Around revision [900], MutekH got support for Flattened device trees.

Rationale

Flattened device trees (FDT) are useful to get a list of all available hardware where no hardware-based enumeration exists (aka PnP, like PCI, USB, ... provides).

FDT provides normalized representation of the hardware platform without adding specific initialization code.

The normalization comes from IEEE1275 (aka Open Firmware); while Open Firmware also defined heavy things (like a Forth interpreter), we only use the FDT information.

Using FDT is mainstream, is also used by Linux and BSD, supported by U-Boot and other bootloaders.

References

- You may learn many things ?googling for "flattened device tree".
- The `dtc` utility used to compile `.dts` files in MutekH is mainteained at ?http://git.jdl.com/. You may:
  - Browse the git there ?http://git.jdl.com/gitweb/
  - Download a snapshot ?http://git.jdl.com/software/ This repository also includes some documentation and a reference library for handling FDTs.
- The Linux kernel documentation tree contains ?an useful document about device trees (`Documentation/powerpc/booting-without-of.txt`)

Implementation

In MutekH, FDT is handled through an hardware enumerator device driver, it behaves like the other enumerators (PCI, ISAPnP).

MutekH-Specific Node syntax quick reference

Processors

Processor nodes look like:

```
Mips,32@0 {
        name = "Mips,32";
        device_type = "cpu";
        reg = <0>;
};
```

`reg`
: This is the CPU identification number

`device_type`
: must be `"cpu"`

There is a couple of optional attributes:

`ipi`
: This is a reference to the ICU device handling IPIs for this CPU, and the ipi number, like

```
        ipi = <&{/xicu@0/out@2} 2>;
```

Memories

Memory nodes look like:

```
memory@0 {
        device_type = "memory";
        cached;
        reg = <0x61100000 0x00100000>;
};
```

`device_type`
: must be "memory"

`reg`
: must be a couple of <address size> with both the values respecting `#address-cells` and `#size-cells`.

There are two optional attributes:

`cached`
: The memory is cacheable

`coherent`
: The memory is cached and coherent (`cached` is implied, setting it is optional)

References to interrupt controllers

Interrupts controller are referenced from one node to another in order to describe the interrupt tree. References are handled through the following properties:

`irq`
: Must be a path to an existing ICU device, and the icu irq number, enclosed in `<&{path} no>`. This syntax is mandatory. Example:

```
        irq = <&{/cpus/Mips,32@0} 0>;
```

Example:

```
icu@0 {
        device_type = "soclib:icu";
        input_count = <2>;
        reg = <0x20600000 0x20>;
        irq = <&{/cpus/Mips,32@0} 0>;
};

tty@0 {
        device_type = "soclib:tty";
        tty_count = <1>;
        reg = <0x90600000 0x10>;
        irq = <&{/icu@0} 1>;
};
```

Here the ICU device for `/tty@0` is `/icu@0` (device at address 0x20600000), which in turn references `/cpus/Mips,32@0` as its ICU device.

The /chosen node

The `/chosen` node contains informations about the global system peripherals and configuration. In MutekH, we use the `/chosen` node to reference preipherals:

`timer`
: Select the global timer device

`root`
: Select the root file system device

`console`
: Select the console tty

Referencing the correct devices does not implicitly make their drivers available. You still have to select the driver in your configuration file.

Example:

```
chosen {
        console = &{/tty@0};
        root = &{/ramdisk@0};
};
```

Parameter structure construction for calling `_init` functions

Some devices require a structure containing parameters in order to correctly initialize them. This case is handled in the FDT description. Let's see the example of the `soclib:xicu` component. It needs a structure containing:

```
struct xicu_root_param_s
{
        size_t input_lines;
        size_t ipis;
        size_t timers;
};
```

In the driver, the id definition is:

```
static const struct driver_param_binder_s xicu_root_binder[] =
{
        PARAM_BIND(struct xicu_root_param_s, input_lines, PARAM_DATATYPE_INT),
        PARAM_BIND(struct xicu_root_param_s, ipis, PARAM_DATATYPE_INT),
        PARAM_BIND(struct xicu_root_param_s, timers, PARAM_DATATYPE_INT),
        { 0 }
};

static const struct devenum_ident_s     xicu_root_ids[] =
{
        DEVENUM_FDTNAME_ENTRY("soclib:xicu:root", sizeof(struct xicu_root_param_s), xicu_root_bi
        { 0 }
};
```

This informs the FDT parser this device will need a parameter structure, with the parameters described in the `xicu_param_binder` correctly filled-in.

In this table, there is one entry telling the `output_line_no` parameter is an integer.

Available data types are:

`PARAM_DATATYPE_INT`
: a simple integer

`PARAM_DATATYPE_DEVICE_PTR`
: a device reference (`&{/node/path}` in the device tree source), which will be transparently translated to a `struct device_s *` before filling the structure. Device must exist in the tree.

`PARAM_DATATYPE_ADDR`
: an address, `#address-cells` will be honored

`PARAM_DATATYPE_BOOL`
: a simple boolean, i.e. a property with no value, if present it is true, if absent it is false (like the `cached` attribute in memory nodes)

Example

Drivers may export themselves as FDT-aware, and define which device name string to match. For instance, the following subtree defines a tty device:

```
tty@0 {
        device_type = "soclib:tty";
        tty_count = <1>;
        reg = <0x90600000 0x10>;
        irq = <&{/icu@0} 1>;
};
```

In turn, the SoCLib tty driver declares itself (in source:trunk/mutekh/drivers/device/char/tty-soclib/tty-soclib.c#L146) as:

Note there is no parameter structure definition, so the two last arguments of `DEVENUM_FDTNAME_ENTRY` are 0.

```
static const struct devenum_ident_s     tty_soclib_ids[] =
{
        DEVENUM_FDTNAME_ENTRY("soclib:tty", 0, 0),
        { 0 }
};

const struct driver_s   tty_soclib_drv =
{
  .class                = device_class_char,
  .id_table             = tty_soclib_ids,
  .f_init               = tty_soclib_init,
  .f_cleanup            = tty_soclib_cleanup,
  .f_irq                = tty_soclib_irq,
  .f.chr = {
    .f_request          = tty_soclib_request,
  }
};
```

This will make the FDT enumerator use the correct driver, matching `"soclib:tty"`