Jtagfs: Acid your ARM

Gorka Guardiola Múzquiz

Lsub, Rey Juan Carlos University

IWP9 2012

Intro: JTAG

- JTAG: architecture to test digital circuits
- JTAG support on µprocessors
  - Freeze the µprocessor, do anything to it, start it again
  - Like debugger but for µp
- All ARMs have JTAG support
- Many machines include: JTAG to USB chip

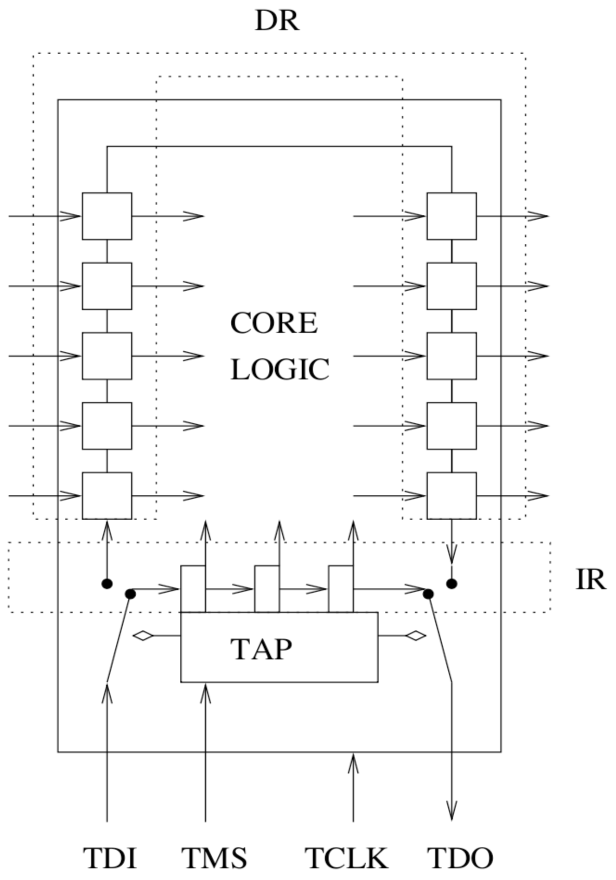
What I wanted JTAG for

- Had rewritten assembler for exceptions
- Could not figure out what was wrong
- A lot of people where having ARM hw debugging problems

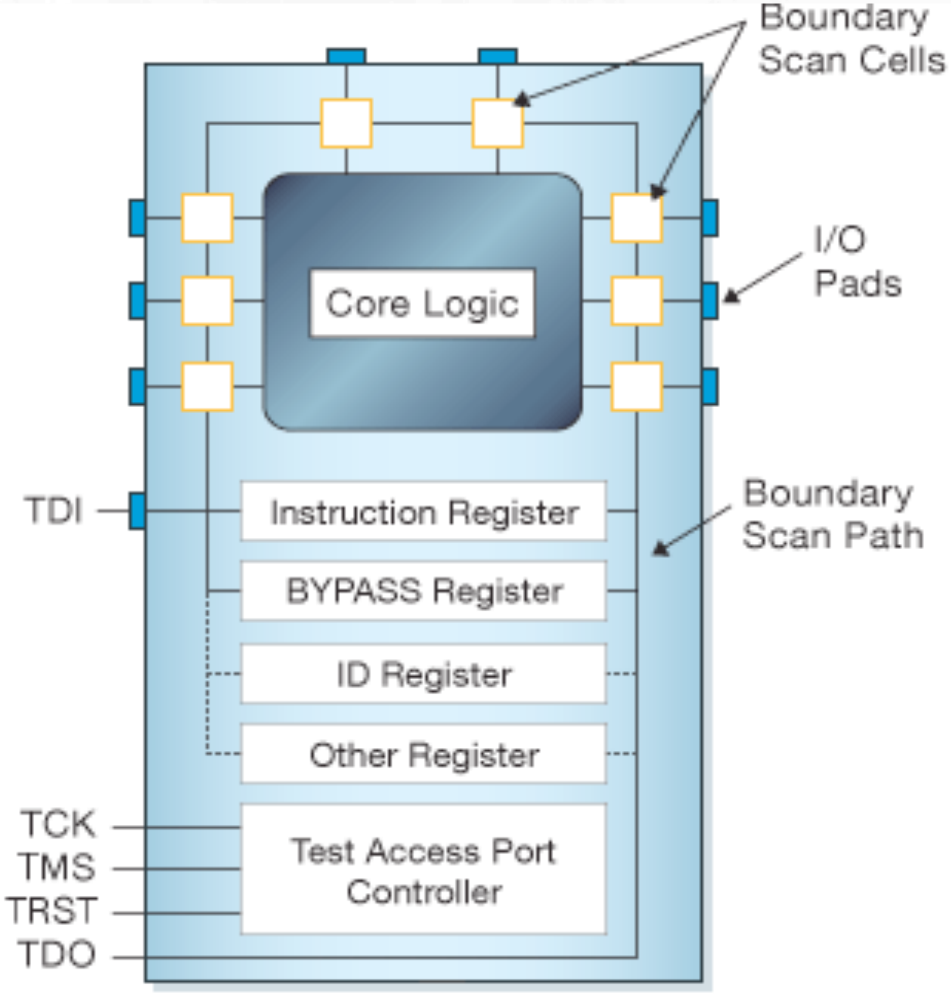
JTAG basics

- Shifting registers: **shift a chain of data** in/out
- Two main registers
  - Data (DR)
  - Instruction (IR)
  - (there are some others, BYPASS...)
- Five cables, data in (TDI), data out (TDO), clock (TCK), Test Mode Select (TMS), reset (TRST)
- TMS: transverse the state machine
- Registers + Signals = TAP (test access point)

