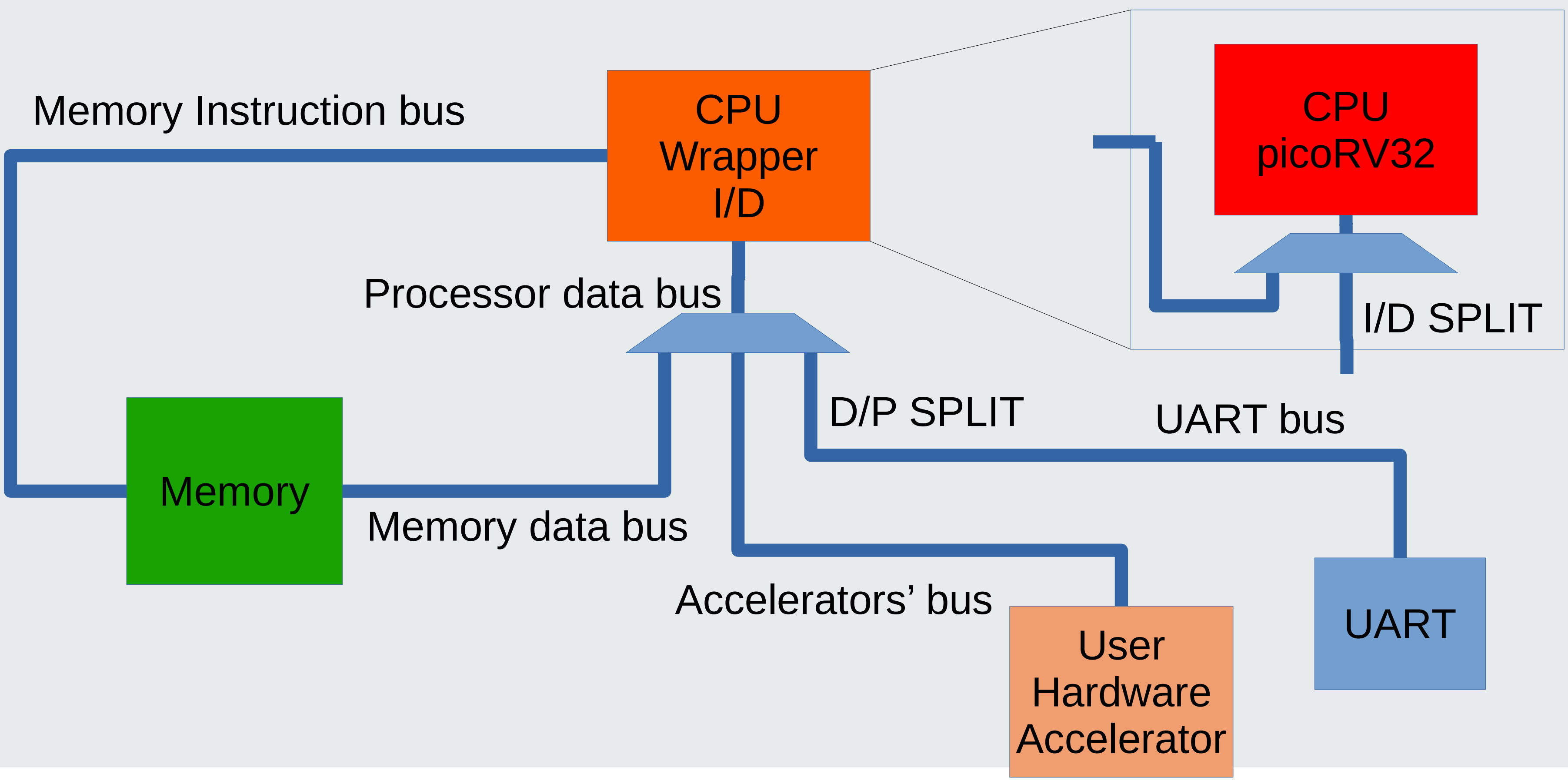
