Hardware Layer in the VISQ Project
— Initial Specifications —

Jean-Luc DANGER & Sylvain GUILLEY
<{jean-luc.danger, sylvain.guilley}@enst.fr>

GET / Télécom Paris
CNRS – LTCI (UMR 5141)

Thursday February 28th, 2006.

Room "A310", ENST, Paris, France.

Presentation Outline

"Classical Hardware" Requirements

Pieces of Hardware

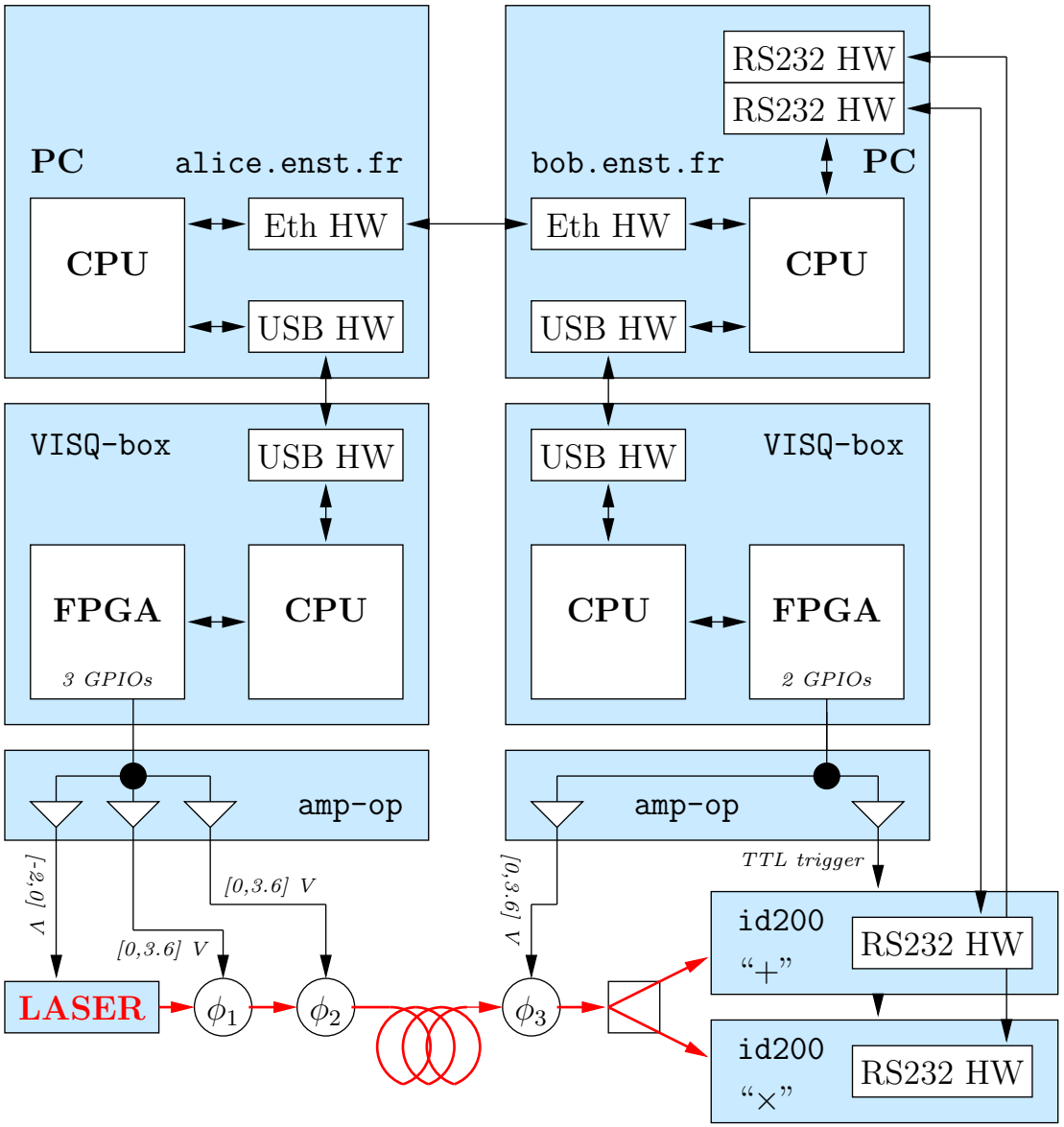
- **Digital** → **Analog** converters to control the quantum apparatus:
  - **Operational amplifiers** (*op-amp*), supplied by various voltages and installed on a daughterboard
- A **general-purpose** and **real-time** circuit to generate the optical stimuli:
  - An **FPGA** (*Field Programmable Gate Array*)
- A **fast** link to a **PC** (*to begin with*):
  - An **RS232/USB** serial port
- Probably some **computational power**, to pilot the PC link:
  - A **microprocessor**