JTAG basics



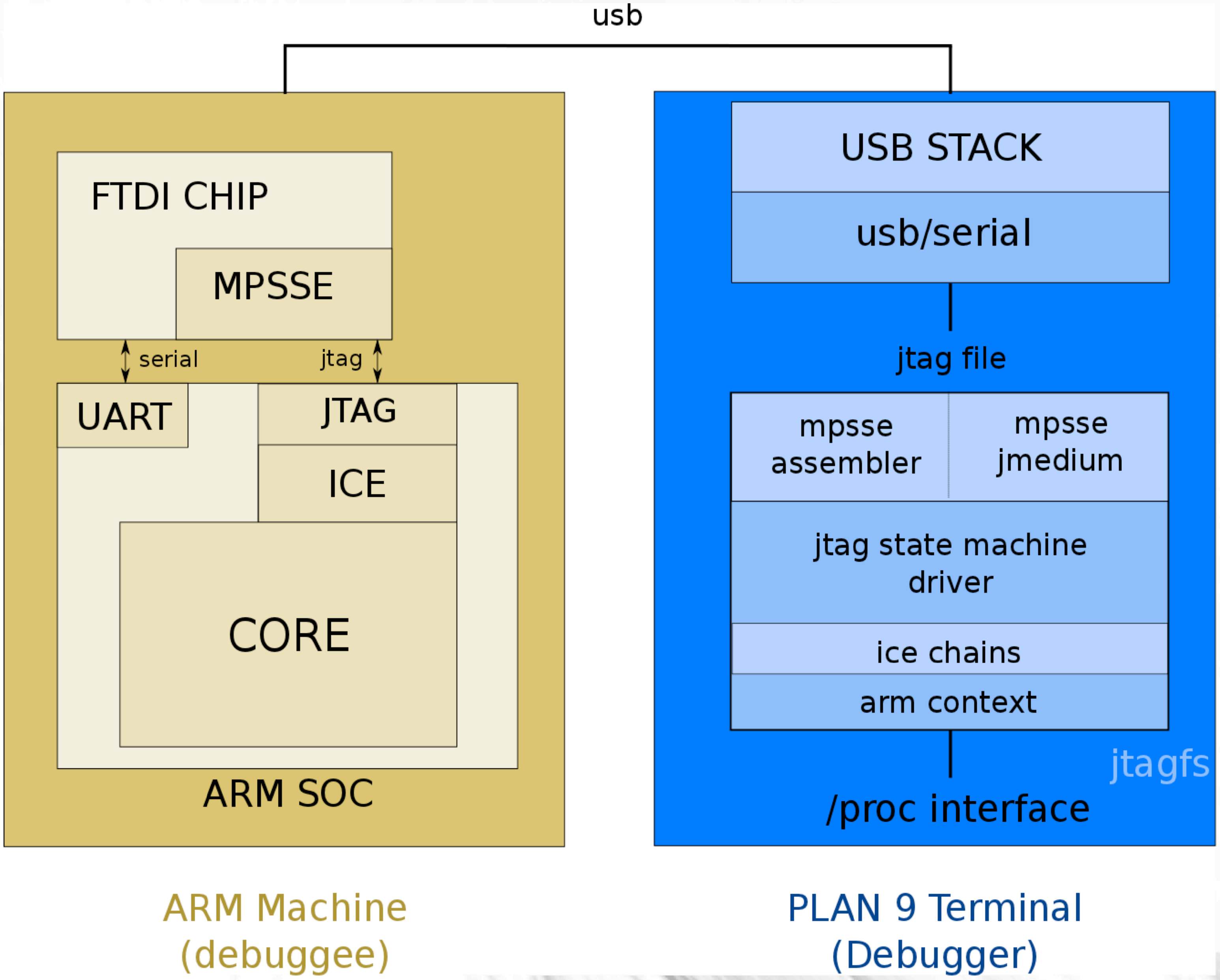
JTAG basics


Boundary Scan Cells
I/O Pads
Core Logic
Boundary Scan Path
TDI
Instruction Register
BYPASS Register
ID Register
Other Register
TCK
TMS
TRST
TDO
Test Access Port Controller

