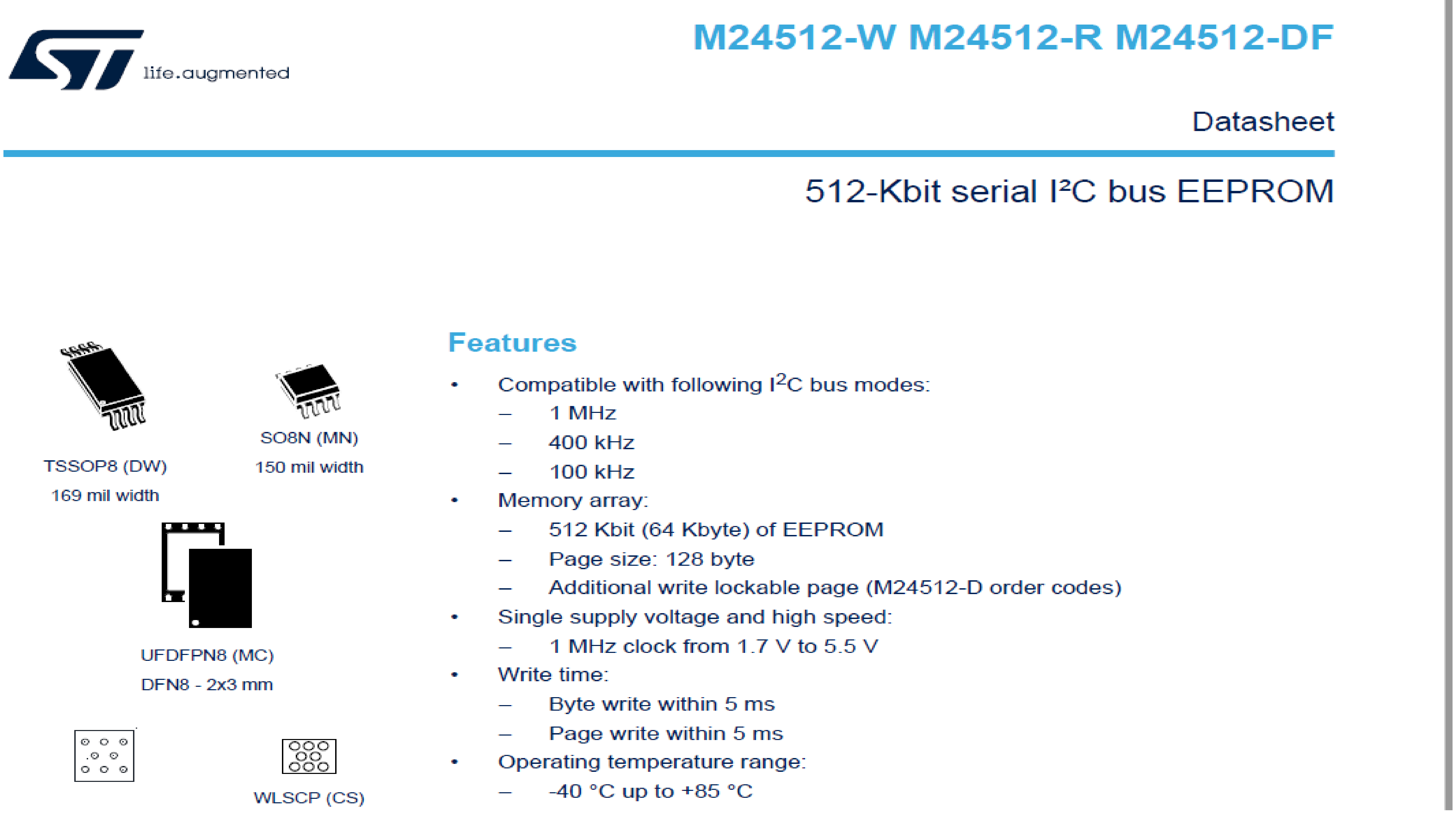
EEPROM driver porting

- I2C_tool verification
- Porting EEPROM driver
- Driver && eeprom
- User Space test case

I2C_tool verification



life.augmented

M24512-W M24512-R M24512-DF

Datasheet

512-Kbit serial I²C bus EEPROM

TSSOP8 (DW)
169 mil width

SO8N (MN)
150 mil width

UFDFPN8 (MC)
DFN8 - 2x3 mm

WLSCP (CS)

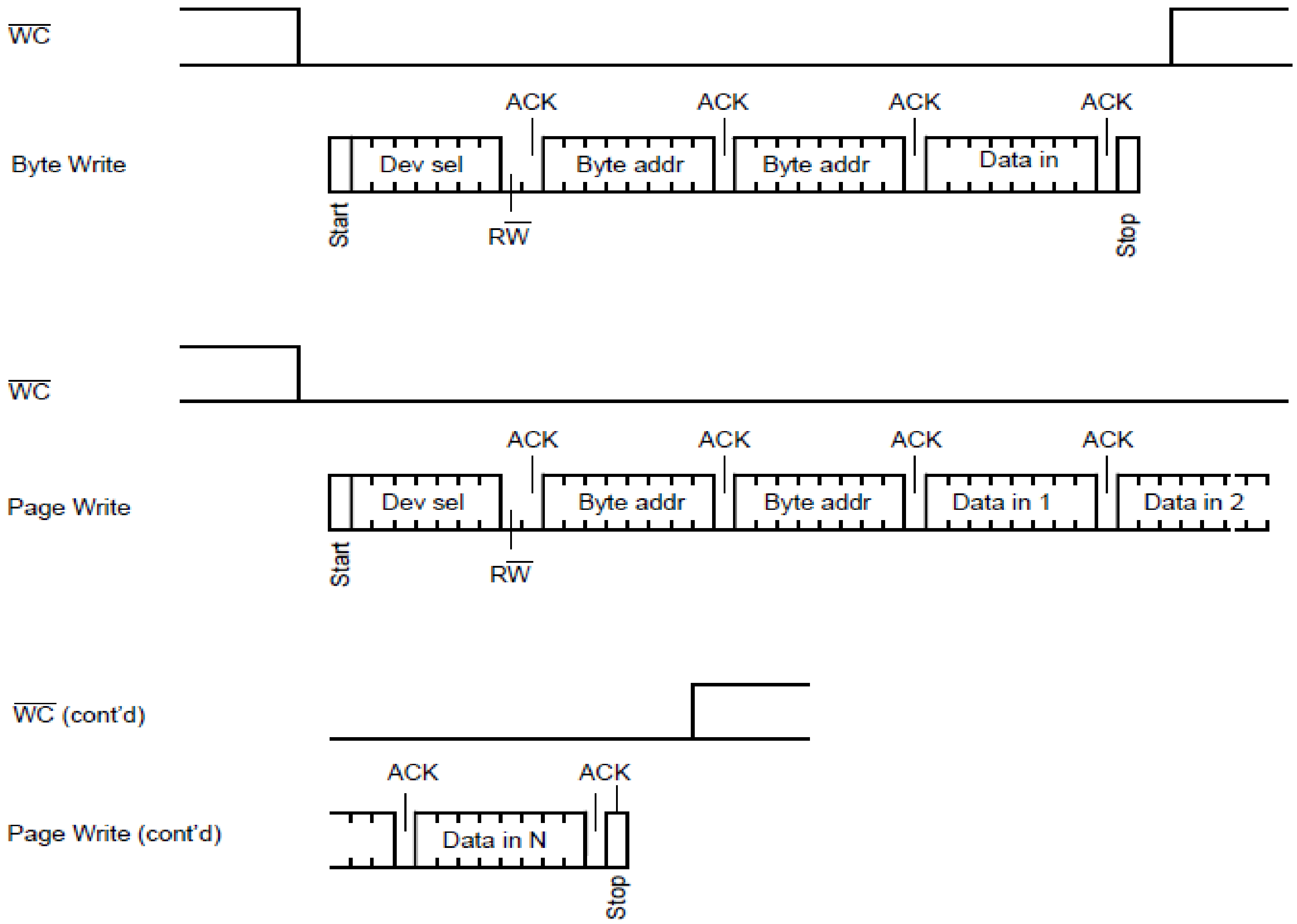
Features

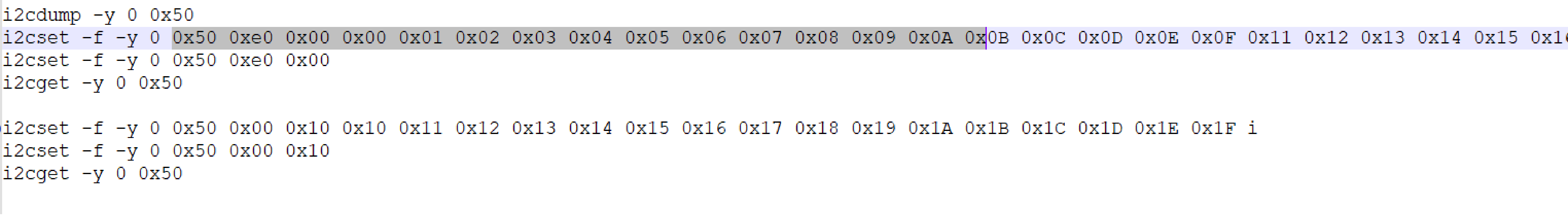
- Compatible with following I^{2}C bus modes:
  - 1 MHz
  - 400 kHz
  - 100 kHz
- Memory array:
  - 512 Kbit (64 Kbyte) of EEPROM
  - Page size: 128 byte
  - Additional write lockable page (M24512-D order codes)
- Single supply voltage and high speed:
  - 1 MHz clock from 1.7 V to 5.5 V
- Write time:
  - Byte write within 5 ms
  - Page write within 5 ms
- Operating temperature range:
  - -40 °C up to +85 °C

5.1.1 Byte write

After the device select code and the address bytes, the bus master sends one data byte. If the addressed location is write-protected, by write control ($\overline{\text{WC}}$) being driven high, the device replies with NoAck, and the location is not modified. If, instead, the addressed location is not write-protected, the device replies with Ack. The bus master terminates the transfer by generating a stop condition, as shown in Figure 7.

