Zephyr® Project
Developer Summit 2022

Deep Dive into Pin Control in Zephyr

Gerard Marull-Paretas
gerard.marull@nordicsemi.no

Nordic Semiconductor ASA

9th June 2022

Outline

Introduction

What is pin control?

- Pin control refers to the hardware blocks that control **pin multiplexing** and **pin configuration parameters**, e.g. pull-up/down, drive mode, slew-rate, etc.
- **Essential** for **SoC peripherals** that have a physical interface, e.g. I2C, SPI, UART, etc.
- The way pin control is implemented in hardware is **vendor/SoC specific**, i.e. non-portable

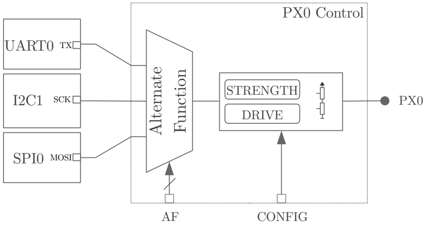


Figure: Example of pin control centralized into a single per-pin block

Pin control vs. GPIO

- Some functionality covered by a pin controller driver **overlaps** with GPIO drivers, e.g. setting pull-up resistor
- The main purpose of **pin control** is to perform both **signal multiplexing** and **pin configuration** (e.g. slew-rate) required for the correct operation of a **peripheral**
- **GPIO** drivers are for **general purpose control** of a pin, that is, when its logic level is read or **controlled manually**
- **GPIO** can be usually seen as a **subset of pin control**. In the future GPIO drivers could **re-use** pin control infrastructure

Pin Control in Zephyr < 3.1

- Each **vendor** implemented its **own** solution
- `pinmux` API existed, but had **design limitations** and was **not always used**
- **Custom** Devicetree representations, some platforms did not even use Devicetree but harcoded **configuration in board C code**

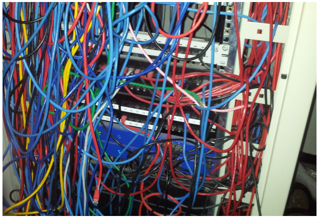


Figure: Pin control in Zephyr < 3.1

Pin Control in Zephyr < 3.1: Example (I)

```
/* boards/arm/nrf52840dk_nrf52840.dts */
&uart0 {
    ...
    tx-pin = <6>;
    rx-pin = <8>;
    rx-pull-up;
    ...
};
```

```
/* drivers/serial/uart_nrfx_uarte.c */
static int uarte_instance_init(const struct device *dev, ...)
{
    ...
    nrf_gpio_pin_write(cfg->pseltxd, 1);
    nrf_gpio_cfg_output(cfg->pseltxd);

    if (cfg->pselrxd != NRF_UARTE_PSEL_DISCONNECTED) {
        nrf_gpio_cfg_input(cfg->pselrxd, cfg->rxd_pull);
    }

    nrf_uarte_txrx_pins_set(uarte, cfg->pseltxd, cfg->pselrxd);
    ...
}
```

Listing: *Pin control* in nRF, Zephyr 2.7

Pin Control in Zephyr < 3.1: Example (II)

```
/* boards/arm/nucleo_h743zi.dts */
&usart3 {
    pinctrl-0 = <&usart3_tx_pd8 &usart3_rx_pd9>;
    ...
};
```

```
/* drivers/serial/uart_stm32.c */
static int uart_stm32_init(const struct device *dev)
{
    ...
    /* Configure dt provided device signals when available */
    err = stm32_dt_pinctrl_configure(config->pinctrl_list, ...);
    ...
}

#define STM32_UART_INIT(index)                                      \
    static const struct soc_gpio_pinctrl uart_pins_##index[] =      \
        ST_STM32_DT_INST_PINCTRL(index, 0);                         \
                                                                    \
    static const struct uart_stm32_config uart_stm32_cfg_##index = { \
        ...                                                         \
        .pinctrl_list = uart_pins_##index,                          \
        .pinctrl_list_size = ARRAY_SIZE(uart_pins_##index),         \
    };                                                              \
    ...
```