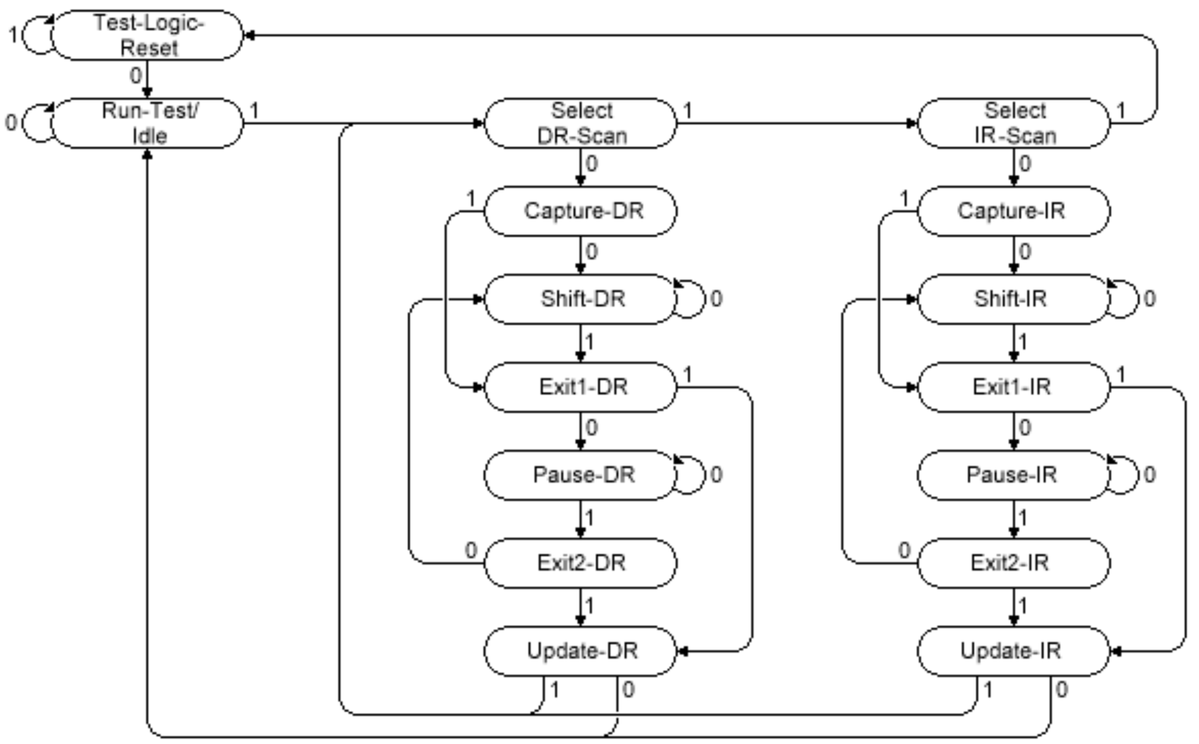
JTAG basics

- TAPs: serial or in parallel
- **Serial** means all the chains are longer
- **Parallels** means they need to be selected
  - Chain identifier
  - Chain selection instruction

Architecture



JTAG state machine

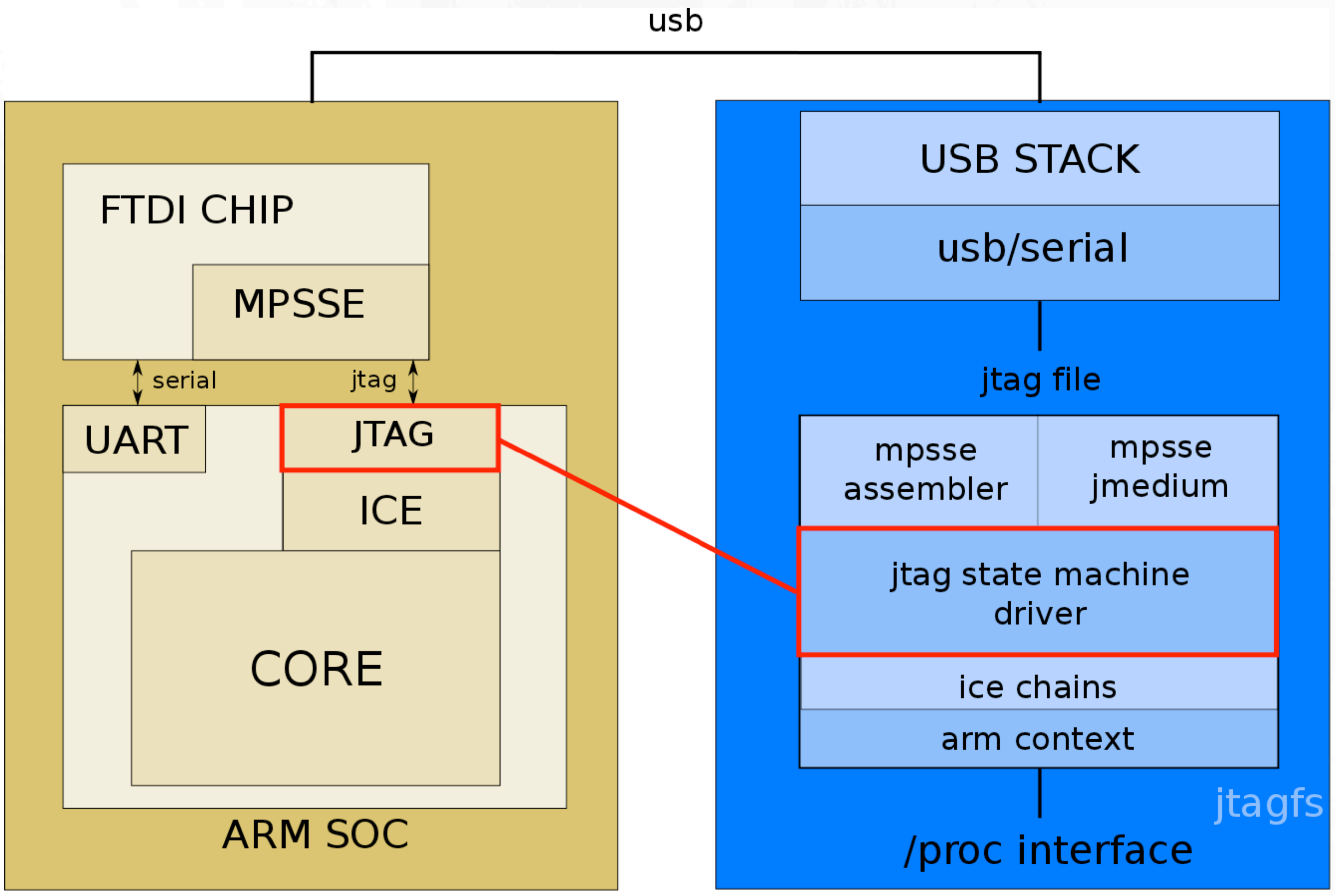


On each TCLK, TMS is read and the state updated

JTAG state machine

- Select, Capture, Shift, Update
- **Select**, connect TDI, TDO
- **Capture** means read from pads
- **Shift** is shift in data on each clock
- **Update** actually make the data shifted count
  - DR: put it in the pads, make it work
  - IR: make the instruction “run”
  - Depends on current instruction, see core manual

JTAG s.m. Driver



ARM Machine
(debuggee)

PLAN 9 Terminal
(Debugger)

JTAG s.m driver

- Two parts
- An abstraction, **JMedium**
- A general **state machine navigator**

