

**INTI**



# USB Framework, IP Core and related software

**Tropea S.E., Melo R.A.**





# Why?

- We develop embedded systems that usually connect to a PC.
- Parallel and serial ports obsolete in favor of USB.
  - Faster
  - Plug & play



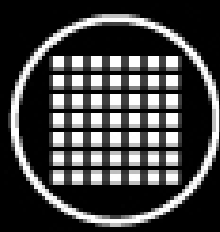


# Objetives

- Synthetizable for most of the FPGAs available in the market (even ASICs) => VHDL93
- Low cost, compared with an external solution => < Cypress EX-USB FX2 ≈ FPGA 200k
- Allow for very compact configurations with minimal area and external components requirements => to allow the use of S2Proto







INTI

# Layers



- Electric layer: voltage and/or current level conversion.
- Physical layer or PHY: parallel  $\Leftrightarrow$  serial, encoding, clock, start/end packet (UTMI)
- Handshake layer or SIE (Serial Interface Engine): basic USB protocol transactions.
- SIE adaptation layer: this layer adapts the SIE signals for the upper layer.
- Protocol layer: higher part of the USB protocol, including plug & play.
- Application layer: here we put the application, this is our device.



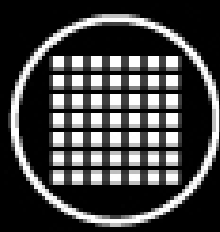


# Electric layer

- LS (1.5 Mb/s) and FS (12 Mb/s): compatible with FPGA I/O, but not compliant. External driver to be compliant.
- HS (480 Mb/s): currents for signaling => external driver is mandatory.







