System-on-Chip Architectures and Modelling

“A Secure Embedded Smart-Card Reader”

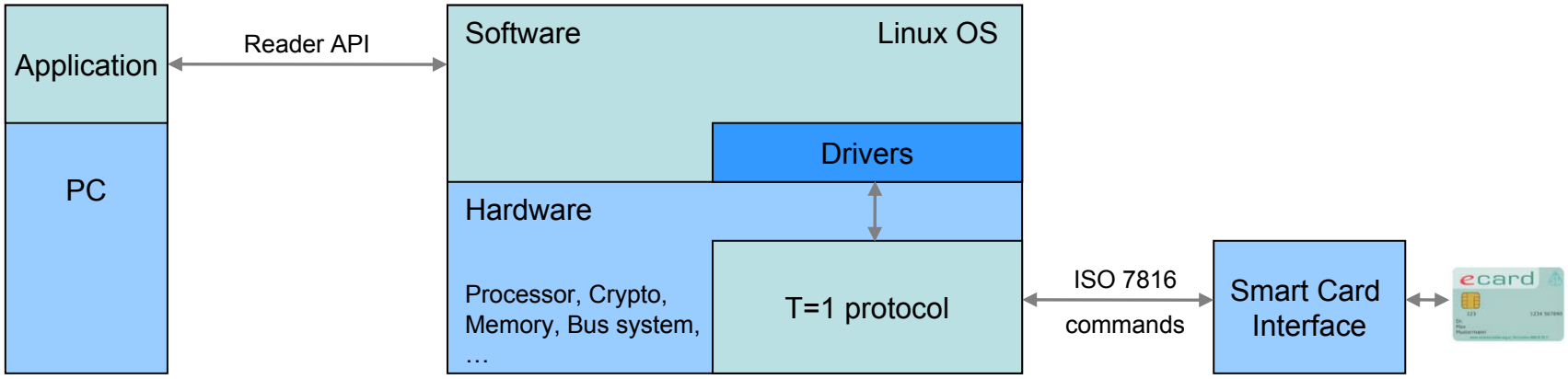
Michael Hutter
Alexander Szekely
Christopher Bischof
Eral Tuerkyilmaz
Georg Neubauer
Hermann Kureck
Andreas Reiter
Andreas Schuster
Paul Rouschal
Alfred Gefahrt
Driton Kuçi
Lukas Pfeifenberger
Jeremias Kleer
Christoph Klug

System on Chip 2009

- The course
  - 5 ECTS credits (3VU)
  - Blocked lecture (Oct. till Dec.)
- Content
  - Its about the design of embedded systems on silicon
  - Designing HW modules
  - IP integration (system-on-a-chip)
  - HW/SW interaction
  - Drivers, embedded OS, communication,…
  - Soft skills (English, presentations, discussions,…)
- 12 Participants
  - Organized in 5 groups (work packages)

The SoC Project

- Design of “a Secure Embedded Smart-Card Reader”
  - Used for e-commerce, e-voting, …
  - Authentication of users with the Austrian citizen card
    - E.g. e-card generates digital signatures using a PIN
  - Reader displays the hash of the message on a screen



Organization

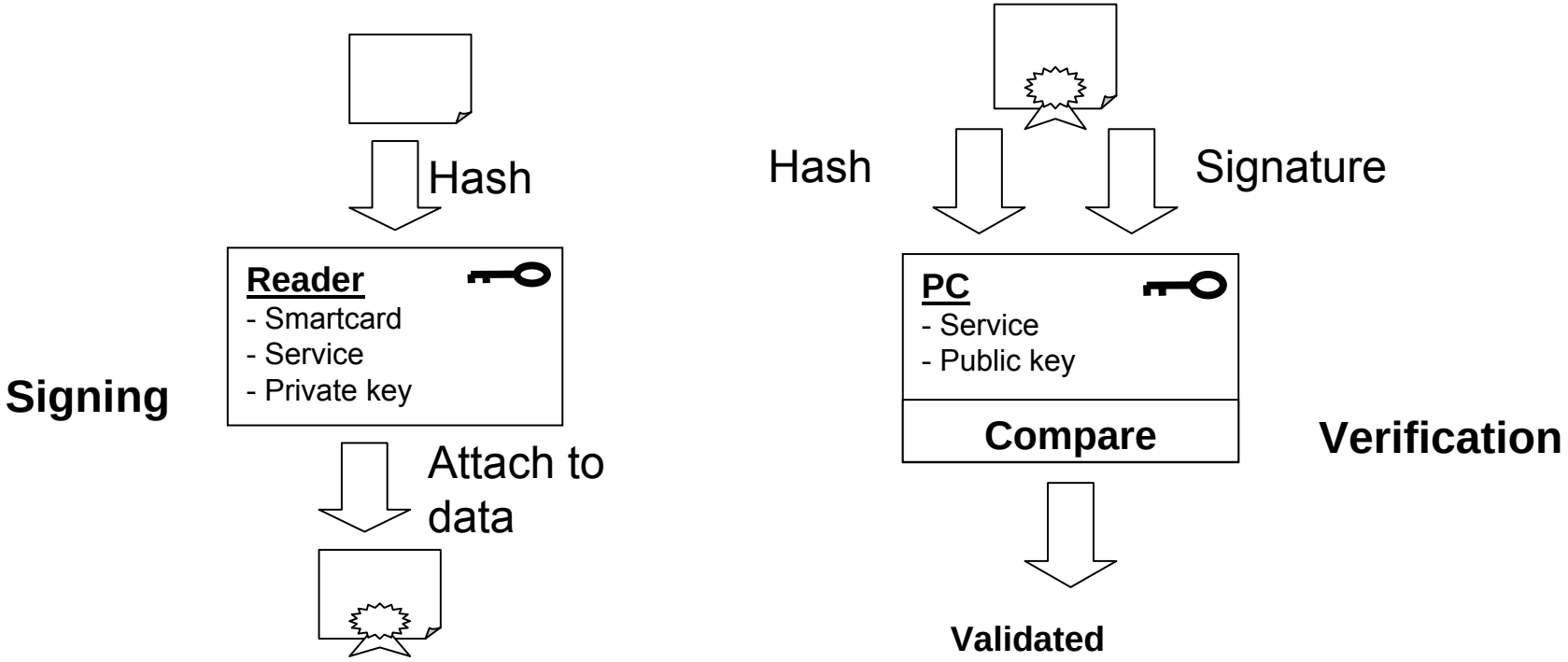
- 5 groups (work packages)
  - WP0 – Frontend
    - Interface to smart card, reusable T=1 protocol HW module, ...
  - WP1 – ML403
    - Worked with ML403 board, PowerPC, drivers, …
  - WP2 – ML4010
    - Worked with ML410 board, Leon processor, drivers, …
  - WP3 – Linux
    - Porting Linux on both platforms, cross compilation,…
  - WP4 – Host application
    - Host software, reader communication, demo application