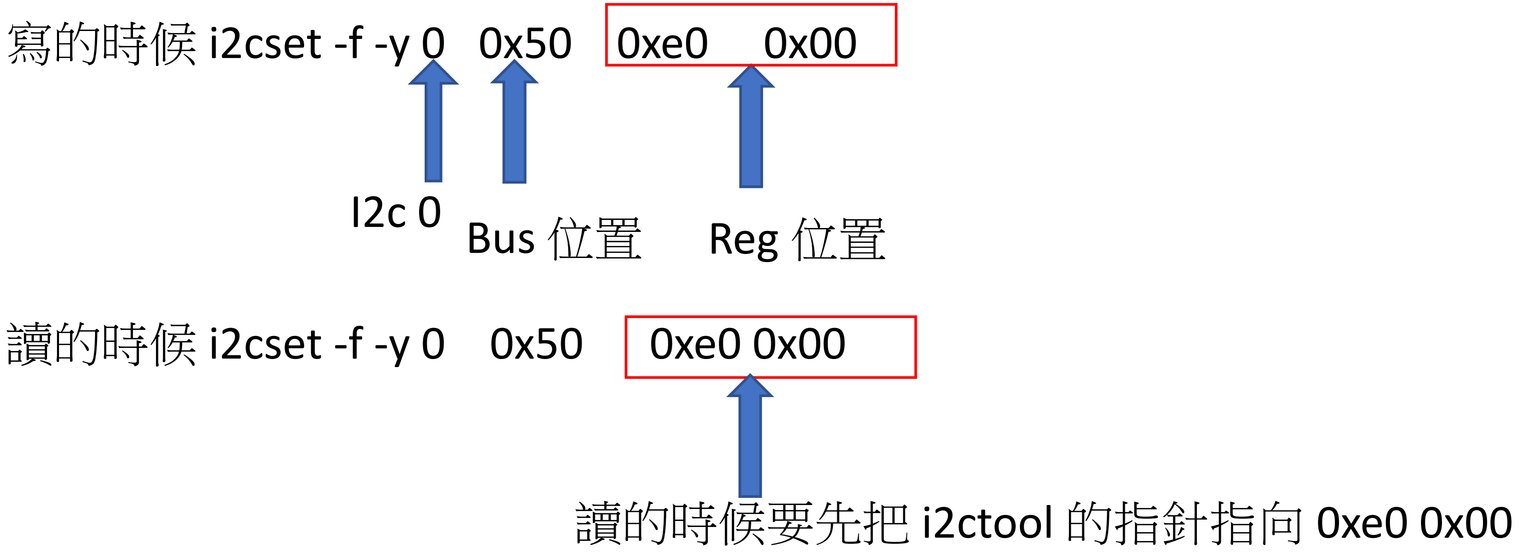
Figure 7. Write mode sequences with $\overline{\text{WC}}$ = 0 (data write enabled)





```
i2cdump -y 0 0x50
i2cset -f -y 0 0x50 0xe0 0x00 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F 0x11 0x12 0x13 0x14 0x15 0x1
i2cset -f -y 0 0x50 0xe0 0x00
i2cget -y 0 0x50

i2cset -f -y 0 0x50 0x00 0x10 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1A 0x1B 0x1C 0x1D 0x1E 0x1F i
i2cset -f -y 0 0x50 0x00 0x10
i2cget -y 0 0x50
```

