

Nokia c110 driver

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-02/7629.html>

From: Kliment Yanev (*Kliment.Yanev_at_helsinki.fi*)

Date: 02/28/04

Date: Sat, 28 Feb 2004 14:23:46 +0200

To: linux-kernel@vger.kernel.org

-----BEGIN PGP SIGNED MESSAGE-----

Hash: SHA1

I was attempting to compile the nokia c110 linux driver, meant for 2.4 kernels, on 2.6.3. Unsurprisingly, it gave me some errors. I suspect that the modification to the source would not be too difficult, it just seems to use some deprecated functions. I would be very willing to fix it myself, however, I am not familiar with the kernel code at all. Here are the errors the compile gives. Do you think you could point me in the right direction? Below is the source of the file causing the errors as well.

gcc output starts here:

```
set -e; for d in src dctl; do make -C $d ; done
make[1]: Entering directory `/home/kliment/Desktop/nokia_c110/src'
gcc -O2 -Wall -Wstrict-prototypes -fomit-frame-pointer -pipe
-D__KERNEL__ -DMODULE -I./include -I/usr/src/linux/include -c dllc.c
In file included from /usr/src/linux/include/asm/processor.h:18,
~ from /usr/src/linux/include/asm/thread_info.h:13,
~ from /usr/src/linux/include/linux/thread_info.h:21,
~ from /usr/src/linux/include/linux/spinlock.h:12,
~ from /usr/src/linux/include/linux/capability.h:45,
~ from /usr/src/linux/include/linux/sched.h:7,
~ from /usr/src/linux/include/linux/module.h:10,
~ from nokia_info.h:42,
~ from dllc.c:29:
/usr/src/linux/include/asm/system.h: In function `__set_64bit_var':
/usr/src/linux/include/asm/system.h:193: warning: dereferencing
type-punned pointer will break strict-aliasing rules
/usr/src/linux/include/asm/system.h:193: warning: dereferencing
type-punned pointer will break strict-aliasing rules
In file included from /usr/src/linux/include/linux/irq.h:20,
~ from /usr/src/linux/include/asm/hardirq.h:6,
~ from /usr/src/linux/include/linux/interrupt.h:11,
~ from nokia_info.h:52,
~ from dllc.c:29:
```

Linux-Kernel: Nokia c110 driver

```
/usr/src/linux/include/asm/irq.h:16:25: irq_vectors.h: No such file or
directory
In file included from /usr/src/linux/include/asm/hardirq.h:6,
~ from /usr/src/linux/include/linux/interrupt.h:11,
~ from nokia_info.h:52,
~ from dllc.c:29:
/usr/src/linux/include/linux/irq.h: At top level:
/usr/src/linux/include/linux/irq.h:70: error: `NR_IRQS' undeclared here
(not in a function)
In file included from /usr/src/linux/include/linux/irq.h:72,
~ from /usr/src/linux/include/asm/hardirq.h:6,
~ from /usr/src/linux/include/linux/interrupt.h:11,
~ from nokia_info.h:52,
~ from dllc.c:29:
/usr/src/linux/include/asm/hw_irq.h:28: error: `NR_IRQ_VECTORS'
undeclared here (not in a function)
/usr/src/linux/include/asm/hw_irq.h:31: error: `NR_IRQS' undeclared here
(not in a function)
In file included from nokia_info.h:89,
~ from dllc.c:29:
nokia_priv.h:43:1: warning: "HZ" redefined
In file included from /usr/src/linux/include/linux/sched.h:4,
~ from /usr/src/linux/include/linux/module.h:10,
~ from nokia_info.h:42,
~ from dllc.c:29:
/usr/src/linux/include/asm/param.h:5:1: warning: this is the location of
the previous definition
dllc.c: In function `dllc_devopen':
dllc.c:156: warning: `MOD_INC_USE_COUNT' is deprecated (declared at
/usr/src/linux/include/linux/module.h:488)
dllc.c: In function `dllc_devstop':
dllc.c:169: warning: `MOD_DEC_USE_COUNT' is deprecated (declared at
/usr/src/linux/include/linux/module.h:500)
make[1]: *** [dllc.o] Error 1
make[1]: Leaving directory `/home/kliment/Desktop/nokia_c110/src'
make: *** [all] Error 2
```

gcc errors end here

dllc.c starts here:

```
/******
~ * The contents of this file are subject to the NOKOS License Version 1.0
~ * (the "License"); you may not use this file except in compliance with the
~ * License.
~ *
~ * Software distributed under the License is distributed on an "AS IS"
basis,
~ * WITHOUT WARRANTY OF ANY KIND, either express or implied. See the
License
~ * for the specific language governing rights and limitations under the
~ * License.
```

