Listing 1. Struktury i funkcje niezbędne do obsługi MCP4716

```
/*struktura przechowująca wskaźnik do klienta i2c oraz aktualną, 10-bitową
wartość na wyjściu przetwornika*/
struct mcp4716_data {
    struct i2c_client *client;
    u16 dac_value;
};

/*struktura zawierająca dane dotyczące pojedynczego kanału przetwornika
(typ obsługiwanej wartości, włączenie indeksowania kanałów, typ kanału
- wyjście, numer kanału, informacje specyficzne dla kanału - wartość
wyjściowa)*/
static const struct iio_chan_spec mcp4716_channel = {
    .type           = IIO_VOLTAGE,
    .indexed        = 1,
    .output         = 1,
    .channel        = 0,
    .info_mask_separate = BIT(IIO_CHAN_INFO_RAW),
};

/*funkcja wysyłająca przez i2c nową wartość wyjściową przetwornika (według
specyfikacji mcp4716, 10-bitowa wartość musi być podczas transmisji
przesunięta o dwa bity w lewo)*/
static int mcp4716_set_value(struct iio_dev *indio_dev, int val)
{
    struct mcp4716_data *data = iio_priv(indio_dev);
    u8 outbuf[2];
    int ret;

    if (val >= (1 << 10) || val < 0) return -EINVAL;
    val <<= 2;
    outbuf[0] = (val >> 8) & 0xf;
    outbuf[1] = val & 0xfc;
    ret = i2c_master_send(data->client, outbuf, 2);
    if (ret < 0)
        return ret;
    else if (ret != 2)
        return -EIO;
    else
        return 0;
}

/*funkcja obsługująca odczyt aktualnej wartości wyjściowej przetwornika
(jest ona zapisywana w sterowniku po każdym zapisie, dlatego nie ma
potrzeby odczytywania jej przez i2c)*/
static int mcp4716_read_raw(struct iio_dev *indio_dev,
                            struct iio_chan_spec const *chan,
                            int *val, int *val2, long mask)
{
    struct mcp4716_data *data = iio_priv(indio_dev);

    switch (mask) {
    case IIO_CHAN_INFO_RAW:
        *val = data->dac_value;
        return IIO_VAL_INT;
    }
    return -EINVAL;
}

/*funkcja obsługująca zapis nowej wartości wyjściowej przetwornika*/
static int mcp4716_write_raw(struct iio_dev *indio_dev,
```

```
                                struct iio_chan_spec const *chan,
                                int val, int val2, long mask)
{
    struct mcp4716_data *data = iio_priv(indio_dev);
    int ret;
    switch (mask) {
    case IIO_CHAN_INFO_RAW:
        ret = mcp4716_set_value(indio_dev, val);
        data->dac_value = val;
        break;
    default:
        ret = -EINVAL;
        break;
    }
    return ret;
}

/*struktura definiująca funkcjonalność sterownika (odczyt i zapis surowych
danych)*/
static const struct iio_info mcp4716_info = {
Listing 1. cd.
    .read_raw = mcp4716_read_raw,
    .write_raw = mcp4716_write_raw,
    .driver_module = THIS_MODULE,
};

/*funkcja rejestrująca sterownik w jądrze i odczytująca początkową wartość
na wyjściu przetwornika*/
static int mcp4716_probe(struct i2c_client *client,
                         const struct i2c_device_id *id)
{
    struct mcp4716_data *data;
    struct iio_dev *indio_dev;
    u8 inbuf[3];
    int err;

    indio_dev = iio_device_alloc(sizeof(*data));
    if (indio_dev == NULL) {
        err = -ENOMEM;
        goto exit;
    }
    data = iio_priv(indio_dev);
    i2c_set_clientdata(client, indio_dev);
    data->client = client;
    indio_dev->dev.parent = &client->dev;
    indio_dev->name = id->name;
    indio_dev->info = &mcp4716_info;
    indio_dev->channels = &mcp4716_channel;
    indio_dev->num_channels = 1;
    indio_dev->modes = INDIO_DIRECT_MODE;
    /* read current DAC value */
    err = i2c_master_recv(client, inbuf, 3);
    if (err < 0) {
        dev_err(&client->dev, „failed to read DAC value”);
        goto exit_free_device;
    }
    data->dac_value = (inbuf[1] << 2) | (inbuf[2] >> 6);
    err = iio_device_register(indio_dev);
    if (err) goto exit_free_device;
        dev_info(&client->dev, „MCP4716 DAC registered\n”);
        return 0;
```

```
exit_free_device:
    iio_device_free(indio_dev);
exit:
    return err;
}

/*funkcja usuwająca sterownik i zwalniająca jego zasoby*/
static int mcp4716_remove(struct i2c_client *client)
{
    struct iio_dev *indio_dev = i2c_get_clientdata(client);
    iio_device_unregister(indio_dev);
    iio_device_free(indio_dev);
    return 0;
}

/*struktura zawierająca nazwy wszystkich urządzeń sterownika podłączanych
do magistrali i2c (w tym przypadku jest to tylko jeden przetwornik)*/
static const struct i2c_device_id mcp4716_id[] = {
    { „mcp4716”, 0 },
    { }
};

/*makro tworzące nazwę modułu używaną przez jądro*/
MODULE_DEVICE_TABLE(i2c, mcp4716_id);

/*struktura reprezentująca sterownik urządzenia podłączonego do magistrali
i2c i zawierająca wskaźniki do funkcji rejestrującej i usuwającej sterownik
oraz jego nazwę*/
static struct i2c_driver mcp4716_driver = {
    .driver = {
        .name   = MCP4716_DRV_NAME,
    },
    .probe          = mcp4716_probe,
    .remove         = mcp4716_remove,
    .id_table       = mcp4716_id,
};

/*makro pomocnicze do rejestracji sterownika przez jądro*/
module_i2c_driver(mcp4716_driver);
```