

# RV3028 Linux driver documentation

Bootlin

December 15, 2018

## Contents

---

<b>Building the driver</b>	<b>1</b>
out of tree . . . . .	1
in-tree . . . . .	2
<b>Connecting the RV-3028-C7 development board</b>	<b>2</b>
Atmel Sama5d3 Xplained . . . . .	2
Device Tree . . . . .	3
Beaglebone Black . . . . .	4
Device Tree . . . . .	5
<b>Using the RTC</b>	<b>5</b>
Loading the driver . . . . .	5
Finding the RTC device . . . . .	6
Reading the date . . . . .	6
Setting the date . . . . .	6
Reading and clearing the Voltage Low flags . . . . .	6
Enabling the trickle charger . . . . .	8
Reading and setting the crystal frequency offset . . . . .	8
Reading and resetting the timestamp . . . . .	8

## Building the driver

---

### out of tree

---

Simply use make in the driver directory.

```
$ make
make -C /lib/modules/`uname -r`/build M=$PWD
make[1]: Entering directory '/usr/src/linux-headers-4.19.0-1-amd64'
CC [M] /usr/src/rv3028/rtc-rv3028.o
Building modules, stage 2.
```

```
MODPOST 1 modules
CC      /husr/src/rv3028/rtc-rv3028.mod.o
LD [M]  /usr/src/rv3028/rtc-rv3028.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.19.0-1-amd64'
```

If cross-compiling or if the kernel headers directory detection fails, pass KDIR:

```
$ make KDIR=/usr/src/linux
make -C /usr/src/linux/ M=$PWD
make[1]: Entering directory '/usr/src/linux'
CC [M]  /usr/src/rv3028/rtc-rv3028.o
Building modules, stage 2.
MODPOST 1 modules
CC      /husr/src/rv3028/rtc-rv3028.mod.o
LD [M]  /usr/src/rv3028/rtc-rv3028.ko
make[1]: Leaving directory '/usr/src/linux'
```

## in-tree

---

For Linux 4.19 :

```
$ git apply < 0001-rtc-rv3028-new-driver.patch
```

Then select CONFIG\_RTC\_DRV\_RV3028 in your kernel configuration.






## Connecting the RV-3028-C7 development board

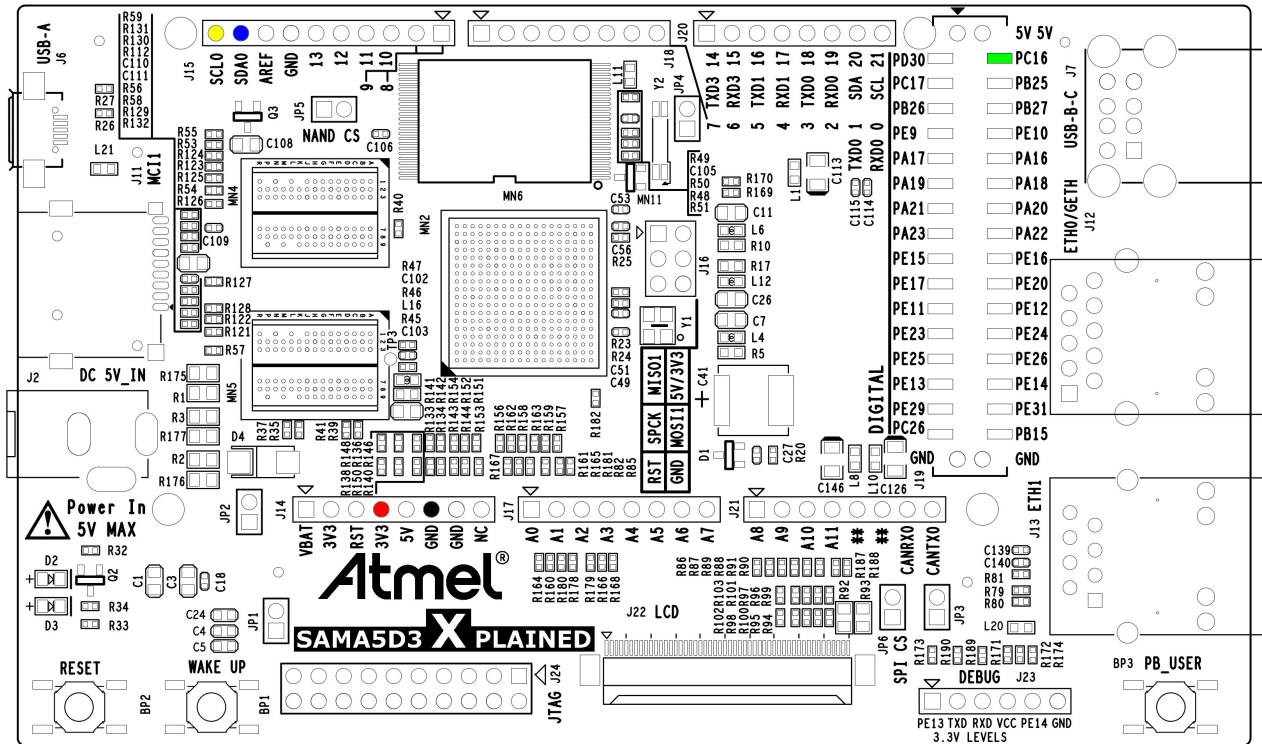
---

### Atmel Sama5d3 Xplained

---

The following configuration has been tested:

RV3028	SAMA5D3 Xplained	
VDD	J14 3V3	
VSS	J14 GND	
SDA	J15 SDA0	
SCL	J15 SCL0	
nINT	J19 PC16	



## Device Tree

In the SAMA5D3 Xplained devicetree (arch/arm/boot/dts/at91-sama5d3\_xplained.dts), the rtc node should be added as a child of the existing i2c0 node. The end result should look like:

```
i2c0: i2c@f0014000 {
    pinctrl-0 = <&pinctrl_i2c0_pu>;
    status = "okay";

    rtc: rv3028@52 {
        compatible = "microcrystal,rv3028";
        reg = <0x52>;
        status = "okay";
        pinctrl-0 = <&pinctrl_rtc_int>;
        interrupts-extended = <&pioC 16 GPIO_ACTIVE_HIGH>;
    };
};
```

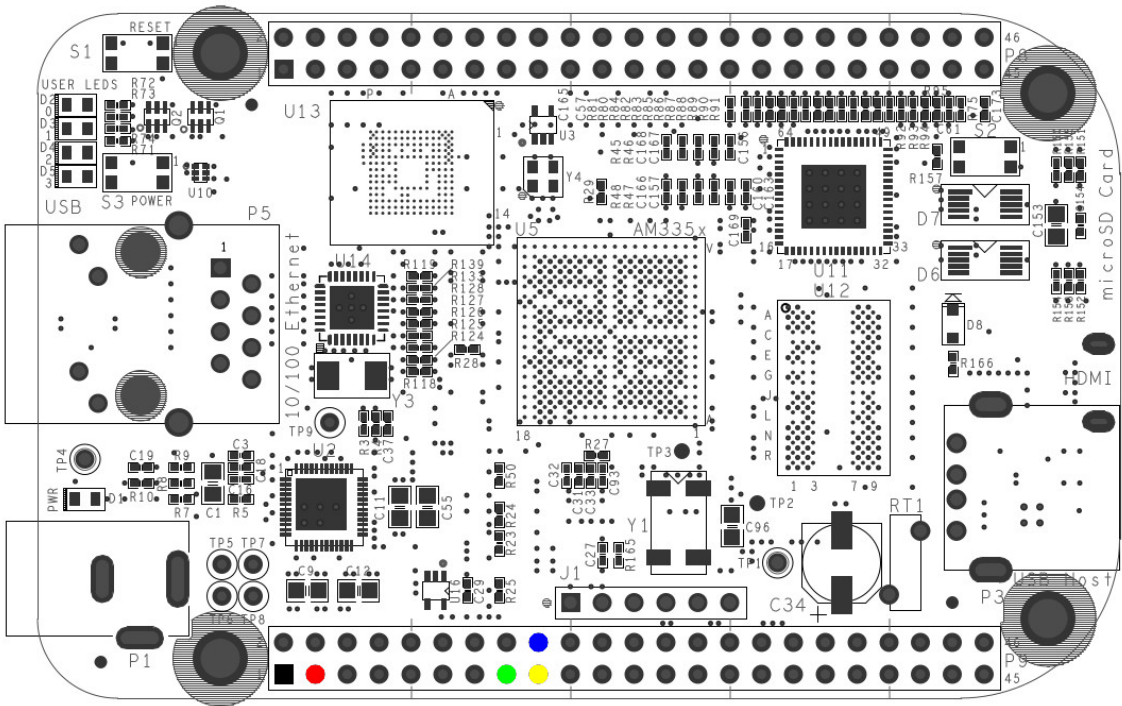
A pinctrl definition has to be added in the pinctrl node:

```
pinctrl@fffff200 {
    rtc {
        pinctrl_rtc_int: rtc_int {
            atmel,pins =
                <AT91_PIOC 16 AT91_PERIPH_GPIO AT91_PINCTRL_PULL_UP_DEGLITCH>;
        };
    };
};
```

Beaglebone Black

The following configuration has been tested:

RV3028	Beaglebone (P9)	
VDD	3	<div style="width: 10px; height: 10px; background-color: red;"></div>
VSS	1	<div style="width: 10px; height: 10px; background-color: black;"></div>
SDA	18	<div style="width: 10px; height: 10px; background-color: blue;"></div>
SCL	17	<div style="width: 10px; height: 10px; background-color: yellow;"></div>
nINT	15	<div style="width: 10px; height: 10px; background-color: green;"></div>



## Device Tree

In the Beaglebone Black device tree (arch/arm/boot/dts/am335x-boneblack.dts), i2c1 should be enabled and the rtc node added as a child. The end result should look like:

```
&i2c1 {
    status = "okay";
    clock-frequency = <100000>;

    pinctrl-names = "default";
    pinctrl-0 = <&i2c1_pins>;

    rv3028: rtc@52 {
        compatible = "microcrystal,rv3028";
        reg = <0x52>;
        status = "okay";
        pinctrl-0 = <&rtc_nint_pins>;
        interrupts-extended = <&gpio1 16 GPIO_ACTIVE_HIGH>;
    };
};
```