**INTI**

# PHY layer

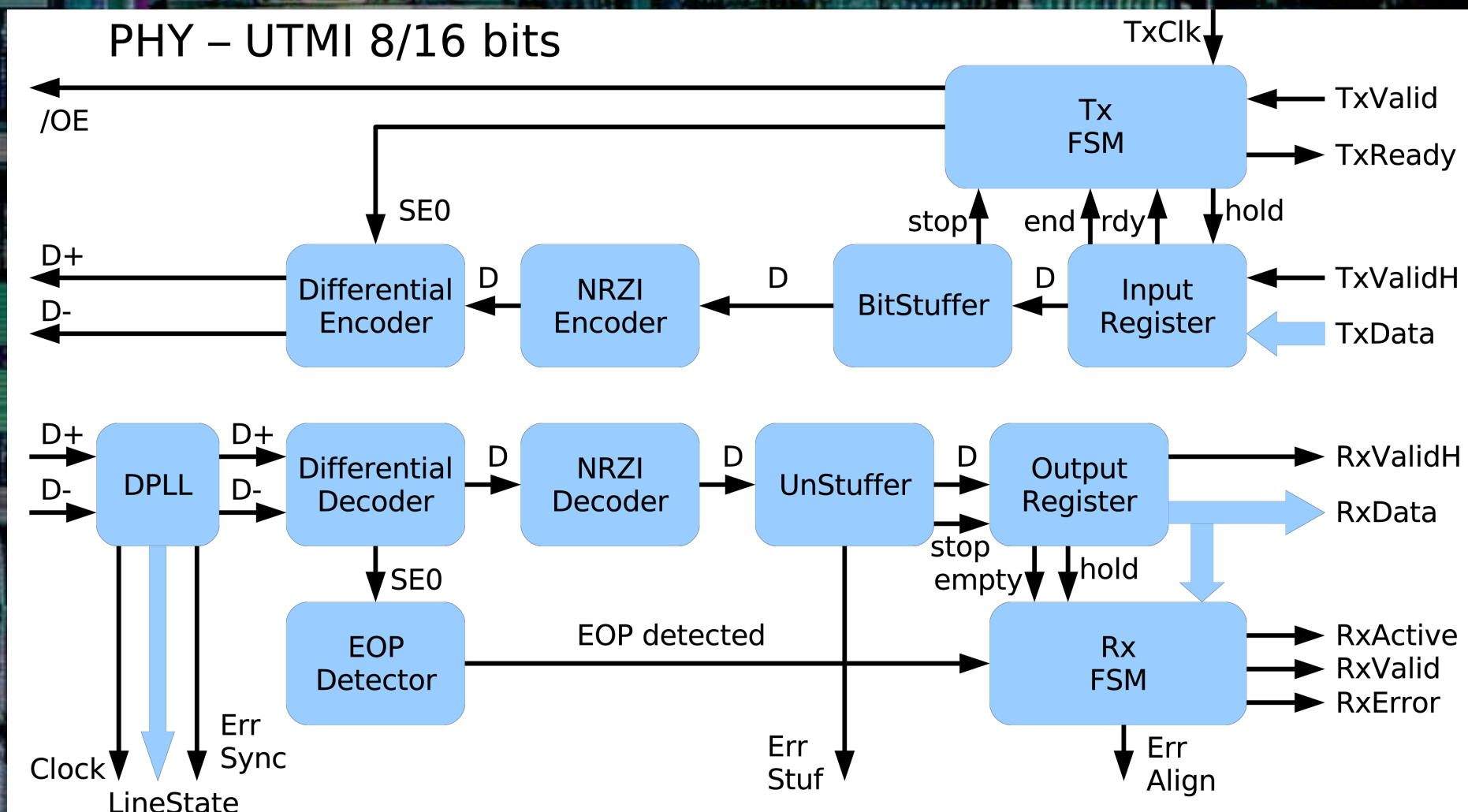


- Implemented for LS/FS, discarded the option to implement a HS PHY.
- UTMI compliant.
- 8/16 bits interface.
- LS: 6 MHz FS: 48 MHz
- Reception: clock recovery and sync (DPLL), bit decoding (NRZI & bitstuffing), serial to parallel and start/end of packet detection.
- Transmitter: parallel to serial, start/end of packet signaling and bit encoding.
- Errors: bitstuffing, aligning and sync.





# PHY layer



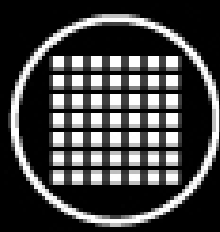


# SIE layer

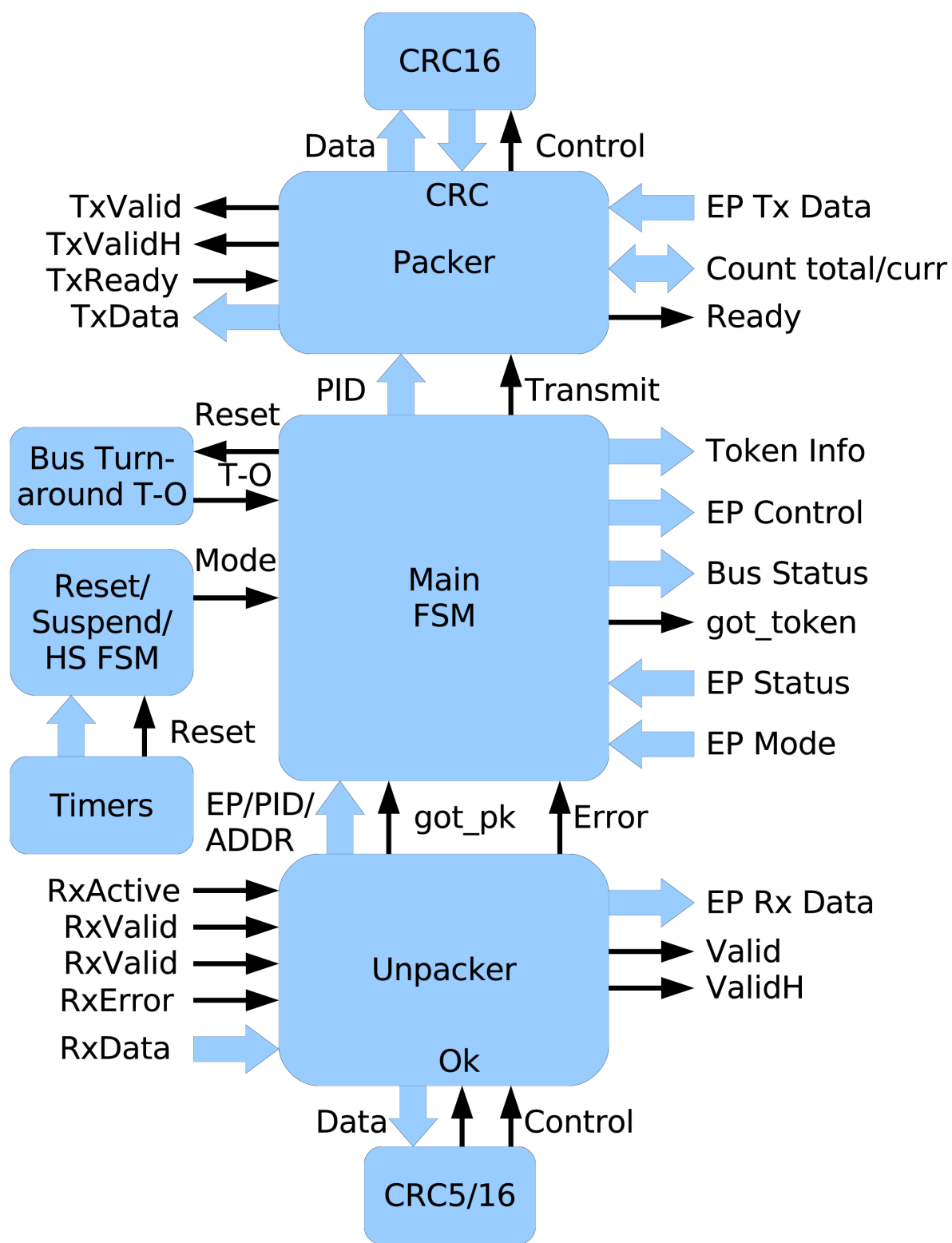
- **Basic USB transactions: 2 or 3 packets exchanged between device and host.**
- **Fast response.**
- **Implemented: control, interrupt and bulk modes, not isochronous.**
- **USB 2.0 features implemented.**







