```
// A. Sheaff 3/15/2019 - wwv driver
// Framework code for creating a kernel driver
// that creates the digital data from WWV station
// Pass in time/date data through ioctl.
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/device.h>
#include <linux/err.h>
#include <linux/fs.h>
#include <linux/spinlock.h>
#include <linux/delay.h>
#include <linux/list.h>
#include <linux/io.h>
#include <linux/ioctl.h>
#include <linux/uaccess.h>
#include <linux/irq.h>
#include <linux/interrupt.h>
#include <linux/slab.h>
#include <linux/gpio.h>
#include <linux/of_gpio.h>
#include <linux/platform_device.h>
#include <linux/pinctrl/consumer.h>
#include <linux/gpio/consumer.h>
#include <linux/jiffies.h>

#include "wwv.h"

// Data to be "passed" around to various functions
struct wwv_data_t {
    struct gpio_desc *gpio_wwv;     // Enable pin
    struct gpio_desc *gpio_unused17;        // Clock pin
    struct gpio_desc *gpio_unused18;        // Bit 0 pin
    struct gpio_desc *gpio_unused22;        // Bit 1 pin
    struct gpio_desc *gpio_shutdown;    // Shutdown input
    int major;          // Device major number
    struct class *wwv_class;    // Class for auto /dev population
    struct device *wwv_dev; // Device for auto /dev population
    // ADD YOUR LOCKING VARIABLE BELOW THIS LINE
    struct mutex lock;
};

// ADD ANY WWV DEFINE BELOW THIS LINE

// WWV data structure access between functions
static struct wwv_data_t *wwv_data_fops;

//************************************
// WWV Data format
//         0       1       2          3             4          5           6          7
        8       9
// +----+-------+-------+-------+--------------+-----------------------------------------
-----+-------+-------+
// |P0  |       |       |       |              |       YEAR Units Value BCD LSb First
    |       |       |
// |    |       |       |       |              +-----------+-----------+-----------+------
-----+       |       |
// |    |Blank  |Zero   |DST    |Leap Sec Warn  |1's Year   |2's Year   |4's Year   |8's Ye
ar  |Zero   |POS ID |
// +----+-------+-------+-------+--------------+-----------+-----------+-----------+------
-----+-------+-------+
//          10          11          12          13        14        15                16
         17              18      19
// +----+-------------------------------------------------+-------+-------------------------
---------------------+-------+-------+
// |P1  |    Minute Units Value BCD LSb First             |       |           Minute Tens Val
ue BCD LSb First    |       |       |
// |    +-----------+-----------+-----------+-----------+       +---------------+----------
-----+---------------+       |       |
```

```
// |    |1's Minute |2's Minute |4's Minute |8's Minute |Zero   |10's Minute    |20's Minut
e   |40's Minute    |Zero   |POS ID |
// +----+-----------+-----------+-----------+-----------+-------+---------------+----------
-----+---------------+-------+-------+
//          20          21          22         23         24          25           26
          27          28      29
// +----+-------------------------------------------------+-------+-------------------------
--------------------+-------+-------+
// |P2  |        Hour Units Value BCD LSb First           |       |           Hour Tens Value
 BCD LSb First      |       |       |
// |    +-----------+-----------+-----------+-----------+       +---------------+----------
-----+---------------+       |       |
// |    |1's Hour   |2's Hour   |4's Hour   |8's Hour   |Zero   |10's Hour      |20's Hour
    |40's Hour      |Zero   |POS ID |
// +----+-----------+-----------+-----------+-----------+-------+---------------+----------
-----+---------------+-------+-------+
//             30                  31                32                33
34          35                  36                37                38                39