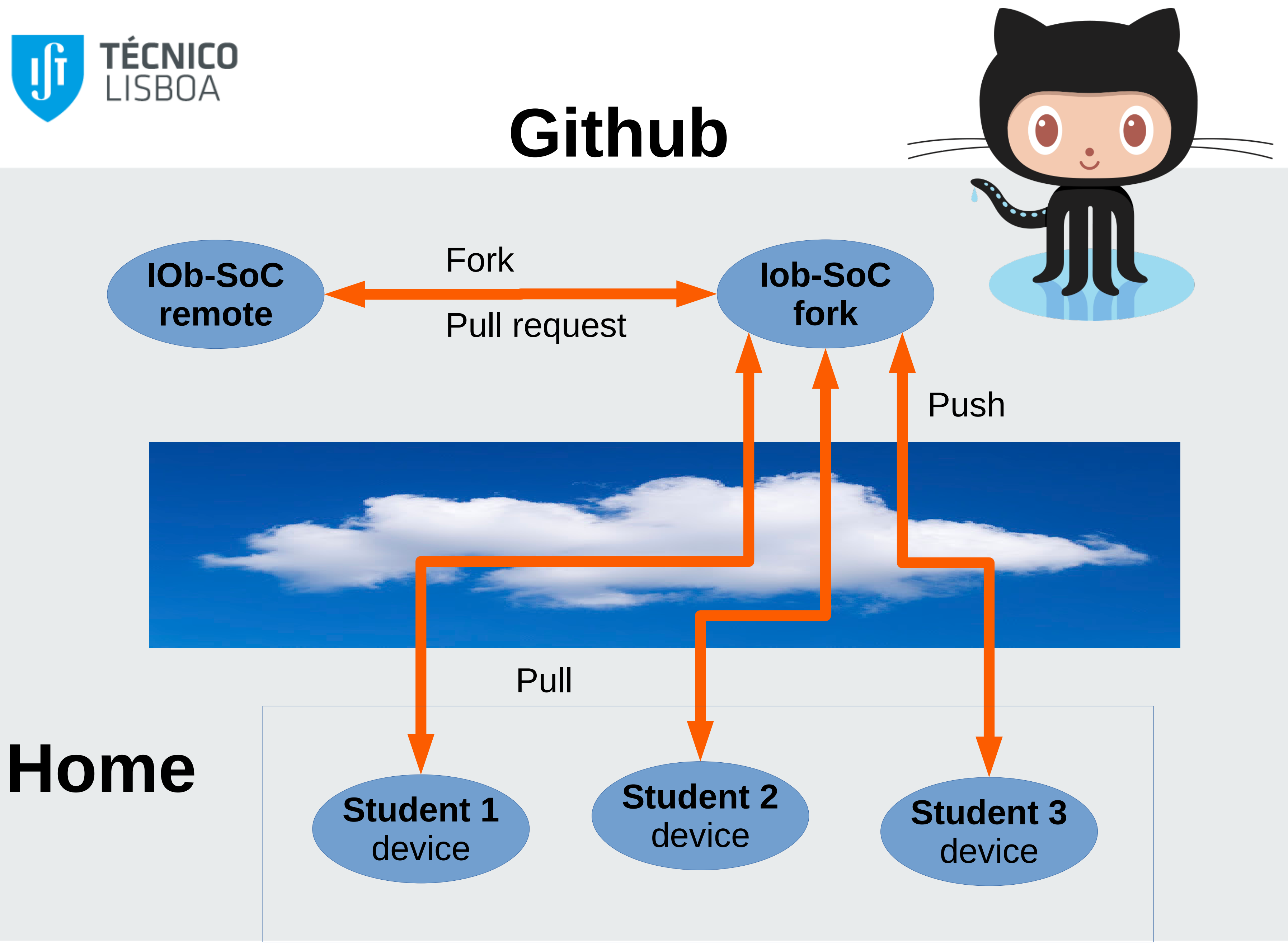
Computer Electronics