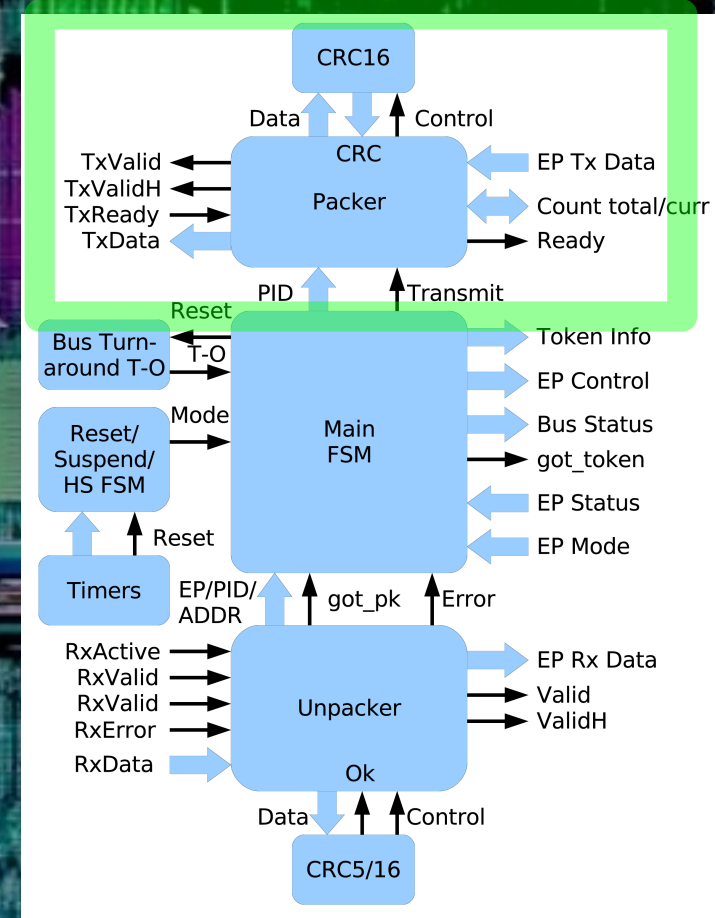
INTI





# SIE: packer

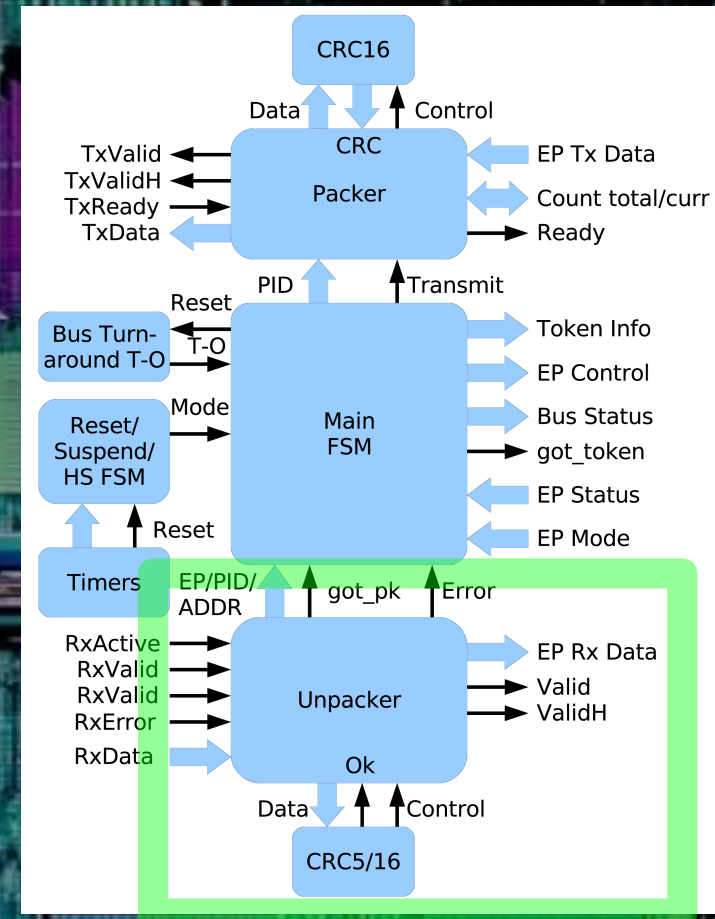
- Assembles packets to be transmitted.
- Supports: DATA0/1 data packets and the ACK, NAK, STALL and NYET handshake packets.
- CRC16 computed for data packets.