I2c bus = 0x50 位置



寫的時候 i2cset -f -y 0 0x50 0xe0 0x00

I2c 0

Bus 位置

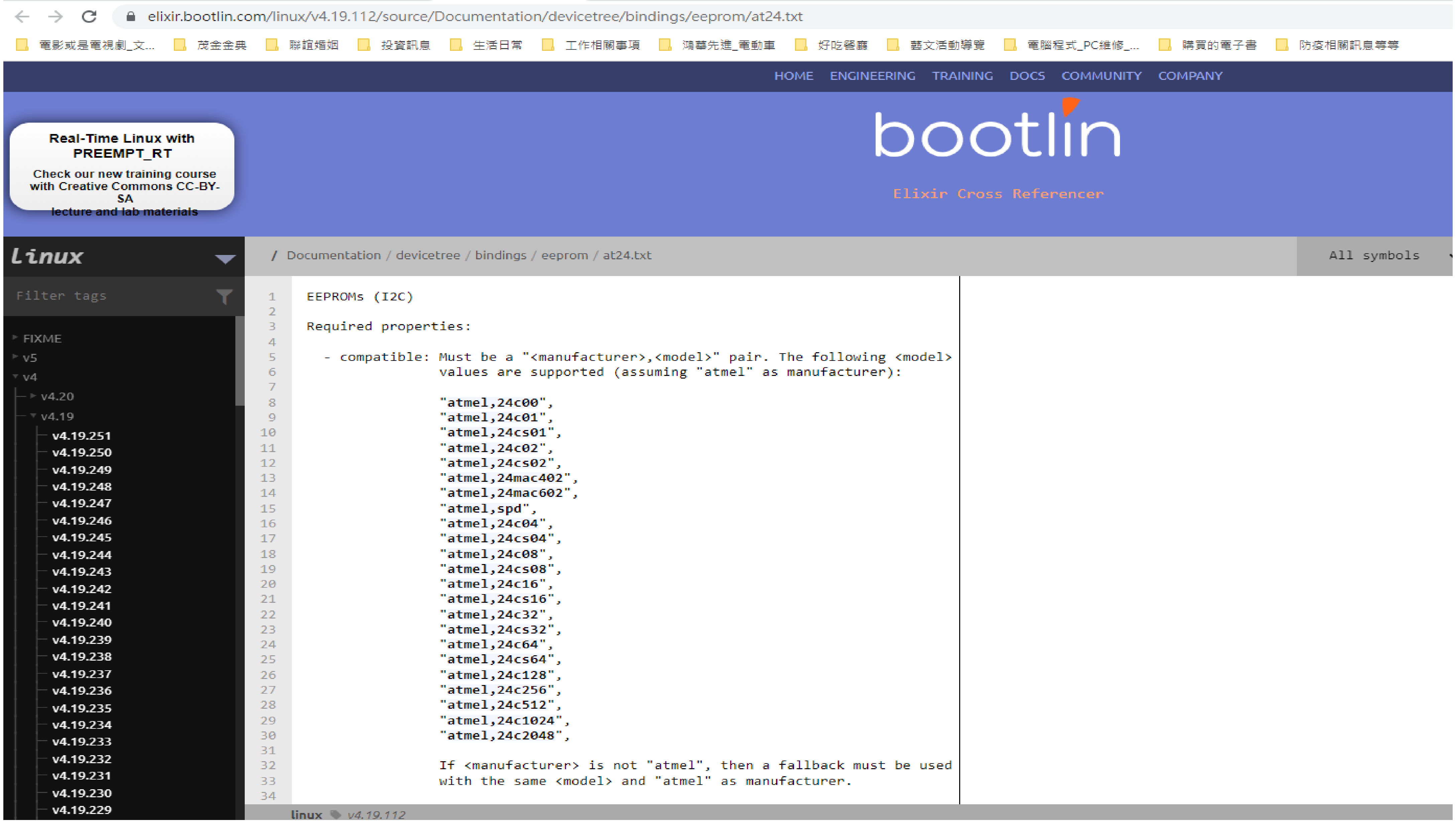
Reg 位置

讀的時候 i2cset -f -y 0 0x50 0xe0 0x00

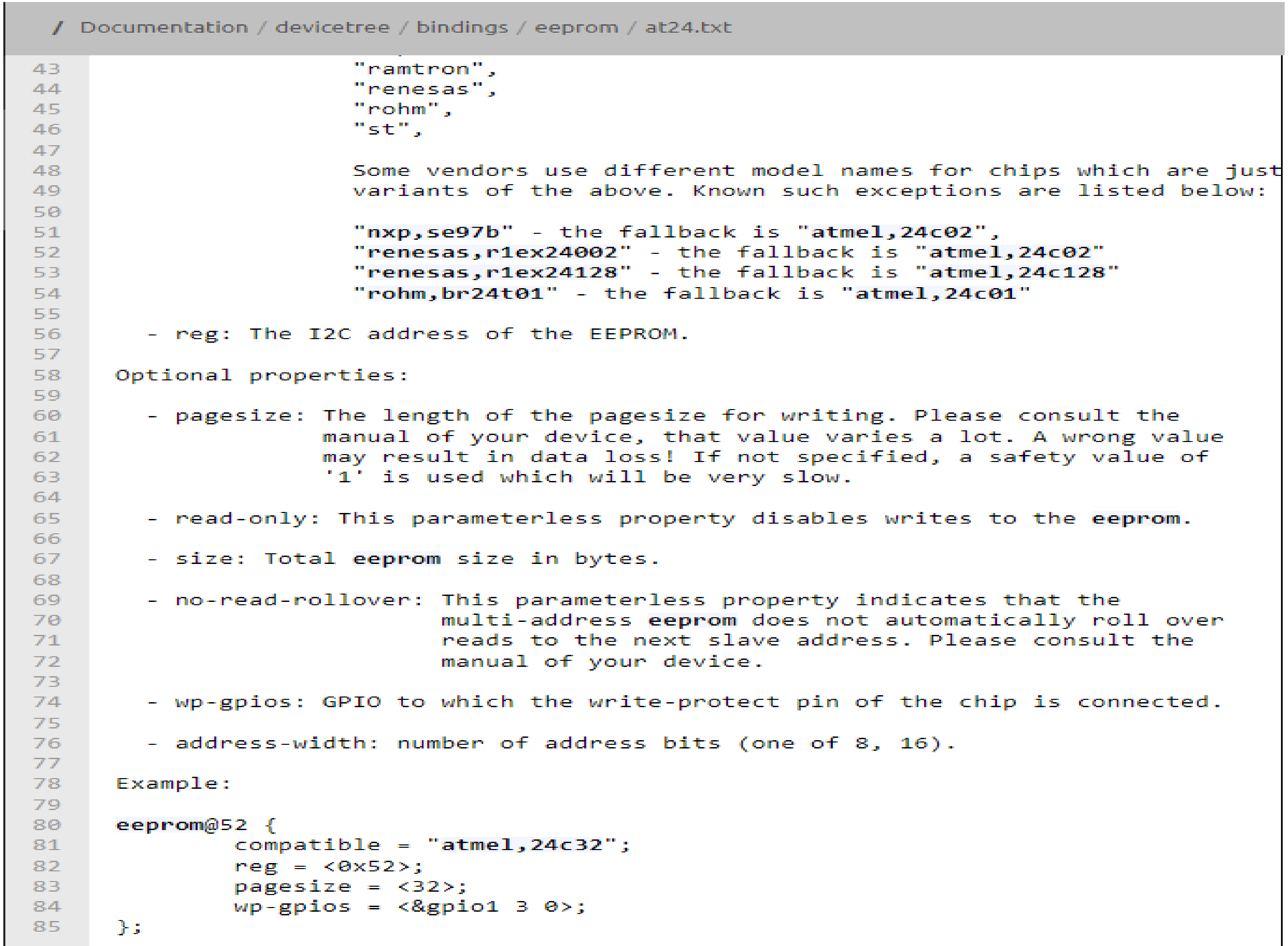
讀的時候要先把 i2ctool 的指針指向 0xe0 0x00

開始讀 i2cget -y 0 0x50 // 這時候指針就是指向0xe0 那個位置了

Porting EEPROM driver

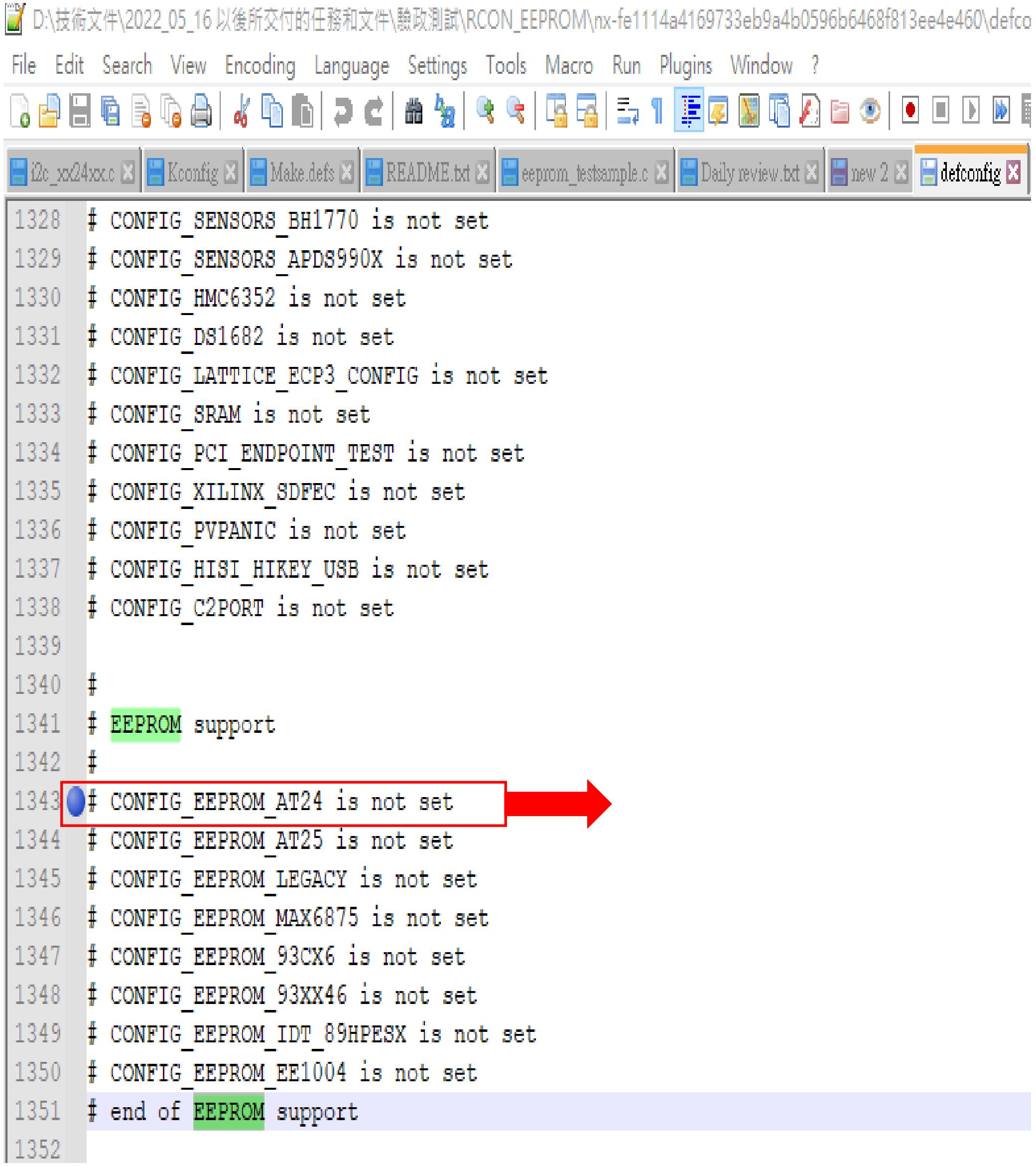


```
EEPROMs (I2C)

Required properties:

  - compatible: Must be a "<manufacturer>,<model>" pair. The following <model>
                values are supported (assuming "atmel" as manufacturer):

                "atmel,24c00",
                "atmel,24c01",
                "atmel,24cs01",
                "atmel,24c02",
                "atmel,24cs02",
                "atmel,24mac402",
                "atmel,24mac602",
                "atmel,spd",
                "atmel,24c04",
                "atmel,24cs04",
                "atmel,24c08",
                "atmel,24cs08",
                "atmel,24c16",
                "atmel,24cs16",
                "atmel,24c32",
                "atmel,24cs32",
                "atmel,24c64",
                "atmel,24cs64",
                "atmel,24c128",
                "atmel,24c256",
                "atmel,24c512",
                "atmel,24c1024",
                "atmel,24c2048",

                If <manufacturer> is not "atmel", then a fallback must be used
                with the same <model> and "atmel" as manufacturer.