JTAG s.m. driver JMedium

```
Struct JMedium{
        int             (*regshift)(JMedium *jmed, ShiftTDesc *req, ShiftRDesc *rep);
        int             (*flush)(void *mdata);
        int             (*term)(void *mdata);
        int             (*resets)(void *mdata, int trst, int srst);
        int             (*rdshiftrep)(JMedium *jmed, uchar *buf, ShiftRDesc *rep);
};
```

- Has buffering (quite important) implemented by **flush()** and **term()**
- Resets to set **trst** and **srst**, both signals
  connected with a circuit, need to know details
- Regshift, rdshift, main functions to implement
  - ShiftRDesc useful for async replies rdshiftrep()
  - Response comes in nbytes (last partially filled), nbits (byte partially filled)

JTAG s.m. Driver

- Uses shortest path (BFS, small branching factor)
- Only need to tell it register, data, operation, fill in a ShiftTDesc, ShiftRDesc to read response

```
req.reg = TapDR;
req.buf = data;
req.nbits = 5;
req.op = ShitPauseIn|ShiftOut|ShiftNoCommit;
```

- Only one active TAP, the rest in BYPASS

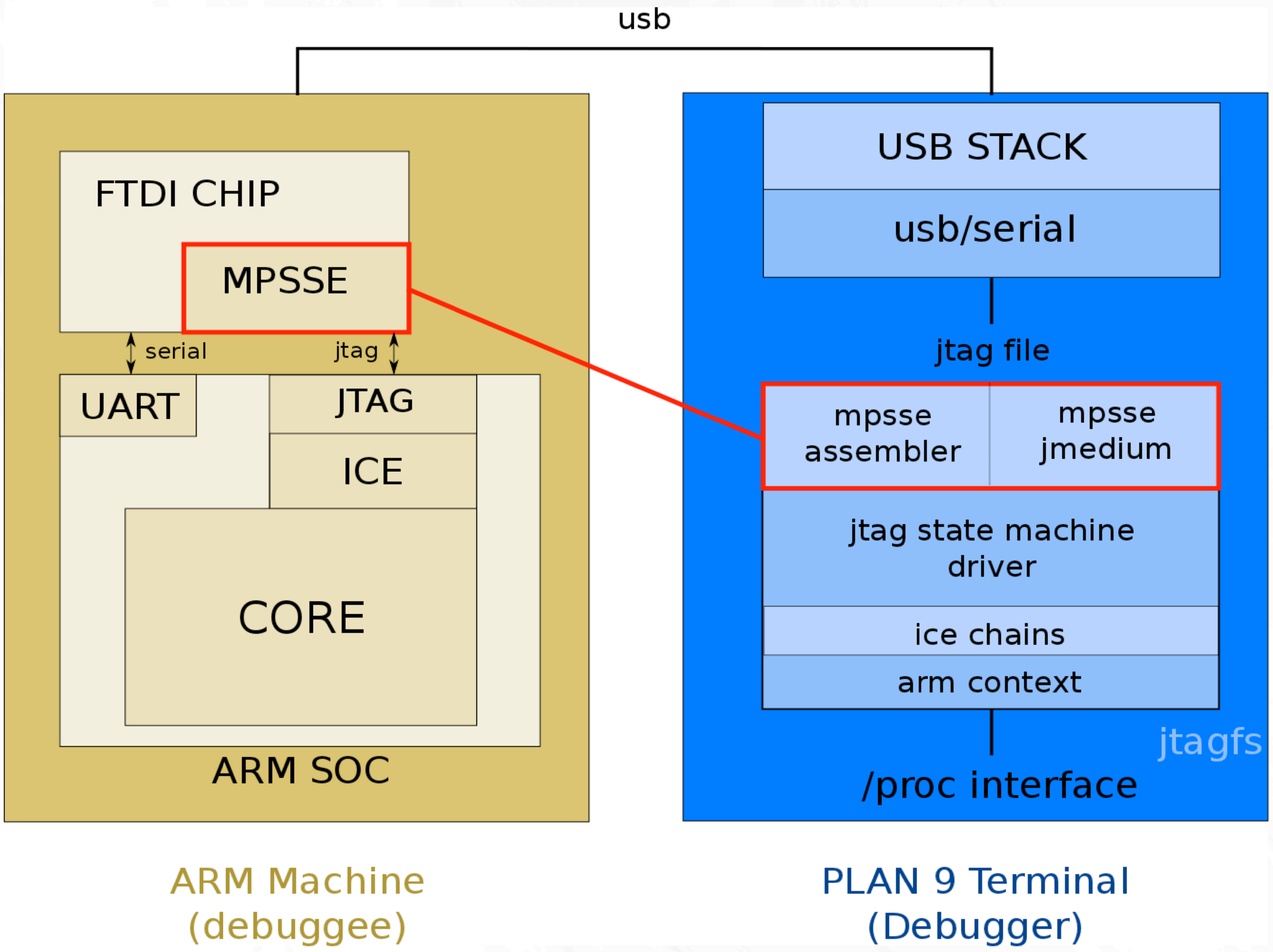
JTAG s.m. Build your own

- To implement regshift, JTAG state navigator,
- tap.[ch]
- Tell it the state, gets you there, shortest path
  - pathto()
  - movepath(), concatpath()