# SIE: unpacker

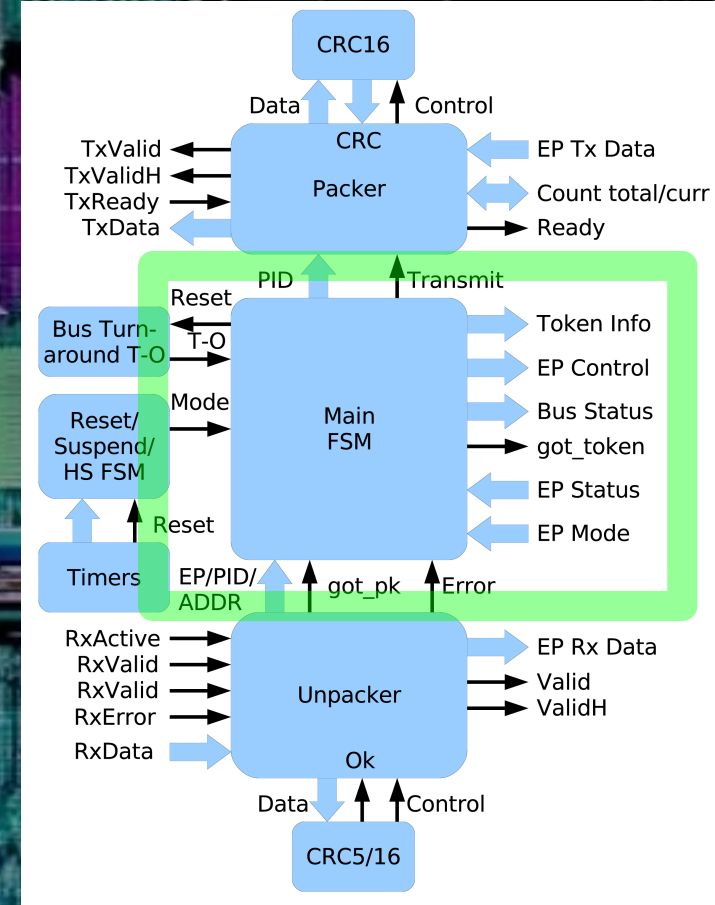
- Disassembles received packets.
- Supports IN, OUT and SETUP tokens, DATA0/1 data packets and ACK, NAK and STALL handshake packets.
- Verifies CRC5 and CRC16





# SIE: Main FSM

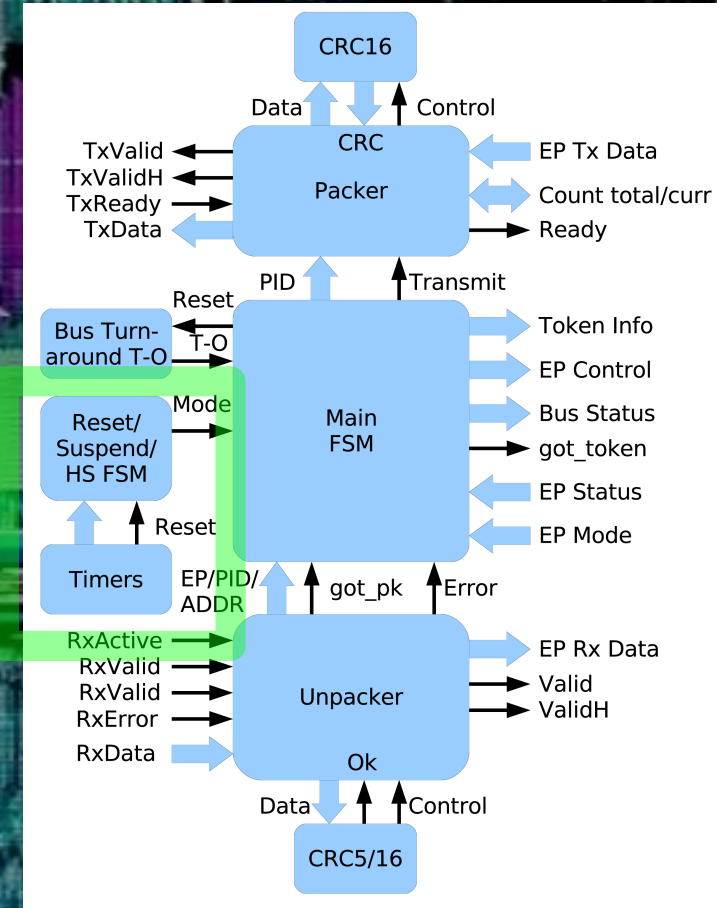
- Ensures a transaction is completed.
- Selects the proper reply for each token.



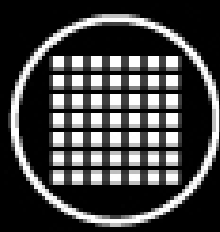


# SIE: other blocks

- Another FSM detects Reset and Suspend signaling. Also negotiates FS to HS.
- A special timer detects “bus turn-around” timeouts.