Linux-Kernel: Nokia c110 driver

```
~ *
~ * The Original Software is Nokia C110/C111 for Linux.
~ *
~ * Copyright (c) Nokia Mobile Phones 2001 and others. All Rights Reserved.
~ *
~ * Nokia is a registered trademark of Nokia Corporation.
~ * Other product and company names mentioned herein may be
~ * trademarks or tradenames of their respective owners.
~ *
~ * Contributor(s): _____
~ *
~
*****/

/******
~ *
~ * Define Linux netdevice above MAC
~ *
~
*****/

#include "nokia_info.h"

/* Static functions' declarations */
static int dllc_devinit (device_t *dev );
static int dllc_devopen (device_t *dev );
static int dllc_devstop (device_t *dev );
static int dllc_devhard_start_xmit (struct sk_buff *skb , device_t *dev );
static struct net_device_stats * dllc_devgetstats (device_t *dev );
static void dllc_devset_multicast_list (device_t *dev );
static int dllc_devdo_ioctl (device_t *dev , void *req , int cmd );
static INT dllc_put_to_queue (dllc_t *dllc, struct sk_buff *skb);
static struct sk_buff *dllc_get_from_queue (dllc_t *dllc);
static struct sk_buff *dllc_remove_from_queue (dllc_t *dllc);
static INT dllc_init_queue(dllc_t *dllc);
static void dllc_close_queue(dllc_t *dllc);
static void dllc_timer (unsigned long dllc);

#define QUEUE_SIZE 10 /* skb queue size */
#define QUEUE_MAX 7 /* when to tell kernel to stop sending */
#define QUEUE_LOW 5 /* when to tell kernel to start again sending */

spinlock_t tx_lock=SPIN_LOCK_UNLOCKED;

dllc_t *dllc_new(driver_priv_t *priv)
{
~ dllc_t *dllc;

~ if(!priv) return NULL;
```

Linux-Kernel: Nokia c110 driver

```
~ dllc = kmalloc(sizeof(dllc_t), GFP_KERNEL);
~ if(!dllc) goto dllc_new_fail;
~ memset(dllc,0,sizeof(dllc_t));

~ if (dllc_init_queue(dllc) == FALSE)
~ goto dllc_new_fail;

~ /* initialize timer */
~ dllc->timer.data = (unsigned long) dllc;
~ dllc->timer.function = dllc_timer;
~ init_timer(&dllc->timer);

~ /* TX packets in buffers */
~ dllc->tx_packets_in_queue = 0;
~ dllc->tx_packets_dropped = 0;
~ dllc->netif_stopped = 0;

~ /* You have llc. Now make the netdevice */

~ dllc->dev = kmalloc(sizeof(device_t), GFP_KERNEL);
~ if(!dllc->dev) goto dllc_new_fail;
~ memset(dllc->dev, 0, sizeof(device_t));

~ /* Link it all together */

~ dllc->priv = priv;
~ dllc->dev->priv = dllc;

~ /* callback pointers */

~ dllc->dev->init = dllc_devinit;
~ dllc->dev->open = dllc_devopen;
~ dllc->dev->stop = dllc_devstop;
~ dllc->dev->hard_start_xmit = dllc_devhard_start_xmit;
~ dllc->dev->get_stats = dllc_devgetstats;
~ dllc->dev->set_multicast_list = dllc_devset_multicast_list;
~ dllc->dev->do_ioctl = (void *)dllc_devdo_ioctl;

~ return dllc;

~ dllc_new_fail:

~ if(dllc) {
~ dllc_delete(dllc);
~ }
~ return NULL;
}

int dllc_register (dllc_t *dllc){

~ if (!dllc) return -1;
```

```

~ ether_setup(dllc->dev);

~ if(register_netdev(dllc->dev)) {
~ ERROR("Failed to register netdev.\n");
~ return -1;
~ }
~ DEBUG("Registered netdev '%s'.\n",dllc->dev->name);

~ return 0;
}

void dllc_delete(dllc_t *dllc)
{
~ if(!dllc) return;

~ DEBUG("dllc delete\n");

~ /* delete timer */
~ del_timer_sync (&dllc->timer);

~ if (dllc->dev) {
~ unregister_netdev(dllc->dev);
~ dfree(dllc->dev,sizeof(device_t));
~ dllc->dev = NULL;
~ }

~ dllc_close_queue (dllc);
~ dfree(dllc, sizeof(dllc_t));
}

/* Called by register_netdev */

static int dllc_devinit(device_t *dev)
{
~ return 0;
}

/* Called when ifconfig sets it up */

static int dllc_devopen(device_t *dev)
{
~ if (!dev) return -1;
~ /* set the flags in the device object */

~ netif_start_queue(dev);

~ MOD_INC_USE_COUNT;

~ return 0;
}

