System Device Tree and Xen Dom0less

- Hypervisor Domains have been discussed (some details are still missing)
  - *they are a target for the next few months*
- System Device Tree Domains were inspired by Xen Dom0less
- Dom0less was recently extended and it is coming to x86 with Hyperlaunch
- System Device Tree and Hyperlaunch today are not identical but they are aligned
  - *in the short term lopper can be used to generated a Hyperlaunch configuration from System Device Tree*
  - *in the long term aim at unifying the two specs*

Domains Hierarchy: the example of Xen

- Domains are naturally hierarchical
- The example of Xen: Xen is running domains (VMs) and it is also a domain in itself
- It makes sense to make Domains hierarchical in System Device Tree
  - *lower privileged Domains are children of higher privileged Domains*
  - *the description of complex systems becomes more readable and understandable*

```
domains {
    xen: domain@0 {
        compatible = "openamp,domain-v1";

        cpus = <&cpus_a72 0x3 0x00000002>;
        memory = <0x0 0x500000 0x0 0x7fb00000>;
        id = <0xffff>;

        linux1: domain@1 {
            compatible = "openamp,domain-v1";

            cpus = <&cpus_a72 0x3 0x00000001>;
            memory = <0x0 0x501000 0x0 0x3faff000>;
            id = <0x0>;

            access = <&mmc0>;
        };

        linux2: domain@2 {
            compatible = "openamp,domain-v1";

            cpus = <&cpus_a72 0x3 0x00000001>;
            memory = <0x0 0x40000000 0x0 0x40000000>;
            id = <0x1>;
        };
    };
};
```

Xilinx Subsystems

- A Subsystem is a Xilinx firmware concept related to power and life-cycle management of hardware resources
  - *Hardware resources that are powered up and reset together belong to the same subsystem*
  - *Xilinx firmware can configure bus-level protection for Subsystems*
  - *a Xilinx subsystem can span multiple CPUs clusters*
    - e.g. Cortex-As and Cortex-Rs can belong to the same Subsystem
- Xilinx Subsystems are similar to System Device Tree Domains
- Describe Xilinx Subsystems in S-DT as the top level Domains of the Hierarchy

Xilinx Subsystems: example

- Things to notice:
  - *Subsystems as top-level Domains with a new compatible string*
  - *new **id** property*
  - *new **sram** property to describe mmio-sram regions*
  - *the **ATF** domain running at EL3*
    - How should ATF be described?

```
domains {
    resource_group0: resource_group@0 {
        compatible = "openamp,group-v1";
        sram = <0x0 0xffff0000 0x0 0x10000>;
    };

    subsystem0: domain@0 {
        compatible = "xlnx,subsystem-v1", "openamp,domain-v1";

        cpus = <&cpus_a72 0x3 0x80000003>;
        memory = <0x0 0x0 0x0 0x80000000>;
        id = <0x4>;
        firewallconf-default = <block 0>;

        atf: domain@1 {
            compatible = "openamp,domain-v1";
            id = <0xffff>;
            cpus = <&cpus_a72 0x3 0x80000003>;
            memory = <0x0 0x0 0x0 0x80000000>;
            access = <&ipi0>;

            linux: domain@2 {
                compatible = "openamp,domain-v1";
                id = <0x0>;
                cpus = <&cpus_a72 0x3 0x00000001>;
                memory = <0x0 0x0 0x0 0x7FF00000>;
                access = <&can1 &i2c0 &ttc0 &ttc1 &watchdog &usb0 &gem0>;
                include = <&resource_group0>;
            };
        };
    };

    subsystem1: domain@3 {
        compatible = "xlnx,subsystem-v1", "openamp,domain-v1";

        cpus = <&cpus_r5 0x3 0x80000001>;
        memory = <0x0 0xffe00000 0x0 0x100000>;
        id = <0x5>;
        firewallconf-default = <block 0>;
```

```
            freertos: domain@4 {
                id = <0x6>;
                cpus = <&cpus_r5 0x3 0x80000001>;
                memory = <0x0 0xffe00000 0x0 0x100000>;
                access = <&ipi1 &watchdog1 &spi1 &ttc2 &usb1>;
                include = <&resource_group0>;
            };
        };
    };
```

Access Flags are Domain Specific

- how do access flags work with hierarchical domains?
- we used to specify access flags as follows:

```
access = <&eth0 0x000f0f0f>
```

- cumbersome: some domains can have very many devices in the access list
- Xilinx Subsystems configure access differently from Xen domains
  - *Xen uses the IOMMU and Xilinx firmware (PLM) uses bus firewalls*
- Xilinx Subsystems and Xen domains need different access flags
- **access flags are domains specific**

Access Flags and Device Sharing

- when a device is shared across multiple domains is specified in a resource group
- a device can be shared across domains of different kinds
  - *e.g. a Xen domain and a Xilinx subsystem sharing a device*
- **we need to be able to specify different access flags for each domain that is sharing the device**
- temporary proposal (discarded):

```
// device access-flags-domain0 access-flags-domain1
access = <&eth0 0x000f0f0f 0xff000000>
```