**INTI**

# SIE layer



- **8 and 16 bits interface with the PHY and upper layer.**
- **Clock can be the same used in the PHY or only half. I.e. 24 MHz for FS**



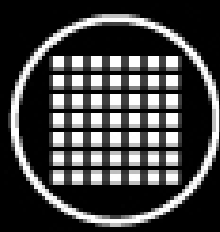


# SIE adaptation layer

- Most cores implements registers and FIFOs here. Versatile and suitable for a microcontroller.
- Many cases doesn't need a CPU.
- In many cases data is volatile, no buffers needed.
- We opted for a simplified layer that only includes the multiplexors used to select the endpoints.







**INTI**

# Protocol layer

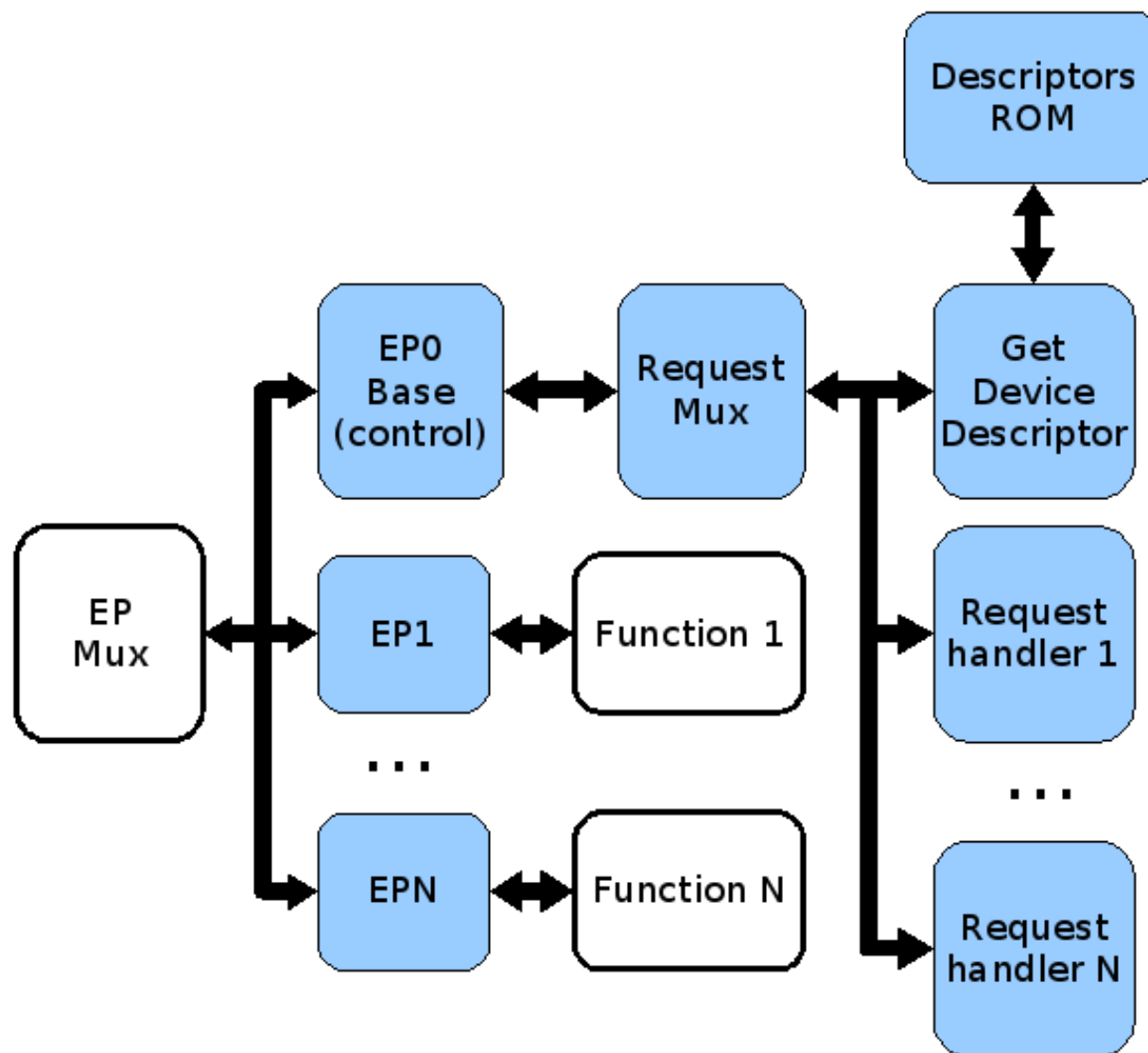


- Solves the high level features: plug & play, multiple configurations. Etc.
- Contains the descriptors used by the host to know how to handle the device (using requests).
- Some cores lets it to software, others implement a limited number of endpoints and features.
- We developed a full hardware implementation.
- A modular approach provides flexibility and allows extensions.
- The most common requests were identified and encapsulated in one extensible module.