```

```

/* Called when ifconfig closes it */

static int dllc_devstop(device_t *dev)
{
~ if (!dev) return -1;

~ netif_stop_queue(dev);

~ MOD_DEC_USE_COUNT;

~ return 0;
}

/* Called when we get packet from the OS */

static int dllc_devhard_start_xmit(struct sk_buff *skb, device_t *dev)
{
~ dllc_t *llc;
~ INT result = FALSE;
~ UINT32 ref = 0;

~ if (!dev) return -1;
~ if (!skb) return -1;

~ llc = dev->priv;
~ if(!llc)
~ return 0;

~ if(!llc->priv || !llc->priv->mgr)
~ return 0;

~ if (!netif_running(dev))
~ return 0;

~ spin_lock_bh(&tx_lock);

~ dev->trans_start = jiffies;
~ if (llc->tx_packets_in_queue == 0) // send immediately if queue
if empty
~ result = dmgr_tx_frame( llc->priv->mgr,
~ &skb->data[0] , // dest
~ &skb->data[6] , // src
~ dt_get16(&skb->data[12]), // type
~ &skb->data[14], // data
~ ref, //reference;
~ (skb->len - 14)); // length
~ if (result == TRUE) //free skb if packet was sent
~ dev_kfree_skb(skb);
~ else {
~ /* Put packet to queue */
~ if ( dllc_put_to_queue (llc, skb) == FALSE ){

```

```

~ ERROR ("Packet queueing failed, dropping
packet...\n");
~ dev_kfree_skb(skb); //can't buffer packet, free skb
~ llc->stats.tx_errors++;
~ llc->tx_packets_dropped++;
~ }
~ else {
~ /* increment packet counter */
~ llc->tx_packets_in_queue++;
~ if ( (llc->tx_packets_in_queue > QUEUE_MAX) &&
(llc->netif_stopped == 0) ){
~ netif_stop_queue (dev);
~ llc->netif_stopped = 1;
~ }
~ mod_timer(&llc->timer, jiffies);
~ }
~ }

~ spin_unlock_bh(&tx_lock);

~ return 0;
}

```

```

#define MAX_RETRIES 5
#define MIN_TIMEOUT HZ/50

```

```

static void dllc_timer (unsigned long dllc_struct){
~ dllc_t *llc = (dllc_t*) dllc_struct;
~ struct sk_buff *skb;
~ INT result = 0;
~ UINT32 ref = 0;
~ INT count = 0;
~ UINT32 timeout;
~ static INT32 errors = 0;

~ spin_lock_bh(&tx_lock);
~ while ( (skb = dllc_get_from_queue (llc)) ) {
~ count++;
~ result = dmgr_tx_frame( llc->priv->mgr,
~ &skb->data[0] , // dest
~ &skb->data[6] , // src
~ dt_get16(&skb->data[12]), // type
~ &skb->data[14], // data
~ ref, //reference;
~ (skb->len - 14)); // length
~ if (result == TRUE){
~ errors = 0;
~ /* remove from queue and free skb */
~ skb = dllc_remove_from_queue (llc);
~ llc->tx_packets_in_queue--;
~ dev_kfree_skb(skb);