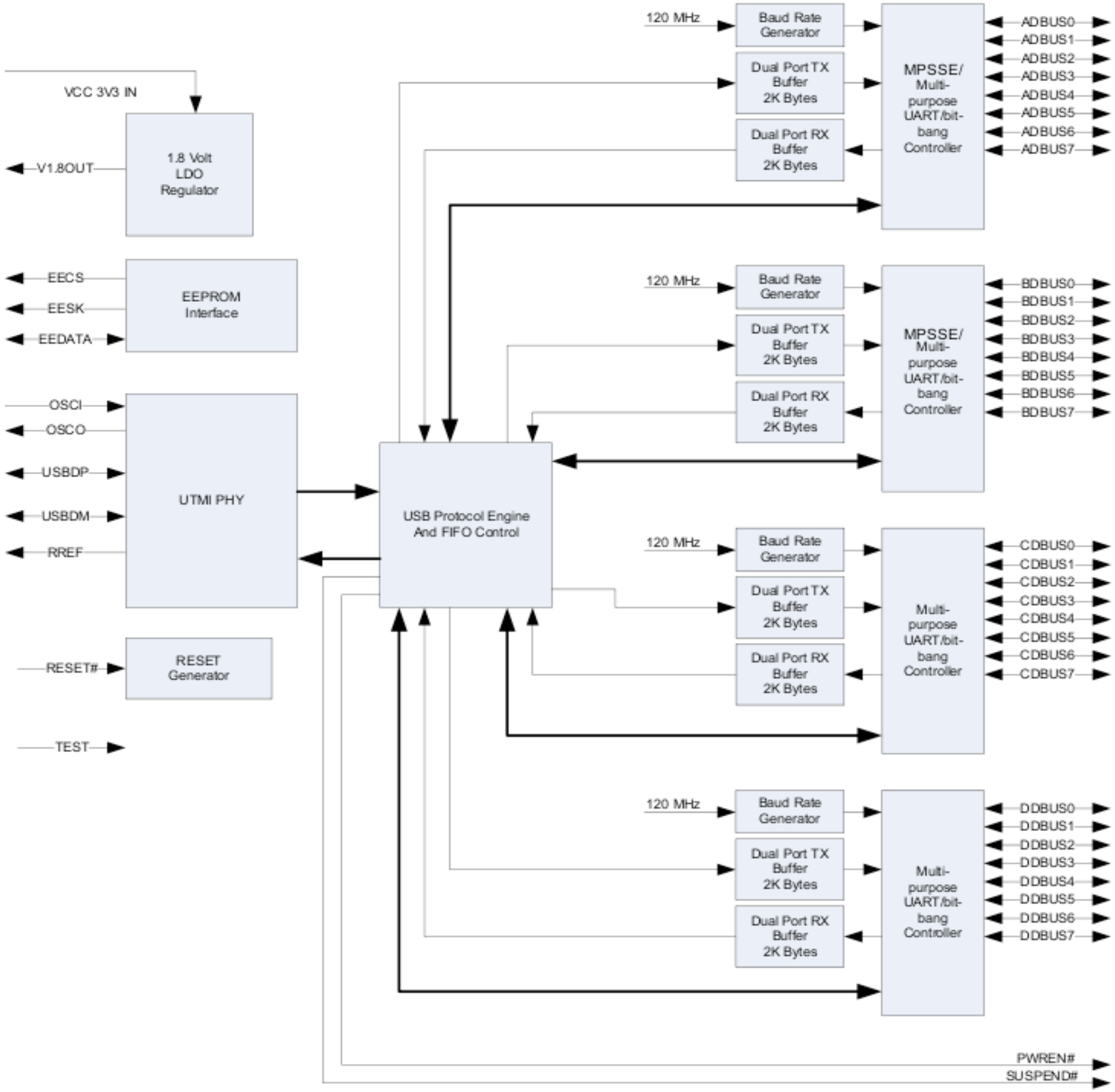
MPSSE

- USB: too much latency
- Small processor on the machine
- You download data with instructions
- MPSSE: Multi-Protocol Synchronous Serial Engine, made by FTDI
- General USB to any kind of serial.
- Can act as a small PIC
- Makes you coffee (almost)

MPSSE


usb
FTDI CHIP
MPSSE
serial
jtag
UART
JTAG
ICE
CORE
ARM SOC
ARM Machine
(debuggee)
USB STACK
usb/serial
jtag file
mpsse
assembler
mpsse
jmedium
jtag state machine
driver
ice chains
arm context
jtagfs
/proc interface
PLAN 9 Terminal
(Debugger)

MPSSE


120 MHz
Baud Rate Generator
Dual Port TX Buffer 2K Bytes
Dual Port RX Buffer 2K Bytes
MPSSE/ Multi-purpose UART/bit-bang Controller
ADBUS0
ADBUS1
ADBUS2
ADBUS3
ADBUS4
ADBUS5
ADBUS6
ADBUS7
VCC 3V3 IN
V1.8OUT
1.8 Volt LDO Regulator
EECS
EESK
EEDATA
EEPROM Interface
120 MHz
Baud Rate Generator
Dual Port TX Buffer 2K Bytes
Dual Port RX Buffer 2K Bytes
MPSSE/ Multi-purpose UART/bit-bang Controller
BDBUS0
BDBUS1
BDBUS2
BDBUS3
BDBUS4
BDBUS5
BDBUS6
BDBUS7
OSCI
OSCO
USBDP
USBDM
RREF
UTMI PHY
USB Protocol Engine And FIFO Control
120 MHz
Baud Rate Generator
Dual Port TX Buffer 2K Bytes
Dual Port RX Buffer 2K Bytes
Multi-purpose UART/bit-bang Controller
CDBUS0
CDBUS1
CDBUS2
CDBUS3
CDBUS4
CDBUS5
CDBUS6
CDBUS7
RESET#
RESET Generator
TEST
120 MHz
Baud Rate Generator
Dual Port TX Buffer 2K Bytes
Dual Port RX Buffer 2K Bytes
Multi-purpose UART/bit-bang Controller
DDBUS0
DDBUS1
DDBUS2
DDBUS3
DDBUS4
DDBUS5
DDBUS6
DDBUS7
PWREN#
SUSPEND#

