Nano Scale Disruptive Silicon-Plasmonic Platform for Chip-to-Chip Interconnection

Plasmonic interconnect VHDL modeling

Milestone no.:	MS6
Due date:	10/31/2013
Actual Submission date:	11/15/2013
Authors:	ST
Work package(s):	WP2
Distribution level:	RE[1] (NAVOLCHI Consortium)
Nature:	document, available online in the restricted area of the NAVOLCHI webpage

List of Partners concerned

Partner number	Partner name	Partner short name	Country	Date enter project	Date exit project
1	Karlsruher Institut für Technologie	KIT	Germany	M1	M36
2	INTERUNIVERSITAIR MICRO-ELECTRONICA CENTRUM VZW	IMEC	Belgium	M1	M36
3	TECHNISCHE UNIVERSITEIT EINDHOVEN	TU/e	Netherlands	M1	M36
4	RESEARCH AND EDUCATION LABORATORY IN INFORMATION TECHNOLOGIES	AIT	Greece	M1	M36
5	UNIVERSITAT DE VALENCIA	UVEG	Spain	M1	M36
6	STMICROELECTRONICS SRL	ST	Italy	M1	M36
7	UNIVERSITEIT GENT	UGent	Belgium	M1	M36

[1] **PU** = Public
PP = Restricted to other programme participants (including the Commission Services)
RE = Restricted to a group specified by the consortium (including the Commission Services)
CO = Confidential, only for members of the consortium (including the Commission Services)

Deliverable Responsible

Organization:	STMicroelectronics
Contact Person:	Alberto Scandurra
Address:	Stradale Primosole, 50 – 95121 Catania Italy
Phone:	+39 095 740 4432
Fax:	+39 095 740 4008
E-mail:	alberto.scandurra@st.com

Executive Summary

This document describes the VHDL model of the plasmonic interconnect. It starts with a brief introduction to the approach used for developing the models and with a general description of the system as a whole, then a description of the behavioural models of each building-block is given.

Change Records

Version	Date	Changes	Author
0.1 (draft)	2013-11-05	Start	Alberto Scandurra, Valentina Cernuto
1 (submission)	2013-11-15	Final version	Alberto Scandurra, Valentina Cernuto

Contents

1. Introduction

The approach followed for the VHDL modelling of the Plasmonic Interconnect is of top-down type, that is to say starting from a specification at high-level of abstraction, where it is expected to use some components properly connected between them, each individual component is specified and modelled individually.

The top-level design consists of the following seven blocks connected in cascade:

- the modulator driver
- the laser
- the plasmonic waveguide
- the plasmonic amplifier
- the plasmonic photodetector
- the TIA (Trans-Impedance Amplifier)
- the CRC (Clock Recovery Circuit)

The block diagram, shown in the following figure, reflects the structural description of the *plasmonic_interconnect* entity as reported in the *plasmonic_interconnect_ent.vhd* file; this entity also includes the analog blocks, present in the transmitter and in the receiver, and the digital Clock Recovery block.
In vhd language an *entity* describes a black box, defining its inputs and outputs together with their type but it gives none information about the internals of the circuit. The uppermost level of the design is the *top-level entity* which contains lower-level entities: it's a *hierarchical design* as above. The function and/or structure of an entity is described by one or multiple *architectures* [1][2].
The transmitted data through the plasmonic interconnect is a 30 bits serial data and the proposed VHDL model concerns a single bus of the four that are present in Navolchi demonstrator.