The VISQ-box motherboard

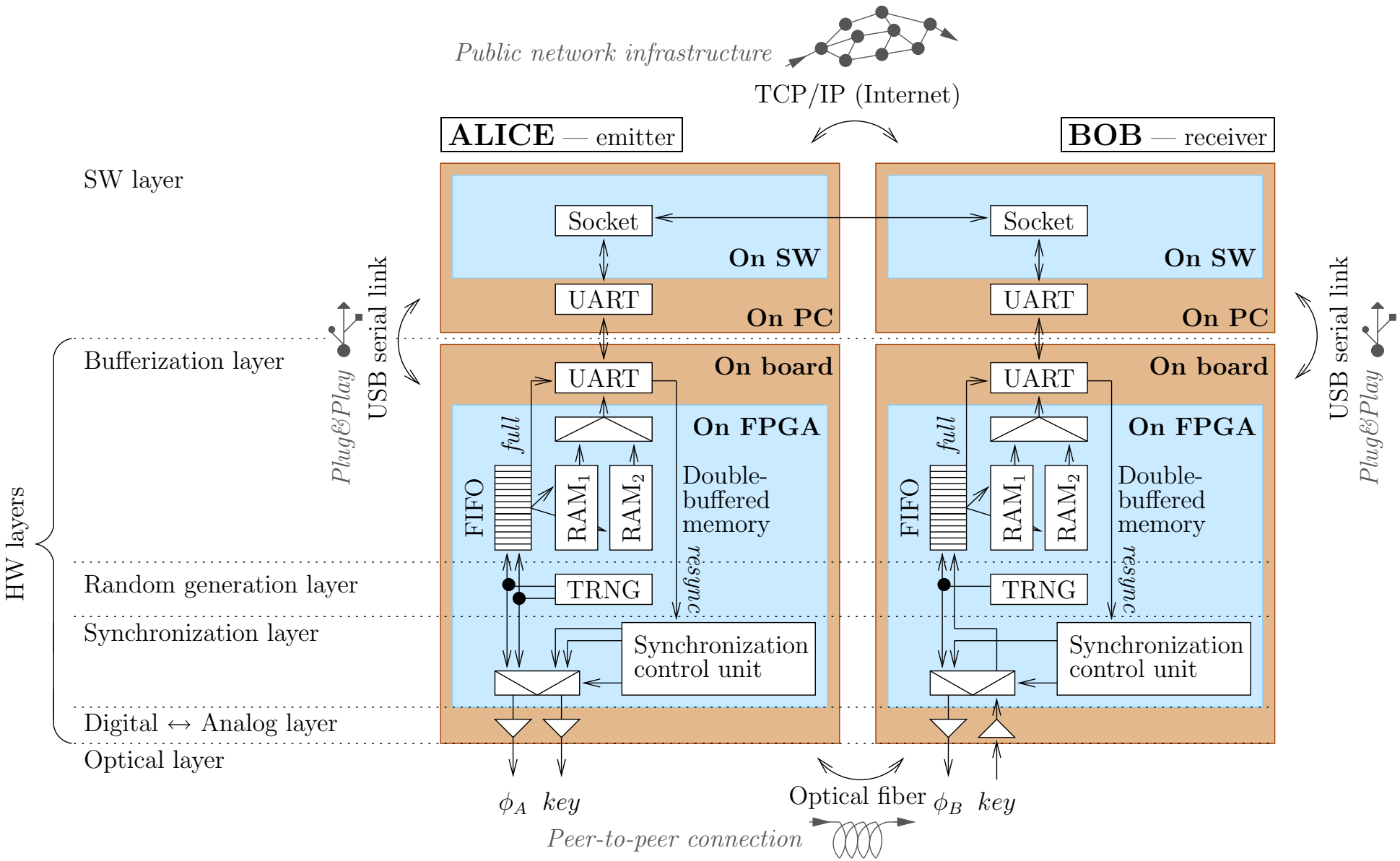
- A board developed at the ENST

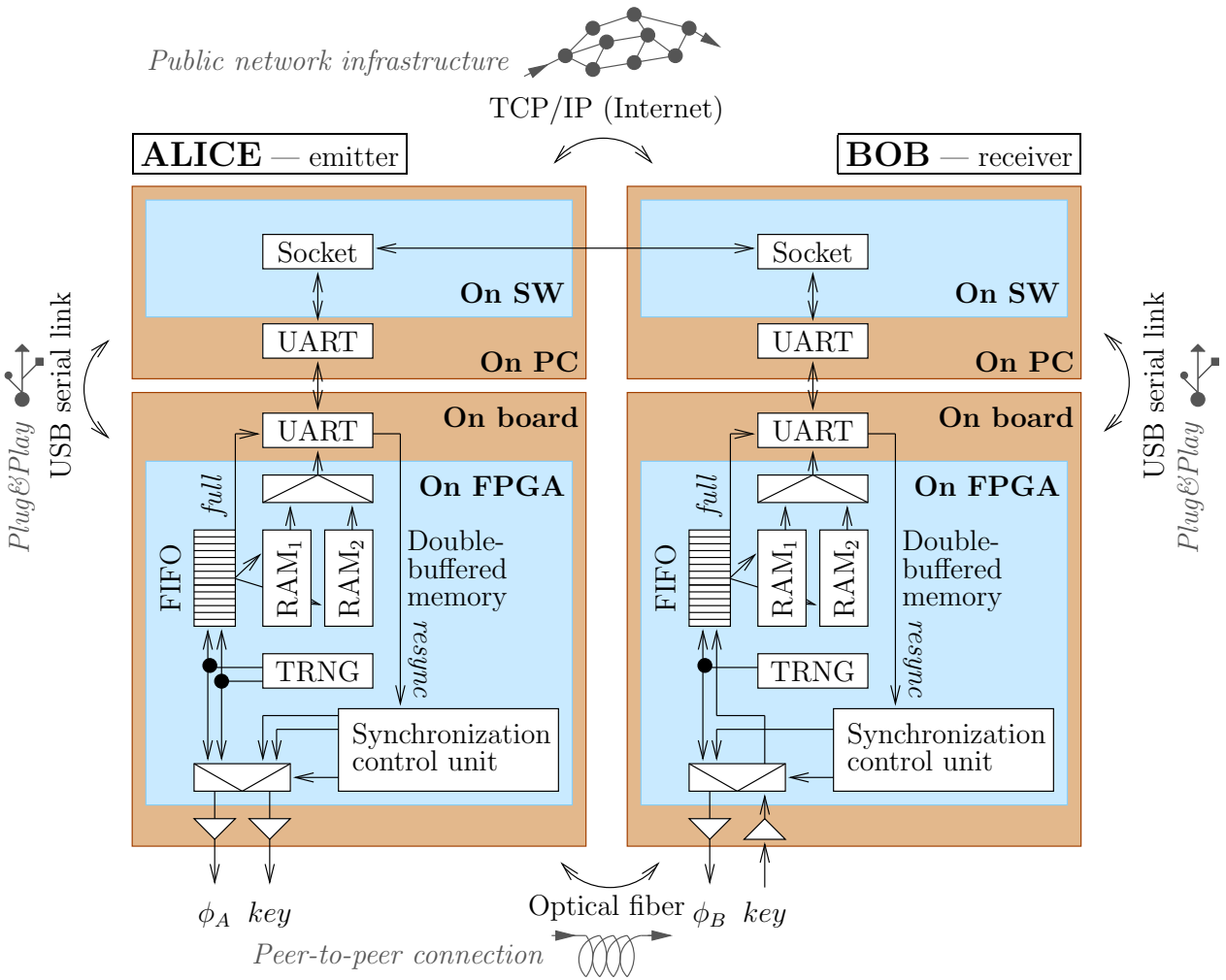
CPU: SH4, 400 Mips
FPGA: Altera STRATIX
OS: GNU/Linux
Peripherals: 1 serial port, 1 USB, 1 screen, *etc.*