Lecture 3: Iob-SoC Tour

Iob-SoC: recap of last lesson & more



Github


IOb-SoC remote
Fork
Pull request
Iob-SoC fork
Push
Pull
Home
Student 1 device
Student 2 device
Student 3 device

IOb-SoC: quick recap

- The PicoRV32 RISC-V CPU (open-source)
- Boot ROM (has small program to load user programs)
- SRAM (on-chip memory for running small programs or storing fast access data)
- UART interface for loading the program and for communication
- *Instruction and Data L1 Caches + shared L2 cache*
- *Other optional modules: Ethernet, UART, DMA*
- **Ready to attach user modules**

Make recap, the include directive

```
include "my_wonderful_makefile_segment.mk"
dead: beef.c lib.c
    gcc -o $@ $^
    cat $<
```

Target

Prerequisites ($^)

Recipes

Short for “beef.c lib.c”

Short for “dead”

Short for “beef.c”

IOb-SoC Tour

- System configuration
- Firmware
- 32-bit CPU memory organization
- SRAM
- Peripherals
- Boot sequence
- Example boot sequence upgrade to use flash memory
- Top Makefile

IOb-SoC: system configuration

- Accomplished using makefile segment **config.mk**
  - Located at *repo* root
- Parameters of most interest in system.mk
  - FIRM_ADDR_W = log2(firmware size in bytes)
  - SRAM_ADDR_W = log2(SRAM size in bytes)
  - PERIPHERALS = list of peripherals

system.mk: FIRM_ADDR_W

- FIRM_ADDR_W = log2(firmware size in bytes)
- Note the _W suffix: stands for WIDTH in bits
- The firmware bytes can be addressed with FIRM_ADDR_W bits
- Then there are 2^FIRM_ADDR_W bytes in the firmware

Firmware

- Firmware is the resident software of an embedded system
- Sits on non-volatile memory: ROM, EPROM, Flash
- It may never change...
- Some more complex devices may allow firmware upgrades (eg. smartphone)

Memory organization for a 32-bit CPU

Full word addresses =

Byte address >> 2

Half word addresses =

Byte address >> 1

0x00	Byte 3	Byte 2	Byte 1	Byte 0
0x04			...	Byte 4
0x08				
0x0C				
0x10				
0x14	Byte 23	...		Byte 20

A 32-bit CPU can

- With single instruction can
  - Read and write single bytes (char, uchar)
  - Read and write single half words (short, ushort)
  - Read and write single long words (int, uint)
- With multiple instructions can
  - Read and write multiple bytes (number array, string)
  - Read and write multiple half words (arrays)
  - Read and write multiple long words (arrays)
- Read = Load, Write = Store

config.mk: SRAM_ADDR_W

- The SRAM bytes can be addressed with SRAM_ADDR_W bits
- Then there are $2^{SRAM\_ADDR\_W}$ bytes in the memory
- If SRAM_ADDR_W is greater than FIRM_ADDR_W then the firmware ***<u>fits</u>*** in the SRAM and can run on it