MPSSE

- Each port connected to a serial interface (different USB endpoints)
  - Serial
  - Jtag
- Two parts, serial communications, PIC
- In usb/serial only serial communications
- Jtag directory like any other serial port (for now already configured, parameters are different)
- Programming, controlling, outside of usb/serial

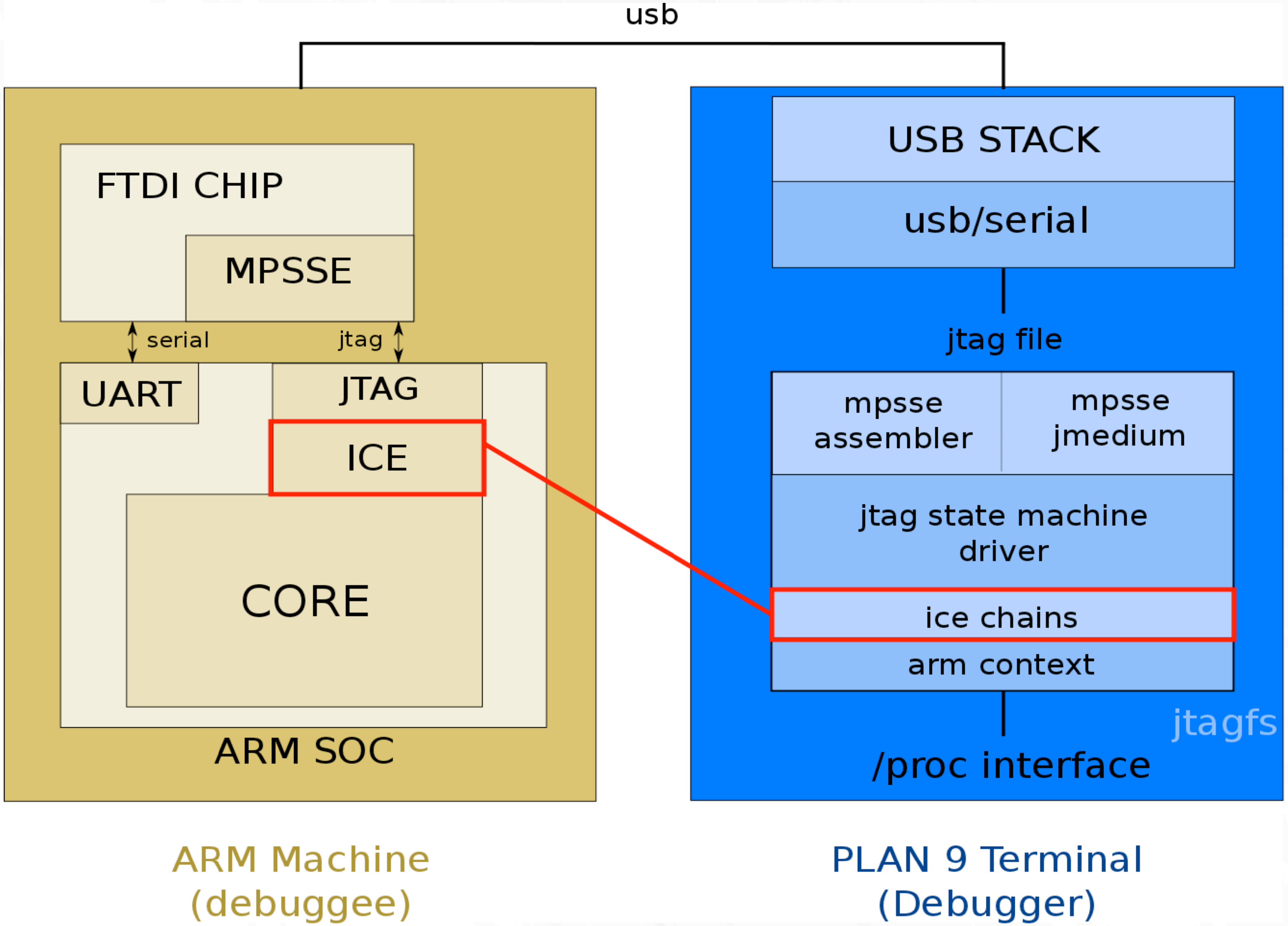
MPSSE

- Complex to program
- Several “addressing modes”, clocking...
  - Shift bits, shift bytes, little bit-endian, byte-endian, edge, level...
- Difficult to debug
- Wrote mini-assembler, assemble on the fly
- Can print mini-assembler to know what is going on

MPSSE assembler (ma)

```
DataOutIn EdgeDown EdgeUp LSB 3 0x42 0x34 0x56
DataOutIn EdgeDown EdgeDown LSB 3 @
DataOutIn EdgeDown EdgeUp LSB B3 0x42
DataOutIn EdgeDown EdgeDown LSB B3 @
TmsCsOut EdgeDown MSB B0x7 0x7
TmsCsOut EdgeDown LSB B7 0x7
TmsCsOutIn EdgeDown EdgeUp LSB B0x7 0x7
MCURd 0x34
SendImm
WaitIOHigh
AdaptClkDisab
```

ICE


usb
FTDI CHIP
MPSSE
serial
jtag
UART
JTAG
ICE
CORE
ARM SOC
USB STACK
usb/serial
jtag file
mpsse assembler
mpsse jmedium
jtag state machine driver
ice chains
arm context
jtagfs
/proc interface
ARM Machine (debuggee)
PLAN 9 Terminal (Debugger)

MPSSE

- Using the assembler implemented a Jmedium
- Could be improved, just good enough
- The assembler can be used independently for other MPSSE endeavours
- The JMedium completely abstracts the MPSSE