Work Package 4

Christoph Klug
Jeremias Kleer
Alfred Gefahrt

The Core Capability – Signing a Document

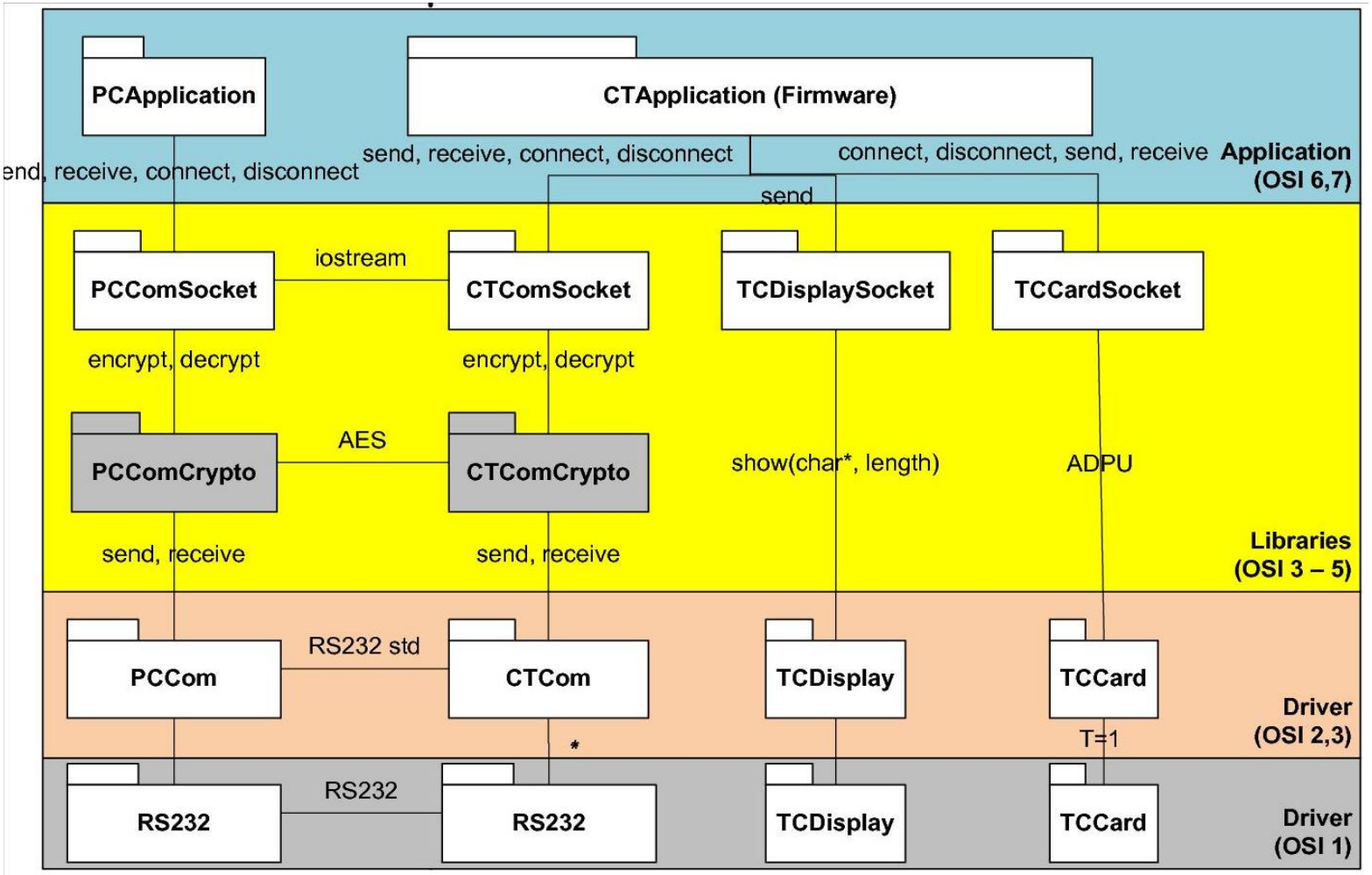
- Austrian Citizen Card
- Authentication of user, signing data, reading certificate,…



Software Part - WP4 Specification

- PC-Application
  - Java
  - Signing documents (IAIK Java JCE)
- Reader-Firmware
  - C++
- Communication
  - Serial port (RS232)
- Security
  - Visual verification with VGA-display

Problem Overview



WP4 Challenge

Windows and Linux
Host-PC on Windows or Linux
Reader on Linux

Linux Cross Compiling
WP3, WP2, WP1

Java and C++
Java for application (Host-PC, Windows or Linux)
C++ for firmware (Reader, Linux)

Separate Data Pipe vs. Hooked on Debug Pipe
Filtered data

Communication Port Settings
RS232, 9.600 bit/sec, 8 data bits, 1 stop bit, Null-Modem

Layers of Communication

Application layer

Basic element: APDU

Generated by the host

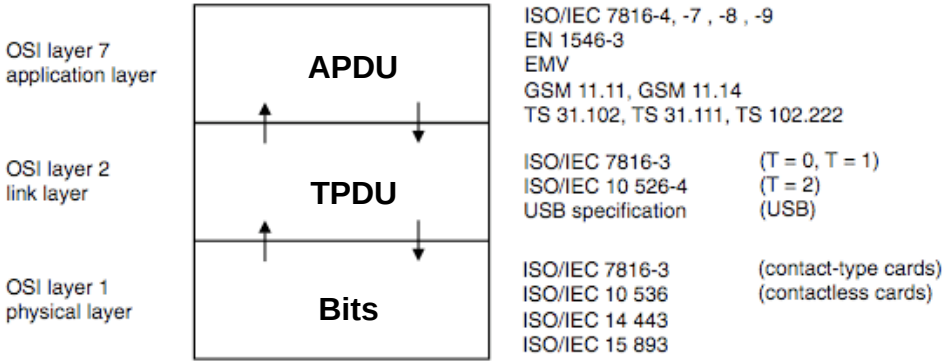
Data link layer

T=1 protocol

Basic element: TPDU

Physical layer

Bitstream to communicate with the smartcard



from SmartCard Handbook by Rankl, Effing

WP4 Application Layer

Application Protocol Data Unit (APDU)

- Request

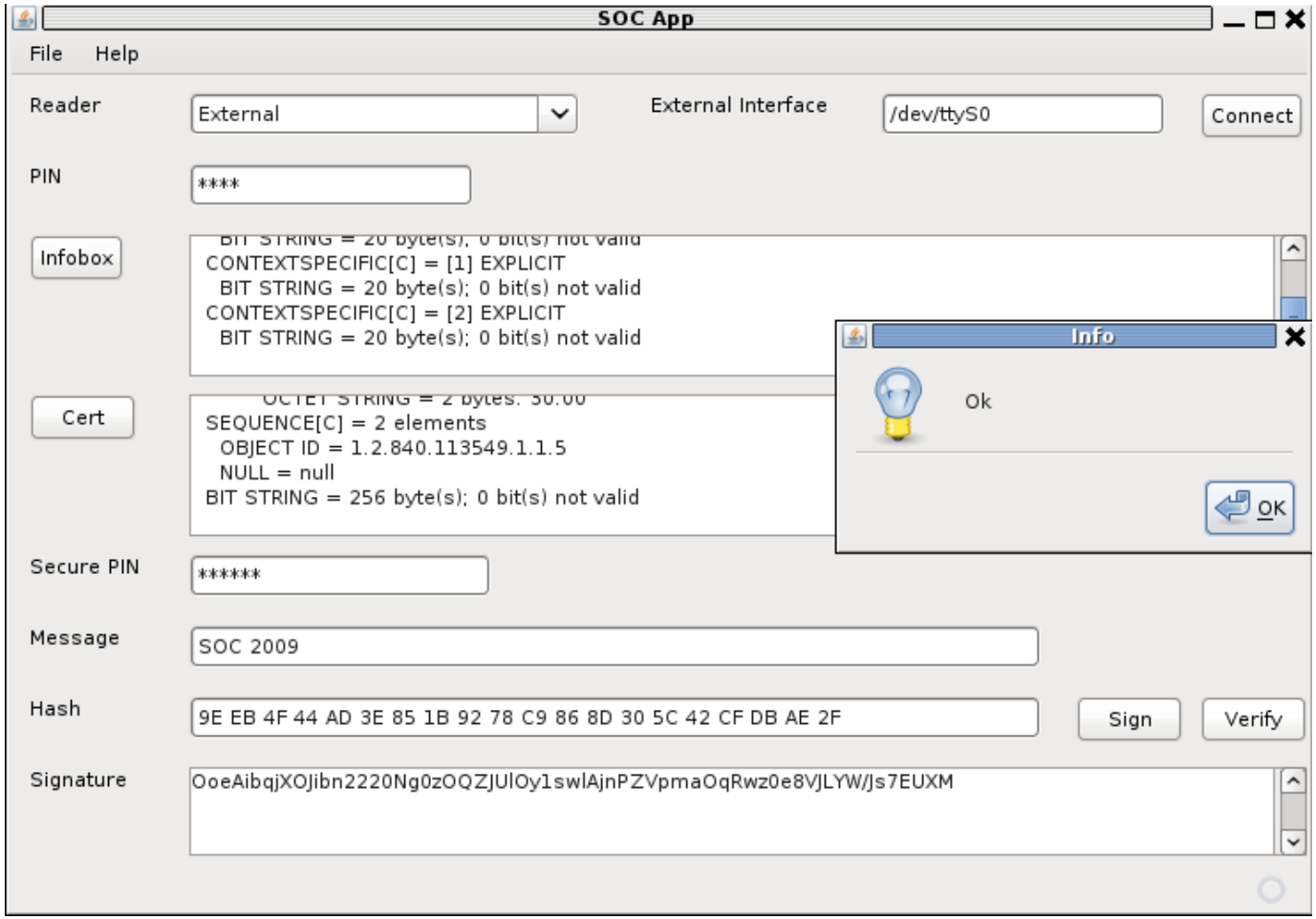
CLA	INS	P1	P2	Lc	Data	Le
Header				Body		

- Response

Data	SW1 SW2
Body	Trailer

- Independent of transmission protocol
- Includes the instructions, parameters and data
- e.g. Austrian Citizen Card
  - Certain order of APDU's for signing a message

PC Application GUI



WP4 Firmware

- „Channel“ between host-software and smart-card-driver and visual verification of the hash
- Communication-interfaces
  - Device-file to smart-card
  - std-in/out to host
- Data-conversion
  - Host: Each byte coded in hex with spaces between
    - e.g. 00 a4 04 00 08 d0 40 00 00 17 00 13 01 00
  - Smart-card: Sequence of characters
- Display on a VGA
  - Device-file for character-based output