```

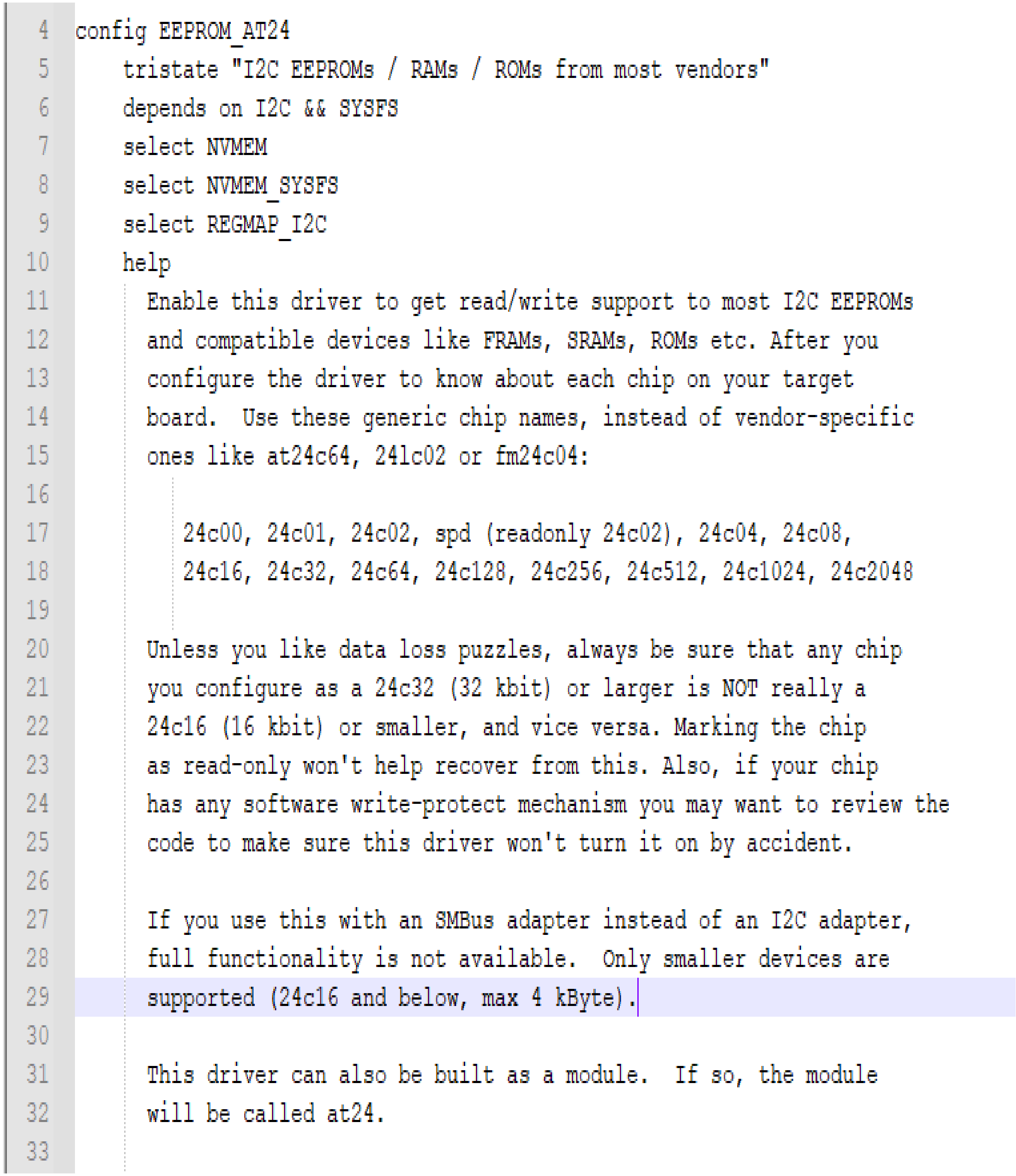


/ Documentation / devicetree / bindings / eeprom / at24.txt

```
                "ramtron",
                "renesas",
                "rohm",
                "st",

                Some vendors use different model names for chips which are just
                variants of the above. Known such exceptions are listed below:

                "nxp,se97b" - the fallback is "atmel,24c02",
                "renesas,r1ex24002" - the fallback is "atmel,24c02"
                "renesas,r1ex24128" - the fallback is "atmel,24c128"
                "rohm,br24t01" - the fallback is "atmel,24c01"

  - reg: The I2C address of the EEPROM.

Optional properties:

  - pagesize: The length of the pagesize for writing. Please consult the
              manual of your device, that value varies a lot. A wrong value
              may result in data loss! If not specified, a safety value of
              '1' is used which will be very slow.

  - read-only: This parameterless property disables writes to the eeprom.

  - size: Total eeprom size in bytes.

  - no-read-rollover: This parameterless property indicates that the
                      multi-address eeprom does not automatically roll over
                      reads to the next slave address. Please consult the
                      manual of your device.

  - wp-gpios: GPIO to which the write-protect pin of the chip is connected.

  - address-width: number of address bits (one of 8, 16).

Example:

eeprom@52 {
        compatible = "atmel,24c32";
        reg = <0x52>;
        pagesize = <32>;
        wp-gpios = <&gpio1 3 0>;
};
```


D:\技術文件\2022_05_16 以後所交付的任務和文件\驗證測試\RCON_EEPROM\nx-fe1114a4169733eb9a4b0596b6468f813ee4e460\defco
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
i2c_xx24xx.c
Kconfig
Make.defs
README.txt
eeprom_testsample.c
Daily review.txt
new 2
defconfig

```
# CONFIG_SENSORS_BH1770 is not set
# CONFIG_SENSORS_APDS990X is not set
# CONFIG_HMC6352 is not set
# CONFIG_DS1682 is not set
# CONFIG_LATTICE_ECP3_CONFIG is not set
# CONFIG_SRAM is not set
# CONFIG_PCI_ENDPOINT_TEST is not set
# CONFIG_XILINX_SDFEC is not set
# CONFIG_PVPANIC is not set
# CONFIG_HISI_HIKEY_USB is not set
# CONFIG_C2PORT is not set

#
# EEPROM support
#
# CONFIG_EEPROM_AT24 is not set
# CONFIG_EEPROM_AT25 is not set
# CONFIG_EEPROM_LEGACY is not set
# CONFIG_EEPROM_MAX6875 is not set
# CONFIG_EEPROM_93CX6 is not set
# CONFIG_EEPROM_93XX46 is not set
# CONFIG_EEPROM_IDT_89HPESX is not set
# CONFIG_EEPROM_EE1004 is not set
# end of EEPROM support
```

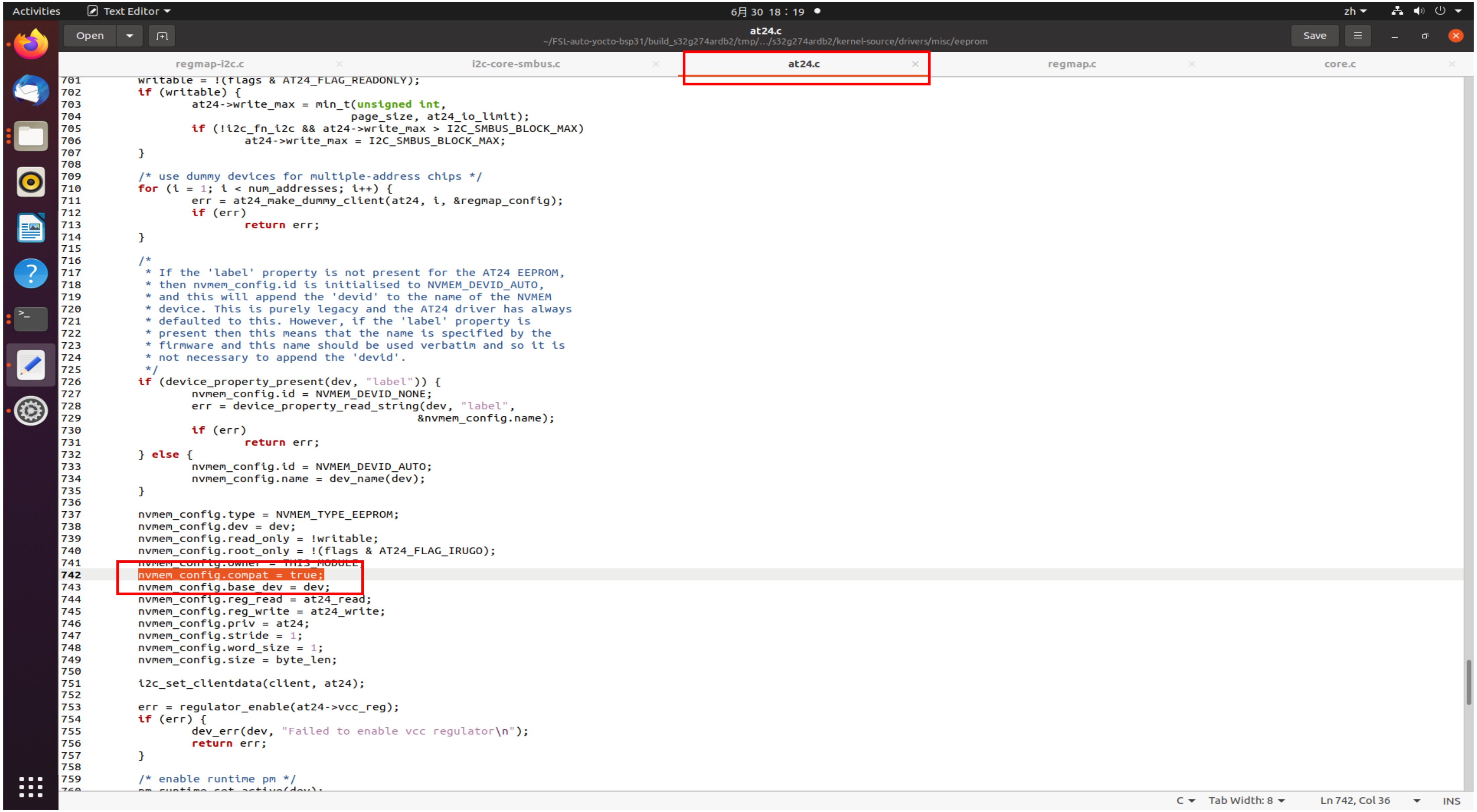


```
config EEPROM_AT24
	tristate "I2C EEPROMs / RAMs / ROMs from most vendors"
	depends on I2C && SYSFS
	select NVMEM
	select NVMEM_SYSFS
	select REGMAP_I2C
	help
	  Enable this driver to get read/write support to most I2C EEPROMs
	  and compatible devices like FRAMs, SRAMs, ROMs etc. After you
	  configure the driver to know about each chip on your target
	  board.  Use these generic chip names, instead of vendor-specific
	  ones like at24c64, 24lc02 or fm24c04:

	     24c00, 24c01, 24c02, spd (readonly 24c02), 24c04, 24c08,
	     24c16, 24c32, 24c64, 24c128, 24c256, 24c512, 24c1024, 24c2048

	  Unless you like data loss puzzles, always be sure that any chip
	  you configure as a 24c32 (32 kbit) or larger is NOT really a
	  24c16 (16 kbit) or smaller, and vice versa. Marking the chip
	  as read-only won't help recover from this. Also, if your chip
	  has any software write-protect mechanism you may want to review the
	  code to make sure this driver won't turn it on by accident.

	  If you use this with an SMBus adapter instead of an I2C adapter,
	  full functionality is not available.  Only smaller devices are
	  supported (24c16 and below, max 4 kByte).

	  This driver can also be built as a module.  If so, the module
	  will be called at24.
```