# Protocol layer





# Application layer

- Our device goes here.
- Can be very simple thanks to the hardware implementation of the protocol, i.e. just some registers.
- If the endpoint needs buffering the buffers goes here, otherwise avoided.





# Validation & Tools

- Using automatic testbenches to test the individual modules and then big blocks.
- Modular test using the UNIX approach.
- C/C++ tools, using pipes for communication.



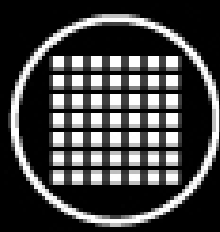


# PHY Validation tools

- **gen-dpll-diff**: random data with clock drift (for DPLL+Differential decoder)
- **nrzi\_en/nrzi\_de/bitsuffer/bitunstuffer**: golden models for the bit encoding.
- **usb-make-packet**: dis/assemble packets using the other tools (for the whole PHY)







INTI

# SIE Validation tools



- **crc**: computes CRC5 and CRC16.
- **usb\_to\_bits**: generates whole USB transactions.
  - Uses a language
  - Can inject errors
  - Uses **usb-make-packet** and **crc**
  - For the unpacker and the whole SIE



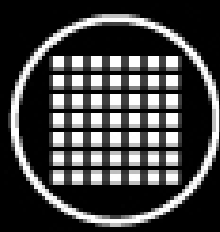


# Validation+development

- Tools also useful for developing USB devices:
- **hid-report**: interpreter for the language used in the HID spec. Generates the descriptor.
- **usb-descriptor**: interprets a language used to describe our device and generates the VHDL ROM.
  - Simplifies one of the most annoying steps.
  - Uses hid-report for HID descriptors.







INTI

# Validation



- Using a keyboard we decoded a full initialization sequence (oscilloscope).
- Translated it to `usb_to_bits` and `usb-descriptor` languages.
- We tested our cores with real initialization sequences based on it.
- We also tested injecting errors.





# Implementation

- VHDL93, avoiding vendor specific stuff.
- GHDL 0.27 (+FPGALibre tools) for validation.
- Hardware validation using Spartan II and Spartan 3 (ISE 9.2)
- LS/FS: ISP1106 driver (cost 5%), also ok without it!



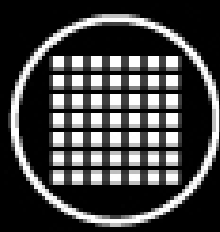


# Implementation cont.

- **HS: CY7C68000 UTMI, 16 bits @ 30 MHz (cost 15%).**
- **Host: PC running Debian GNU/Linux**
- **Language for the tests: C++**
- **Linux APIs:**
  - hiddev
  - libusb







INTI

# Demonstration devs



- Analog joystick to USB adapter (2 external comparator for the A/D). HID, 1 ep. 708 LUTs 427 FFs (493 slices). [Linux+XP]
- Generic HID (LEDs and switches) HID, 2 eps. 692 LUTs 402 FFs (488 slices) [hiddev]
- Generic USB (LEDs and switches+ROM) 3 eps 584 LUTs 349 FFs (418 slices), HS implementation: 860 LUTs 347 FFs (511 slices) [libusb]
- USB to WISHBONE bridge FS 716 LUTs 404 FFs (496 slices)
- All of them consume 1 BRAM

