CMS overview on infrastructure

Marc Dobson

SoC Interest Group, 29th April 2020

Acknwledgements: P. Zejdl, N. Dzemaili, K. Suresh Mor

Workshop June 2019

- See the ATLAS & CMS SysAdmin talk for more details
- Will follow up of some of the issues raised then
- Topics:
  - Operating system:
    - Intro by Ralf and presentation by Petr
  - Network: the main gist of this presentation
    - MAC address storage and retrieval
    - ClientID:
      - For crate, shelf manager, IPMC
      - For Zynq SoC
  - Linux Serial console
    - (some questions)

Ethernet MAC Address

- Ethernet networking = MAC address !
  - Unique like a serial number
  - Single per network end-point
  - Requires storage of MAC on the HW

- Suggest MAC is stored in I2C EEPROM connected to Zynq SoC for all designs going in to the network (same mechanism for all)
  - Unfortunately not as simple as that !
  - I2C EEPROM chips can be different
  - I2C EEPROM is usually behind a multiplexer
  - MAC may be stored at different location of EEPROM
  - Network Boot:
    - U-Boot needs the device tree info (how to get to EEPROM)
    - U-Boot network device driver needs to know the offset for MAC info in EEPROM
  - U-boot needs the above also to provide the MAC to the Linux kernel via editing the device tree passed to the kernel

ClientID: reminder from workshop

- ATCA spec. foresees usage of Client ID based DHCP
  - Each Shelf is identified by a Shelf address (an arbitrary <=20 character long ASCII string):
    - Should be unique in a DHCP domain
    - For us it could be “Building”, “Rack” and “U” (always unique)
    - Could drop “Building” in experiment networks
  - IPMC gets this info (FRU data) from Shelf Manager
  - ATCA boards, actually IPMC on the board, are located in the shelf by the Physical Slot Number
  - AMC Modules on an ATCA Carrier (actually MMC), are located by the carrier Slot Number and by a Module position (e.g. B1)
  - ComExpress or Zynq controllers could get this info from IPMC
- ClientIDs could look something like:
  - Crate-slot-component
    - E.g. atca-c2e32-10-08-ipmc “simplistic” more news at next IG
  - Need to understand how to identify the end point (i.e. components) foreseen
  - Exact scheme will converge with testing of different HW boards

How ?
(see next slide)

ClientID: Zynq usage

- For the Zynq or ComE control endpoint to use ClientID it needs to get:
  - Shelf Address
  - Physical Slot Number
- IPMC knows this info:
  - gets it from the Shelf Manager via I2C connection
- Need link between Zynq/ComE and IPMC
- Suggest (HW developers please comment):
  - a Serial line between the two
  - Implement a function (available in the pingeon point framework) to retrieve the data over the link
  - Suggest all boards implement same serial line on Zynq side and IPMC side (same configuration to retrieve the information from U-boot or Linux on all boards)

Serial Console from Zynq/ComE

- The typical debug option on Zynq is a serial line (e.g. VT100 like console), for ComE probably screen, but console is probably sufficient
- Board developers sometimes thinking of bringing line to front panel for debugging
  - Good for Lab
  - Difficult in production
- Servers use Serial over Lan, using the IPMI interface
- Could this be transposed to Zynq/ComE with IPMC ?
  - Serial line from Zynq/ComE to IPMC
  - Remote access via IPMC to the serial line
- Questions (to HW & IPMC developers):
  - How ?
  - Same serial line as the one used for ClientID or second one ?
  - Switch to use to from front panel or IPMC ?