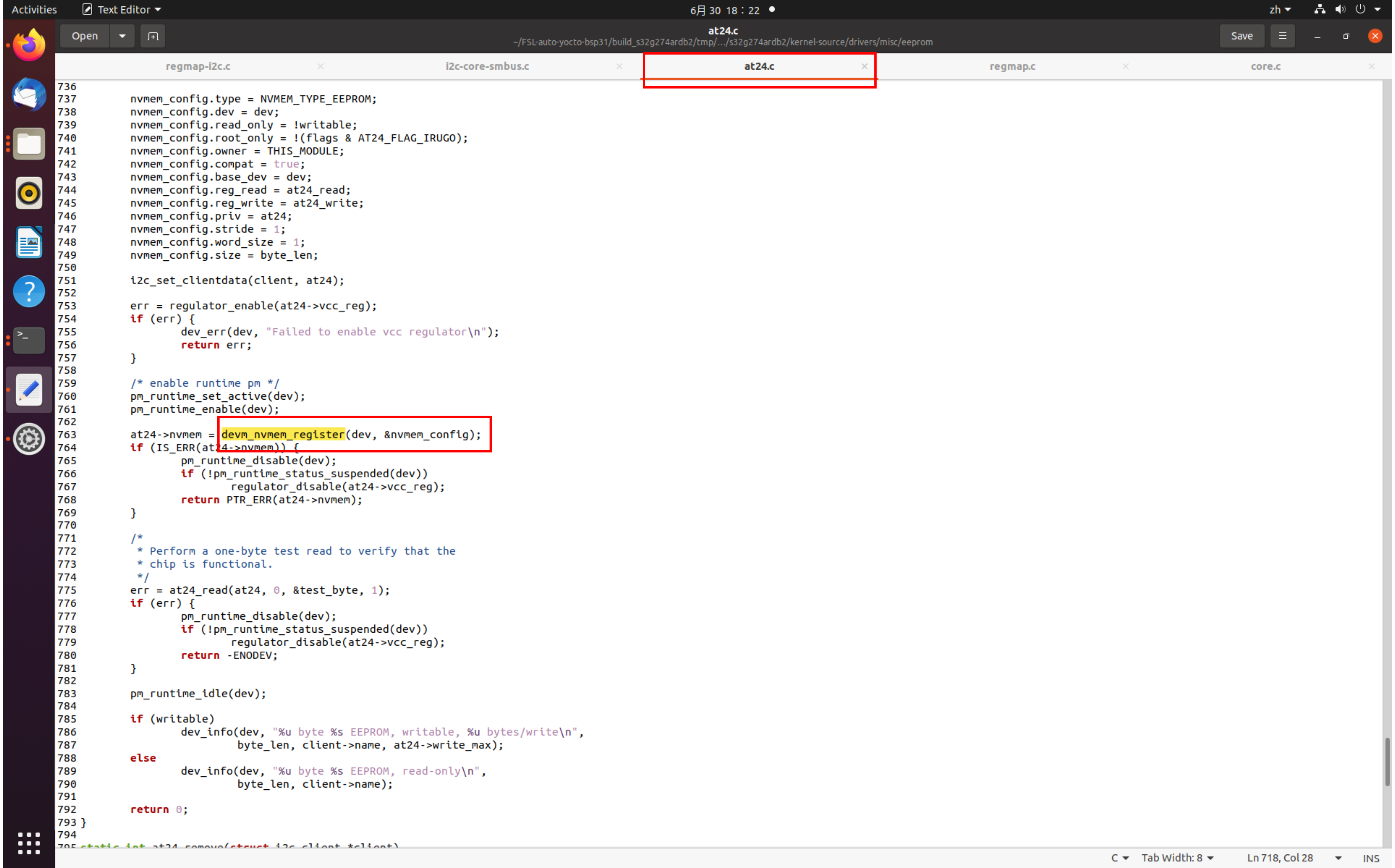
```
cat /sys/bus/i2c/device/i2c-0/0-0050/name
eeprom
```