WP4 Firmware

Communication-interface to smart-card

- Device-file provided by WP1/WP2
- File-functions of Linux:
  - Open
  - Write
  - Read
  - Close

WP4 Firmware

Communication-interface to host

- Abstracted layer to unify access
  - Hide access method
    - RS232 direct
    - std-in/out redirected to RS232
  - Issue of signaling end transmission

Work Package 3

Andreas Reiter
Georg Neubauer

WP3 Requirements

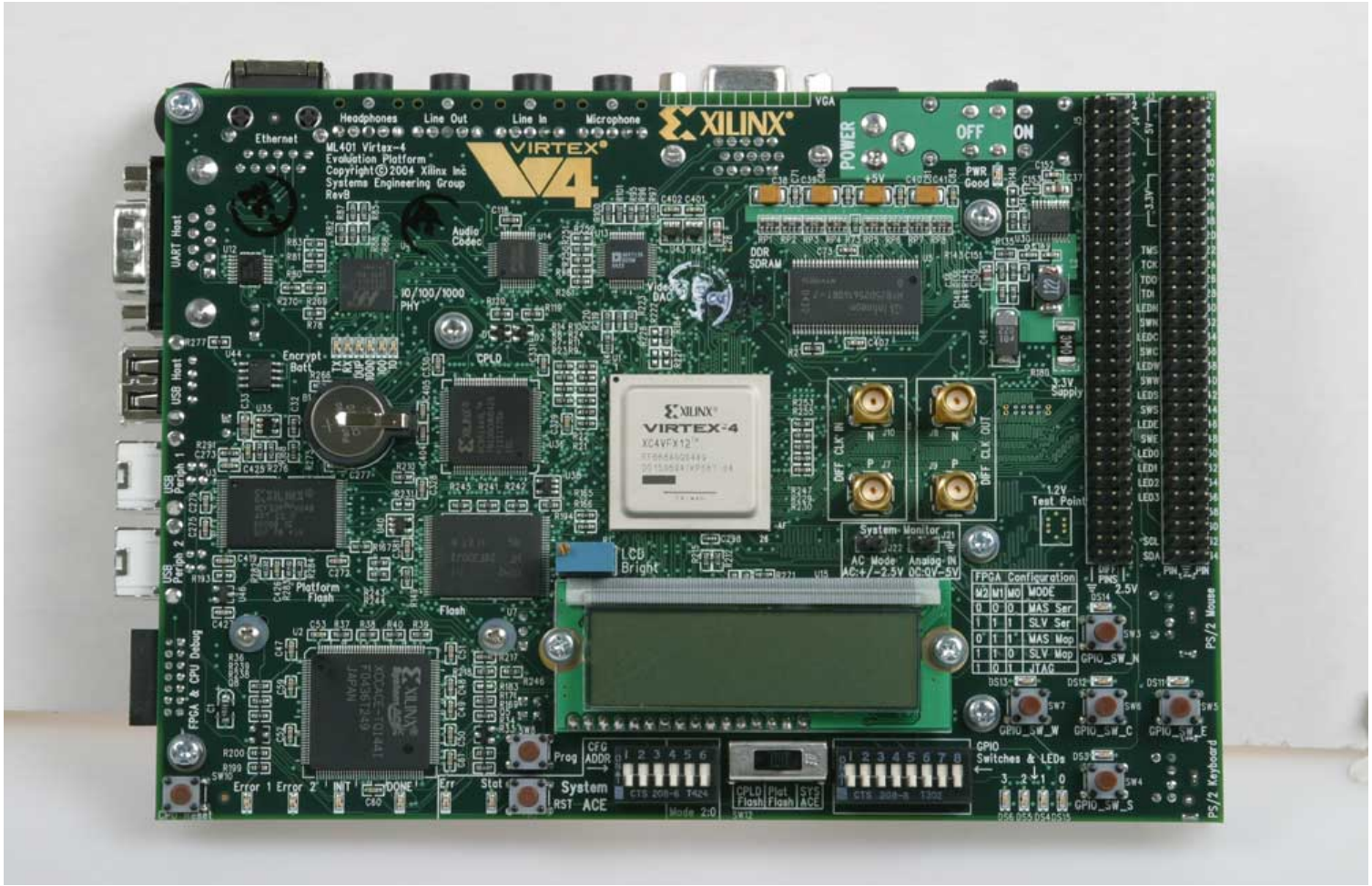
- Provide linux for both boards (ML403 + ML410)
  - ✓ Kernel
  - ✓ Root file system
  - ✓ Start scripts
- Provide toolchain for both boards (PowerPC and Sparc-LEON3)
  - ✓ Compilers
  - ✓ Libraries (glibc, c++, …)
  - ✓ Include drivers and application in build process

ML403 Specifications

Relevant features

- Virtex-4 FPGA with PowerPC hardmacro
- 64MB DDR SDRAM
- One RS-232 port
- VGA output
- System ACE CF-controller
- 16-character x 2-line LCD display
- etc.

ML403



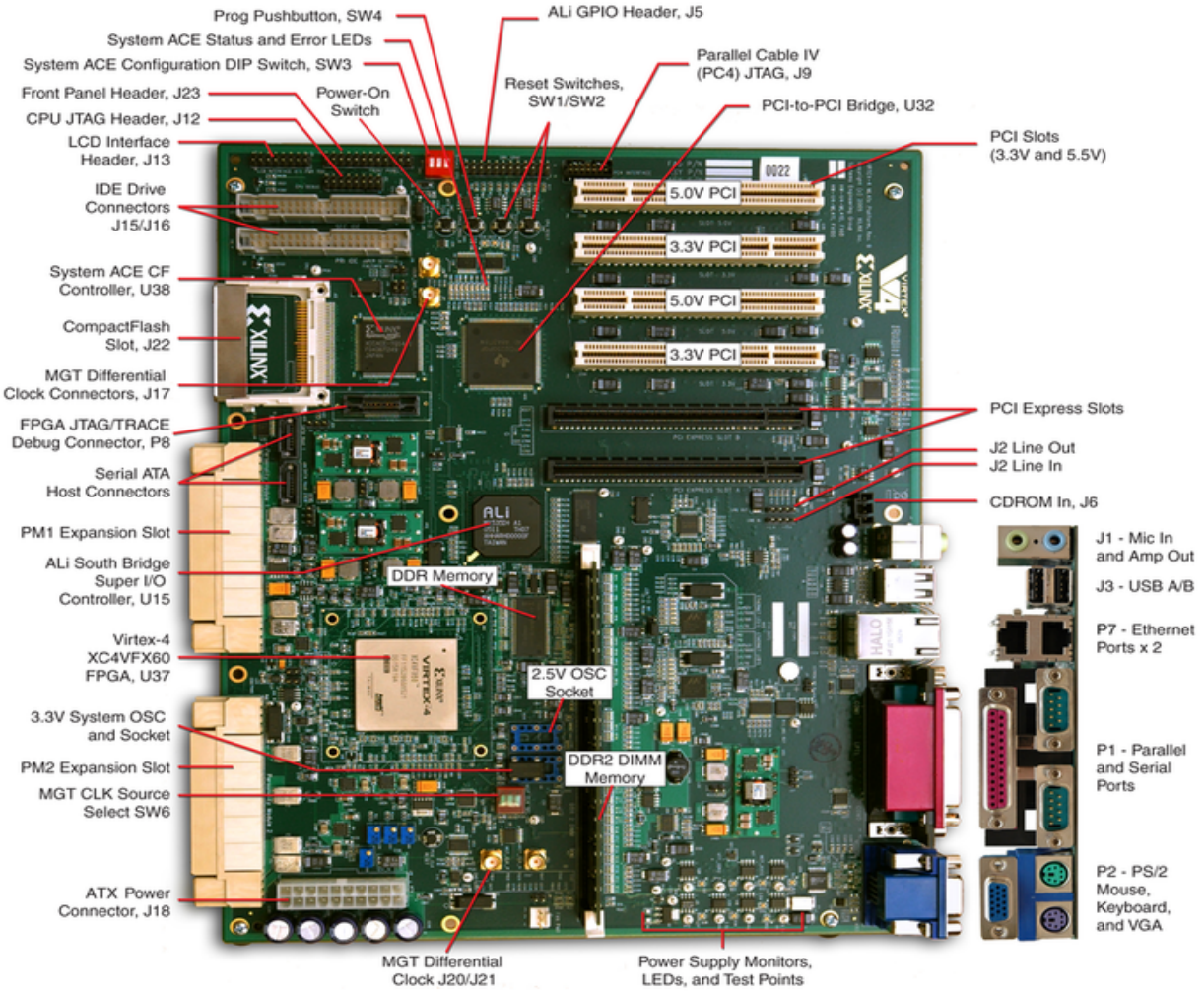
http://www.xilinx.com/images/boards/ml403/side_back_l.jpg

ML410 Specifications

Relevant features

- Virtex-4 FPGA (also with PowerPC hardmacro but not used)
  - We use an Sparc-LEON3 softmacro (WP2)
- 64MB DDR SDRAM
- DDR2 DIMM Memory slot
- Two RS-232 ports
  - We use only one
- VGA output
- System ACE CF-controller
- etc.