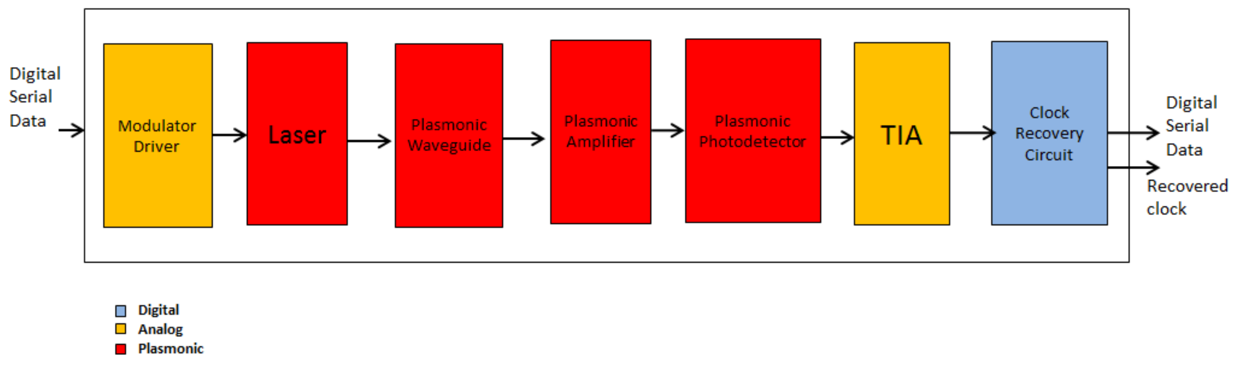

Figure 1-1 : Plasmonic Interconnect Block Diagram (Top-level)

In the plasmonic interconnect VHDL model we used a REAL type for the current and optical power representations. Like this it's possible to represent numbers out of the range of integer values as well as fractional values.

2. Modulator Driver

This block has the task to generate the current levels in order to modulate properly the light emitted by the laser, starting by the stream of bits which have to be transported through the plasmonic interconnect.
The *modulator_driver* entity has a *std_logic* type as input (named *dig_in*) and a REAL type as output (named *analog_out*). It is shown in the following figure.

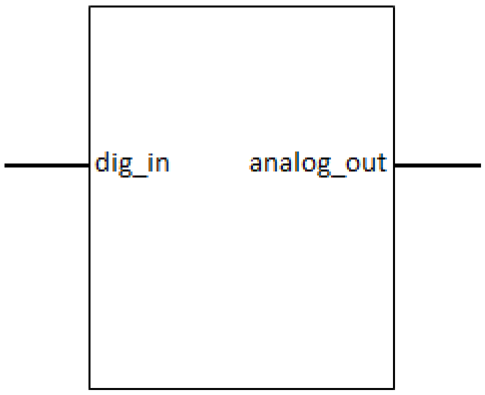

Figure 2-1 : Modulator Driver Entity

In the architecture of *modulator_driver* entity we declared three CONSTANT values which represent the current generated according to the digital input which can be a logic '1', a logic '0' or simple noise, i.e. input signal absence ('U')[2].
VHDL has many possible representations for similar functionality. To describe the functionality of a *modulator_driver* we used a *process statement*: an event on *dig_in*, which represents its *sensitivity list*, causes the process statement to be executed.
The *modulator_driver* behavioural model is summarized in the following table.

dig_in	analog_out
'1'	I_{high}
'0'	I_{low}
'U'	I_{dark}

Table 2-1 : Behavioural Table

[2] An input signal not explicitly initialized takes on the most left value of its type and for the *std_logic* type it is 'U'.

3. Laser

This block acts as a current-to-optical power converter, taking the current generated by the modulator driver and generating a proper optical power.
The *laser* entity has a REAL type as input (named *current_in*) and a REAL type as output (named *power_out*). It is shown in the following figure.

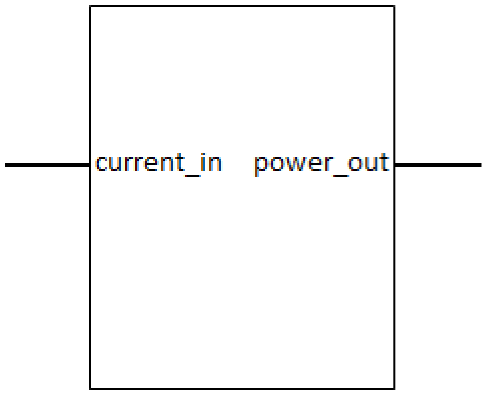

Figure 3-1 : Laser Entity

Also in this case, in the architecture of the *laser* entity we defined the reference values for currents and optical powers as CONSTANT and the behavioural model is described by following table.

current_in	power_out
$\geq I_{high}$	P_{high}
$> I_{dark}$	P_{low}
other values	P_{dark}

Table 3-1 : Behavioural Table

4. Plasmonic Waveguide

This block has the task to transmit the optical power.
The *plasmonic_waveguide* entity has a REAL type as input (named *power_in*) and a REAL type as output (named *power_out*). It is shown in the following figure.