ARM ICE

- Defines several chains (like parallel TAPs), instruction to select them (IR)
- Chain 1: inject instructions
- Chain 2: access to debug registers
- Chain 15: access to MMU
- The **endianness of bits** is particularly weird
  - Functions to pack, unpack
  - mini-language was an overkill, undone
- Chain 15 is different in ARM 7 and ARM 9 (we are using MCR and MRC and Chain 1)

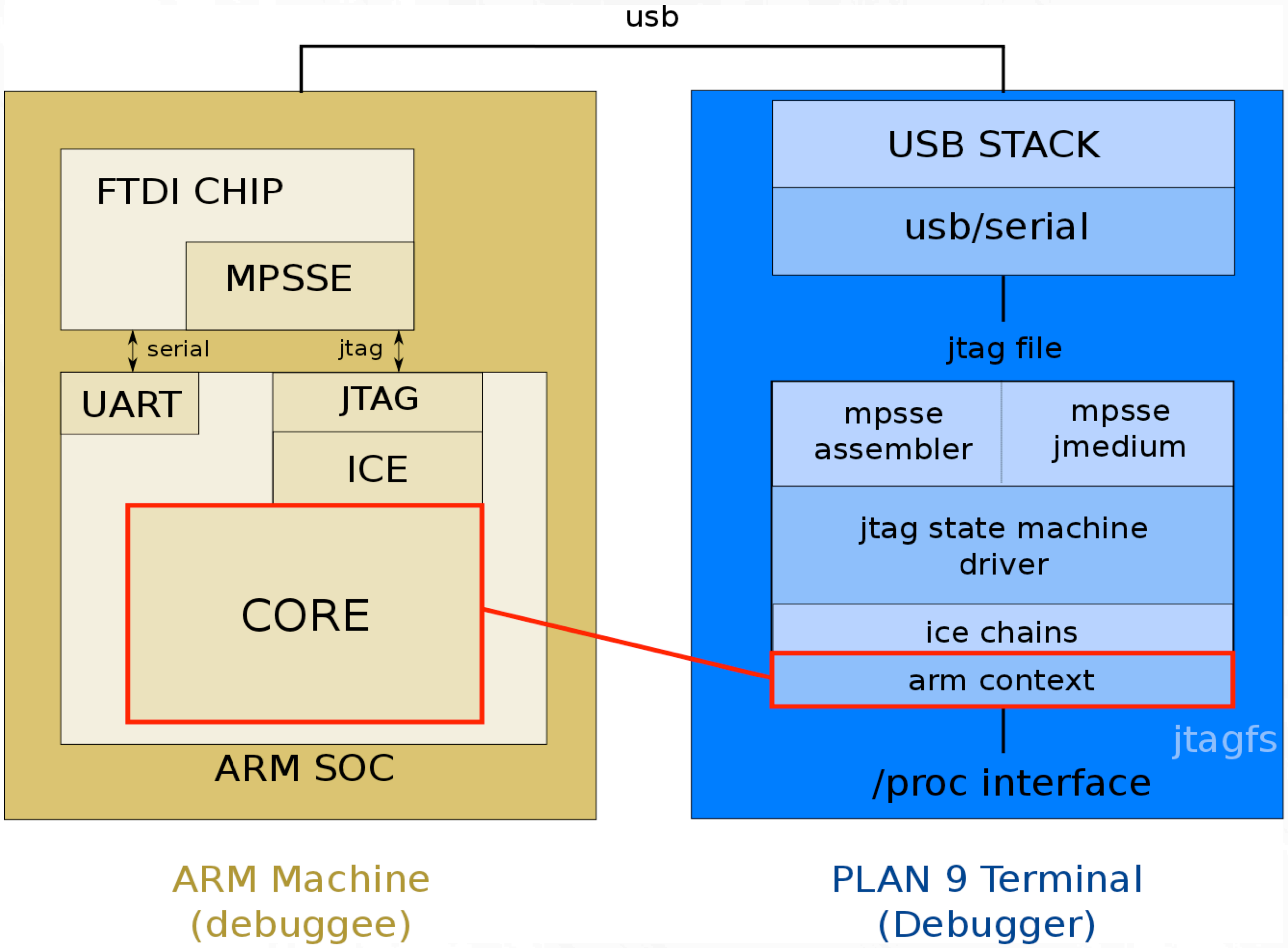
ARM ICE: Chain 1

- To **inject instructions** to a core **in debug mode**
- When on debug mode: processor on a different clock, isolated
- To read memory or access peripherals must go back into real clock
- Inject directly into pipeline (be careful to flush after you, injecting NOPs)

ARM ICE: Chain 2

- **Access debug registers**
- Can be accessed also from inside the core
- Can stop the core, start the core, set watchpoints, breakpoints, etc.
- Only one breakpoint, watchpoint in the Sheeva, more would be better

ARM context


usb
FTDI CHIP
MPSSE
serial
jtag
UART
JTAG
ICE
CORE
ARM SOC
ARM Machine
(debuggee)
USB STACK
usb/serial
jtag file
mpsse assembler
mpsse jmedium
jtag state machine driver
ice chains
arm context
jtagfs
/proc interface
PLAN 9 Terminal
(Debugger)