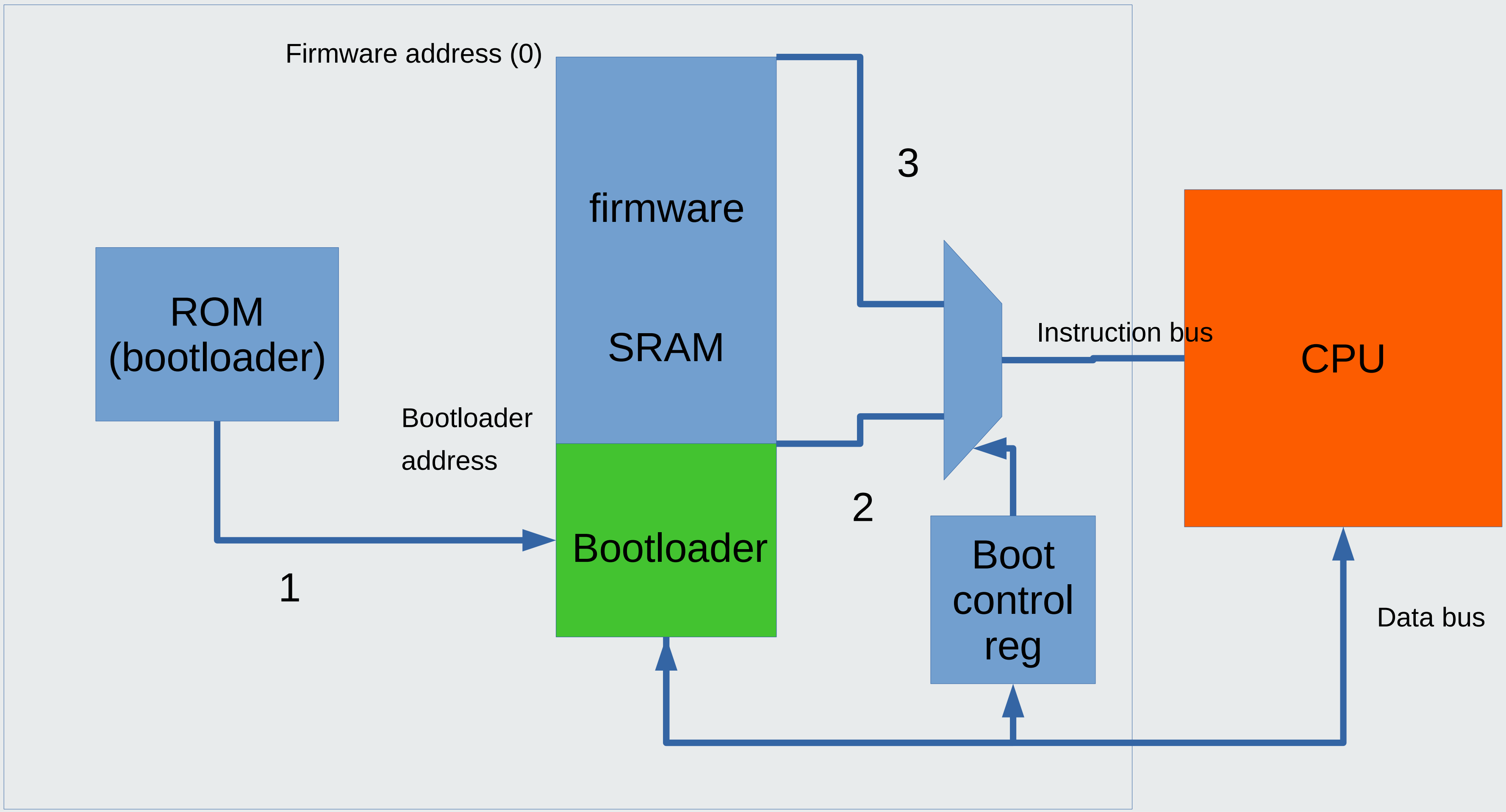
config.mk: PERIPHERALS

- List of peripherals in the system
- By default contains only one peripheral: the UART
- Users can make peripherals and attach them to the IOb-SoC peripheral bus
- For this to happen
  1) the peripheral repository must follow a certain structure
  2) the peripheral repository must be placed in the ***submodules*** directory
  3) the directory name of the peripheral repository must be added to the the PERIPHERALS list

Iob-SoC boot sequence

1) Power on; Hardware reset (hard reset)
2) Bootloader program copied from ROM to SRAM by a special hardware circuit; bootloader runs on SRAM
3) When running, Bootloader awaits for user firmware to arrive via UART
4) Bootloader receives firmware from UART and loads it to SRAM
5) Bootloader sets Boot Control Register and restarts system
6) Boot Control Register tells bootloader to start running from a different address where firmware has been stored