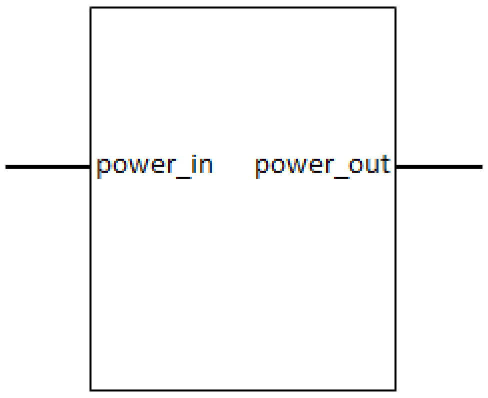

Figure 4-1 : Plasmonic Waveguide Entity

Considering the signal delay and the energy dissipation per transmitted bit introduced by the plasmonic waveguide, we defined a *loss* ($G < 1$) and a *delay* as CONSTANT values and so the output signal is reduced in amplitude and retarded compared to input signal.

5. Plasmonic Amplifier

Plasmonic devices are lossy and the task of the amplifier block is to amplify the input optical power.
The *plasmonic_amplifier* entity has a REAL type as input (named *power_in*) and a REAL type as output (named *power_out*). It is shown in the following figure.

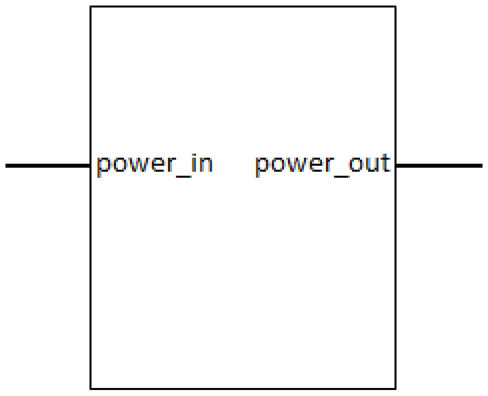

Figure 5-1: Plasmonic Amplifier Entity

In the architecture of *plasmonic_amplifier* entity we defined a *gain* (G >1) as CONSTANT value.

6. Plasmonic Photodetector

This block acts as an optical power-to-current converter, taking the optical power generated by the plasmonic amplifier and generating a current representing a high level optical power, a low level optical power or simple noise (i.e. input signal absence).
The *plasmonic_photodetector* entity has a REAL type as input (named *light_power_in*) and a REAL type as output (named *current_out*). It is shown in the following figure.

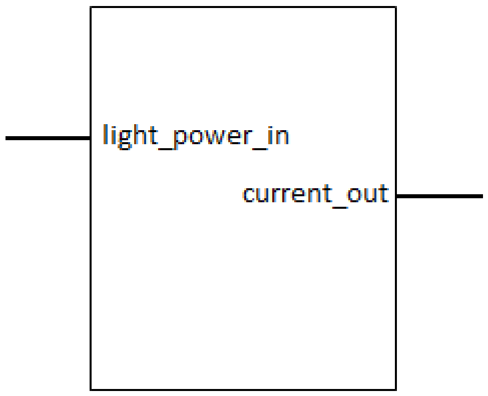

Figure 6-1 : Plasmonic Photodetector Entity

In the architecture of *plasmonic_photodetector* entity we defined the reference values for currents and optical powers as CONSTANT.
The plasmonic photodetector behavioural model is summarized in the following table.

light_power_in	current_out
$\geq P_{high}$	I_{high}
$> P_{dark}$	I_{low}
other values	I_{dark}

Table 6-1 : Behavioural Table

7. TIA (Trans-Impedance Amplifier)

This block acts as a current-to-voltage converter, taking the photocurrent generated by the plasmonic detector and generating a proper voltage, representing a logic ‘0’, a logic ‘1’ or simply noise (i.e. no detection).
The *TIA* entity has a REAL type as input (named *current_in*) and a std_logic type as output (named *voltage_out*). It is shown in the following figure.