ML410



http://www.xilinx.com/images/boards/ml410/images/ml410_pict_callouts_revb.jpg

WP3 Challenges

- Get in contact with embedded linux
- Adapt embedded linux to the boards
  - Buildung standalone application
  - Build first bootable linux kernel, from tutorials using prebuilt toolchains
  - Get on board hardware running
  - Switch to open embedded and build custom toolchain
  - Fully automated build process
- Provide toolchain tutorials to other WPs

WP3 Implementation

- ML403
  - Xilinx Linux distrubution
  - Provided via Sys-Ace file on CF-card
  - Console support (RS-232 and VGA)

- ML410
  - Snapgear Linux distribution
  - Loaded via grmon application (JTAG)
  - Console support (RS-232 and VGA)

WP3 Results

- Linux console (stdin, stdout)
  - Only one RS232 (at least on ml403)
  - No Ethernet or other communication ports
  - Linux console (stdin, stdout) and host communication over same serial port
- VGA support
  - Keep the 'user display' clear from kernel/debug messages
  - Not an alternative for stdin/stdout
  - VGA is only a framebuffer device populated by the firmware

WP3 Results cont'd

- Automatic build
  - Via open embedded for the ML403
  - Via Gaisler toolchain for the ML410

- Root filesystem
  - For ML403 on the CF-card
  - For ML410 via RamFS
    - because of unresolved CF-driver issues

Literature

- Xilinx, Gaisler, open embedded
  - Homepages (tutorials)
    - gaisler.com
    - xilinx.com
    - openembedded.net
  - Newsgroups
  - Forums
  - Mailing lists

Work Package 2

Paul Rouschal
Lukas Pfeifenberger
Hermann Kureck

Tasks

- Get LEON3 running on Xilinx ML410
  - Hardware extensions
  - First test programs
- Cooperate with WP1 for the smartcard driver
- Integrate the smartcard interface from WP0 into Gaisler workflow
  - Connect the module via dual-port BRAM

LEON3 on ML410

- Gaisler design-flow
  - Leon 32bit RISC CPU with a rich set of peripherals
  - ETH MAC, VGA, UART, GPIO, etc.
- Design for ML403 available
  - Used as example for adaptation to ML410
- Adaptations:
  - .ucf – get the pins right for the Virtex4 XC4VFX60 FPGA
  - Makefiles – get the device and package right in the workflow
  - Top level .vhd file - add needed IP cores from GRLIB
    - Second UART
    - System ACE
    - VGA

Custom Hardware Integration

- Familiarize with…
  - Gaisler workflow
  - IP cores in GRLIB
  - Cross compiling standalone and Linux applications
- First experiments:
  - Control GPIO LEDs with standalone application
  - Writing own LED module with dual-port BRAM interface
  - Clock the LED module with a lower frequency
  - Writing a demo driver for the LED module

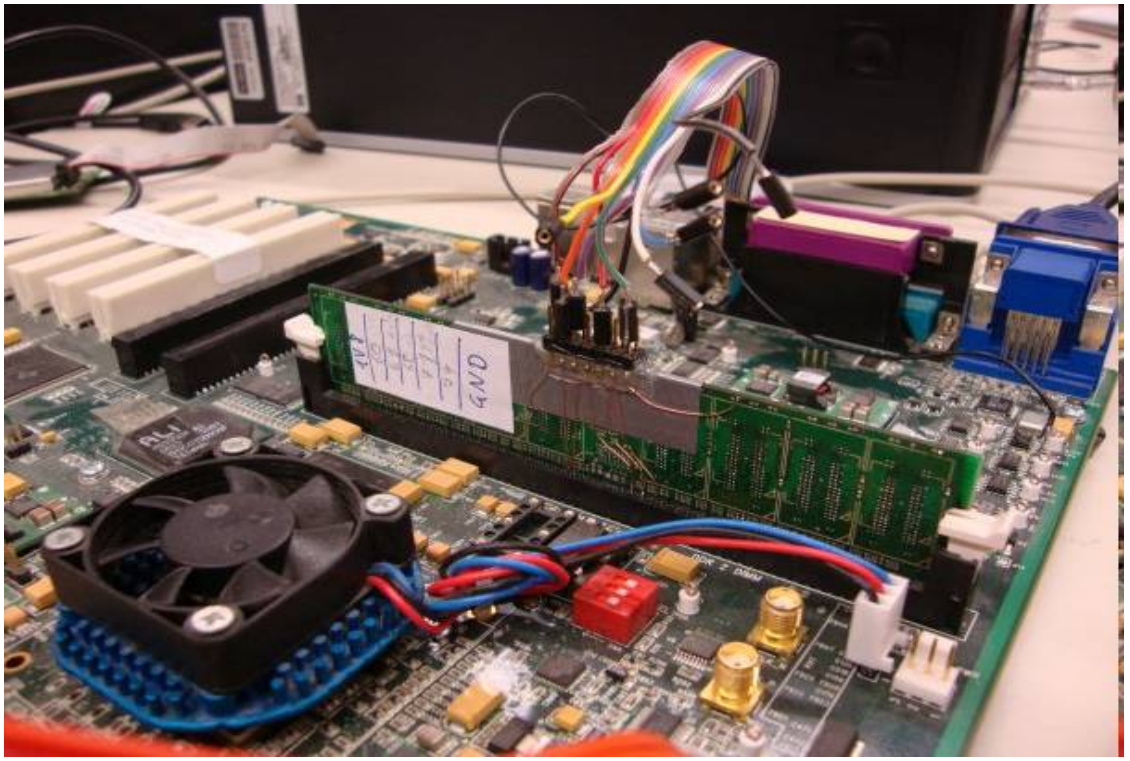
Custom Hardware Integration

- Finally integrate WP0 module
  - Provide 3 I/O pins
  - Connect to DP BRAM
- Writing driver
  - Memory mapped I/O
  - Very simple communication via DP BRAM
  - Control and TX/RX data registers
  - Bytewise transmission of data in each direction
  - Bits in control register for handshake

Challenges

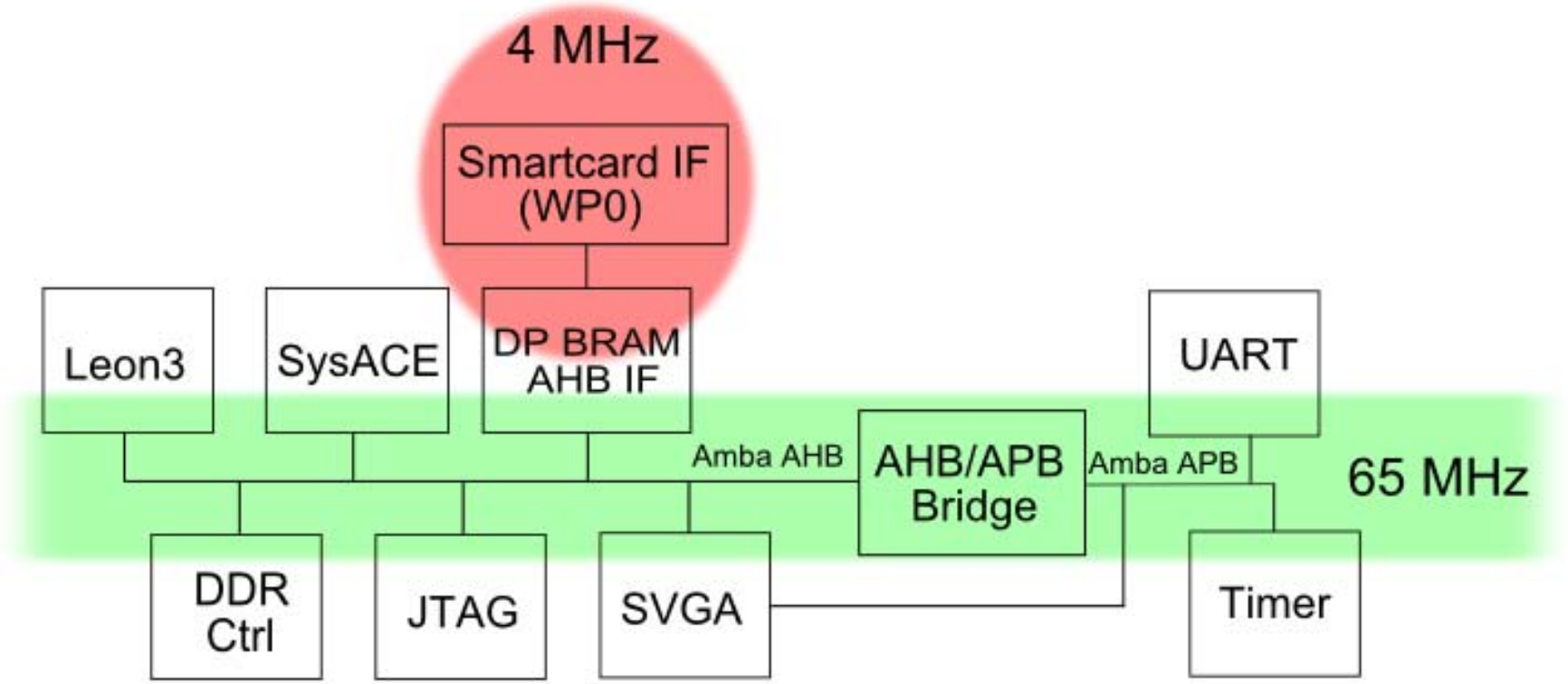
- Different clock domains
  - First Idea: WP0 provides AMBA interface
    - Not a portable solution for both boards
  - → DP BRAM for communication
- Chaotic documentation (by Gaisler and Xilinx)
  - e.g. LVDCI for VGA clock, 2 clock domains for BRAM
- Errors in GRLIB
- … and lots of seem-to-be-configurable, but hard coded values in template designs
- Getting a sufficient number of 3.3V I/O pins
  - ML410 provides only 2 I/O pins directly connected to FPGA (only 2.5V)
  - LCD pins buffered (3.3V), but only one direction possible
  - IDEA: DDR2 unused, lots of pins directly connected