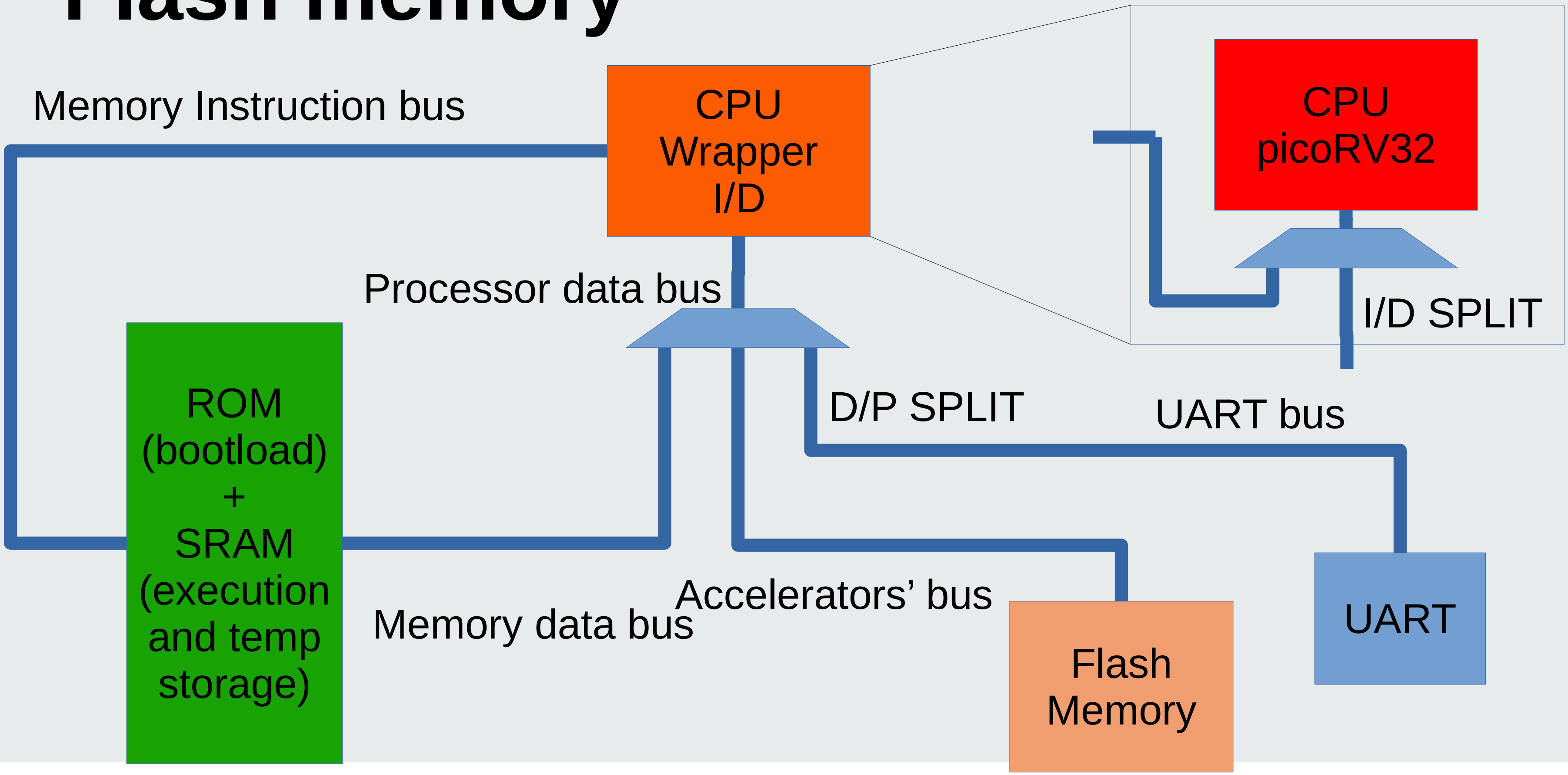
Iob-SoC boot sequence illustrated



Example: Upgrading IOb-SoC with Flash memory

1) Power on; Hardware reset (hard reset)
2) Bootloader program copied from ROM to SRAM by a special hardware circuit; bootloader runs on SRAM
3) Bootloader awaits for firmware to arrive by UART
4) Bootloader receives firmware from UART and loads it to FLASH
5) Boot loader reboots system (soft reset)
6) Boot loader starts again and awaits for firmware to arrive by UART
7) Bootloader times out and copies firmware from FLASH to SRAM
8) Bootloader sets boot control register and reboots system
9) Boot control register tells bootloader to start running from a different address where firmware has been stored

Example: Upgrading IOb-SoC with Flash memory



IOb-SoC: top Makefile

- The top Makefile contains the following ***targets***
  - sim-build: compile for simulation but do not run. Used for checking compile errors
  - sim-run: simulate, eg. make sim-run INIT_MEM=1
  - sim-waves: simulates and displays waveforms using gtkwave
  - sim-clean: cleans all simulation generated files
  - sim-test: automatic simulation test (takes a long time due to simulation speed)
  - fpga-build: compile for fpga but do not run
  - fpga-run: run on FPGA, eg. make fpga-run INIT_MEM=0
  - fpga-clean: cleans all fpga compile generated files
  - fpga-test: automatic fpga test (takes a long time due to several compiles)
  - fw-build: builds the firmware and bootloader to check compile errors (called by simulation or fpga)
  - fw-clean: cleans embedded software compiler generated files
  - doc-build: makes all the documents in the document directory
  - doc-clean: cleans all document compilation generated files
  - doc-test: automatic documentation test
  - clean: complete detox, cleans every generated files

IOb-SoC: README, LICENSE and test directory

- The README file
  - Several styles exist
  - Adopted style: brief description of the repo followed by what you can do with it, prompting action!
- LICENSE file
  - States under which conditions the files in repo can been used
  - Adopted license created by MIT with 2 clauses only
    - Keep copyright notice in the main files
    - Keep license text in the main files
- Testing
  - Golden test logs are saved in the simulation, fpga and documentation directories
  - Generated test log is compared to golden logs (.expected files) to yield pass/fail result