How to use MCHP Linux on GPIO

Index

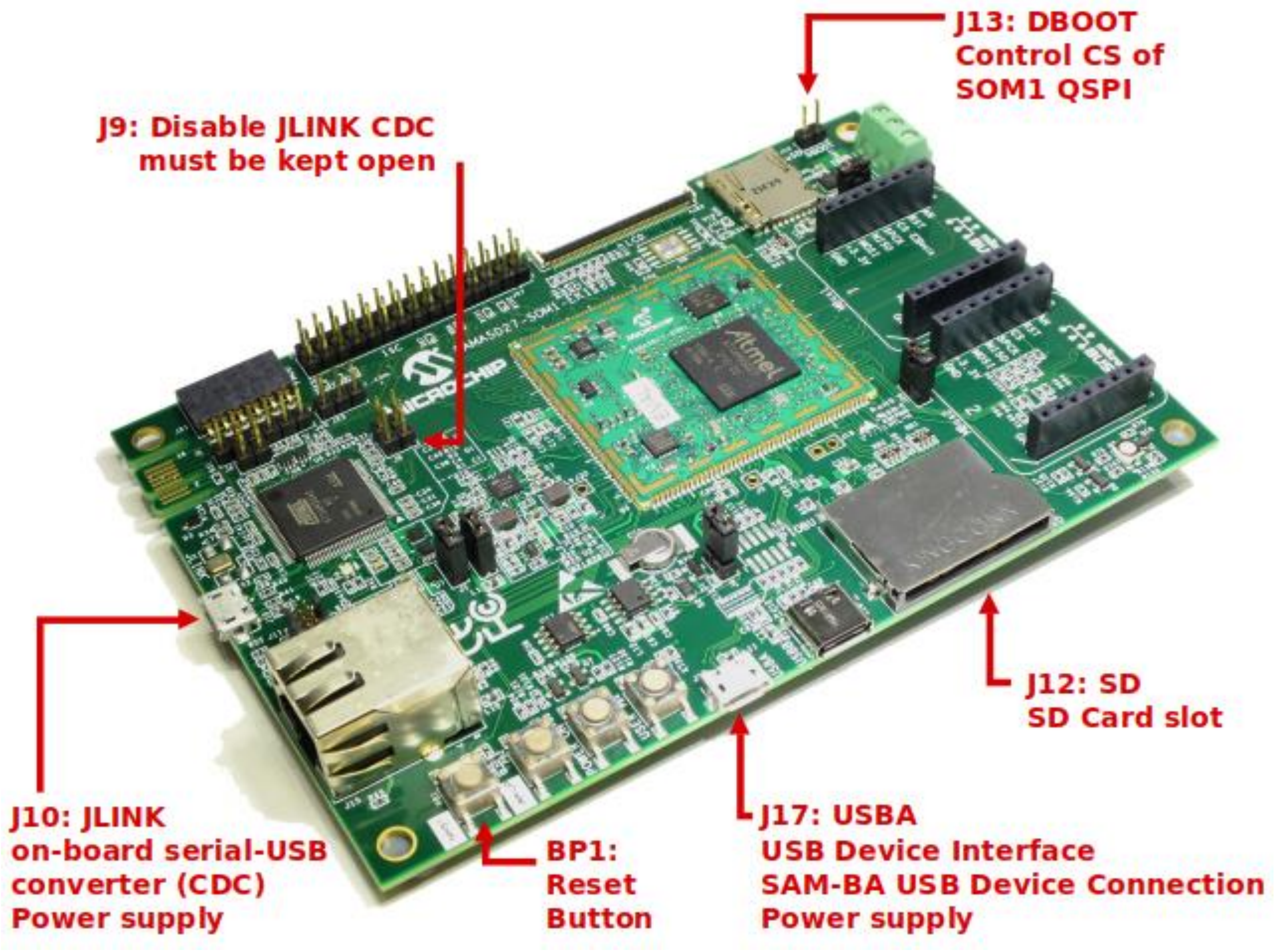
Table of Contents

Prerequisite

- Development environment
  This demo is running on AT91 Linux platform which built by Buildroot.
  Please setup AT91 Buildroot development environment via this document first:
  HowTo_AT91_Buildroot_Setup_Based_on_Ubuntu.pdf