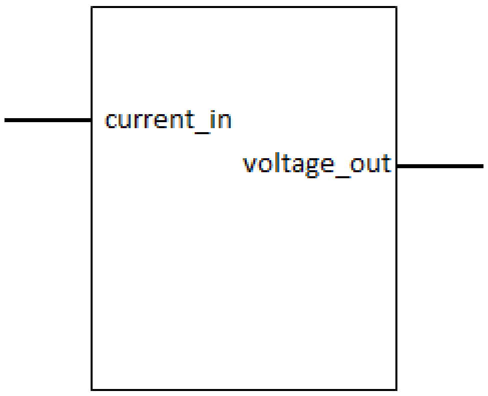

Figure 7-1 : TIA Entity

In the architecture of *TIA* entity we defined the reference currents as CONSTANT values. The digital output is ‘1’, ‘0’ or ‘U’ in accordance with the following table which summarizes the behavioural model of the TIA.

current_in	voltage_out
$\geq I_{high}$	‘1’
$> I_{dark}$	‘0’
other cases	‘U’

Table 7-1 : Behavioural Table

8. CRC (Clock Recovery Circuit)

Serial data stream could have multiple interpretations in the Receiver as shown in the following figure.

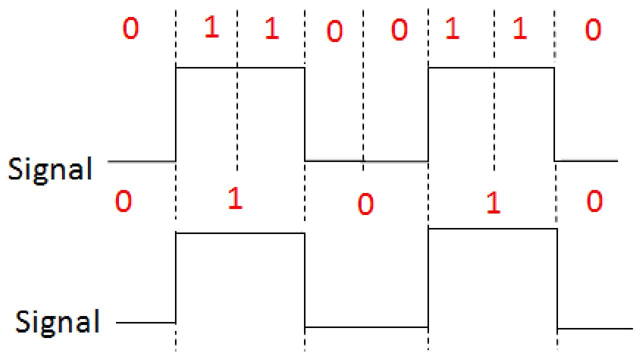

Figure 8-1: Example of double interpretation of one and only signal

There is need to recover the clock from the data stream and to use it to sample the data in order to extract the individual data bits. This task is carried out by a *Clock Recovery Circuit*. It allows the Receiver to regenerate serial data with the fewest bit errors and to obtain a correct interpretation of itself.
To describe *crc* entity we used a structural model in which some subcomponents, that perform smaller operations of the complete model, are instantiate [1].
This description uses three lower-level components to model the behavior of the CRC:

- an edge detection circuit component;
- a counter component;
- a D-flip flop component.

The *crc* top-level is shown in the following figure.