Listing: *Pin control* in STM32, Zephyr 2.7

Pin Control in Zephyr < 3.1: Example (III)

```
/* boards/arm/mimxrt1010_evk/pinmux.c */
static int mimxrt1010_evk_init(const struct device *dev)
{
    ...
    CLOCK_EnableClock(kCLOCK_Iomuxc);
    CLOCK_EnableClock(kCLOCK_IomuxcSnvs);
    ...
    #if DT_NODE_HAS_STATUS(DT_NODELABEL(lpuart1), okay) && CONFIG_SERIAL
      /* LPUART1 TX/RX */
      IOMUXC_SetPinMux(IOMUXC_GPIO_09_LPUART1_RXD, 0);
      IOMUXC_SetPinMux(IOMUXC_GPIO_10_LPUART1_TXD, 0);

      IOMUXC_SetPinConfig(IOMUXC_GPIO_09_LPUART1_RXD,
                          IOMUXC_SW_PAD_CTL_PAD_PKE_MASK |
                          IOMUXC_SW_PAD_CTL_PAD_SPEED(2) |
                          IOMUXC_SW_PAD_CTL_PAD_DSE(6));

      IOMUXC_SetPinConfig(IOMUXC_GPIO_10_LPUART1_TXD,
                          IOMUXC_SW_PAD_CTL_PAD_PKE_MASK |
                          IOMUXC_SW_PAD_CTL_PAD_SPEED(2) |
                          IOMUXC_SW_PAD_CTL_PAD_DSE(6));
    #endif
    ...
}

SYS_INIT(mimxrt1010_evk_init, PRE_KERNEL_1, 0);
```

Listing: *Pin control* in i.MX RT, Zephyr 2.7

Pin control in Zephyr >= 3.1

- Introduced a **new** `pinctrl` API in Zephyr 3.0
- **Standardizes** a few aspects of pin control:
  - **State model**, inspired by Linux Kernel approach
  - Standardizes **common properties**, e.g. `bias-pull-up`
  - All **vendor-specific** bits are contained in **Devicetree** and a **SoC specific header**
  - All **drivers** use the **same mechanism** to configure pins
- `pinctrl` will be **mandatory** for any new platform starting from Zephyr 3.1 onwards

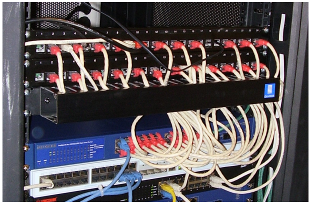


Figure: Pin control in Zephyr >= 3.1 (CC-BY-SA-4.0, Wikipedia)

State Model

State Model: Background

- Some device drivers need a certain **pin configuration** to be applied to **work correctly**. This includes **signal multiplexing** and **other configurations** such as pull-up resistors
- Pin configuration **requirements** may **change at runtime**, e.g. when suspending the device
- Each **required** pin configuration is **modeled** as a state, following Linux Kernel approach
- States **encode** all necessary pin configurations and are **independent** of each other (they can be applied in any order)
- States **isolate** driver code from pin configuration: *a driver just applies a state*

State Model: Example

I2C0 peripheral (master)			
	`default` state		`sleep` state
SDA	▪ Pin: PA0 ▪ Drive: Open-Drain ▪ Low-Power: No	SDA	▪ Pin: PA0 ▪ Drive: Default ▪ Low-Power: Yes
SCL	▪ Pin: PA1 ▪ Drive: Open-Drain ▪ Low-Power: No	SCL	▪ Pin: PA1 ▪ Drive: Default ▪ Low-Power: Yes

Standard States

- In general, pin control states can have **arbitrary names**
- A **naming convention** has been established for the most **common** use cases
- Standardization brings **consistency** and paths for **optimization**. Example: `sleep` state is automatically discarded if `CONFIG_PM_DEVICE=n`