↦ http://www.enst.fr/~polti/realisations/shix20/
(*public web page*)
↦ http://www.enst.fr/~polti/realisations/shix20/doc/
(*personal wiki*)



- `{alice,bob}.enst.fr` are coming soon
- They are powerful enough to do their VISQ-related job
- They are not powerful enough to develop on them
- SW development under Linux (GCC)
- FPGA development under Linux (Altera / quartus)
- The two `VISQ-boxes` are the same
- Global synchronization: Alice is the **reference**, Bob is **enslaved**.
- Hence the `VISQ-box` has two clocks inputs: **internal** and **external**.


Public network infrastructure
TCP/IP (Internet)
ALICE — emitter
BOB — receiver
SW layer
Bufferization layer
Random generation layer
Synchronization layer
Digital ↔ Analog layer
Optical layer
HW layers
Socket
On SW
UART
On PC
On board
On FPGA
FIFO
full
RAM1
RAM2
Double-buffered memory
resync
TRNG
Synchronization control unit
USB serial link
Plug&Play
ϕ_A key
Optical fiber
ϕ_B key
Peer-to-peer connection



- The FPGA controls **GPIOs** (actually *GPOs*)
- It also accommodates the two **asynchronous domains**: optical/network
- Hence the use of **FIFOs**
- The FIFO **is not data-driven**, to prevent from desynch caused by undetected photons
- Thus the FIFO is **clocked** by the `id200` trigger signal
- Initially: **open loop**

"Classical Hardware" Design – Organization

Hardware Development Organization into "Task Forces"

1. Quantum optics / HW interface (*internship proposal*)
2. HW design (*internship proposal* — attributed)
3. SW support of the VISQ-box (drivers)

Pure HW Design: two Challenges

1. True Random Numbers Generation (*alias TRNGs*)
2. Side-Channel Attacks (*alias SCAs*) resistance

Random Numbers Generation

Definitions

- **Pseudo-RNG**: $\exists$ algorithm that generates the sequence
- **True-RNG**: $\nexists$ such algorithm. It is not simulable

Facts

- QKD **does not need** perfect RNG ...
- ... although **any bias impedes** the security by giving Eve an advantage
- Cryptography-wise, **True-RNG** is better than **Pseudo-RNG**
- Hence:
  - $\phi_{1,2,3}$ are issued by a TRNG and
  - "*sequence replenishment*" by a PRNG (since this sequence must be identical to Alice and Bob)

TRNG Technology

DFF metastability:

1. Phase-noise of a PLL ($\sim 100kb/s$)
2. Interference between two free oscillators ($\sim 100kb/s$)
3. ... open-loop delay-chain ($\sim 100Mb/s$)!

Presentation Outline

Quantum Optics ↔ HW Problematics

- `amp-op` daughterboard: **generic** or **dedicated**?
- **Transition** from something that works w/o electronics to something that does not work (at least at the first time) with electronics! (*humour* :-))
- **Physical access** to the A102 lab (key problem!)

Presentation Outline

HW ↔ SW Service

```
#include <cstdlib>
class usb_driver { /***/ }; // To be written...

/**
 * @brief Interface provided by HW to the SW layer.
 * The objects are constant: the SW can neither command
 * the HW nor the underlying optical layers.
*/

class qkd_driver: protected usb_driver
{
  static size_t const N=1<<20; // Buffers size
public:
  /// The QKD buffer index (buffer-to-buffer synch):
  volatile int qkd_index() const; // Increasing number
  /// The QKD buffer itself:
  volatile bool* get_qkd_bits() const; // Packet of N bits
};
```