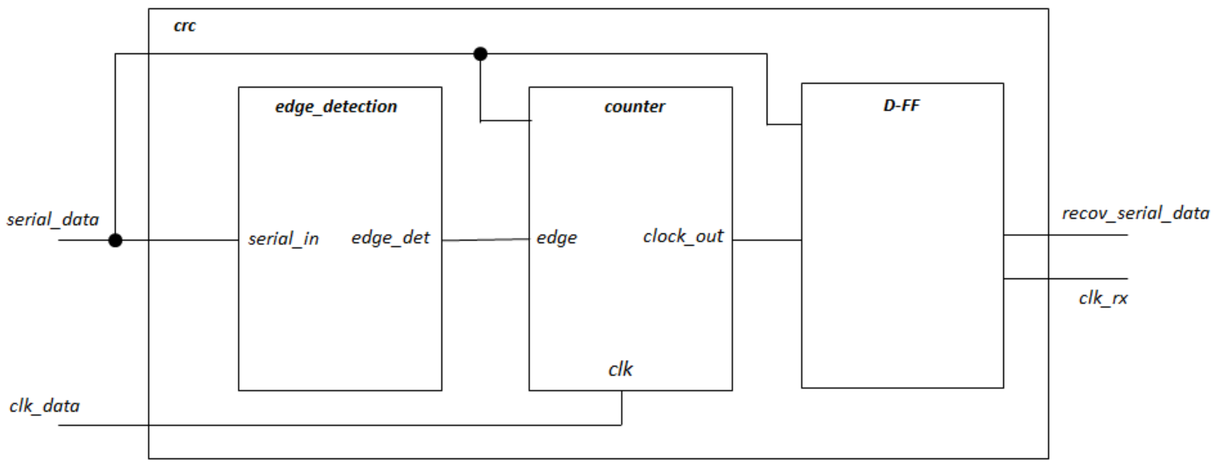

Figure 8-2 : CRC Top-level

Its inputs are:

- *serial_data* declared as *std_logic* type;
- *clk_data* declared as *std_logic* type. It's the clock of transmitted serial data.

Its outputs are *recov_serial_data* and *clk_rx*, declared as *std_logic* types.
The *edge_detection* entity has a *std_logic* type as input (named *serial_in*) and a *std_logic* type as output (named *edge_det*). Its functionality, described in the architecture of *edge_detection* entity, is to generate a pulse when '0' to '1' or '1' to '0' transition of the incoming data occurs. Edge detection circuit is similar to a Phase Locked Loop (PLL) device detecting Phase Changes [3].
To describe this block we used a structural model consisting of three INVERTER gates, a NAND gate e two D-flip flop.
The edge detection circuit top-level is shown in the following figure.

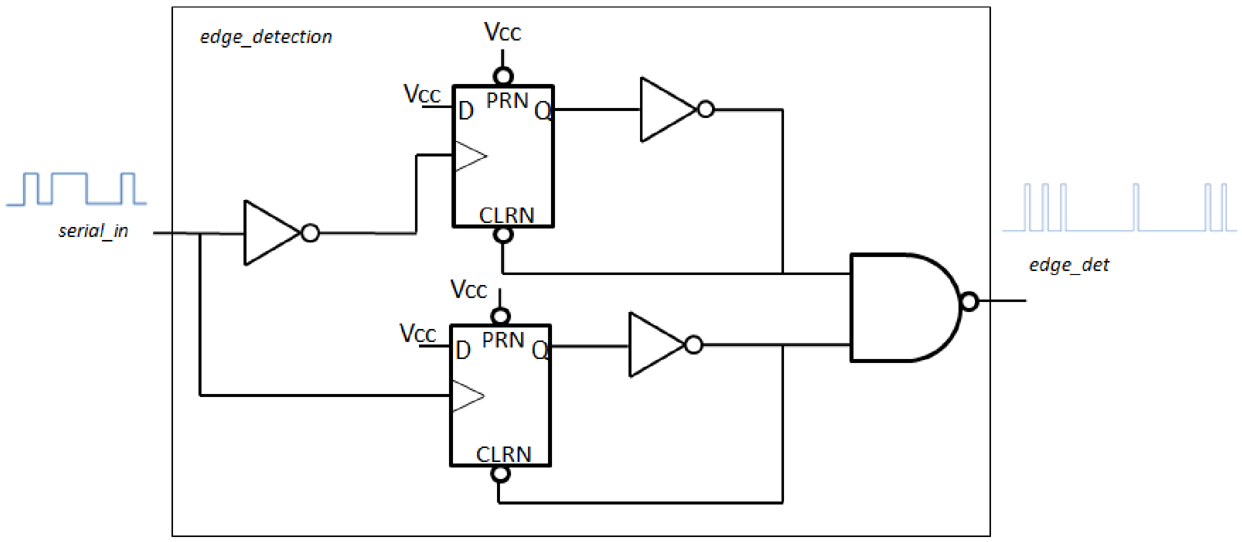

Figure 8-3 : Edge Detection Circuit Entity (Top-level)

The upper flip-flop generates a pulse from a negative transition of the incoming data, while the lower generates a pulse from a positive transition. The logic high propagates to the flip-flop's Q output pin, then through the inverter gate and back to the flip-flop's asynchronous clear pin (CLRN) which will asynchronously reset the flip-flop but only after the combined propagation delay of the inverter and the flip-flop has occurred [3]. Pulse width depends on this delay.
The second block contains a 3 bit counter whose clock signal is approximately 8 times the data rate and the MBS of its output is used as the Sampling Point of data. This is approximately in the centre of the data bit. Moreover, an input transition gets reset the counter.
The *counter* entity has two *std_logic* types as inputs (respectively named *edge* and *clk*), a *std_logic* type as output (named *clk_out*).