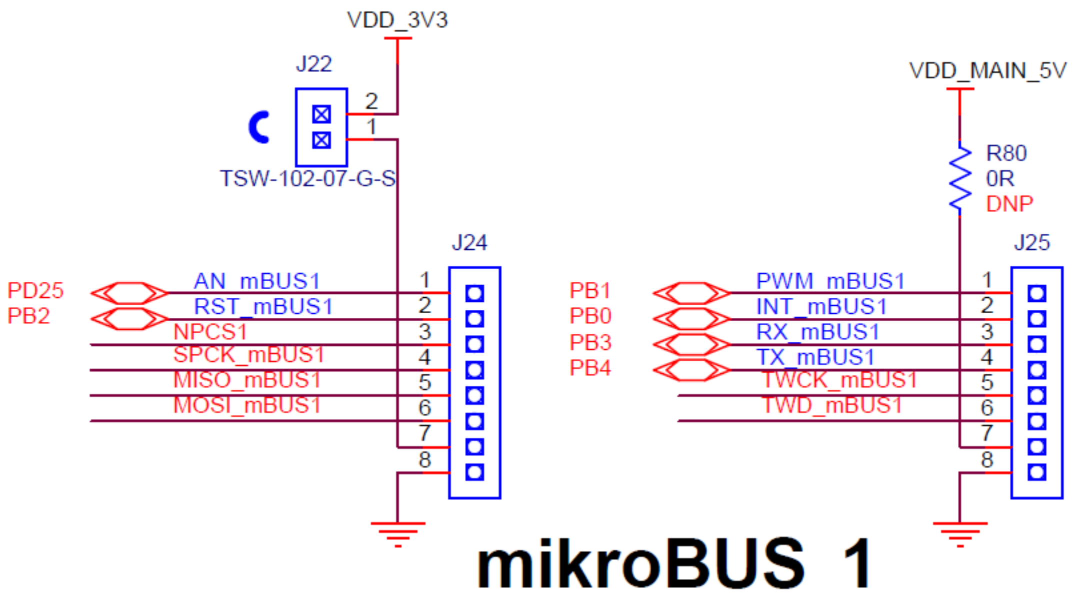
Hardware requirements

- PC
- SAMA5D27 SOM1 Evaluation Kit
- SD Card



1. Hardware design

1.1 Interface



"mikroBUS 1" was used for easy testing and monitoring.

This document will show you how to control GPIO PB2 (J24 pin2) on Linux.

2. Software design

Our Linux platform was built by buildroot with following configuration:
`atmel_sama5d27_som1_ek_mmc_dev_defconfig`
GPIO device driver should works under this default configuration without any adjustment.
Then let's check each part in needed.

2.1 Device tree

- Action: no need to change
- Location: buildroot-at91/output/build/linux-linux4sam_6.0/arch/arm/boot/dts
- Sources: sama5d2.dtsi

Device tree for GPIO function:

```
pioA: pinctrl@fc038000 {
    compatible = "atmel,sama5d2-pinctrl"; // specify which driver will be used for this pioA device
    reg = <0xfc038000 0x600>; // pioA base address is 0xfc038000, size is 0x600
    interrupts = <18 IRQ_TYPE_LEVEL_HIGH 7>, // 128 gpios were divided into four banks
            <68 IRQ_TYPE_LEVEL_HIGH 7>, // each gpio bank has its own irq line
//check buildroot-at91/output/build/linux-linux4sam_6.0/include/dt-bindings/interrupt-controller/irq.h
// for the definitions of TRQ_TYPE….
            <69 IRQ_TYPE_LEVEL_HIGH 7>,
            <70 IRQ_TYPE_LEVEL_HIGH 7>;
    interrupt-controller;
    #interrupt-cells = <2>;
    gpio-controller;
    #gpio-cells = <2>;
    clocks = <&pioA_clk>; // definition for pioA clock source
};

pioA_clk: pioA_clk {
    #clock-cells = <0>;
    reg = <18>; // PID of pioA is 18, this definition of offset will be used to enable pioA clock in PMC
    atmel,clk-output-range = <0 83000000>; // pioA input clock, max frequency is 83MHz
};
```