„Memory Mapped I/O“ ☺



- Unfortunately all data and address wires are terminated with 50Ω to 0.9V
  - Not suitable for I/O at lower frequencies or higher voltages
- Fortunately 2.5V are sufficient for the I/Os

On-chip Architecture



Work Package 1

Driton Kuçi
Andreas Schuster

WP1 – Main Tasks

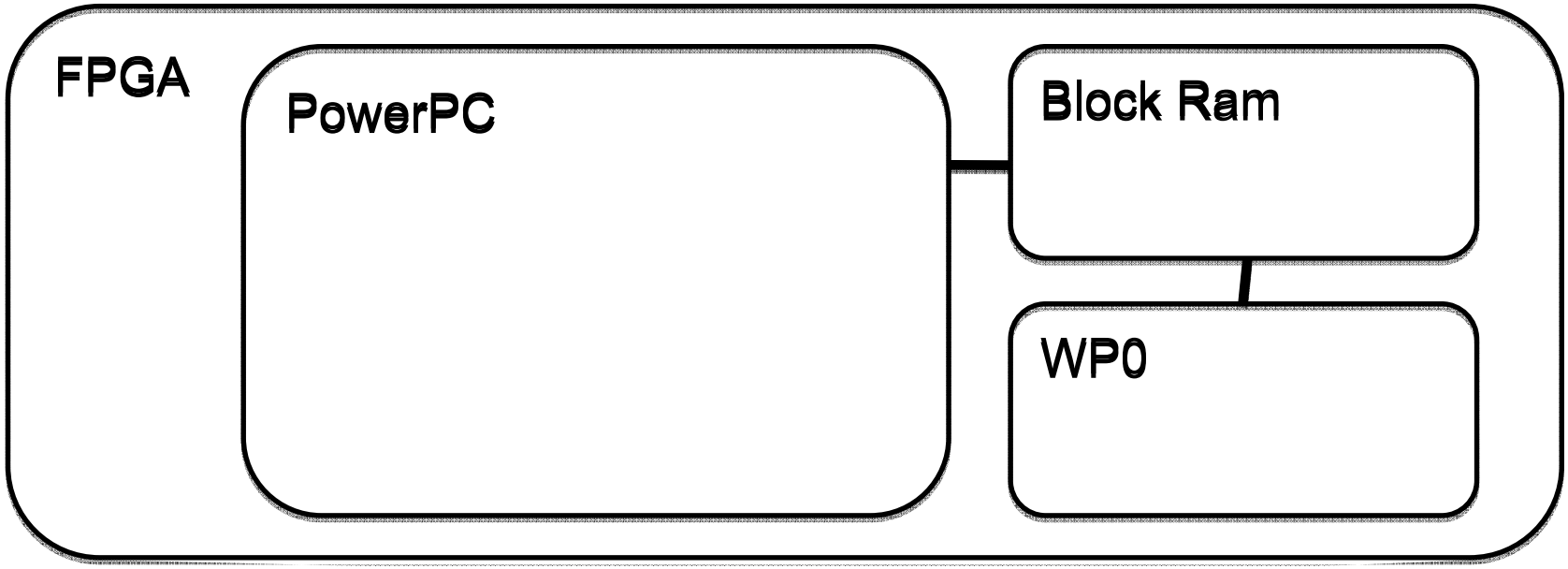
- Hardware base (ML403)
  - Integrating XPS PowerPC module into ISE for ML403 board
  - Providing interface for WP0
- Communication SW to HW
  - Driver
  - Providing an interface for WP4

Hardware Base

- Integrating XPS PowerPC module into ISE
  - PowerPC
  - UART
  - Memory
  - …
  - VGA (done by WP3)

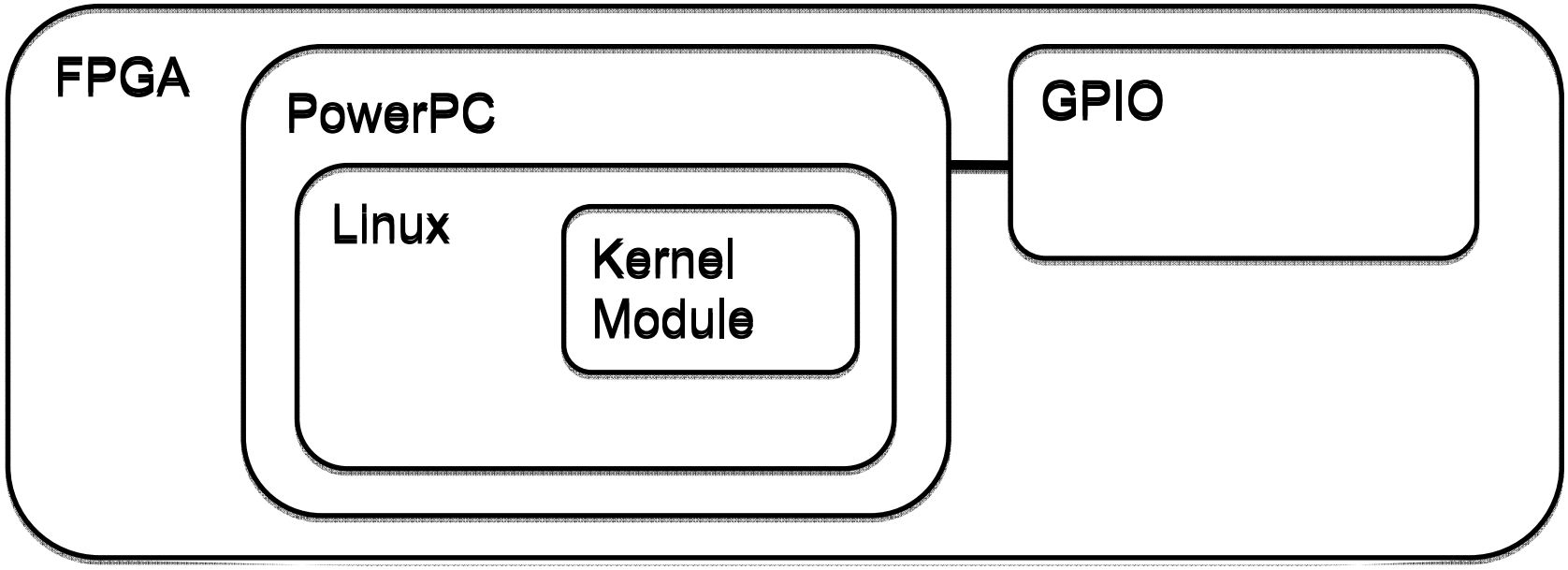
Hardware Base

- Providing Interface for WP0
  - Interface for Block Ram
  - Connection to FPGA Board Pins