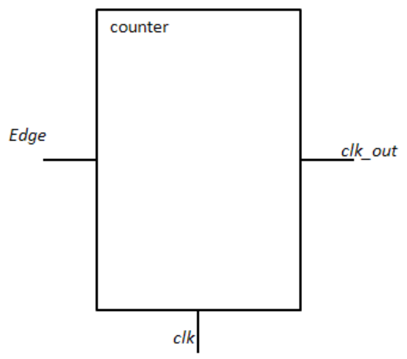

Figure 8-4 : Counter Entity

If an edge signal occurs, the counter is reset early. If there are successive bits of the same states, the counter *free-runs* and continues to sample the data correctly [3]. It is shown in the following figure.

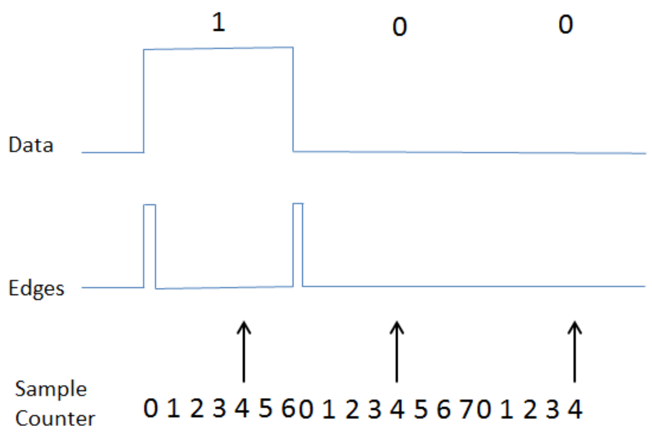

Figure 8-5: Example of the counter functionality

Then the output of Sampling Circuit is the clock signal for a D-flip flop, which is the last block, to regenerate the real content of the data as shown in the following figure [3].

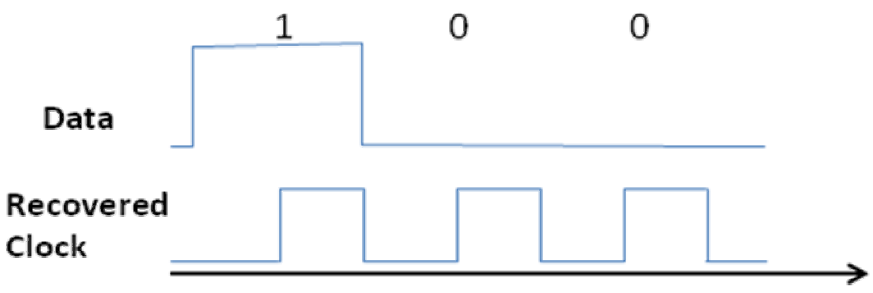

Figure 8-6: Example of a CRC output signal

9. Conclusion

The above document presents a VHDL behavioural model of a Plasmonic Interconnect. Traditionally to recover the clock from a serial data stream uses Phase Locked Loop (PLL) [3] but, in order to be implemented with FPGA, we proposed a purely Digital Method of Clock Recovery. In particular, it's the same used inside all Universal Asynchronous Receiver Transmitter (UARTs[3]) devices. The next step of the work will be to run simulations of these VHDL models.

[3] They traditionally have a Sampling Clock that is 16 Times the Data Rate [3].

References

[1] VHDL: Programming by Example. Douglas L. Perry, McGraw-Hill
[2] Digital System Design with VHDL, Mark Zwolinski, Pearson Prentice Hall
[3] http://www.twyman.org.uk/clock_recovery/