State	Identifier	Purpose
`default`	`PINCTRL_STATE_DEFAULT`	State of the pins when the device is in **operational** state
`sleep`	`PINCTRL_STATE_SLEEP`	State of the pins when the device is in **low power or sleep modes**

Custom States

- Some device drivers may require **custom states** beyond `default` or `sleep`
- **Solution**: define custom state identifiers named as `PINCTRL_STATE_{NAME}`, where `{NAME}` is the capitalized state name
- It is **important** that custom state identifiers both are in **driver's scope** and **start from** `PINCTRL_STATE_PRIV_START` to avoid clashes with standard states

Custom States: Example

- A device driver needs different states depending on the runtime bus operating speed: **`slow`** or **`fast`**
- **Solution**: define `PINCTRL_STATE_SLOW` and `PINCTRL_STATE_FAST` custom state identifiers

```
/* identifier for 'slow' state */
#define PINCTRL_STATE_SLOW PINCTRL_STATE_PRIV_START
/* identifier for 'fast' state */
#define PINCTRL_STATE_FAST PINCTRL_STATE_PRIV_START + 1
```

Listing: Identifiers for custom states `slow` and `fast`

Skipping states

- In some software configurations, certain pin control **states** may be left **unused**, thus wasting ROM space
- States can be skipped (not stored) if a definition named `PINCTRL_SKIP_{STATE_NAME}` **expanding to 1** is in driver's scope when pin control configuration is defined

```
#ifndef CONFIG_PM_DEVICE
/* If device power management is not enabled, "sleep"
 * state will not be stored, even if defined in Devicetree.
 */
#define PINCTRL_SKIP_SLEEP 1
#endif
```

Listing: `sleep` state is skipped if `CONFIG_PM_DEVICE=n`

Devicetree Representation

States

- Each device's pin control state is represented in Devicetree by **`pinctrl-N`** properties, where `N` is the state index starting from zero. Property content is vendor specific.
- The **`pinctrl-names`** property is then used to assign the state to each state property by index

```
periph0: periph@0 {
    ...
    /* state 0 ("default") */
    pinctrl-0 = <...>;
    ...
    /* state N ("mystate") */
    pinctrl-N = <...>;
    /* names for state 0 up to state N */
    pinctrl-names = "default", ..., "mystate";
    ...
};
```

Listing: Pin control states for `periph0`

Pin configuration

- There are **multiple** ways to represent the pin configurations in Devicetree
- All representations encode the **same information**: the pin multiplexing and the pin configuration parameters
- Representation choice depends largely on **vendor preferences**
- A couple of popular choices: **grouping** and **node-based**
- **Standardized properties are mandatory**. For example, `bias-pull-up` has to be used to activate a pin pull-up resistor. They are **pre-defined** in `pincfg-node-group.yaml` or `pincfg-node.yaml`

Pin configuration: grouping approach (I)

- Multiple signals can be **grouped** if they share the same pin configuration
- Pin configuration parameters for a particular state are enclosed in a **single** Devicetree node
- Platforms using this approach: Atmel, Gigadevice, Nordic, NXP*, Raspberry Pi

Pin configuration: grouping approach (II)

```
/* board-pinctrl.dtsi */
#include <vnd-soc-pkgxx.h>

&pinctrl {
    /* Node with pin configuration for default state */
    periph0_default: periph0_default {
        group1 {
            /* Mappings: PERIPH0_SIGA->PX0, PERIPH0_SIGC->PZ1 */
            pinmux = <PERIPH0_SIGA_PX0>, <PERIPH0_SIGC_PZ1>;
            /* Pins PX0 and PZ1 have pull-up enabled */
            bias-pull-up;
        };
        ...
        groupN {
            /* Mappings: PERIPH0_SIGB->PY7 */
            pinmux = <PERIPH0_SIGB_PY7>;
        };
    };
};
```