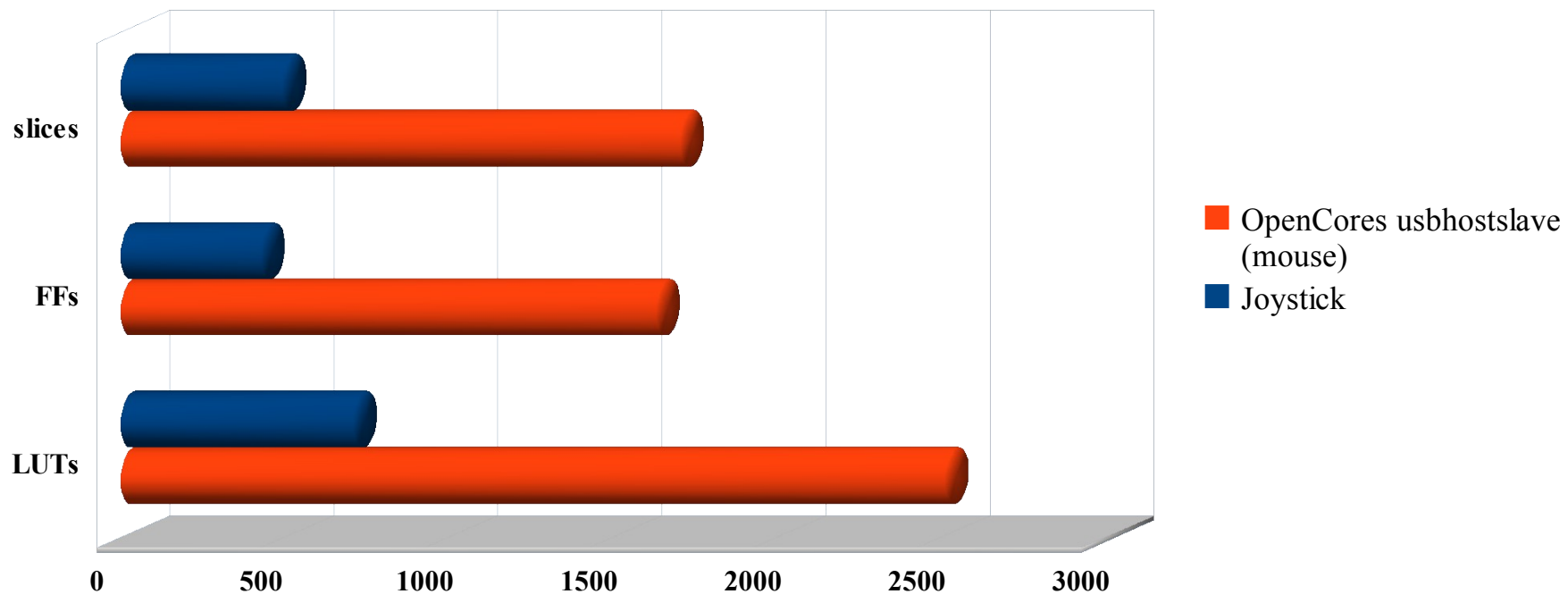




# Comparisons

- With other full implementation:
- OpenCores ushhostslave project.
- Device: mouse

OpenCores mouse vs our joystick

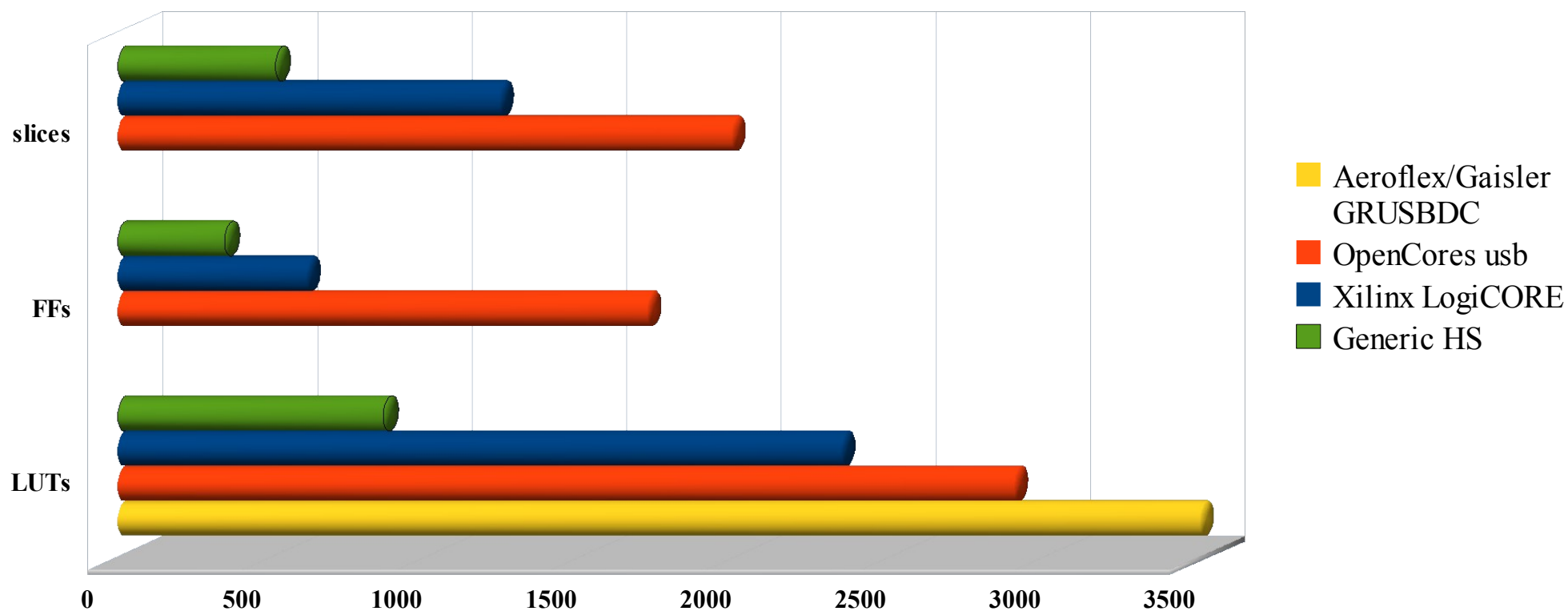




# Comparison cont.

- With USB cores (only SIE and CPU interface! no protocol or app.) HS

USB cores (SIE+CPU interface)