```

```

~ if (!llc->tx_packets_in_queue &&
llc->netif_stopped){
~ netif_wake_queue(llc->dev);
~ llc->netif_stopped = 0;
~ }
~ }
~ else {
~ if (!llc->netif_stopped){ // stop the netif if
error in sending
~ netif_stop_queue (llc->dev);
~ llc->netif_stopped = 1;
~ }

~ timeout = (errors > 0) ? (MIN_TIMEOUT * errors)
: MIN_TIMEOUT;
~ if (errors > MAX_RETRIES){
~ errors = 0;
~ skb = dllc_remove_from_queue (llc);
~ llc->tx_packets_in_queue--;
~ dev_kfree_skb(skb);
~ llc->stats.tx_errors++;
~ llc->tx_packets_dropped++;
~ }
~ else {
~ errors++;
~ }
~ mod_timer(&llc->timer, jiffies + timeout);
~ break;
~ }
~ }

~ if (count == 0){
~ DEBUG ("No data to send\n");
~ if (!llc->tx_packets_in_queue && llc->netif_stopped){
~ netif_wake_queue(llc->dev);
~ llc->netif_stopped = 0;
~ }
~ }

~ spin_unlock_bh(&tx_lock);

~ return ;
}

/* Called when tx completes to radio, should update stats */

void dllc_ontxcomplete( driver_priv_t *priv, UINT32 ref, UINT32 txresult)
{
~ dllc_t *dllc;

```

```

~ if (!priv) return;
~ dllc = priv->llc;
~ if (!dllc) return;

~ if (txresult != 0) {
~ dllc->stats.tx_errors++;
~ }
~ else {
~ dllc->stats.tx_packets++;
~ }
~ dllc_flush_queue(priv);
}

void dllc_flush_queue(driver_priv_t* priv)
{
~ dllc_t *dllc;

~ if (!priv) return;
~ dllc = priv->llc;
~ if (!dllc) return;

~ spin_lock_bh(&tx_lock);

~ if (!dllc->tx_packets_in_queue && dllc->netif_stopped){
~ netif_wake_queue(dllc->dev);
~ dllc->netif_stopped = 0;
~ }
~ /* if there is data to send */
~ if (dllc_get_from_queue (dllc))
~ mod_timer(&dllc->timer, jiffies);

~ spin_unlock_bh(&tx_lock);
}

/* Called by e.g ifconfig to get device stats */

static struct net_device_stats* dllc_devgetstats(device_t *dev)
{
~ dllc_t *dllc;

~ if (!dev) return NULL;
~ dllc = dev->priv;
~ if (!dllc) return NULL;

~ return &dllc->stats;
}

/* Set multicast list for netdev */
#define MAX_MULTI_ADDRESSES 15

```

Linux-Kernel: Nokia c110 driver

```

static void dllc_devset_multicast_list(device_t *dev)
{
~ dllc_t *dllc;
~ struct dev_mc_list *dmi;
~ int i, length = 0;
~ UINT8 mac_addrs[MAC_ADDR_SIZE*MAX_MULTI_ADDRESSES]; // 15 MAC
addresses
#ifdef D_DEBUG
~ UINT8 mac_buf[MAC_ADDR_SIZE+BUFEXTRA];
#endif

~ if (!dev){
~ DEBUG ("dev NULL\n");
~ return;
~ }

~ dmi=dev->mc_list;
~ dllc = dev->priv;
~ if(!dllc || !dllc->priv || !dmi){
~ DEBUG ("NULL parameters\n");
~ return;
~ }

~ if (dev->flags & IFF_PROMISC) {
~ /* Promiscuous mode not supported */
~ ERROR ("Promiscuous mode not supported\n");
~ }
~ if ((dev->flags & IFF_ALLMULTI) ||
(dev->mc_count>MAX_MULTI_ADDRESSES)){
~ DEBUG("%s: setting all multicast Rx mode (%02X, %d
addresses).\n", dev->name, dev->flags, dev->mc_count);
~ /* receive all multicasts, mc_count=1 and
mac_addrs==NULL is magic */
~ dhw_set_multicast_addresses(dllc->priv->hw, 1, NULL,
length);
~ return;
~ }
~ else if (dev->mc_count) {
~ DEBUG("%s: setting multicast Rx mode %02X to %d
addresses.\n", dev->name, dev->flags, dev->mc_count);
~ /* use HW filter for multicasts */
~ for (i=0;i<dev->mc_count;i++) {
~ DEBUG("Multicast MAC:
%s\n",dt_printmac(dmi->dmi_addr, mac_buf));
~ memcpy(mac_addrs+length, dmi->dmi_addr,
MAC_ADDR_SIZE);
~ dmi = dmi->next;
~ length = length + MAC_ADDR_SIZE;
~ }
~ length = dev->mc_count * MAC_ADDR_SIZE;
~ dhw_set_multicast_addresses(dllc->priv->hw,