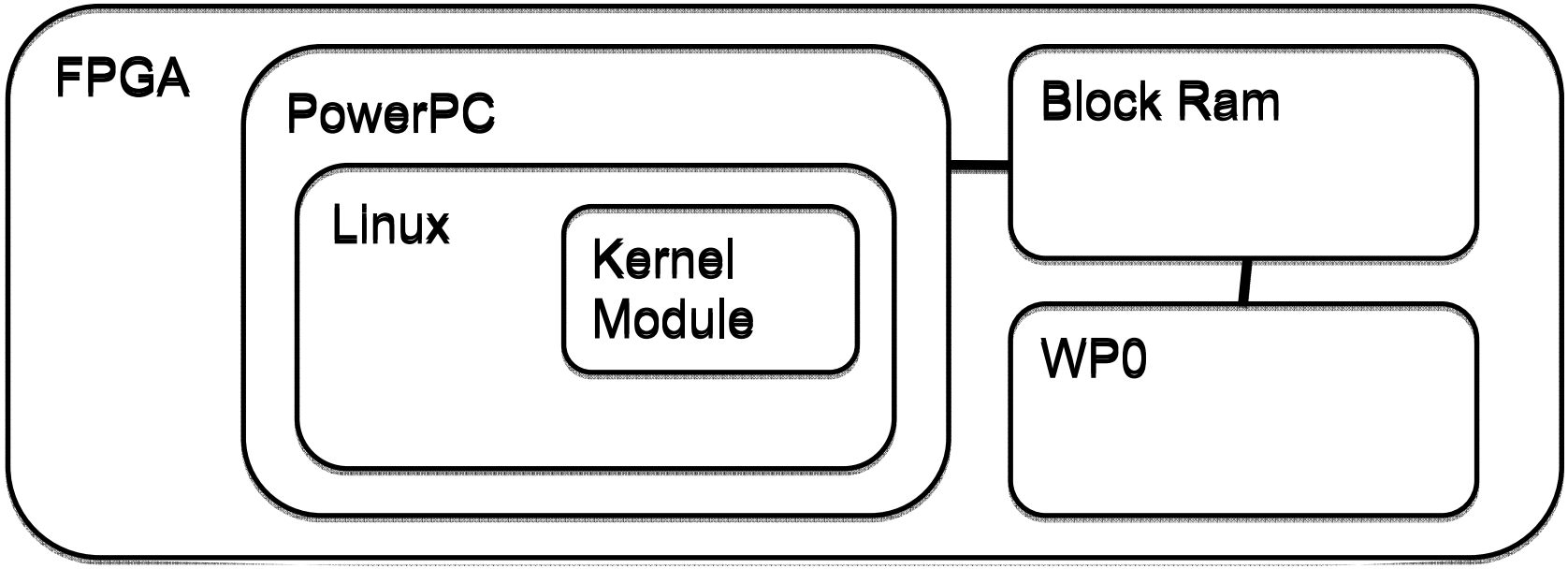
Software for Hardware

- Kernel Module
  - Drive Led through GPIO Module



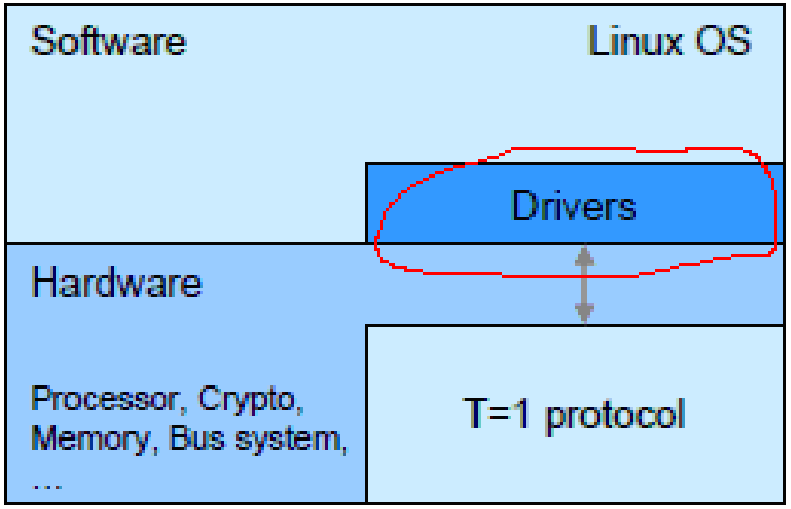
Software for Hardware

- Kernel Module
  - Read and write Block Ram
  - Demo applications to drive WP0



Driver General Description

- Character driver
- User applications use: *fopen()*, *fclose()*, *fread()*, *fwrite()* to /dev/smartcard
- System calls proceeded further by the kernel



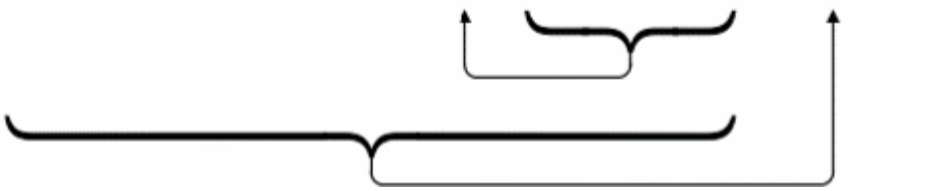
Driver Development

- First version
  - Read/write only one character
- Second version
  - Read/write the whole TPDU
  - In defined memory address
- Third version
  - Conditions and functions
- Final version

Driver Functions (1)

- Open
  - Sends RESET signal and reads ATR
- Write
  - Creates the TPDU command
  - TPDU command/response contains:

prologue field			information field	epilogue field
node address NAD	protocol control byte PCB	length LEN	APDU	EDC
1 byte	1 byte	1 byte	0 ... 254 bytes	1 ... 2 bytes



  - RESEND_TPDU = TPDU
  - Writes the TPDU command
    - Checks if previous byte is read
    - Last byte ?

Driver Functions (2)

- Read
  - Checks if new byte is available
  - Checks BWT bit
  - Reads the new byte (is the last byte?)
  - Checks if R-block received
  - Checks parity, checksum and TPDU length
  - Copies to user buffer the APDU response
- Close

Literature

- Jonathan Corbet, Alessandro Rubini and Greg Kroah-Hartman. (2005). **Linux Device Drivers**. 3rd ed. O'Reilly Media.
- Wolfgang Rankl and Wolfgang Effing. (2003). **Smartcard Handbook**. 3rd ed. John Wiley & Sons
- Xavier Calbet. **Writing device drivers in Linux: A brief tutorial.**

Work Package 0

Christopher Bischof
Eral Türkyilmaz

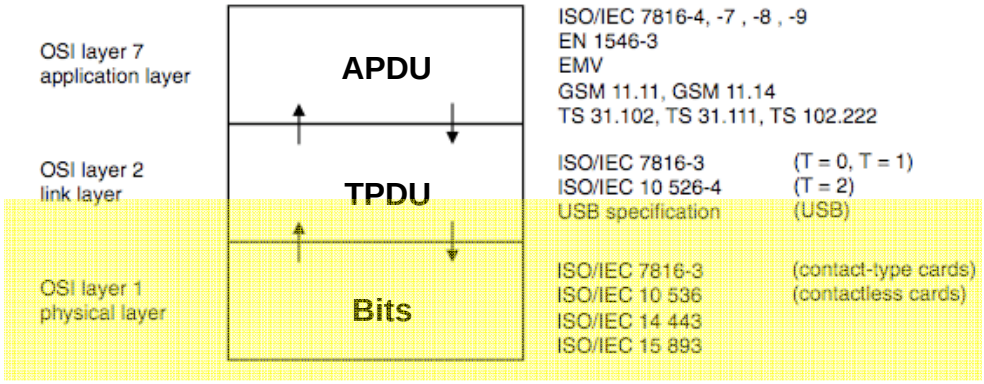
WP0 Requirements

Implementation of a smartcard interface

- Communication with the smartcard
  - Serial communication
  - Timing designated by the smartcard

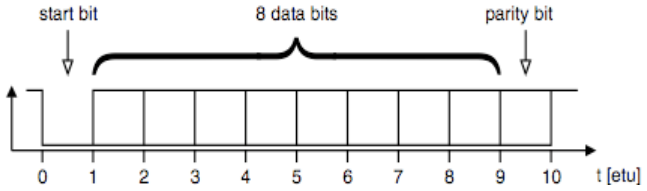
- Communication with the driver
  - Receive data from driver
  - Send data to driver

Data Link Layer To Physical Layer



Generate bitstream from the TPDU

- Serial communication on physical layer
- RS232-like communication but
  - Different voltage levels (VCC – GND)
  - Only one communication “wire” (bidirectional) I/O
  - Even parity
  - Start bit – 8 data bits – parity bit – 2 stop bits