// +----+-------------------------------------------------------------------------------+--
-----+-------------------------------------------------------------------------------+-----
--+
// |P3  |               Day of Year Units Value BCD LSb First                           |
    |                  Day of Year Tens Value BCD LSb First                          |
|
// +    +------------------+------------------+------------------+------------------+
     +------------------+------------------+------------------+------------------+
  +
// |    |1's Day of Year    |2's Day of Year    |4's Day of Year    |8's Day of Year    |Ze
ro  |10's Day of Year   |20's Day of Year   |40's Day of Year   |80's Day of Year   |POS ID
|
// +----+------------------+------------------+------------------+------------------+--
-----+------------------+------------------+------------------+------------------+-----
--+
//              40                  41                42      43       44     45
46       47     48      49
// +----+-------------------------------------------------+-------+-------+-------+-------+--
-----+-------+-------+-------+
// |P4  |    Day of Year Hundreds Value BCD LSb First     |       |       |       |       |
    |       |       |       |
// |    +-----------+-----------+-----------+-----------+       |       |       |       |
    |       |       |       |
// |    |100's Day of Year      |200's Day of Year      |Zero   |Zero   |Zero   |Blank  |Bl
ank |Blank  |Blank  |Blank  |
// +----+-----------------------+-----------------------+-------+-------+-------+-------+--
-----+-------+-------+-------+
//        50       51      52     53       54      55      56     57       58      59
// +----+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
// |P5  |Blank  |Blank  |Blank  |Blank  |Blank  |Blank  |Blank  |Blank  |Blank  |Blank  |
// +----+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+

// ADD YOUR WWV ENCODING/TRANSMITING/MANAGEMENT FUNCTIONS BELOW THIS LINE
static int encodebit(int width, struct wwv_data_t *wwv_dat)
{
    int i;                  // Iterator

    // 100Hz signal for width ms
    for(i = 0; i < width/10; i++) {
        gpiod_set_value(wwv_dat->gpio_wwv,1);
        usleep_range(5000,5010);
        gpiod_set_value(wwv_dat->gpio_wwv,0);
        usleep_range(5000,5010);

        // Extra time for accuracy
        if(i == (width/10 - 1)) {
            gpiod_set_value(wwv_dat->gpio_wwv,1);
            usleep_range(5000,5010);
        }
    }
```

```
    // zero for rest of second
    gpiod_set_value(wwv_dat->gpio_wwv,0);
    usleep_range((1000-width)*1000,(1000-width)*1000 + 10);
    return 0;
}

// Gets data and time and encodes it for wwv
static int encodedatetime(int *data, struct wwv_data_t *wwv_dat)
{
    int i;                  // Iterator
    int ret;                // Return Value
    int *array;             // Holds widths for wwv
    int yearones;           // Date and Time variables
    int minones;
    int mintens;
    int hourones;
    int hourtens;
    int doyones;
    int doytens;
    int doyhundreds;

    // Initializes to size of 60 ints
    array = (int *)kcalloc(61,sizeof(int),GFP_ATOMIC);
    if (array == NULL) {
        printk(KERN_INFO "Memory allocation failed!\n");
        ret = -ENOMEM;
        goto fail;
    }

    // Gets ones, tens and hundreds place for date and time
    yearones = (data[0] + 1900) % 10;
    minones = (data[3]) % 10;
    mintens = ((data[3]) % 100) / 10;
    hourones = (data[2]) % 10;
    hourtens = ((data[2]) % 100) / 10;
    doyones = (data[1]) % 10;
    doytens = ((data[1]) % 100) / 10;
    doyhundreds = ((data[1]) % 1000) / 100;

    // Places 1s, 2s, 4s, and 8s bit in wwv array
    for (i = 4; i < 8; i++) {
        array[i] = ((yearones & (0x01 << (i - 4))) ? 470 : 170);
        array[i + 6] = ((minones & (0x01 << (i - 4))) ? 470 : 170);
        array[i + 16] = ((hourones & (0x01 << (i - 4))) ? 470 : 170);