Pinctrl definitions have to be added in the pinctrl node:

```
&am33xx_pinmux {
    rtc_nint_pins: pinmux_rtc_nint_pins {
        pinctrl-single,pins = <
            0x040 (PIN_INPUT | MUX_MODE7)    /* gpmc_a1.gpio1_16 */
        >;
    };

    /* Pins 17 (I2C1_SCL) and 18 (I2C1_SDA) of connector P9 */
    i2c1_pins: pinmux_i2c1_pins {
        pinctrl-single,pins = <
            0x158 (PIN_INPUT_PULLUP | MUX_MODE2)    /* spi0_d1.i2c1_sda */
            0x15c (PIN_INPUT_PULLUP | MUX_MODE2)    /* spi0_cs0.i2c1_scl */
        >;
    };
};
```

## Using the RTC

### Loading the driver

If the driver has been built statically in the kernel, nothing has to be done. If it is built as a module, then load it with:

```
$ modprobe rtc-rv3028
```

or if that is not working:

```
$ insmod /path/to/rtc-rv3028.ko
```

## Finding the RTC device

---

On initialization, the driver will log the following message:

```
$ dmesg | grep rv3028
rtc-rv3028 1-0052: registered as rtc1
```

This means that `rtc1` is the RV3028 rtc.

## Reading the date

---

```
$ hwclock -r -f /dev/rtc1
```

You'll get the following message when the date is invalid:

```
hwclock: RTC_RD_TIME: Invalid argument
```

You will also see the following message in your kernel log:

```
$ dmesg | grep rv8803
rtc-rv3028 1-0052: Voltage low, data is invalid.
```

## Setting the date

---

Set the system date:

```
$ date -s "2016-04-20 09:30"
```

Write the date to the RTC:

```
$ hwclock -w -f /dev/rtc1
```

Writing the date will clear the voltage low flags.

## Reading and clearing the Voltage Low flags

---

The `RTC_VL_READ` and `RTC_VL_CLR` ioctls are implemented. The following program will read the flags then clear them and read again:

```
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <linux/rtc.h>
```



```
#ifndef RTC_VL_READ
#include <linux/ioctl.h>

#define RTC_VL_READ    _IOR('p', 0x13, int)    /* Voltage low detector */
#define RTC_VL_CLR     _IO('p', 0x14)         /* Clear voltage low informat

#endif

int main(int argc, char **argv)
{
    int fd, retval;
    const char *rtc = "/dev/rtc1";
    int vl;

    fd = open(rtc, O_RDWR);

    if ( fd == -1 ) {
        perror(rtc);
        exit(errno);
    }

    retval = ioctl(fd, RTC_VL_READ, &vl);
    if (retval < 0) {
        perror("RTC_VL_READ failed");
        exit(errno);
    }
    printf("%s flags: %d\n", rtc, vl);

    retval = ioctl(fd, RTC_VL_CLR, NULL);
    if (retval < 0) {
        perror("RTC_VL_CLR failed");
        exit(errno);
    }

    retval = ioctl(fd, RTC_VL_READ, &vl);
    if (retval < 0) {
        perror("RTC_VL_READ failed");
        exit(errno);
    }
    printf("%s flags: %d\n", rtc, vl);

    close(fd);

    return 0;
}
```

## Enabling the trickle charger

---

To enable the trickle charger, simply set the `trickle-resistor-ohms` property in the `rtc` node. For example:

```
rv3028: rtc@52 {
    compatible = "microcrystal,rv3028";
    reg = <0x52>;
    status = "okay";
    pinctrl-0 = <&rtc_nint_pins>;
    interrupts-extended = <&gpio1 16 GPIO_ACTIVE_HIGH>;
    trickle-resistor-ohms = <11000>;
};
```

## Reading and setting the crystal frequency offset

---

The `/sys/class/rtc/rtcX/offset` file exposes the current frequency offset correction in parts per billion.

For example, to read the offset:

```
# cat /sys/class/rtc/rtc1/offset
20027
```

And to set the offset:

```
# echo 21000 > /sys/class/rtc/rtc1/offset
# cat /sys/class/rtc/rtc1/offset
20981
```

You can notice that the closest value is automatically chosen.

## Reading and resetting the timestamp

---

The `/sys/class/rtc/rtcX/timestamp0` file exposes the stored timestamp as a UNIX timestamp. The timestamp count is also exposed in `/sys/class/rtc/rtcX/timestamp0_count`.

For example, to read the timestamp:

```
# cat /sys/class/rtc/rtc1/timestamp0
1544829583
# cat /sys/class/rtc/rtc1/timestamp0_count
2
```

And to reset the timestamp:

```
# echo 0 > /sys/class/rtc/rtc1/timestamp0
# cat /sys/class/rtc/rtc1/timestamp0
#
```



Note that using `select` or `poll` is working to get notice as soon as the event interrupt happens.