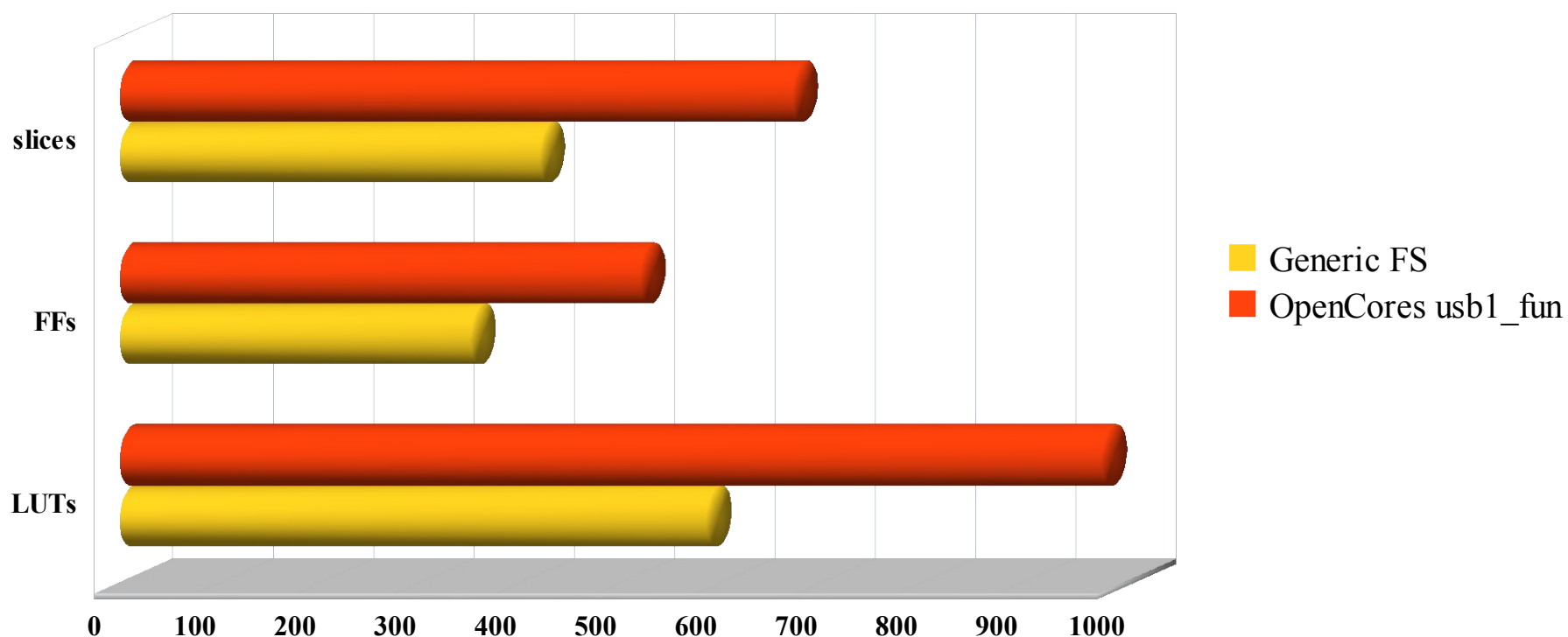




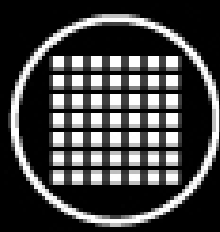
# Comparison cont.

- With a fixed (6 ep) implementation that includes a protocol. No app! FS

Generic FS vs Fixed eps







INTI

# Conclusions



- **Very compact, a full USB device takes just 35% of a Spartan II 100**
- **Needs few external components (only resistors for not fully compliant LS/FS)**
- **Cost effective compared with an external chip.**
- **Using UTMI enabled HS**
- **hid-report and usb-descriptor reduced the development time of the demonstration devices.**



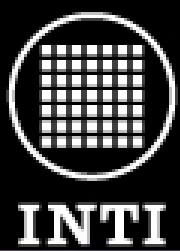


# Conclusions cont.

- The use of small modular testing tools enabled a good code reuse.
- The FPGALibre tool chain was effective for this development
- Very small when compared to other solutions. We think the main reason is the simplified interface between the SIE and the application. It also simplifies the application layer.







# Contact



## Staff:

- Brengi, Diego J. <[brengi@inti.gov.ar](mailto:brengi@inti.gov.ar)>
- Melo, Rodrigo A. <[melo@inti.gov.ar](mailto:melo@inti.gov.ar)>
- Tropea, Salvador E. <[salvador@inti.gov.ar](mailto:salvador@inti.gov.ar)>
- <http://utic.inti.gov.ar>

## FPGALibre

- <http://fpgalibre.sf.net/>

¡Thank you!