Listing: Pin configuration for `periph0`, `default` state

Pin configuration: grouping approach (III)

```
/* vnd-soc-pkgxx.h */
...
/* encode pinmux entry in a 32-bit field */
#define VNDSOC_PIN(...)
...
/* valid mappings for SoC package 'xx' (may be autogenerated) */
#define PERIPH0_SIGA_PX0 VNDSOC_PIN(X, 0, MUX0)
#define PERIPH0_SIGB_PY7 VNDSOC_PIN(Y, 7, MUX4)
#define PERIPH0_SIGC_PZ1 VNDSOC_PIN(Z, 1, MUX2)
...
```

Listing: Pre-defined valid mappings for `periph0` (optional)

Pin configuration: grouping approach (IV)

```
/* board.dts */
#include "board-pinctrl.dtsi"

&periph0 {
    pinctrl-0 = <&periph0_default>;
    pinctrl-names = "default";
};
```

Listing: States definition for `periph0`

Pin configuration: node approach (I)

- A **state** is assigned with a **set of nodes**, each one containing the configuration for a single pin
- It requires a **node per pin and per state** (nodes can not be re-used for multiple states)
- **Recommended** only if nodes can be **pre-defined**, it can become too verbose otherwise
- Platforms using this approach: ITE, Microchip, Renesas R-Car, SiFive, STM32, TI, ITE

Pin configuration: node approach (II)

```
  /* vnd-soc-pkgxx.dtsi
   * File with valid nodes for a specific package (may be autogenerated).
   * This file is optional, but recommended. Note the /omit-if-no-ref/
   * keyword is added to discard node if not used.
   */

&pinctrl {
    /* Mapping for PERIPH0_SIGA -> PX0, to be used for default state */
    /omit-if-no-ref/ periph0_siga_px0_default: periph0_siga_px0_default {
        pinmux = <VNDSOC_PIN(X, 0, MUX0)>;
    };

    /* Mapping for PERIPH0_SIGB -> PY7, to be used for default state */
    /omit-if-no-ref/ periph0_sigb_py7_default: periph0_sigb_py7_default {
        pinmux = <VNDSOC_PIN(Y, 7, MUX4)>;
    };

    /* Mapping for PERIPH0_SIGC -> PZ1, to be used for default state */
    /omit-if-no-ref/ periph0_sigc_pz1_default: periph0_sigc_pz1_default {
        pinmux = <VNDSOC_PIN(Z, 1, MUX2)>;
    };
};
```

Listing: Pre-defined nodes with valid mappings for `periph0` (optional)

Pin configuration: node approach (III)

```
/* board-pinctrl.dts */
#include <vnd-soc-pkgxx.dtsi>

/* Enable pull-up for PX0 (default state) */
&periph0_siga_px0_default {
    bias-pull-up;
};

/* Enable pull-up for PZ1 (default state) */
&periph0_sigc_pz1_default {
    bias-pull-up;
};
```

Listing: Properties are assigned by board as needed

Pin configuration: node approach (IV)

```
/* board.dts */
#include "board-pinctrl.dtsi"

&periph0 {
    pinctrl-0 = <&periph0_siga_px0_default
                 &periph0_sigb_py7_default
                 &periph0_sigc_pz1_default>;
    pinctrl-names = "default";
};
```

Listing: States definition for `periph0`

The Pin Control API

Pin Control Drivers

- There is a single pin control driver per SoC, so it is effectively a **singleton**.
- Pin control drivers only need to implement a **single** function call: `pinctrl_configure_pins`
- **Vendor specific** bits are defined in `pinctrl_soc.h`. This header that must be in the include path and must define:
  - `pinctrl_soc_pin_t`, the pin configuration opaque type
  - `Z_PINCTRL_STATE_PINS_INIT`, the macro responsible for Devicetree *parsing*
- Reference implementation and test (uses grouping approach): `tests/drivers/pinctrl/api/src`