```

Linux-Kernel: Nokia c110 driver

```
dev->mc_count, mac_addrs, length);
~ return;
~ }
~ else if (dev->mc_count == 0) {
~ DEBUG("%s: setting multicast disabled Rx mode (%02X, %d
addresses).\n", dev->name, dev->flags, dev->mc_count);
~ /* Normal mode */
~ dhwc_set_multicast_addresses(dllc->priv->hw, 0, 0, 0);
~ }
~ return;
}

/* Ioctl interface to user apps */

static int dllc_devdo_ioctl(device_t *dev, void *req, int cmd)
{
~ return 0;
}

/* Frame received from radio, and you should send it to system */

void dllc_rxframe(driver_priv_t *priv, MAC src, MAC dest, INT16 type,
UINT8 *frm, UINT32 len)
{
~ struct sk_buff *skb;
~ dllc_t *dllc;

~ if (!priv) return;
~ dllc = priv->llc;
~ if(!dllc) return;
~ if(!dllc->dev) return;

~ if ( !netif_running(dllc->dev) )
~ return;

~ skb = dev_alloc_skb(len + 14 + 2);
~ if (!skb) {
~ return;
~ }

~ dllc->dev->last_rx = jiffies;

~ skb->dev = dllc->dev;

~ skb->ip_summed = CHECKSUM_UNNECESSARY;

~ skb_put( skb, len+14);

~ memcpy( &skb->data[0] , dest , sizeof(MAC));
~ memcpy( &skb->data[6] , src , sizeof(MAC));
~ memcpy( &skb->data[12], &type, sizeof(INT16));
```

```

~ memcpy( &skb->data[14], frm , len);

~ skb->protocol = eth_type_trans( skb, dllc->dev);

~ dllc->stats.rx_packets++;

~ netif_rx(skb);
}

INT dllc_get_my_mac_addr(driver_priv_t* priv, char* mac){
~ if (!priv) return FALSE;
~ if (!priv->llc) return FALSE;
~ if (!priv->llc->dev) return FALSE;
~ if (!mac) return FALSE;
~ memcpy (mac, priv->llc->dev->dev_addr, MAC_ADDR_SIZE);
~ return TRUE;
}

void dllc_set_my_mac_addr(driver_priv_t* priv, char* mac){
~ if (!priv || !priv->llc || !priv->llc->dev || !mac){
~ ERROR ("dllc_set_my_mac_addr: NULL parameters\n");
~ return;
~ }
~ memcpy(priv->llc->dev->dev_addr, mac, MAC_ADDR_SIZE);
}

/* queue handling */

static buf_t* dllc_get_empty(dllc_t *dllc);
static void dllc_insert_empty(dllc_t *dllc, buf_t* bd);
static buf_t* dllc_get_tail (dllc_t *dllc);
static buf_t* dllc_remove_tail (dllc_t *dllc);
static void dllc_put_first (dllc_t *dllc, buf_t* bd);

static struct sk_buff *dllc_get_from_queue (dllc_t *dllc){
~ struct sk_buff *skb = NULL;
~ buf_t *bd = dllc_get_tail(dllc);
~ if (!bd)
~ return NULL;

~ skb = bd->skb;
~ return skb;
}

static struct sk_buff *dllc_remove_from_queue (dllc_t *dllc){
~ struct sk_buff *skb;
~ buf_t *bd = dllc_remove_tail(dllc);
~ if (!bd){
~ DEBUG ("bd NULL\n");
~ return NULL;
~ }
}