        array[i + 26] = ((doyones & (0x01 << (i - 4))) ? 470 : 170);
        array[i + 31] = ((doytens & (0x01 << (i - 4))) ? 470 : 170);
    }

    // Places 1s, 2s, and 4s bit in wwv array
    for (i = 15; i < 18; i++) {
        array[i] = ((mintens & (0x01 << (i - 15))) ? 470 : 170);
        array[i + 10] = ((hourtens & (0x01 << (i - 15))) ? 470 : 170);
    }

    // Places 1s, and 2s bit in wwv array
    for (i = 40; i < 42; i++) {
        array[i] = ((doyhundreds & (0x01 << (i - 40))) ? 470 : 170);
    }

    // Places P indentifiers in wwv array
    for (i = 1; i < 5; i++) {
        array[i * 10 - 1] = 770;
    }

    // Places zero bits in wwv array
    for (i = 0; i < 3; i++) {
        array[i + 1] = 170;
```

```
        array[i + 42] = 170;
        array[i * 10 + 8] = 170;
        array[i * 10 + 14] = 170;
    }

    // Encodes each element
    for(i = 0; i < 60; i++)
    {
        encodebit(array[i], wwv_dat);
    }

    // Clean Up
    kfree(array);
    array = NULL;

    return 0;

fail:

    return ret;
}

// Does checking that is described in comments above ioctl function
static int encodemanagement(struct file * filp, struct wwv_data_t *wwv_dat, unsigned long a
rg)
{
    int ret = 0;    // Return value
    int *kdata;     // Memory for data transfer
    int size = 16;  // Size of data transfer

    // Checks if another process is using pins and if it is opened with O_NONBLOCK return
    if ((mutex_is_locked(&(wwv_dat->lock))) && (filp->f_flags & O_NONBLOCK)) {
        printk(KERN_INFO "Lock taken and Nonblock!\n");
        kdata = NULL;
        ret = -EACCES;
        goto fail;
    }

    // Checks if another process is using pins. If so wait and if receive a signal handles
error.
    if(mutex_lock_interruptible(&(wwv_dat->lock))) {
        printk(KERN_INFO "Error getting lock!\n");
        kdata = NULL;
        ret = -EACCES;
        goto fail;
    }

    // Allocates memory for data
    kdata = (int *)kmalloc(size+1,GFP_ATOMIC);
    if (kdata == NULL) {
        printk(KERN_INFO "Memory allocation failed!\n");
        mutex_unlock(&(wwv_dat->lock));
        ret = -ENOMEM;
        goto fail;
    }

    // Gets datetime from userspace
    if(copy_from_user(kdata,(const void __user *)arg,size) != 0) {
        printk(KERN_INFO "Copying from userspace failed!\n");
        mutex_unlock(&(wwv_dat->lock));
        kfree(kdata);
        kdata = NULL;
        ret = -EFAULT;
        goto fail;
    }

    // Encodes data
    ret = encodedatetime(kdata, wwv_dat);
```

```
    if(ret != 0) {
        printk(KERN_INFO "Encoding Error!\n");
        mutex_unlock(&(wwv_dat->lock));
        kfree(kdata);
        kdata = NULL;
        goto fail;
    }

    // Unlocks and cleans up
    mutex_unlock(&(wwv_dat->lock));
    printk(KERN_INFO "Cleaning up!\n");
    kfree(kdata);
    kdata = NULL;

    return 0;

    fail:

        return ret;
}
// ioctl system call
// If another process is using the pins and the device was opened O_NONBLOCK
//   then return with the appropriate error
// Otherwise
//   If another process is using the pins
//     then block/wait for the pin to be free.  Clean up and return an error if a signal is
 received.
//   Otherwise
//     Copy the user space data using copy_from_user() to a local kernel space buffer that
you allocate
//     Encode to the copied data using your encoding system from homework 5 (your "wwv" cod
e) to another kernel buffer that you allocate
//     Toggle pins as in homework 04 gpio code.  While delaying, do not consume CPU resourc
es. *** SEE TIMERS-HOWTO.TXT IN THE KERNEL DOCUMENTATION ***
//       Use a variable for the clock high and low pulse widths that is shared with the ioc
tl class.  The
//  CLEAN UP (free all allocated memory and any other resouces you allocated) AND RETURN AP
PROPRAITE VALUE

// You will need to choose the type of locking yourself.  It may be atmonic variables, spin
locks, mutex, or semaphore.