Pin Control Drivers: Example (I)

```
...
compatible: "vnd,pinctrl"
...
include:
    - name: base.yaml
    - name: pincfg-node-group.yaml
      child-binding:
        child-binding:
          property-allowlist:
            - bias-pull-down
            - bias-pull-up

child-binding:
  child-binding:
    properties:
      pinmux:
        required: true
        type: array
```

Listing: Example Devicetree binding for a pin control driver (grouping approach)

Pin Control Drivers: Example (II)

```
/* board-pinctrl.dtsi */
&pinctrl {
    periph0_default: periph0_default {
        group1 {
            pinmux = <PERIPH0_SIGA_PA5>,
                     <PERIPH0_SIGB_PA6>;
            bias-pull-up;
        };
    };
};

/* board.dts */
&periph0 {
    pinctrl-0 = <&periph0_default>;
    pinctrl-names = "default";
};
```

Listing: Devicetree definition of `periph0` default state

Pin Control Drivers: Example (III)

```
/**
 * State initializer (invoked for every device state)
 *
 * It takes the node stored in prop first index, e.g. for
 * pinctrl-0 'periph0_default', and iterates over its children,
 * i.e. groups. In each group, it iterates over all 'pinmux'
 * elements, using the 'Z_PINCTRL_STATE_PIN_INIT' initializer.
 *
 * @param node_id Device's node (e.g. periph0)
 * @param prop Property holding n-th state, e.g. pinctrl-0
 */
#define Z_PINCTRL_STATE_PINS_INIT(node_id, prop) \
  {                                              \
      DT_FOREACH_CHILD_VARGS(                    \
        DT_PROP_BY_IDX(node_id, prop, 0),        \
        DT_FOREACH_PROP_ELEM,                    \
        pinmux, Z_PINCTRL_STATE_PIN_INIT         \
      )                                          \
  }
```

Listing: Example of `Z_PINCTRL_STATE_PINS_INIT` macro

Pin Control Drivers: Example (IV)

```c
/** Pin configuration type (here a 32-bit bit field) */
typedef uint32_t pinctrl_soc_pin_t;

/**
 * Pin configuration initializer (VND_XXX macros are used to
 * encode configuration in a 32-bit field)
 *
 * @param node_id Group node, e.g. group1
 * @param prop Property holding pinmux (always 'pinmux')
 * @param idx Index of the pinmux element, e.g. 0, 1...
 */
 #define Z_PINCTRL_STATE_PIN_INIT(node_id, prop, idx)    \
    (DT_PROP_BY_IDX(node_id, prop, idx) |                 \
     ((VND_PULL_UP * DT_PROP(node_id, bias_pull_up))      \
      << VND_PULL_POS) |                                  \
     ((VND_PULL_DOWN * DT_PROP(node_id, bias_pull_down)) \
      << VND_PULL_POS)                                    \
    ),
```

Listing: Pin configuration initializer used by `Z_PINCTRL_STATE_PINS_INIT`

Pin Control Drivers: Example (V)

```
{
    (PERIPH0_SIGA_PA5 | VND_PULL_UP),
    (PERIPH0_SIGB_PA6 | VND_PULL_UP),
}
```

Listing: Result of Z_PINCTRL_STATE_PINS_INIT when invoked for periph0, default state

Using pin control in drivers (I)

- Driver binding needs to **include** `pinctrl-device.yaml`, **pre-defines** `pinctrl-0..4` and `pinctrl-names` properties
- `pinctrl-0` and `pinctrl-names` properties may be flagged as required (improves error diagnostics)

```yaml
# vnd,periph.yaml
...
compatible: "vnd,periph"
...
include: [base.yaml, pinctrl-device.yaml]
...
```

Listing: `vnd,periph` binding includes `pinctrl-device.yaml`

Using pin control in drivers (II)

- Driver needs to **define** its pin control configuration by using `PINCTRL_DT_DEFINE` or `PINCTRL_DT_INST_DEFINE` macros
- Driver needs to **keep a reference** to the defined pin control configuration by using `PINCTRL_DT_DEV_CONFIG_GET` or `PINCTRL_DT_INST_CONFIG_GET` macros