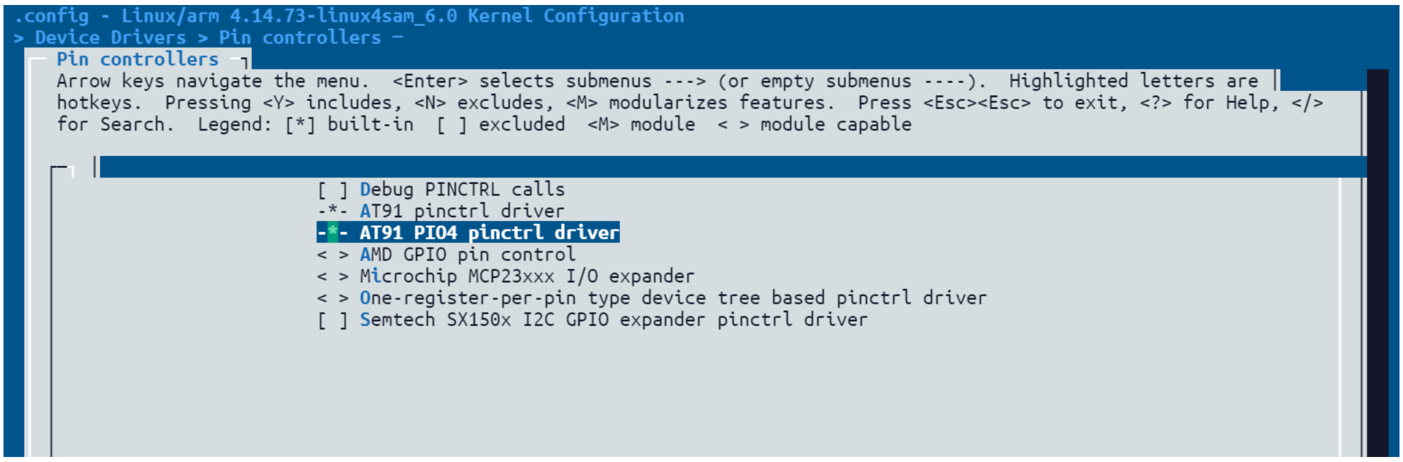
2.2 Kernel

- Action: no need to change
- Location: buildroot-at91/output/build/linux-linux4sam_6.0/
- Defconfig: sama5_defconfig
- Driver files: drivers/pinctrl/pinctrl-at91-pio4.c

Check kernel configuration for GPIO function:

```
user@at91:~/buildroot-at91$ make linux-menuconfig
```

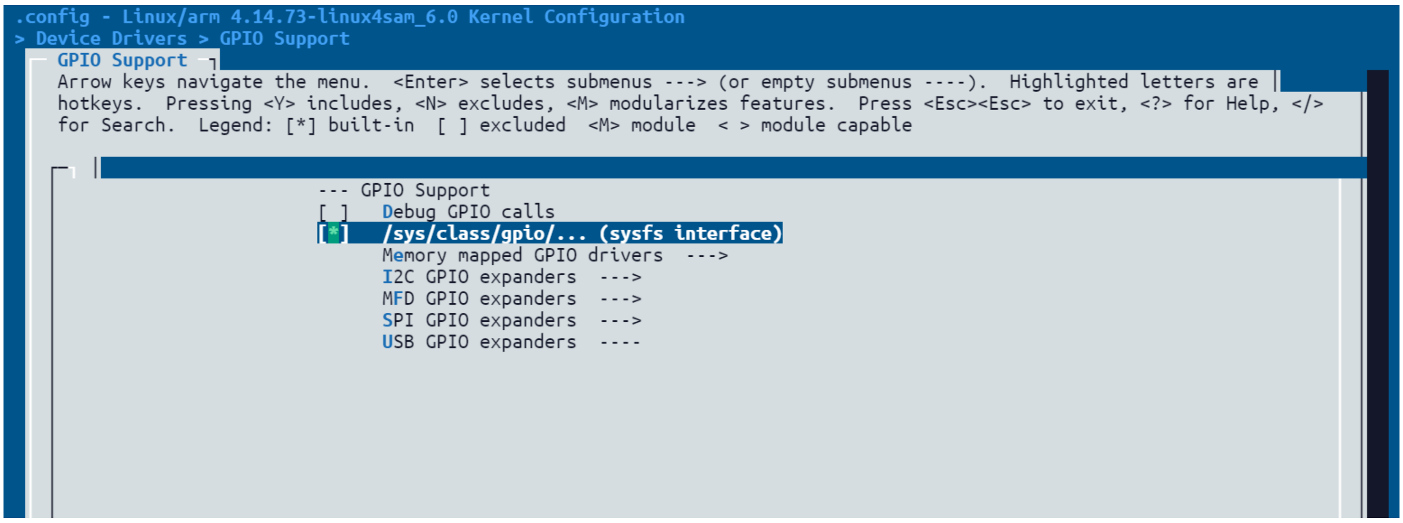
➢ Device Drivers > Pin controllers > **AT91 PIO4 pinctrl driver**
With this setting "pinctrl" and "gpio" driver for AT91 will be built into kernel.
Then we can access GPIO driver via device node in rootfs (/dev/gpiochip0)



```
.config - Linux/arm 4.14.73-linux4sam_6.0 Kernel Configuration
> Device Drivers > Pin controllers ─
  ┌─ Pin controllers ─┐
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are
  hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                        [ ] Debug PINCTRL calls
                        -*- AT91 pinctrl driver
                        -*- AT91 PIO4 pinctrl driver
                        < > AMD GPIO pin control
                        < > Microchip MCP23xxx I/O expander
                        < > One-register-per-pin type device tree based pinctrl driver
                        [ ] Semtech SX150x I2C GPIO expander pinctrl driver
```