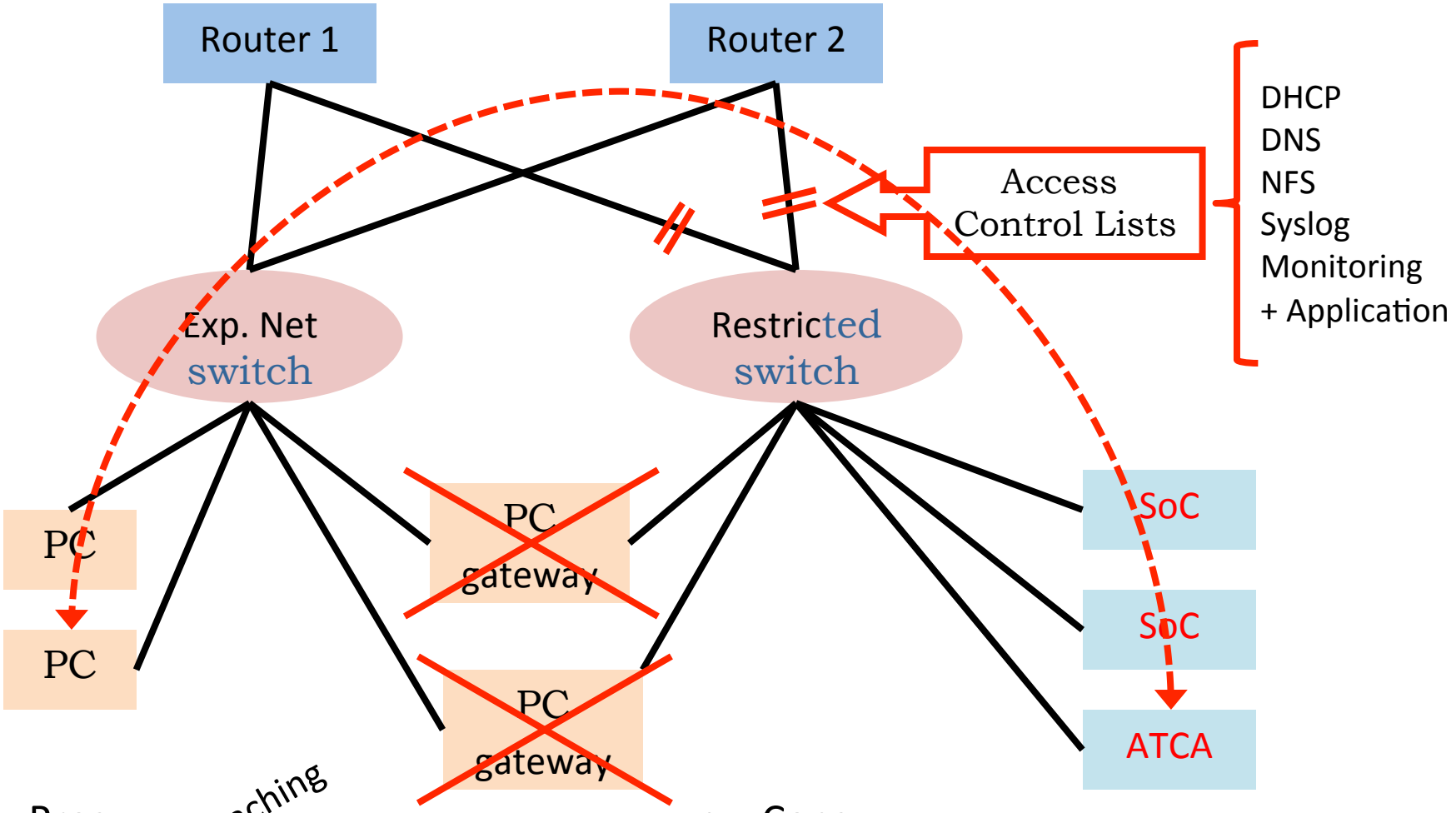
Future Steps

As soon as one can get back into the labs…

- Produce guideline on how to configure Shelf Manager: ClientID and others
- Produce a DHCP server configuration for Client ID
- Package in a VM/Docker image or install script:
  - DHCP server, DNS caching/forwarding server, remote log server
  - TFTP and NFS servers (network boot, see talk by Petr)
  - HTTP proxy server for access to YUM, Git, etc…
- Can be used on a local server to act like a experiment isolated network control host
- Will be extensively tested in the labs at CERN

BACKUP

Restricted switch: limited comm.