Using pin control in drivers (III)

```c
/* A driver for the "vnd,periph" compatible device */
#define DT_DRV_COMPAT vnd_periph
...
#include <zephyr/drivers/pinctrl.h>
...
struct vnd_periph_config {
    ...
    /* Reference to instance's pinctrl configuration */
    const struct pinctrl_dev_config *pcfg;
};

#define VND_PERIPH_DEFINE(i)                                            \
    /* Define pinctrl configuration for instance "i" */                 \
    PINCTRL_DT_INST_DEFINE(i);                                          \
    ...                                                                 \
    static const struct vnd_periph_config vnd_periph_config_##i = {     \
        ...                                                             \
        /* Keep a ref. to the pinctrl configuration for instance "i" */ \
        .pcfg = PINCTRL_DT_INST_DEV_CONFIG_GET(i),                      \
    };                                                                  \
    ...                                                                 \
                                                                        \
    DEVICE_DT_INST_DEFINE(i, vnd_periph_init, NULL, &vnd_periph_data##i, \
                          &vnd_periph_config##i, ...);

DT_INST_FOREACH_STATUS_OKAY(VND_PERIPH_DEFINE)
```

Listing: Example of `vnd,periph` driver defining pin control

Using pin control in drivers (IV)

- Driver can apply any defined pin control state by using `pinctrl_apply_state`

```c
static int vnd_periph_init(const struct device *dev)
{
    const struct vnd_periph_config *config = dev->config;
    int ret;
    ...
    /* Select "default" state at initialization time */
    ret = pinctrl_apply_state(config->pcfg,
                              PINCTRL_STATE_DEFAULT);
    if (ret < 0) {
        return ret;
    }
    ...
}
```

Listing: `default` state applied at driver initialization stage

Dynamic Pin Control

Dymamic Pin Control

- Dynamic pin pontrol refers to the capability of **changing** pin configuration at **runtime**
- Useful in situations where the **same firmware** needs to run onto **slightly different** boards
- Can be enabled by setting `CONFIG_PINCTRL_DYNAMIC=y`
- Device pin control configuration is stored in **RAM instead of ROM**, thus allowing state swapping. States are still kept in ROM
- An alternative set of states can be set at runtime using `pinctrl_update_states`
- Limited to **uninitialized devices**

Dymamic Pin Control: Example (I)

- `uart0` is routed to one or another set of pins at boot time depending on the Button 1 state.
- `samples/boards/nrf/dynamic_pinctrl`

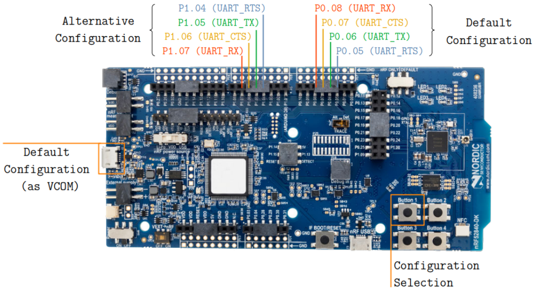


Figure: Configuration for nRF52840 DK

Dymamic Pin Control: Example (II)

```
/ {
    zephyr,user {
        uart0_alt_default = <&uart0_alt_default>;
        uart0_alt_sleep = <&uart0_alt_sleep>;
    };
};

&pinctrl {
    /* Alternative pin configuration for UART0 */
    uart0_alt_default: uart0_alt_default {
        group1 {
            psels = <NRF_PSEL(UART_TX, 1, 5)>,
                    <NRF_PSEL(UART_RTS, 1, 4)>;
        };
        ...
    };
    ...
};
```