➢ Device Drivers > GPIO Support > **/sys/class/gpio/... (sysfs interface)**
With this setting sysfs for "gpio" feature will be built into kernel.
Then we can access GPIO driver via sysfs in rootfs (/sys/class/gpio)



```
.config - Linux/arm 4.14.73-linux4sam_6.0 Kernel Configuration
> Device Drivers > GPIO Support
  ┌─ GPIO Support ─┐
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are
  hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                        --- GPIO Support
                        [ ]   Debug GPIO calls
                        [*]   /sys/class/gpio/... (sysfs interface)
                              Memory mapped GPIO drivers  --->
                              I2C GPIO expanders  --->
                              MFD GPIO expanders  --->
                              SPI GPIO expanders  --->
                              USB GPIO expanders  ----
```

2.3 Rootfs

- Action: no need to change
- Location: buildroot-at91/output/images/rootfs.tar

Two paths (file node) could be used to access GPIO driver:

- /dev/gpiochip0
  The dev node interface only could be accessed by C language, because most of operations must be done by ioctl().
- /sys/class/gpio
  The sysfs interface is more friendly for accessing, because all needed operations could be done by read() and write(). Normally this interface will be used in Scripts program or in command line.

2.4 Application

Following is an C language demo for accessing GPIO driver which based on dev node:

gpio.c

- How to compile

```
user@at91:~$ buildroot-at91/output/host/bin/arm-buildroot-linux-uclibcgnueabihf-gcc
gpio.c -o gpio_test
```

- Source code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <linux/gpio.h>
#include <sys/ioctl.h>

#define DEV_GPIO    "/dev/gpiochip0"

int main(int argc, char *argv[])
{
    int fd;
    int ret;

    struct gpiochip_info cinfo;
    struct gpioline_info linfo;
    struct gpiohandle_request req;
    struct gpiohandle_data data;

    /* open gpio */
    fd = open(DEV_GPIO, 0);
    if (fd < 0) {
        printf("ERROR: open %s ret=%d\n", DEV_GPIO, fd);
        return -1;
    }

    /* get gpio chip info */
    ret = ioctl(fd, GPIO_GET_CHIPINFO_IOCTL, &cinfo);
    if (ret < 0) {
        printf("ERROR get chip info ret=%d\n", ret);
        return -1;
    }
    printf("GPIO chip: %s, \"%s\", %u GPIO lines\n",
                    cinfo.name, cinfo.label, cinfo.lines);
    ret = ioctl(fd, GPIO_GET_LINEINFO_IOCTL, &linfo);
    if (ret < 0) {
        printf("ERROR get line info ret=%d\n", ret);
        return -1;
    }
    printf("line %2d: %s\n", linfo.line_offset,
                linfo.name);

    /* set gpio_pb2 output */
    // 128 gpio in gpiochip0
    // 0   ~ 31      PA0 -> PA31
    // 32 ~ 63    PB0 -> PB31
    // 33 ~ 95    PC0 -> PC31
    // 96 ~ 127 PD0 -> PD31
    req.lineoffsets[0] = 34;
    req.lines = 1;
    req.flags = GPIOHANDLE_REQUEST_OUTPUT;
    strcpy(req.consumer_label, "RST_mBUS1");
    int lhfd = ioctl(fd, GPIO_GET_LINEHANDLE_IOCTL, &req);
    if (lhfd < 0) {
        printf("ERROR get line handle lhdf=%d\n", lhfd);
        return -1;
    }
    data.values[0] = 1;
    ret = ioctl(req.fd, GPIOHANDLE_SET_LINE_VALUES_IOCTL, &data);
    if (ret < 0) {
        printf("ERROR set line value ret=%d\n", ret);
        return -1;
    }

    while (1) {
        // set gpio_pb2 low
        data.values[0] = 0;
        ioctl(req.fd, GPIOHANDLE_SET_LINE_VALUES_IOCTL, &data);
        usleep(5*1000);

        // set gpio_pb2 high
        data.values[0] = 1;
        ioctl(req.fd, GPIOHANDLE_SET_LINE_VALUES_IOCTL, &data);
        usleep(5*1000);
    }

    /* close gpio */
    close(fd);

    return 0;
}
```

3. Hands-on

As we talked before, there are two different paths for accessing GPIO driver:

- Accessing via dev node

Copy gpio_test app to target and execute it, then 100Hz wave will be generated via PB2 (J24 pin2)

```
# chmod +x gpio_test
# ./gpio_test
```

- Accessing via sysfs

// Export PB2

```
# echo 34 > /sys/class/gpio/export
```

// Set PB2 output

```
# echo out > /sys/class/gpio/PB2/direction
```

// Set PB2 low

```
# echo 0 > /sys/class/gpio/PB2/value
```

// Set PB2 high

```
# echo 1 > /sys/class/gpio/PB2/value
```

Number to GPIO port:

0 ~ 31	PA0 -> PA31
32 ~ 63	PB0 -> PB31
33 ~ 95	PC0 -> PC31
96 ~ 127	PD0 -> PD31