```

```

~ skb = bd->skb;
~ dllc_insert_empty (dllc, bd);
~ return skb;
}

```

```

static INT dllc_put_to_queue (dllc_t *dllc, struct sk_buff *skb){
~ buf_t *bd;
~ if (!skb)
~ return FALSE;

~ bd = dllc_get_empty(dllc);
~ if (!bd)
~ return FALSE;

~ bd->skb = skb;
~ dllc_put_first(dllc, bd);
~ return TRUE;
}

```

```

static INT dllc_init_queue(dllc_t *dllc){
~ int i;
~ buf_t* bd = NULL;

~ dllc->tail_queue = NULL;
~ dllc->head_queue = NULL;
~ dllc->head_empty = NULL;

~ for ( i=0; i < QUEUE_SIZE; i++){
~ bd = kmalloc (sizeof(buf_t), GFP_KERNEL); //should we
allocate a single memory reg?
~ if (!bd){
~ ERROR ("memory allocation failed\n");
~ return FALSE;
~ }
~ dllc_insert_empty (dllc, bd);
~ }
~ return TRUE;
}

```

```

static void dllc_close_queue(dllc_t *dllc){
~ buf_t* tmp;
~ struct sk_buff *skb;

~ while ( (skb = dllc_remove_from_queue(dllc)) ){
~ dev_kfree_skb(skb);
~ dllc->stats.tx_errors++;
~ dllc->tx_packets_dropped++;
~ }
~ while ( (tmp = dllc_get_empty(dllc)) ){
~ kfree (tmp);
~ }
}

```

```

}

/* get buffer from the tail of the queue */

static buf_t* dllc_get_tail (dllc_t *dllc){
~ if (!dllc->tail_queue){ /* empty list */
~ return NULL;
~ }
~ return dllc->tail_queue;
}

static buf_t* dllc_remove_tail (dllc_t *dllc){
~ buf_t* tmp;
~ if (!dllc->tail_queue){ /* empty list */
~ return NULL;
~ }
~ tmp = dllc->tail_queue;
~ dllc->tail_queue = dllc->tail_queue->prev;
~ if (dllc->tail_queue)
~ dllc->tail_queue->next = NULL;
~ else
~ dllc->head_queue = NULL;

~ return tmp;
}

/* put new buffer to head of the queue, according to the sortkey */

static void dllc_put_first (dllc_t *dllc, buf_t* bd){
~ bd->next = NULL;
~ bd->prev = NULL;
~ if (!dllc->head_queue){ /* empty list */
~ dllc->head_queue = bd;
~ dllc->tail_queue = bd;
~ return;
~ }
~ dllc->head_queue->prev = bd;
~ bd->next = dllc->head_queue;
~ dllc->head_queue = bd;

~ return;
}

static void dllc_insert_empty(dllc_t *dllc, buf_t* bd){
~ buf_t* tmp;
~ bd->next = NULL;
~ bd->prev = NULL;
~ bd->skb = NULL;

~ if (!dllc->head_empty){ /* empty list */
~ dllc->head_empty = bd;

```

Linux-Kernel: Nokia c110 driver

```
~ return;
~ }

~ tmp = dllc->head_empty;
~ while (tmp->next != NULL)
~ tmp = tmp->next;

~ tmp->next = bd;
}

static buf_t* dllc_get_empty(dllc_t *dllc){
~ buf_t* tmp;
~ if (!dllc->head_empty){ /* empty list */
~ return NULL;
~ }
~ tmp = dllc->head_empty;
~ dllc->head_empty = dllc->head_empty->next;

~ tmp->next = NULL;
~ tmp->prev = NULL;
~ tmp->skb = NULL;
~ return tmp;
}
```

-----BEGIN PGP SIGNATURE-----

Version: GnuPG v1.2.4 (GNU/Linux)

Comment: Using GnuPG with Mozilla - <http://enigmail.mozdev.org>

iD8DBQFAQIhRrPQTyNB9u9YRAo5kAJwOb/GOyFfZMPMAI+AXHSfyzaPiPACgifie
jIv7h518L3raCtbJu2ktCYM=
=4EaU

-----END PGP SIGNATURE-----

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>