It shall have Device node

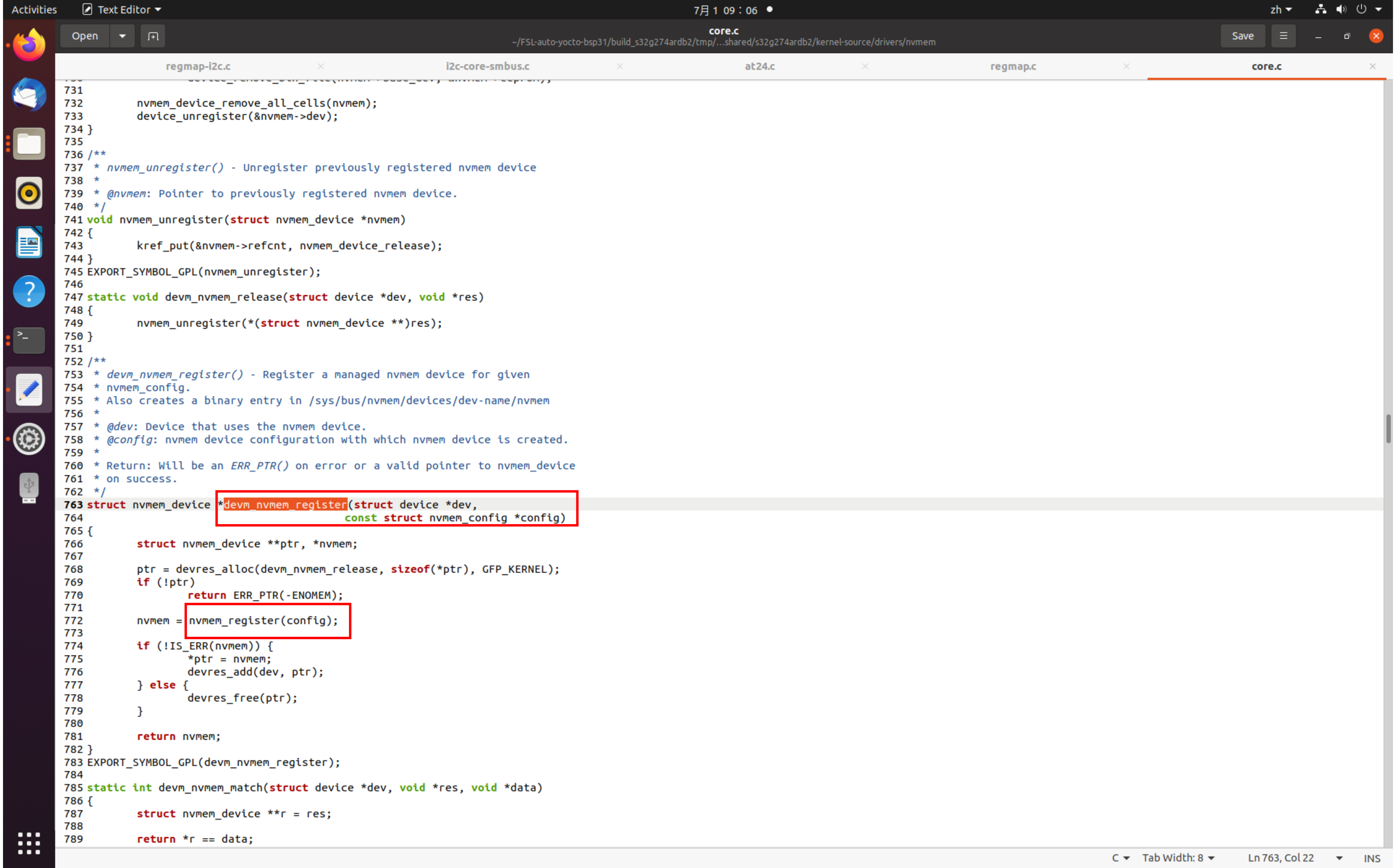
Driver && eeprom



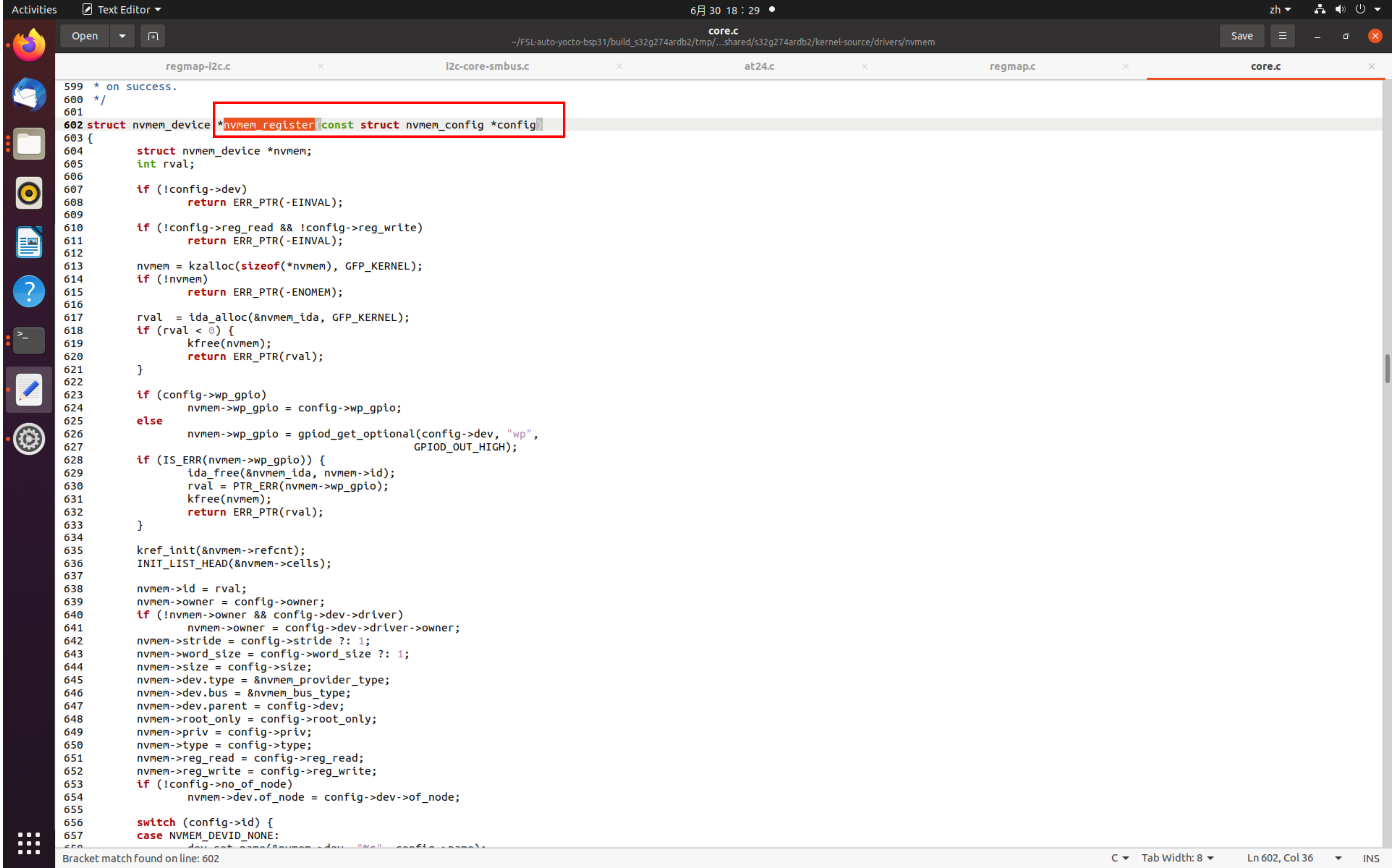
```c
        writable = !(flags & AT24_FLAG_READONLY);
        if (writable) {
                at24->write_max = min_t(unsigned int,
                                        page_size, at24_io_limit);
                if (!i2c_fn_i2c && at24->write_max > I2C_SMBUS_BLOCK_MAX)
                        at24->write_max = I2C_SMBUS_BLOCK_MAX;
        }

        /* use dummy devices for multiple-address chips */
        for (i = 1; i < num_addresses; i++) {
                err = at24_make_dummy_client(at24, i, &regmap_config);
                if (err)
                        return err;
        }

        /*
         * If the 'label' property is not present for the AT24 EEPROM,
         * then nvmem_config.id is initialised to NVMEM_DEVID_AUTO,
         * and this will append the 'devid' to the name of the NVMEM
         * device. This is purely legacy and the AT24 driver has always
         * defaulted to this. However, if the 'label' property is
         * present then this means that the name is specified by the
         * firmware and this name should be used verbatim and so it is
         * not necessary to append the 'devid'.
         */
        if (device_property_present(dev, "label")) {
                nvmem_config.id = NVMEM_DEVID_NONE;
                err = device_property_read_string(dev, "label",
                                                  &nvmem_config.name);
                if (err)
                        return err;
        } else {
                nvmem_config.id = NVMEM_DEVID_AUTO;
                nvmem_config.name = dev_name(dev);
        }

        nvmem_config.type = NVMEM_TYPE_EEPROM;
        nvmem_config.dev = dev;
        nvmem_config.read_only = !writable;
        nvmem_config.root_only = !(flags & AT24_FLAG_IRUGO);
        nvmem_config.owner = THIS_MODULE;
        nvmem_config.compat = true;
        nvmem_config.base_dev = dev;
        nvmem_config.reg_read = at24_read;
        nvmem_config.reg_write = at24_write;
        nvmem_config.priv = at24;
        nvmem_config.stride = 1;
        nvmem_config.word_size = 1;
        nvmem_config.size = byte_len;

        i2c_set_clientdata(client, at24);

        err = regulator_enable(at24->vcc_reg);
        if (err) {
                dev_err(dev, "Failed to enable vcc regulator\n");
                return err;
        }

        /* enable runtime pm */
        pm_runtime_set_active(dev);
```



```

        nvmem_config.type = NVMEM_TYPE_EEPROM;
        nvmem_config.dev = dev;
        nvmem_config.read_only = !writable;
        nvmem_config.root_only = !(flags & AT24_FLAG_IRUGO);
        nvmem_config.owner = THIS_MODULE;
        nvmem_config.compat = true;
        nvmem_config.base_dev = dev;
        nvmem_config.reg_read = at24_read;
        nvmem_config.reg_write = at24_write;
        nvmem_config.priv = at24;
        nvmem_config.stride = 1;
        nvmem_config.word_size = 1;
        nvmem_config.size = byte_len;

        i2c_set_clientdata(client, at24);

        err = regulator_enable(at24->vcc_reg);
        if (err) {
                dev_err(dev, "Failed to enable vcc regulator\n");
                return err;
        }

        /* enable runtime pm */
        pm_runtime_set_active(dev);
        pm_runtime_enable(dev);

        at24->nvmem = devm_nvmem_register(dev, &nvmem_config);
        if (IS_ERR(at24->nvmem)) {
                pm_runtime_disable(dev);
                if (!pm_runtime_status_suspended(dev))
                        regulator_disable(at24->vcc_reg);
                return PTR_ERR(at24->nvmem);
        }

        /*
         * Perform a one-byte test read to verify that the
         * chip is functional.
         */
        err = at24_read(at24, 0, &test_byte, 1);
        if (err) {
                pm_runtime_disable(dev);
                if (!pm_runtime_status_suspended(dev))
                        regulator_disable(at24->vcc_reg);
                return -ENODEV;
        }

        pm_runtime_idle(dev);

        if (writable)
                dev_info(dev, "%u byte %s EEPROM, writable, %u bytes/write\n",
                         byte_len, client->name, at24->write_max);
        else
                dev_info(dev, "%u byte %s EEPROM, read-only\n",
                         byte_len, client->name);

        return 0;
}

static int at24_remove(struct i2c_client *client)
```