static long wwv_ioctl(struct file * filp, unsigned int cmd, unsigned long arg)
{
    long ret = 0;               // Return value
    struct wwv_data_t *wwv_dat; // Driver data - has gpio pins

    // Get our driver data
    wwv_dat=(struct wwv_data_t *)filp->private_data;

    // IOCTL cmds
    switch (cmd) {
        case WWV_TRANSMIT:
            // PLACE A CALL TO YOUR FUNCTION AFTER THIS LINE
            ret = encodemanagement(filp, wwv_dat, arg);
            break;
        default:
            ret = -EINVAL;
            break;
    }

    // Clean up
    gpiod_set_value(wwv_dat->gpio_wwv,0);
    goto fail;

fail:

    return ret;
}
```

```
// Write system call
//   Just return 0
static ssize_t wwv_write(struct file *filp, const char __user * buf, size_t count, loff_t *
 offp)
{
    return 0;
}

// Open system call
// Open only if the file access flags (NOT permissions) are appropiate as discussed in clas
s
// Return an appropraite error otherwise
static int wwv_open(struct inode *inode, struct file *filp)
{
    // SUCESSFULLY THE FILE IF AND ONLY IF THE FILE FLAGS FOR ACCESS ARE APPROPRIATE
    // RETURN WITH APPROPRIATE ERROR OTHERWISE
    if ((filp->f_flags&O_ACCMODE)==O_RDONLY) return -EOPNOTSUPP;
    if ((filp->f_flags&O_ACCMODE)==O_RDWR) return -EOPNOTSUPP;

    filp->private_data=wwv_data_fops;  // My driver data (afsk_dat)

    return 0;
}

// Close system call
// What is there to do?
static int wwv_release(struct inode *inode, struct file *filp)
{
    return 0;
}

// File operations for the wwv device
static const struct file_operations wwv_fops = {
    .owner = THIS_MODULE,   // Us
    .open = wwv_open,       // Open
    .release = wwv_release,// Close
    .write = wwv_write, // Write
    .unlocked_ioctl=wwv_ioctl,  // ioctl
};

static struct gpio_desc *wwv_dt_obtain_pin(struct device *dev, struct device_node *parent,
char *name, int init_val)
{
    struct device_node *dn_child=NULL;  // DT child
    struct gpio_desc *gpiod_pin=NULL;   // GPIO Descriptor for setting value
    int ret=-1; // Return value
    int pin=-1; // Pin number
    char *label=NULL;   // DT Pin label

    // Find the child - release with of_node_put()
    dn_child=of_get_child_by_name(parent,name);
    if (dn_child==NULL) {
        printk(KERN_INFO "No child %s\n",name);
        gpiod_pin=NULL;
        goto fail;
    }

    // Get the child pin number - does not appear to need to be released
    pin=of_get_named_gpio(dn_child,"gpios",0);
    if (pin<0) {
        printk(KERN_INFO "no %s GPIOs\n",name);
        gpiod_pin=NULL;
        goto fail;
    }
    // Verify pin is OK
    if (!gpio_is_valid(pin)) {
        gpiod_pin=NULL;
```

```
        goto fail;
    }
    printk(KERN_INFO "Found %s pin %d\n",name,pin);

    // Get the of string tied to pin - Does not appear to need to be released
    ret=of_property_read_string(dn_child,"label",(const char **)&label);
    if (ret<0) {
        printk(KERN_INFO "Cannot find label\n");
        gpiod_pin=NULL;
        goto fail;
    }
    // Request the pin - release with devm_gpio_free() by pin number
    if (init_val>=0) {
        ret=devm_gpio_request_one(dev,pin,GPIOF_OUT_INIT_LOW,label);
        if (ret<0) {