- Pros
  - ~~Gateway~~ PC not compulsory
  - Selective connectivity
  - Can use 10Gb or more
- Cons
  - Potential traffic bottleneck
  - Potential traffic DoS

U-Boot Configuration

U-Boot can be configured using the U-Boot menuconfig or meta-user layer in PetaLinux. (meta-user layer allows for manual configuration and does not get overwritten by PetaLinux)

Configuration options for enabling I2C EEPROM with MAC-address:

```
CONFIG_I2C_EEPROM                       = y
CONFIG_SYS_I2C_EEPROM_ADDR              = 0x54
CONFIG_SYS_I2C_EEPROM_ADDR_OVERFLOW     = 0x0
CONFIG_ZYNQ_GEM_I2C_MAC_OFFSET          = 0x20
```

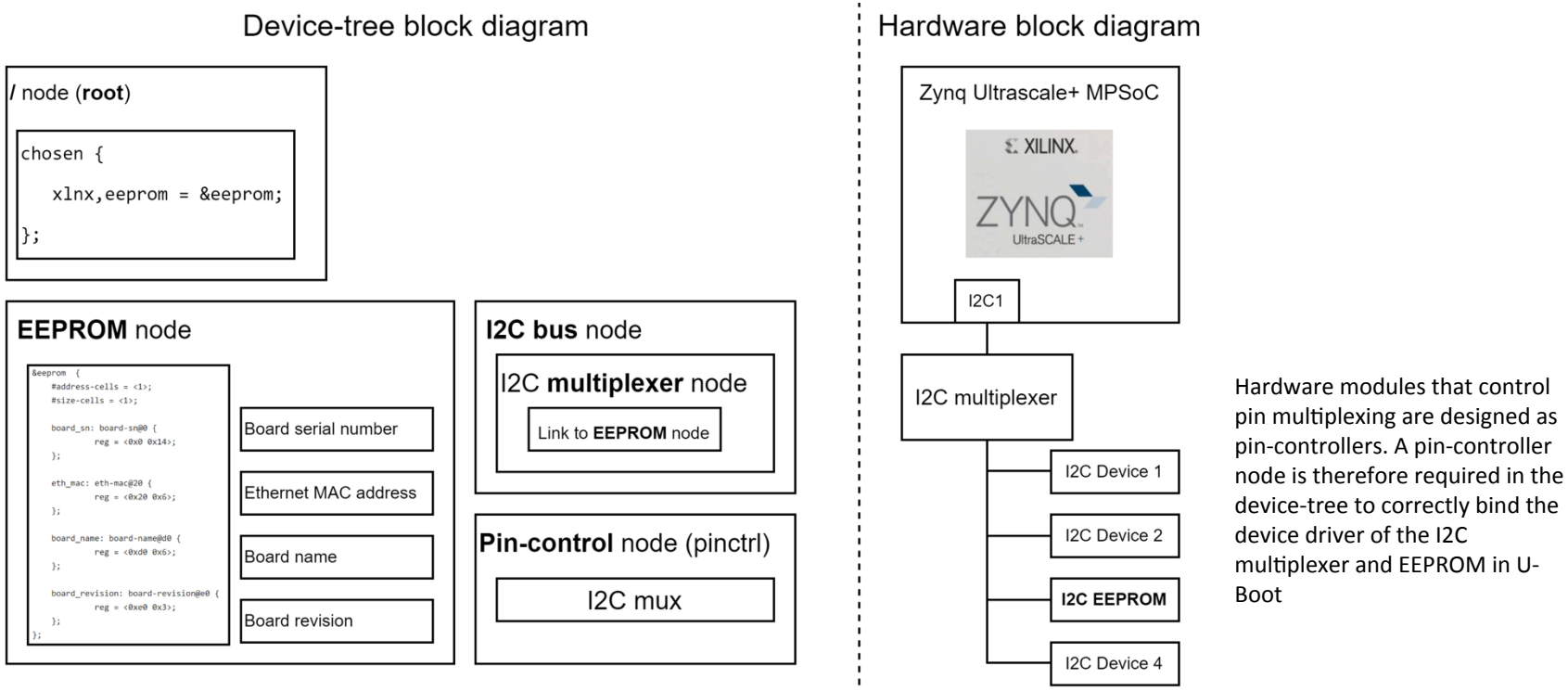
When using the meta-user layer in PetaLinux, your configuration file (**.cfg**) must be appended to BitBake to be built:

```
SRC_URI_append = " file://platform-top.h"
SRC_URI += "file://eeprom.cfg"

FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
```

EEPROM in device-tree

U-Boot and Linux can have access to the I2C EEPROM if it is properly defined in the device-trees. The I2C bus, I2C multiplexer and EEPROM need to be defined



Hardware modules that control pin multiplexing are designed as pin-controllers. A pin-controller node is therefore required in the device-tree to correctly bind the device driver of the I2C multiplexer and EEPROM in U-Boot