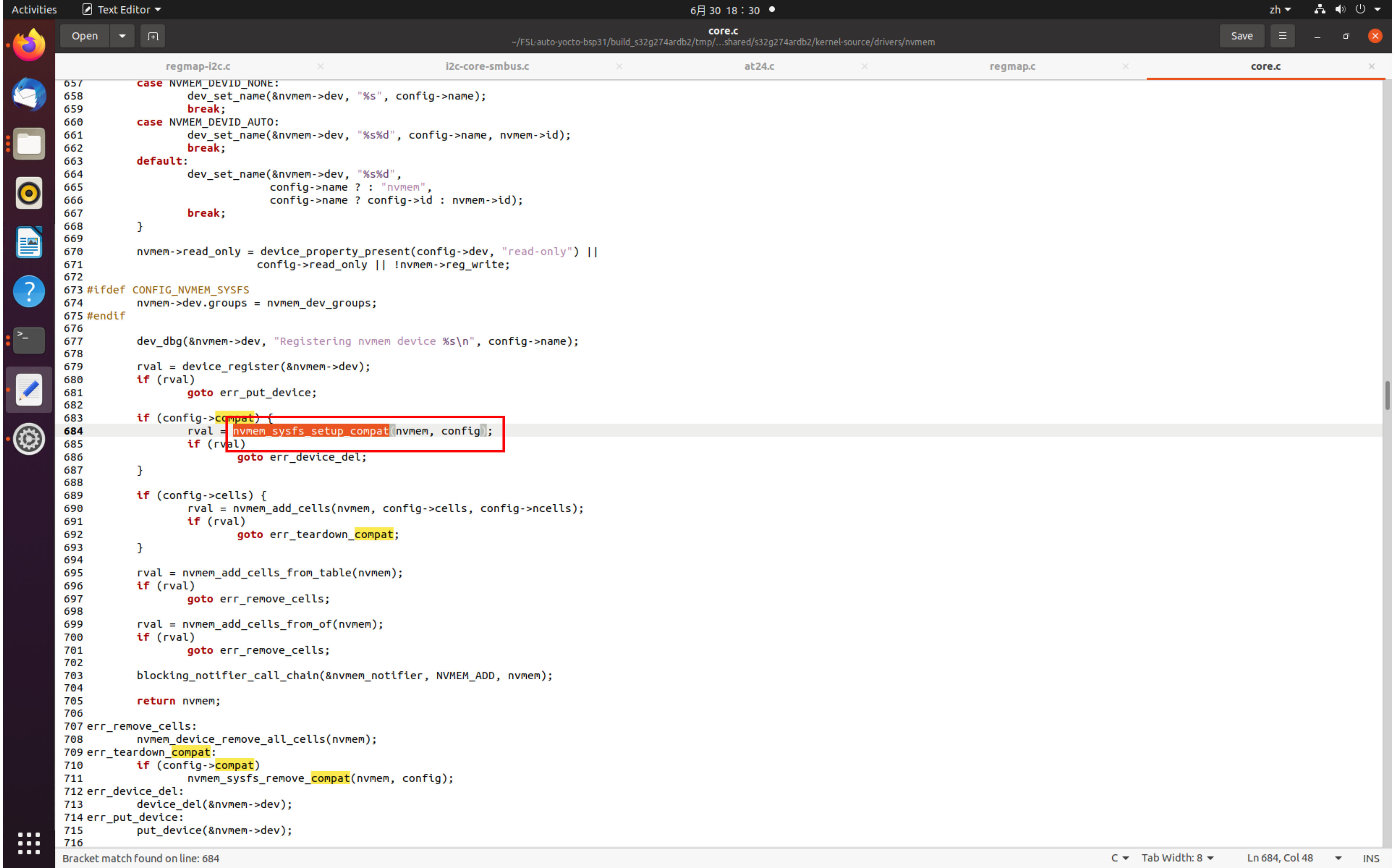


```

        nvmem_device_remove_all_cells(nvmem);
        device_unregister(&nvmem->dev);
}

/**
 * nvmem_unregister() - Unregister previously registered nvmem device
 *
 * @nvmem: Pointer to previously registered nvmem device.
 */
void nvmem_unregister(struct nvmem_device *nvmem)
{
        kref_put(&nvmem->refcnt, nvmem_device_release);
}
EXPORT_SYMBOL_GPL(nvmem_unregister);

static void devm_nvmem_release(struct device *dev, void *res)
{
        nvmem_unregister(*(struct nvmem_device **)res);
}

/**
 * devm_nvmem_register() - Register a managed nvmem device for given
 * nvmem_config.
 * Also creates a binary entry in /sys/bus/nvmem/devices/dev-name/nvmem
 *
 * @dev: Device that uses the nvmem device.
 * @config: nvmem device configuration with which nvmem device is created.
 *
 * Return: Will be an ERR_PTR() on error or a valid pointer to nvmem_device
 * on success.
 */
struct nvmem_device *devm_nvmem_register(struct device *dev,
                                        const struct nvmem_config *config)
{
        struct nvmem_device **ptr, *nvmem;

        ptr = devres_alloc(devm_nvmem_release, sizeof(*ptr), GFP_KERNEL);
        if (!ptr)
                return ERR_PTR(-ENOMEM);

        nvmem = nvmem_register(config);

        if (!IS_ERR(nvmem)) {
                *ptr = nvmem;
                devres_add(dev, ptr);
        } else {
                devres_free(ptr);
        }

        return nvmem;
}
EXPORT_SYMBOL_GPL(devm_nvmem_register);

static int devm_nvmem_match(struct device *dev, void *res, void *data)
{
        struct nvmem_device **r = res;

        return *r == data;
```



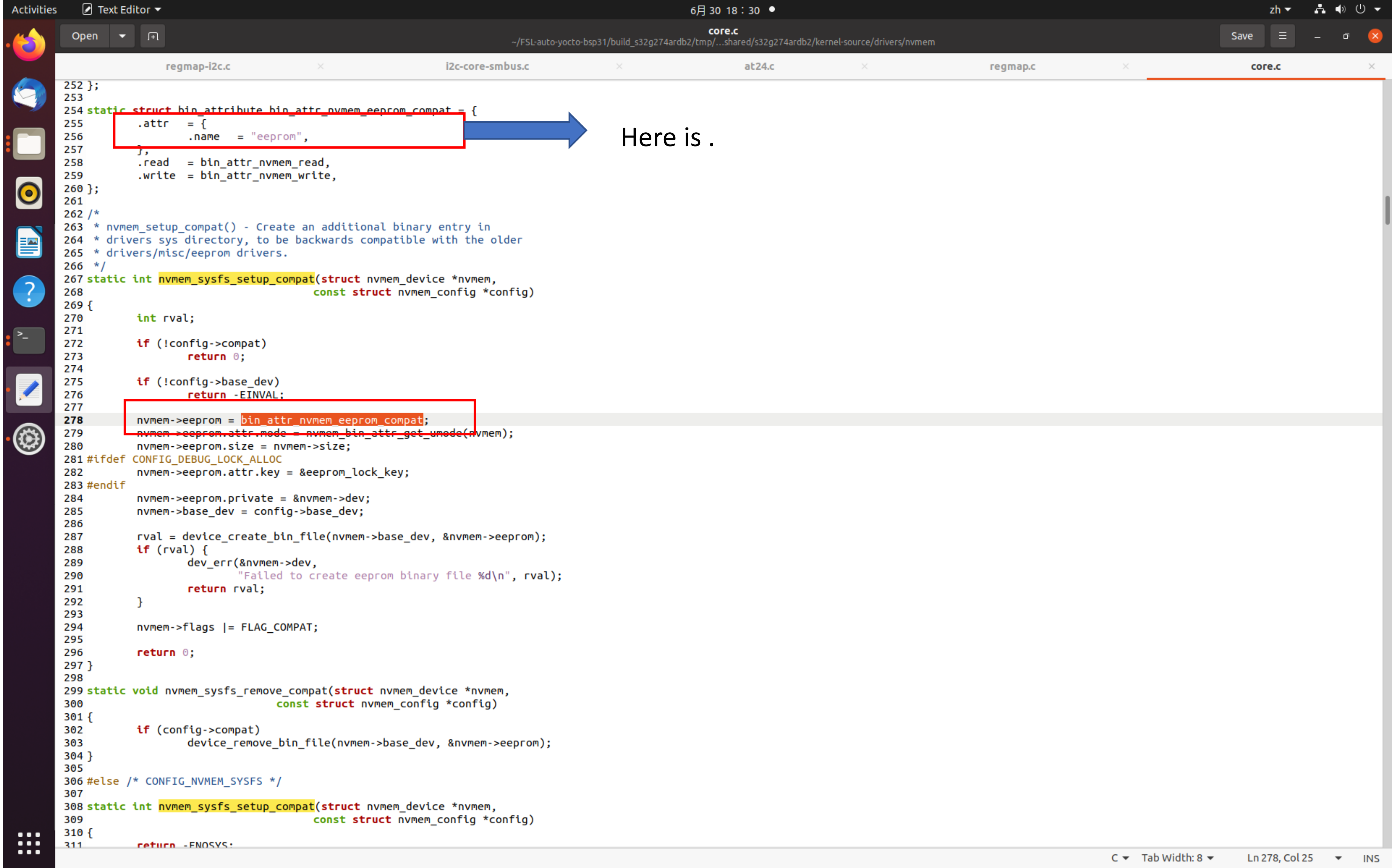
```c
 * on success.
 */

struct nvmem_device *nvmem_register(const struct nvmem_config *config)
{
        struct nvmem_device *nvmem;
        int rval;

        if (!config->dev)
                return ERR_PTR(-EINVAL);

        if (!config->reg_read && !config->reg_write)
                return ERR_PTR(-EINVAL);

        nvmem = kzalloc(sizeof(*nvmem), GFP_KERNEL);
        if (!nvmem)
                return ERR_PTR(-ENOMEM);

        rval  = ida_alloc(&nvmem_ida, GFP_KERNEL);
        if (rval < 0) {
                kfree(nvmem);
                return ERR_PTR(rval);
        }

        if (config->wp_gpio)
                nvmem->wp_gpio = config->wp_gpio;
        else
                nvmem->wp_gpio = gpiod_get_optional(config->dev, "wp",
                                                   GPIOD_OUT_HIGH);
        if (IS_ERR(nvmem->wp_gpio)) {
                ida_free(&nvmem_ida, nvmem->id);
                rval = PTR_ERR(nvmem->wp_gpio);
                kfree(nvmem);
                return ERR_PTR(rval);
        }

        kref_init(&nvmem->refcnt);
        INIT_LIST_HEAD(&nvmem->cells);

        nvmem->id = rval;
        nvmem->owner = config->owner;
        if (!nvmem->owner && config->dev->driver)
                nvmem->owner = config->dev->driver->owner;
        nvmem->stride = config->stride ?: 1;
        nvmem->word_size = config->word_size ?: 1;
        nvmem->size = config->size;
        nvmem->dev.type = &nvmem_provider_type;
        nvmem->dev.bus = &nvmem_bus_type;
        nvmem->dev.parent = config->dev;
        nvmem->root_only = config->root_only;
        nvmem->priv = config->priv;
        nvmem->type = config->type;
        nvmem->reg_read = config->reg_read;
        nvmem->reg_write = config->reg_write;
        if (!config->no_of_node)
                nvmem->dev.of_node = config->dev->of_node;

        switch (config->id) {
        case NVMEM_DEVID_NONE:
                dev_set_name(&nvmem->dev, "%s", config->name);
```