            dev_err(dev,"Cannot get %s gpio pin\n",name);
            gpiod_pin=NULL;
            goto fail;
        }
    } else {
        ret=devm_gpio_request_one(dev,pin,GPIOF_IN,label);
        if (ret<0) {
            dev_err(dev,"Cannot get %s gpio pin\n",name);
            gpiod_pin=NULL;
            goto fail;
        }
    }

    // Get the gpiod pin struct
    gpiod_pin=gpio_to_desc(pin);
    if (gpiod_pin==NULL) {
        printk(KERN_INFO "Failed to acquire wwv gpio\n");
        gpiod_pin=NULL;
        goto fail;
    }

    // Make sure the pin is set correctly
    if (init_val>=0) gpiod_set_value(gpiod_pin,init_val);


    // Release the device node
    of_node_put(dn_child);

    return gpiod_pin;

fail:
    if (pin>=0) devm_gpio_free(dev,pin);
    if (dn_child) of_node_put(dn_child);

    return gpiod_pin;
}


// Sets device node permission on the /dev device special file
static char *wwv_devnode(struct device *dev, umode_t *mode)
{
    if (mode) *mode = 0666;
    return NULL;
}

// My data is going to go in either platform_data or driver_data
//  within &pdev->dev. (dev_set/get_drvdata)
// Called when the device is "found" - for us
// This is called on module load based on ".of_match_table" member
static int wwv_probe(struct platform_device *pdev)
{
    struct device *dev = &pdev->dev;    // Device associcated with platform
```

```
    struct wwv_data_t *wwv_dat;     // Data to be passed around the calls
    struct device_node *dn=NULL;

    int ret=-1; // Return value


    // Allocate device driver data and save
    wwv_dat=kmalloc(sizeof(struct wwv_data_t),GFP_ATOMIC);
    if (wwv_dat==NULL) {
        printk(KERN_INFO "Memory allocation failed\n");
        return -ENOMEM;
    }

    memset(wwv_dat,0,sizeof(struct wwv_data_t));

    dev_set_drvdata(dev,wwv_dat);

// Find my device node
    dn=of_find_node_by_name(NULL,"wwv");
    if (dn==NULL) {
        printk(KERN_INFO "Cannot find device\n");
        ret=-ENODEV;
        goto fail;
    }
    wwv_dat->gpio_wwv=wwv_dt_obtain_pin(dev,dn,"WWV",0);
    if (wwv_dat->gpio_wwv==NULL) goto fail;
    wwv_dat->gpio_unused17=wwv_dt_obtain_pin(dev,dn,"Unused17",0);
    if (wwv_dat->gpio_unused17==NULL) goto fail;
    wwv_dat->gpio_unused18=wwv_dt_obtain_pin(dev,dn,"Unused18",0);
    if (wwv_dat->gpio_unused18==NULL) goto fail;
    wwv_dat->gpio_unused22=wwv_dt_obtain_pin(dev,dn,"Unused22",0);
    if (wwv_dat->gpio_unused22==NULL) goto fail;
    wwv_dat->gpio_shutdown=wwv_dt_obtain_pin(dev,dn,"Shutdown",-1);
    if (wwv_dat->gpio_shutdown==NULL) goto fail;


    // Create the device - automagically assign a major number
    wwv_dat->major=register_chrdev(0,"wwv",&wwv_fops);
    if (wwv_dat->major<0) {
        printk(KERN_INFO "Failed to register character device\n");
        ret=wwv_dat->major;
        goto fail;
    }

    // Create a class instance
    wwv_dat->wwv_class=class_create(THIS_MODULE, "wwv_class");
    if (IS_ERR(wwv_dat->wwv_class)) {
        printk(KERN_INFO "Failed to create class\n");
        ret=PTR_ERR(wwv_dat->wwv_class);
        goto fail;
    }

    // Setup the device so the device special file is created with 0666 perms
    wwv_dat->wwv_class->devnode=wwv_devnode;