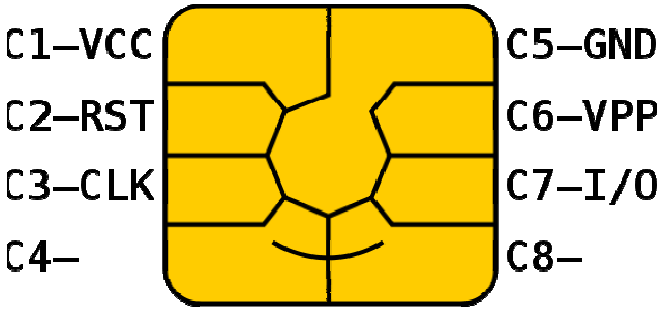


from SmartCard Handbook by Rankl, Effing
©John Wiley & Sons Ltd, 2003

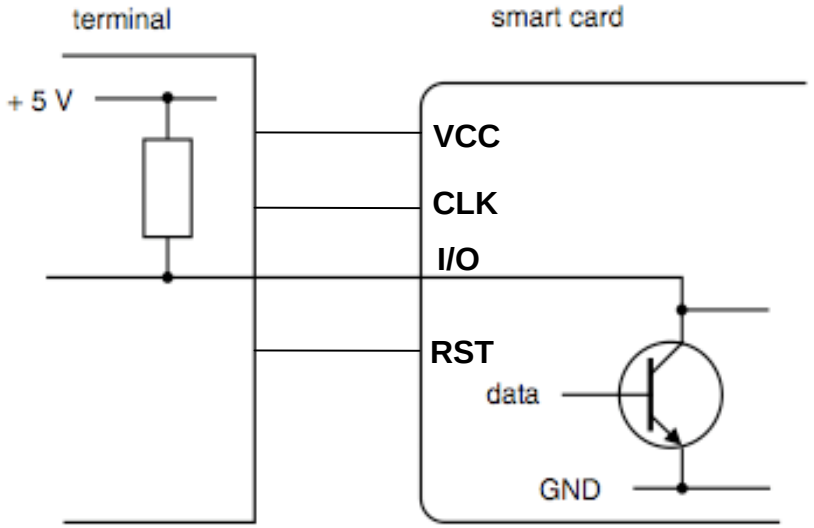
Physical Layer

Connecting the smartcard

- VCC: supply voltage
- RST: reset signal
- CLK: clock for the smartcard
- I/O: bidirectional
  - IDLE equals HIGH voltage level
    - Pull-up resistor
  - Tri-state output on terminal
    - High impedance = logic high
- GND
- VPP: programming voltage
  - Not used



http://commons.wikimedia.org/wiki/File:SmartCardPinout.svg



from SmartCard Handbook by Rankl, Effing
©John Wiley & Sons Ltd, 2003

WP0 Challenges

Learning VHDL

Howto design a hardware module

Parsing the ATR

- Sending PPS

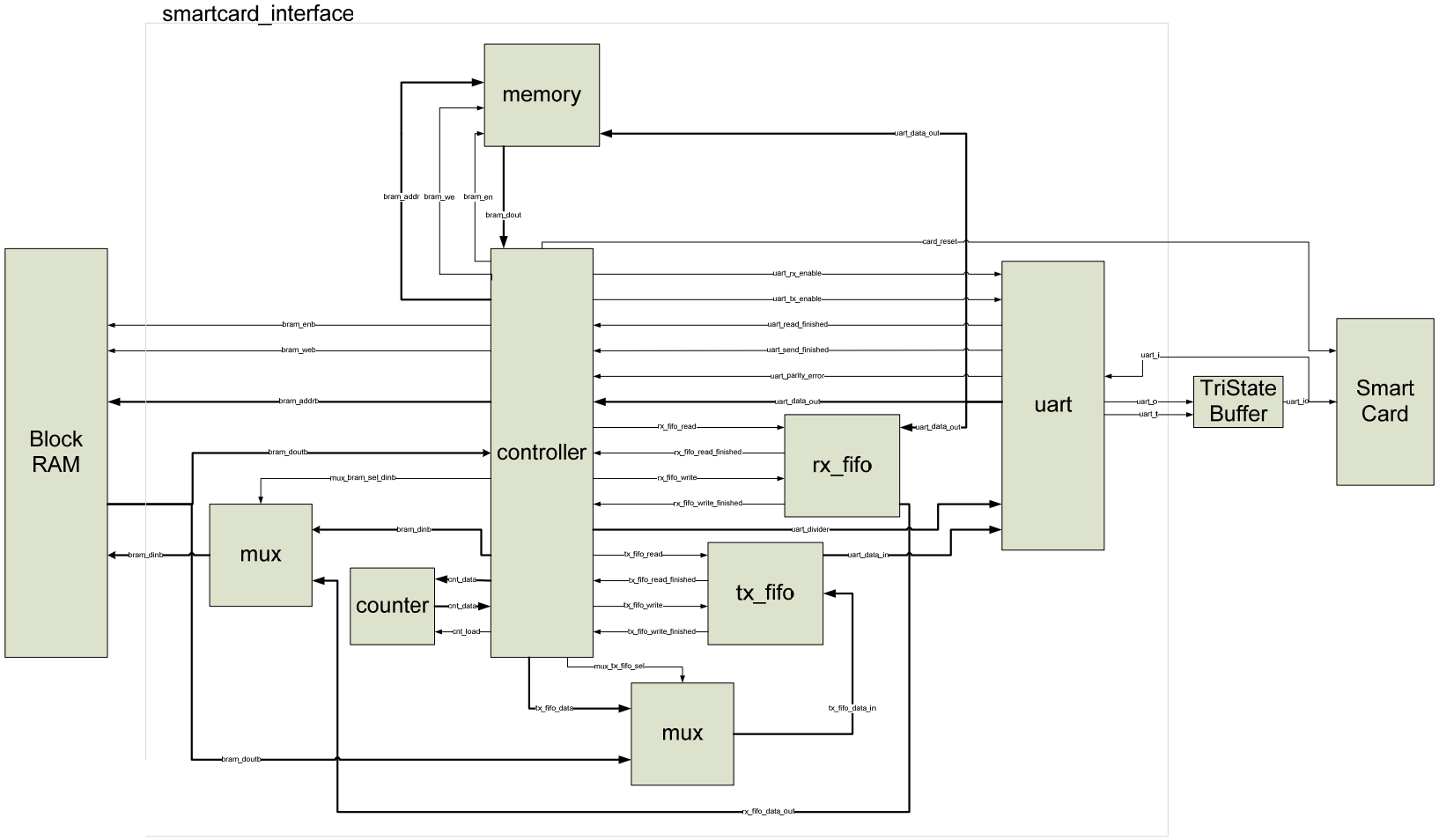
UART implementation

Getting the timing right

Error recognition

Communication with the driver

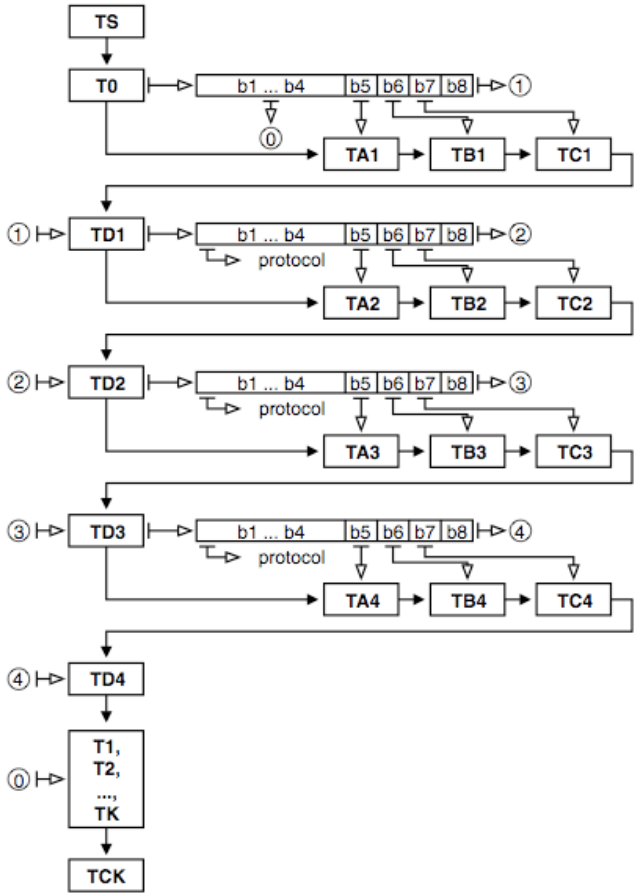
Design


smartcard_interface
memory
uart_data_out
bram_addr
bram_we
bram_en
bram_dout
card_reset
Block RAM
controller
uart
TriState Buffer
Smart Card
uart_rx_enable
uart_tx_enable
uart_read_finished
uart_send_finished
uart_parity_error
uart_data_out
uart_i
uart_o
uart_t
uart_io
bram_enb
bram_web
bram_addrb
bram_doutb
mux_bram_sel_dinb
bram_dinb
rx_fifo_read
rx_fifo_read_finished
rx_fifo_write
rx_fifo_write_finished
rx_fifo
uart_divider
mux
counter
cnt_data
cnt_data
cnt_load
tx_fifo_read
tx_fifo_read_finished
tx_fifo_write
tx_fifo_write_finished
tx_fifo
uart_data_in
mux_tx_fifo_sel
tx_fifo_data
mux
tx_fifo_data_in
bram_doutb
rx_fifo_data_out