- not easy to read
- requires specifying which is the first domain and which is the second domain

```
resource_group_1: resource_group_1 {
        compatible = "openamp,resource-group-v1";
        access = <&ethernet 0x0 0x1 0x2>, <&serial0 0x0 0x1 0x2>;
        access-flags-index = <0 1>;
};
domain@0 {
        #access-flags-cells = <0x1>;
        compatible = "openamp,domain-v1";
        access = <&mmc0 0x0>;
        include = <&resource_group_1 0x0>;
};

domain@1 {
        #access-flags-cells = <0x2>;
        compatible = "openamp,domain-v1";
        access = <&can0 0x1 0x2>;
        include = <&resource_group_1 0x1>;
};
```

Access Flags: a better way

- access flags are domain specific --> define them at the domain level
- give a name to each access flags group
- use the group name to select which set of flags to use for each assigned devices
- the same flags name can be defined differently by different domains
  - *solve the problem of devices in resource groups*

```
resource_group_1: resource_group_1 {
    compatible = "openamp,group-v1";

    access = <&ethernet0>;
    access-flags-names = "shared";
};

domain@0 {
        #flags-cells = <1>;
        flags = <0x1 0x0>;
        flags-names = "dev", "shared";

        access = <&mmc0>;
        access-flags-names = "dev";

        include = <&resource_group_1>;
};

domain@1 {
        #flags-cells = <1>;
        flags = <0xf>;
        flags-names = "shared";

        include = <&resource_group_1>;
};
```

Full Example

```
    domains {
        #address-cells = <0x2>;
        #size-cells = <0x2>;

        resource_group_1: resource_group_1 {
            compatible = "openamp,group-v1";

            access = <&ethernet0, &serial0>;
        };

        subsystem1: domain@1 {
            compatible = "xilinx,subsystem-v1", "openamp,domain-v1";

            cpus = <&cpus_a72 0x3 0x80000003>;
            memory = <0x0 0x500000 0x0 0x7fb00000>;

            #flags-cells = <1>;
            flags = <0x1>;
            flags-names = "dev";
            access = <&mmc0>;
            access-flags-names = "dev";
            id = <0x3>;

            xen: domain@2 {
                compatible = "openamp,domain-v1";

                cpus = <&cpus_a72 0x3 0x00000002>;
                memory = <0x0 0x500000 0x0 0x7fb00000>;
                id = <0xffff>;

                linux1: domain@3 {
                    compatible = "openamp,domain-v1";

                    cpus = <&cpus_a72 0x3 0x00000001>;
                    memory = <0x0 0x501000 0x0 0x3faff000>;
                    id = <0x0>;

                    access = <&mmc0>;
                };

                linux2: domain@4 {
                    compatible = "openamp,domain-v1";

                    cpus = <&cpus_a72 0x3 0x00000001>;
                    memory = <0x0 0x40000000 0x0 0x40000000>;
                    id = <0x1>;

                    include = <&resource_group_1>;
                };
            };
        };

        subsystem2: domain@5 {
            compatible = "xilinx,subsystem-v1", "openamp,domain-v1";
```

```
            cpus = <&cpus_r5 0x3 0x80000001 &microblaze0 0x1 0x00000000>;
            memory = <0x0 0x100000 0x0 0x400000>;
            id = <0x4>;

            flags = <0x0>;
            flags-names = "dev";
            access = <&can0>;
            access-flags-names = "dev";
            include = <&resource_group_1>;
        };
    };
```

Lopper Status Update

- Current defined stages in the Lopper Framework / Pipeline
  - *Frontend: yaml, dts*
  - *Translation: yaml (**new**)*
  - *Tree Processing: lops/assists (on LopperTree)*
  - *Backend (yaml, dts, dtb, custom)*
  - *Server / Daemon (ReST)*
- In progress activities:
- Addition of the Zephyr dtlib/edtlib processing as a new Frontend and processing resource
- pypi installation
- Extended YAML processing / expansion
- ReST API
- Updated reference pipeline for SDT specification

Example: Xilinx CDO Generation with Lopper

- Inputs:
  - *System Device Tree (dts, dtsi, overlays)*
  - *Lopper Operations (lops)*
  - *yaml domain/subsystem specification*
  - *python assists*
- Outputs:
  - *Modified system device tree*
  - *Xilinx CDO file for firmware consumption*
- Flow:
  - *Setup / Load:*
    - lops are loaded and assists registered
  - *Parse:*
    - YAML and Device Tree components are parsed
    - Translation on registered inputs
    - A combined LopperTree registration is instantiated
  - *Exec:*
    - lops are applied to the tree
      - Rename/modify/delete/select/etc
    - Registered assists are triggered (CDO)
    - Assists execute, and examine LopperTree
      - Xilinx CDO operations are generated
  - *Output:*
    - modified (combined) device tree
    - CDO output file