD_LOGIC;
```

Checksum error signal: The indicates, that calculated and received ethernet checksums don't match.

```
    CG_RXErr            : OUT STD_LOGIC;
```

Receive error signal: This indicates a far-end receive error, or and invalid 64b/66b code.

```
    CG_TXAv              : OUT STD_LOGIC;
```

Transmit buffer available: This indicates, that the transmit buffer is ready to accept an ethernet frame for transmission (60 – 16kbyte in length)

```
    CG_TXData            : IN  STD_LOGIC_VECTOR(8*64-1 downto 0);
```

Transmit Data for 100G Tx-MAC. Ethernet frames must always start on the first (LSB) byte. Data must be provided from the ethernet destination MAC address, ethernet CRC is generated by the IP core itself.

- 100G: 5*64 bits
- 40G: 4*64 bits

```
    CG_TXMod64           : IN  STD_LOGIC_VECTOR(6-1 downto 0);
```

This shows the number of valid bytes at the end of the frame. 0 means all bytes are valid, b”100111” means 319 bytes are valid. (It is also possible to use the Tx-MAC in 512/320bit mode)

- 100G: 0 – 39
- 40G: 0 – 31

```
    CG_TXDv              : IN  STD_LOGIC;
```

Data valid signal: The Tx-MAC only accepts input data, when this signal is in high-state.

```
    CG_TXSof             : IN  STD_LOGIC;
```

Start of frame signal: This indicates the first valid data-nibble in an ethernet frame.

```
    CG_TXEof             : IN  STD_LOGIC;
```

End of frame signal: This indicates the last valid data-nibble in an ethernet frame. (TXMod64 is only valid when this signal is in high-state)

```
    PCS_RXSof            : OUT STD_LOGIC;
```

PCS start of frame: for debug purposes, and precise timestamping.

```
    CG_Rst               : IN  STD_LOGIC;
```

Reset signal: This signal resets the user portion of the 100G core (fabric reset)

```
    CG_initdone          : OUT STD_LOGIC;
```

GTH initialization complete, core is ready.

```
    CG_locked            : OUT STD_LOGIC;
```

All virtual lanes are locked on the 66b code level.

```
    CG_aligned          : OUT STD_LOGIC;
```

All virtual lanes are aligned and locked, core is ready to receive frames.

5. Constraints

These are the main timing and placement constraints used for implementing a physical FPGA design.
Additional block placement constraints can be defined for partitioning the design, or for partial reconfiguration.

```
INST "RCKKLP"       LOC = "P6";
INST "RCKKLN"       LOC = "P5";
[...]
```

Differential reference clock inputs for the transcievers (XCO @ 156.25 MHz by default).

```
# GTH placements
INST
"UC_GTH/UGTH/v6_gthwizard_v1_11_caui_i/quad0_v6_gthwizard_v1_11_ca
ui_i/gthe1_quad_i"      LOC = GTHE1_QUAD_X1Y0;
INST
"UC_GTH/UGTH/v6_gthwizard_v1_11_caui_i/quad1_v6_gthwizard_v1_11_ca
ui_i/gthe1_quad_i"      LOC = GTHE1_QUAD_X1Y1;
INST
"UC_GTH/UGTH/v6_gthwizard_v1_11_caui_i/quad2_v6_gthwizard_v1_11_ca
ui_i/gthe1_quad_i"      LOC = GTHE1_QUAD_X1Y2;
```

Transciever placement for the specific design.

```
# GTH Tx PLL clock
NET "UC_GTH/clk_GTH_tx" TNM_NET = "clk_GTH_tx";
NET "UC_GTH/clk_GTH_tx" TNM_NET = "clk_GTH_tx_tig";
TIMESPEC "TS_clk_GTH_tx" = PERIOD "clk_GTH_tx" 6206 ps;
# GTH Rx recovered clock
NET "UC_GTH/clk_GTH_rx" TNM_NET = "clk_GTH_rx";
NET "UC_GTH/clk_GTH_rx" TNM_NET = "clk_GTH_rx_tig";
TIMESPEC "TS_clk_GTH_rx" = PERIOD "clk_GTH_rx" 6206 ps;
```

Clock frequency specifications for the serdes reference output, and for the CDR recovered clocks.

```
# Rx MAC clock synthetised by MMCMs (312.5MHz) [auto-constrains by
tools]
NET "UC_GTH/clk_MAC_RX" TNM_NET = "clk_MAC100G_RX";
NET "UC_GTH/clk_MAC_RX" TNM_NET = "clk_MAC100G_RX_tig";
TIMESPEC "TS_clk_MAC100G_RX" = PERIOD "clk_MAC100G_RX" 3200 ps;
# Tx MAC clock synthetised by MMCMs (312.5MHz) [auto-constrains by
tools]
NET "UC_GTH/clk_MAC_TX" TNM_NET = "clk_MAC100G_TX";
NET "UC_GTH/clk_MAC_TX" TNM_NET = "clk_MAC100G_TX_tig";
TIMESPEC "TS_clk_MAC100G_TX" = PERIOD "clk_MAC100G_TX" 3200 ps;
```

Synthetised clocks for the receive and transmit MAC interfaces.