Parsing the ATR

ATR: answer to reset

- Sent by the smartcard after a reset
- Defines the communication parameters needed by the smartcard e.g.
  - BWT: block wait time
  - CWT: character wait time
  - clock divider
  - …
- Implement this parameters for correct communication



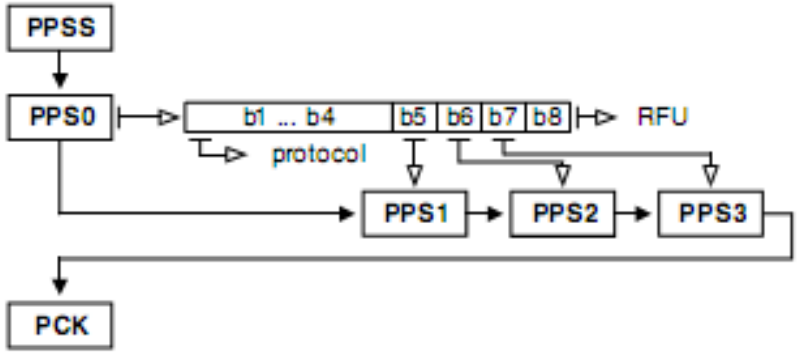
from SmartCard Handbook by Rankl, Effing
©John Wiley & Sons Ltd, 2003

Protocol Parameter Selection PPS

Terminal can change parameters for communication

Citizen card requires PPS

- In order to use higher baud rate (clock divider, …)
- Send parameters to smartcard packed in a PPS
- Smartcard sends the same PPS back if agreed
- Continue communication with new parameters



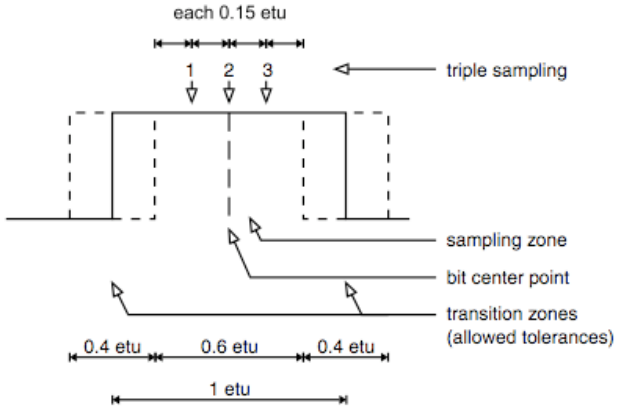
from SmartCard Handbook by Rankl, Effing

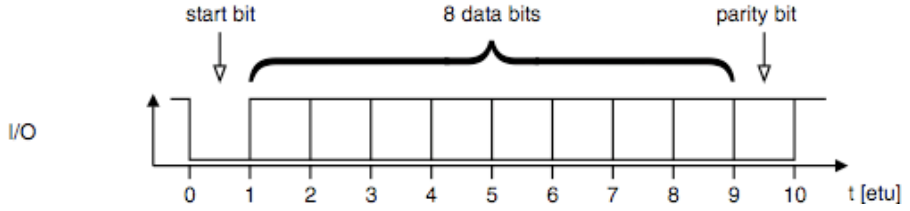
UART

Split up into two submodules

- Receiver:
  - Sampling of the input provided by the smartcard
  - Oversampling
  - 2 out of 3 decision
  - Correct timing
    - Variable baud rate
  - Startbit detection
  - Parity error detection

- Transmitter:
  - Parity bit generation
  - Correct timing
    - Variable baud rate



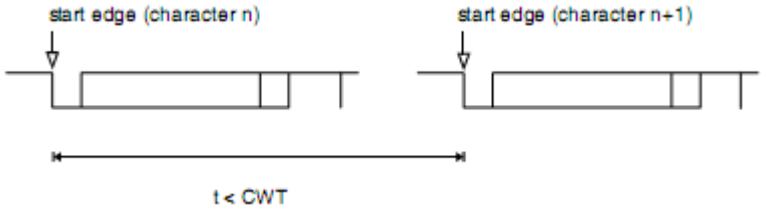


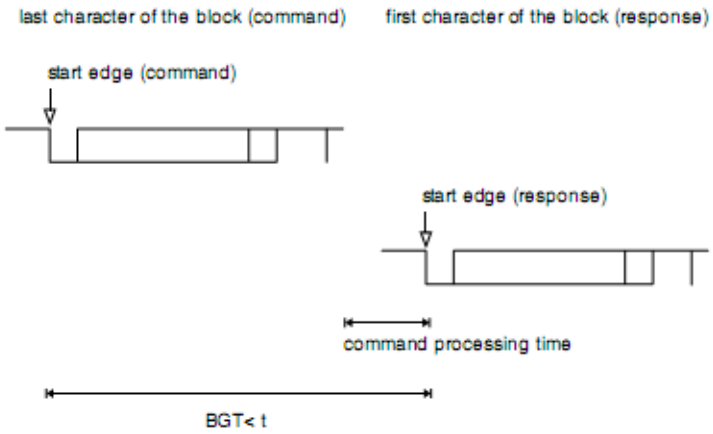
from SmartCard Handbook by Rankl, Effing
©John Wiley & Sons Ltd, 2003

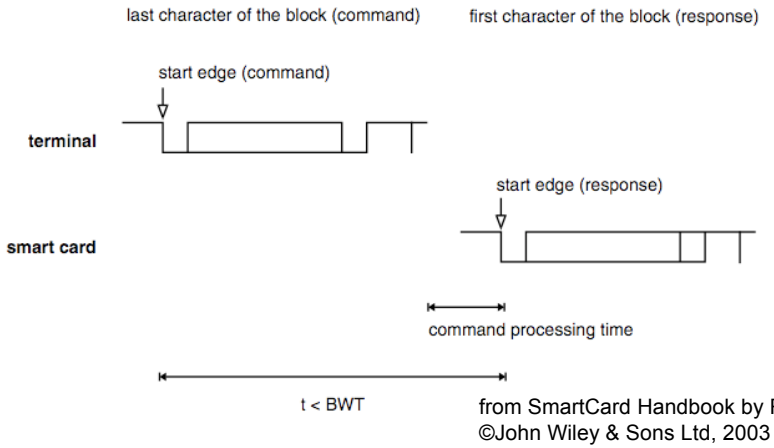
Getting the Timing Right

Timing parameters defined in ATR

- CWT: character wait time
- BWT: block wait time
- BGT: block guard time
- Clock divider: defines the duration of a single bit (higher baud rate)





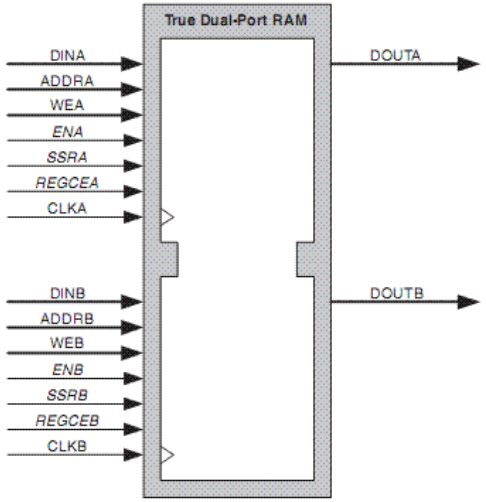


from SmartCard Handbook by Rankl, Effing
©John Wiley & Sons Ltd, 2003

Communication with the Driver

Dual ported Block RAM

- Driver uses A-ports
- Smartcard interface uses B-ports
- Defined timing for byte access
- Send byte
- Receive byte
- Status-control byte



7	6	5	4	3	2	1	0
X	X	BWT_TIMEOUT	RESET_SC	PARITY_ERROR	LAST_BYTE	RECV_DATA_READY	SEND_DATA_READY

from Block Memory Generation Documentation
by Xilinx

Literature

SmartCard Handbook by Rankl, Effing
John Wiley & Sons Ltd, 2003

Live Presentation