Listing: Alternative `uart0` pin configuration defined in Devicetree

Dymamic Pin Control: Example (III)

```c
/* define uart0 alternative configurations (taken from DT) */
PINCTRL_DT_STATE_PINS_DEFINE(DT_PATH(zephyr_user),
                             uart0_alt_default);
#ifdef CONFIG_PM_DEVICE
PINCTRL_DT_STATE_PINS_DEFINE(DT_PATH(zephyr_user),
                             uart0_alt_sleep);
#endif

/* all uart0 alternative states (default and sleep) */
static const struct pinctrl_state uart0_alt[] = {
    PINCTRL_DT_STATE_INIT(uart0_alt_default,
                          PINCTRL_STATE_DEFAULT),
#ifdef CONFIG_PM_DEVICE
    PINCTRL_DT_STATE_INIT(uart0_alt_sleep,
                          PINCTRL_STATE_SLEEP),
#endif
};
```

Listing: Define `uart0` alternative configurations (taken from DT)

Dymamic Pin Control: Example (IV)

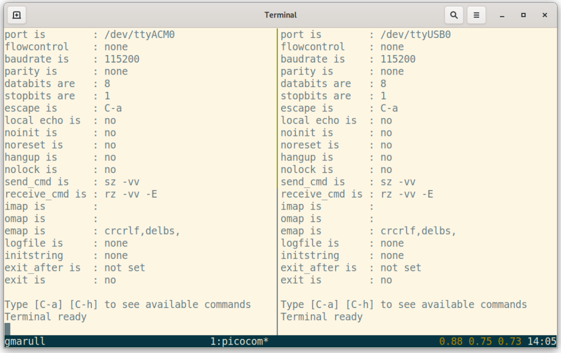
```
/* declare uart0 device pin control configuration
* (defined in the driver)
*/
PINCTRL_DT_DEV_CONFIG_DECLARE(DT_NODELABEL(uart0));

/* obtain a reference to the uart0 device pin control config */
struct pinctrl_dev_config *uart0_config =
    PINCTRL_DT_DEV_CONFIG_GET(DT_NODELABEL(uart0));

/* update uart0 configuration with alternative states */
pinctrl_update_states(uart0_config, uart0_alt,
                      ARRAY_SIZE(uart0_alt));
```

Listing: Update `uart0` states to the alternative set

Dymamic Pin Control: Example (V)



```
port is        : /dev/ttyACM0
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
stopbits are   : 1
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
hangup is      : no
nolock is      : no
send_cmd is    : sz -vv
receive_cmd is : rz -vv -E
imap is        :
omap is        :
emap is        : crcrlf,delbs,
logfile is     : none
initstring     : none
exit_after is  : not set
exit is        : no

Type [C-a] [C-h] to see available commands
Terminal ready
```

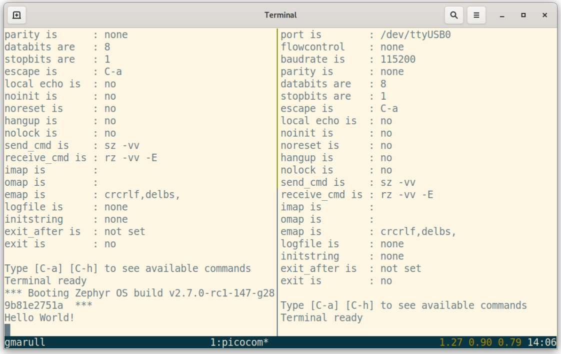
```
port is        : /dev/ttyUSB0
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
stopbits are   : 1
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
hangup is      : no
nolock is      : no
send_cmd is    : sz -vv
receive_cmd is : rz -vv -E
imap is        :
omap is        :
emap is        : crcrlf,delbs,
logfile is     : none
initstring     : none
exit_after is  : not set
exit is        : no

Type [C-a] [C-h] to see available commands
Terminal ready
```

```
gmarull                         1:picocom*                     0.88 0.75 0.73 14:05
```

Figure: Two serial terminals (left: default, right: alternative)

Dymamic Pin Control: Example (VI)