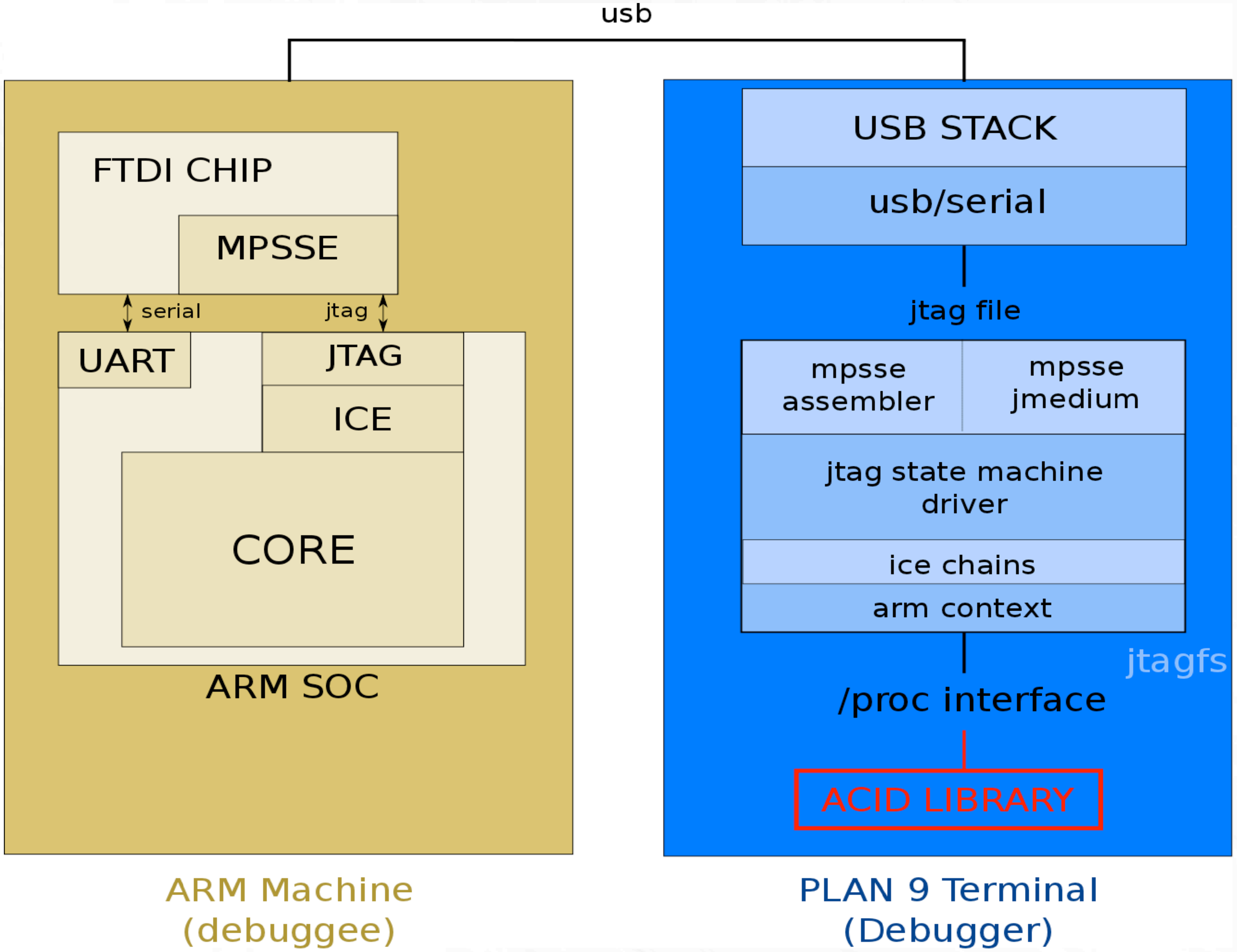
ARM context

- Save state of processor before debug mode
- Careful when restoring it after, very delicate
- The PC and other things are modified by injecting instructions
- Depending on where and how we entered, need to go back differently (calculate PC and set interrupt flags mainly)

/proc interface

- Endianness of interface, like in ARM
- Translated to host of jtagfs after reading
- Similar to rdbfs
- Need to be able to write on some of them
- We export more registers (the mmu registers)
- Offer access to memory out of segments of the binary, MACH, mmap'd registers

ACID Library



ACID Library

- Abstract common operations
- ACID is best for this
- Can set watchpoints, reset to debug, etc.
- Needed to remap the things out of the binary
  - Undocumented map() function
- Able to stop the watchdog, amazing
- Can stop the processor, ask for MMU translations

Jtagfs debugging

- Quite difficult, many layers
- Having debug flags to have each layer print, very useful
- Many corners, bit endianess confusing and even wrong in some documents
- Best documentation, OpenOCD+wireshark
- When getting out of debug mode, if PC is not set, the machine is frozen
- Jtagfs can be used as documentation/to learn

Experience

- Good for debugging
- Found my bug, one of the instructions was not supported in this ARM (store in the other stack)
- Could use more breakpoints/watchpoints
  - Are software breakpoints doable?
- Slow (not for debugging, but other purposes),
  - Could use DCC, multi-instruction inject

Portability (debuggee)

- Add a SOC, add CPUID, SRST/TRST circuit, extra serial TAPs to ignore
- Only one watchpoint/breakpoint, could be changed
- All ARM 7, ARM 9 should work
- Add medium, as explained above (other JTAG controllers)

Future work

- Only Feroceon (in Sheeva) tested
    - Bug: unresponding if too late (hw?, OpenOCD too)
- Added support for Armada, untested
- ETM support (traces), other chains
- Setting debug mode in panic (port-mortem inspection)
- DCC, Chain 2, inside coprocessor 14, MCR, MRC
- Loader (maybe using DCC, small assembly program)
- Acid a linux/U-boot (/proc works) with some more ELF support for the symbol tables
- Multi-ICE support (multicore)

Further in the Future work

- You can stop the processor, do anything to it, start it again
- Migration of the state of the processor, virtual hardware machines
- Debugging of loader+kernel
- Hot patching of the kernel
- Education
- The limit is the imagination

Related work

- OpenOCD and similar:
  - Adapt to a regular debugger like gdb, not really programmable
  - Can write batch scripts for OpenOCD but limited too, cannot access symbols, etc.
  - OpenOCD: ported to any debuggee
- Hw+Sw debug solutions: expensive, unavailable
- Others: similar to OpenOCD or very limited capabilities
- /proc + acid + verbose flags = unique
- Closest is rdbfs, but can do much less

Questions?