Bracket match found on line: 602



```
	case NVMEM_DEVID_NONE:
		dev_set_name(&nvmem->dev, "%s", config->name);
		break;
	case NVMEM_DEVID_AUTO:
		dev_set_name(&nvmem->dev, "%s%d", config->name, nvmem->id);
		break;
	default:
		dev_set_name(&nvmem->dev, "%s%d",
			     config->name ? : "nvmem",
			     config->name ? config->id : nvmem->id);
		break;
	}

	nvmem->read_only = device_property_present(config->dev, "read-only") ||
			   config->read_only || !nvmem->reg_write;

#ifdef CONFIG_NVMEM_SYSFS
	nvmem->dev.groups = nvmem_dev_groups;
#endif

	dev_dbg(&nvmem->dev, "Registering nvmem device %s\n", config->name);

	rval = device_register(&nvmem->dev);
	if (rval)
		goto err_put_device;

	if (config->compat) {
		rval = nvmem_sysfs_setup_compat(nvmem, config);
		if (rval)
			goto err_device_del;
	}

	if (config->cells) {
		rval = nvmem_add_cells(nvmem, config->cells, config->ncells);
		if (rval)
			goto err_teardown_compat;
	}

	rval = nvmem_add_cells_from_table(nvmem);
	if (rval)
		goto err_remove_cells;

	rval = nvmem_add_cells_from_of(nvmem);
	if (rval)
		goto err_remove_cells;

	blocking_notifier_call_chain(&nvmem_notifier, NVMEM_ADD, nvmem);

	return nvmem;

err_remove_cells:
	nvmem_device_remove_all_cells(nvmem);
err_teardown_compat:
	if (config->compat)
		nvmem_sysfs_remove_compat(nvmem, config);
err_device_del:
	device_del(&nvmem->dev);
err_put_device:
	put_device(&nvmem->dev);

```

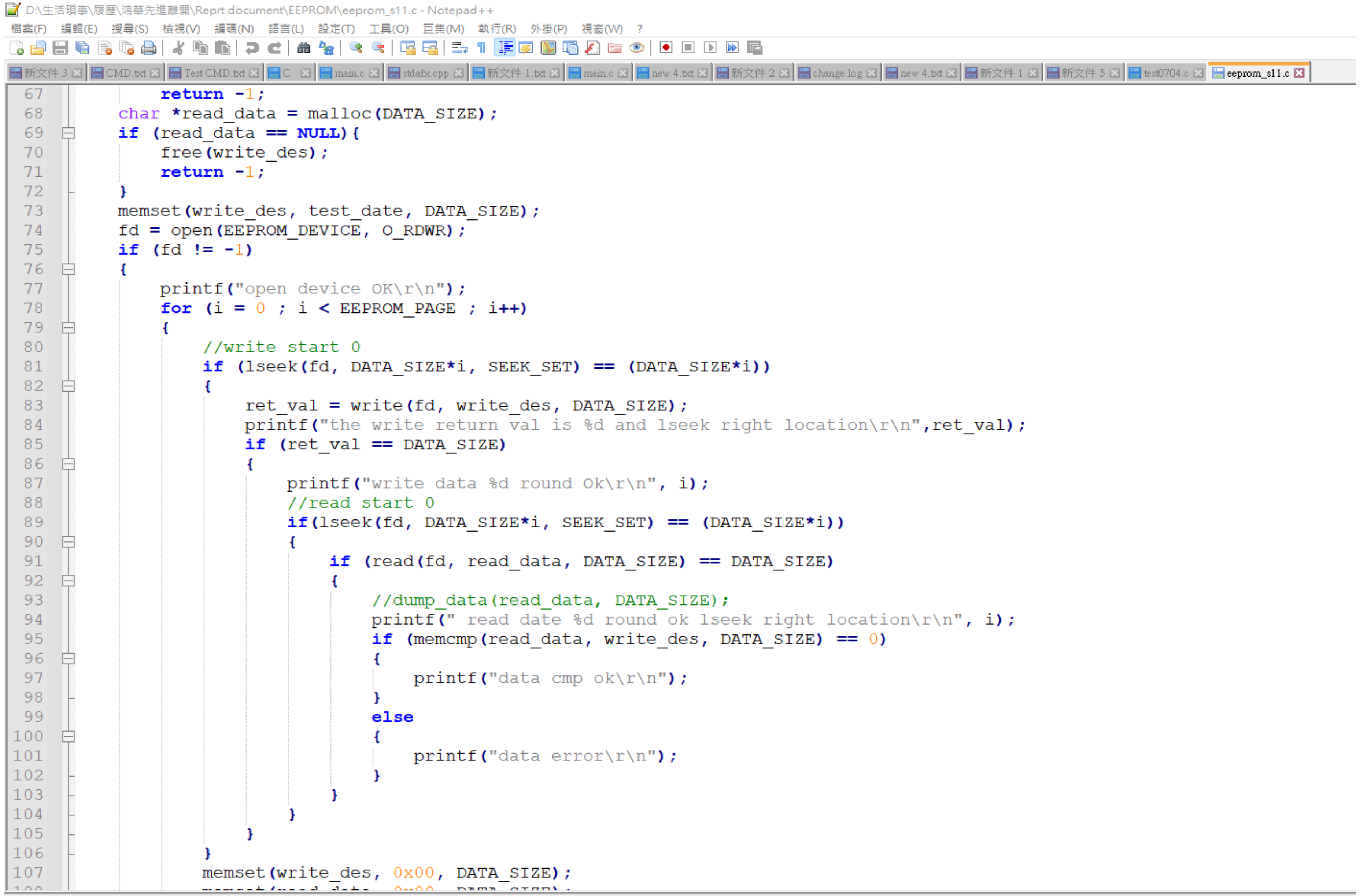
Bracket match found on line: 684



```
};

static struct bin_attribute bin_attr_nvmem_eeprom_compat = {
        .attr   = {
                .name   = "eeprom",
        },
        .read   = bin_attr_nvmem_read,
        .write  = bin_attr_nvmem_write,
};

/*
 * nvmem_setup_compat() - Create an additional binary entry in
 * drivers sys directory, to be backwards compatible with the older
 * drivers/misc/eeprom drivers.
 */
static int nvmem_sysfs_setup_compat(struct nvmem_device *nvmem,
                                    const struct nvmem_config *config)
{
        int rval;

        if (!config->compat)
                return 0;

        if (!config->base_dev)
                return -EINVAL;

        nvmem->eeprom = bin_attr_nvmem_eeprom_compat;
        nvmem->eeprom.attr.mode = nvmem_bin_attr_get_umode(nvmem);
        nvmem->eeprom.size = nvmem->size;
#ifdef CONFIG_DEBUG_LOCK_ALLOC
        nvmem->eeprom.attr.key = &eeprom_lock_key;
#endif
        nvmem->eeprom.private = &nvmem->dev;
        nvmem->base_dev = config->base_dev;

        rval = device_create_bin_file(nvmem->base_dev, &nvmem->eeprom);
        if (rval) {
                dev_err(&nvmem->dev,
                        "Failed to create eeprom binary file %d\n", rval);
                return rval;
        }

        nvmem->flags |= FLAG_COMPAT;

        return 0;
}

static void nvmem_sysfs_remove_compat(struct nvmem_device *nvmem,
                              const struct nvmem_config *config)
{
        if (config->compat)
                device_remove_bin_file(nvmem->base_dev, &nvmem->eeprom);
}

#else /* CONFIG_NVMEM_SYSFS */

static int nvmem_sysfs_setup_compat(struct nvmem_device *nvmem,
                                    const struct nvmem_config *config)
{
        return -ENOSYS;
```

Here is .

User Space test case


D:\生活瑣事\履歷\鴻華先進離開\Reprt document\EEPROM\eeprom_s11.c - Notepad++
檔案(F) 編輯(E) 搜尋(S) 檢視(V) 編碼(N) 語言(L) 設定(T) 工具(O) 巨集(M) 執行(R) 外掛(P) 視窗(W) ?
新文件 3
CMD.txt
Test CMD.txt
C
main.c
stdafx.cpp
新文件 1.txt
main.c
new 4.txt
新文件 2
change.log
new 4.txt
新文件 1
新文件 5
test0704.c
eeprom_s11.c

```c
        return -1;
    char *read_data = malloc(DATA_SIZE);
    if (read_data == NULL){
        free(write_des);
        return -1;
    }
    memset(write_des, test_date, DATA_SIZE);
    fd = open(EEPROM_DEVICE, O_RDWR);
    if (fd != -1)
    {
        printf("open device OK\r\n");
        for (i = 0 ; i < EEPROM_PAGE ; i++)
        {
            //write start 0
            if (lseek(fd, DATA_SIZE*i, SEEK_SET) == (DATA_SIZE*i))
            {
                ret_val = write(fd, write_des, DATA_SIZE);
                printf("the write return val is %d and lseek right location\r\n",ret_val);
                if (ret_val == DATA_SIZE)
                {
                    printf("write data %d round Ok\r\n", i);
                    //read start 0
                    if(lseek(fd, DATA_SIZE*i, SEEK_SET) == (DATA_SIZE*i))
                    {
                        if (read(fd, read_data, DATA_SIZE) == DATA_SIZE)
                        {
                            //dump_data(read_data, DATA_SIZE);
                            printf(" read date %d round ok lseek right location\r\n", i);
                            if (memcmp(read_data, write_des, DATA_SIZE) == 0)
                            {
                                printf("data cmp ok\r\n");
                            }
                            else
                            {
                                printf("data error\r\n");
                            }
                        }
                    }
                }
            }
            memset(write_des, 0x00, DATA_SIZE);
            memset(read_data, 0x00, DATA_SIZE);
```