Terminal

```
parity is       : none
databits are    : 8
stopbits are    : 1
escape is       : C-a
local echo is   : no
noinit is       : no
noreset is      : no
hangup is       : no
nolock is       : no
send_cmd is     : sz -vv
receive_cmd is  : rz -vv -E
imap is         :
omap is         :
emap is         : crcrlf,delbs,
logfile is      : none
initstring      : none
exit_after is   : not set
exit is         : no

Type [C-a] [C-h] to see available commands
Terminal ready
*** Booting Zephyr OS build v2.7.0-rc1-147-g28
9b81e2751a  ***
Hello World!
```

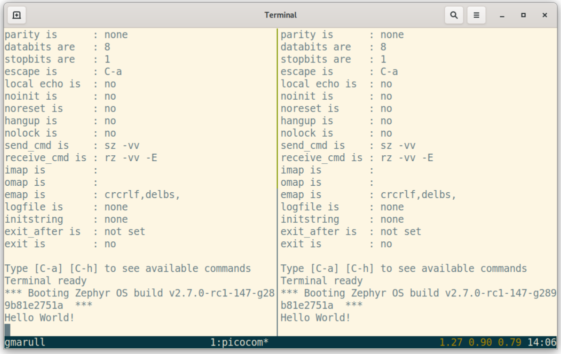
```
port is         : /dev/ttyUSB0
flowcontrol     : none
baudrate is     : 115200
parity is       : none
databits are    : 8
stopbits are    : 1
escape is       : C-a
local echo is   : no
noinit is       : no
noreset is      : no
hangup is       : no
nolock is       : no
send_cmd is     : sz -vv
receive_cmd is  : rz -vv -E
imap is         :
omap is         :
emap is         : crcrlf,delbs,
logfile is      : none
initstring      : none
exit_after is   : not set
exit is         : no

Type [C-a] [C-h] to see available commands
Terminal ready
```

```
gmarull                        1:picocom*                    1.27 0.90 0.79 14:06
```

Figure: Boot without pressing Button 1: `Hello World!` printed on the default set of pins

Dymamic Pin Control: Example (VII)



```
parity is       : none              parity is       : none
databits are    : 8                 databits are    : 8
stopbits are    : 1                 stopbits are    : 1
escape is       : C-a               escape is       : C-a
local echo is   : no                local echo is   : no
noinit is       : no                noinit is       : no
noreset is      : no                noreset is      : no
hangup is       : no                hangup is       : no
nolock is       : no                nolock is       : no
send_cmd is     : sz -vv            send_cmd is     : sz -vv
receive_cmd is  : rz -vv -E         receive_cmd is  : rz -vv -E
imap is         :                   imap is         :
omap is         :                   omap is         :
emap is         : crcrlf,delbs,     emap is         : crcrlf,delbs,
logfile is      : none              logfile is      : none
initstring      : none              initstring      : none
exit_after is   : not set           exit_after is   : not set
exit is         : no                exit is         : no

Type [C-a] [C-h] to see available commands    Type [C-a] [C-h] to see available commands
Terminal ready                                Terminal ready
*** Booting Zephyr OS build v2.7.0-rc1-147-g28 *** Booting Zephyr OS build v2.7.0-rc1-147-g289
9b81e2751a  ***                               b81e2751a  ***
Hello World!                                  Hello World!

gmarull                    1:picocom*                    1.27 0.90 0.79 14:06
```

Figure: Boot pressing Button 1: `Hello World!` printed on the alternative set of pins

Conclusions

Conclusions

- Pin control is heavily **vendor dependent**, non-portable
- State model allows to **isolate** drivers from pin configurations, API is used the **same way** independently of the vendor
- State model allows to solve in a **generic** way problems such as pin configuration in **sleep** or low power modes
- **Devicetree** representation has been **partially standardized** (common properties, e.g. `bias-pull-up`)
- All **vendor specific** bits are contained in **Devicetree** and a **SoC specific header**
- Even if limited, **dynamic pin control** allows to route peripheral signals to another set of pins at **runtime**

THANK YOU!

Questions?

https://github.com/teslabs/zds-2022-pinctrl