    wwv_dat->wwv_dev=device_create(wwv_dat->wwv_class,NULL,MKDEV(wwv_dat->major,0),(void *)
wwv_dat,"wwv");
    if (IS_ERR(wwv_dat->wwv_dev)) {
        printk(KERN_INFO "Failed to create device file\n");
        ret=PTR_ERR(wwv_dat->wwv_dev);
        goto fail;
    }

    wwv_data_fops=wwv_dat;

    mutex_init(&(wwv_dat->lock));

    printk(KERN_INFO "Registered\n");
    dev_info(dev, "Initialized");
```

```
    return 0;

fail:
    // Device cleanup
    if (wwv_dat->wwv_dev) device_destroy(wwv_dat->wwv_class,MKDEV(wwv_dat->major,0));
    // Class cleanup
    if (wwv_dat->wwv_class) class_destroy(wwv_dat->wwv_class);
    // char dev clean up
    if (wwv_dat->major) unregister_chrdev(wwv_dat->major,"wwv");

    if (wwv_dat->gpio_shutdown) devm_gpio_free(dev,desc_to_gpio(wwv_dat->gpio_shutdown));
    if (wwv_dat->gpio_unused22) devm_gpio_free(dev,desc_to_gpio(wwv_dat->gpio_unused22));
    if (wwv_dat->gpio_unused18) devm_gpio_free(dev,desc_to_gpio(wwv_dat->gpio_unused18));
    if (wwv_dat->gpio_unused17) devm_gpio_free(dev,desc_to_gpio(wwv_dat->gpio_unused17));
    if (wwv_dat->gpio_wwv) devm_gpio_free(dev,desc_to_gpio(wwv_dat->gpio_wwv));


    dev_set_drvdata(dev,NULL);
    kfree(wwv_dat);
    printk(KERN_INFO "WWV Failed\n");
    return ret;
}

// Called when the device is removed or the module is removed
static int wwv_remove(struct platform_device *pdev)
{
    struct device *dev = &pdev->dev;
    struct wwv_data_t *wwv_dat; // Data to be passed around the calls

    // Obtain the device driver data
    wwv_dat=dev_get_drvdata(dev);

    // Device cleanup
    device_destroy(wwv_dat->wwv_class,MKDEV(wwv_dat->major,0));
    // Class cleanup
    class_destroy(wwv_dat->wwv_class);
    // Remove char dev
    unregister_chrdev(wwv_dat->major,"wwv");

    // Free the gpio pins with devm_gpio_free() & gpiod_put()
    devm_gpio_free(dev,desc_to_gpio(wwv_dat->gpio_shutdown));
    devm_gpio_free(dev,desc_to_gpio(wwv_dat->gpio_unused22));
    devm_gpio_free(dev,desc_to_gpio(wwv_dat->gpio_unused18));
    devm_gpio_free(dev,desc_to_gpio(wwv_dat->gpio_unused17));
    devm_gpio_free(dev,desc_to_gpio(wwv_dat->gpio_wwv));
#if 0
    // not clear if these are allocated and need to be freed
    gpiod_put(wwv_dat->gpio_shutdown);
    gpiod_put(wwv_dat->gpio_unused22);
    gpiod_put(wwv_dat->gpio_unused18);
    gpiod_put(wwv_dat->gpio_unused17);
    gpiod_put(wwv_dat->gpio_wwv);
#endif

    // Free the device driver data
    dev_set_drvdata(dev,NULL);
    kfree(wwv_dat);

    printk(KERN_INFO "Removed\n");
    dev_info(dev, "GPIO mem driver removed - OK");

    return 0;
}

static const struct of_device_id wwv_of_match[] = {
    {.compatible = "brcm,bcm2835-wwv",},
    { /* sentinel */ },
};
```

```
MODULE_DEVICE_TABLE(of, wwv_of_match);

static struct platform_driver wwv_driver = {
    .probe = wwv_probe,
    .remove = wwv_remove,
    .driver = {
           .name = "bcm2835-wwv",
           .owner = THIS_MODULE,
           .of_match_table = wwv_of_match,
           },
};

module_platform_driver(wwv_driver);

MODULE_DESCRIPTION("WWV pin modulator");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("WWV");
//MODULE_ALIAS("platform:wwv-bcm2835");
```