BCM430x - 802.11 IOCTL Interface

6/10/03

Revision 1.7

16215 Alton Parkway • P.O. Box 57013 • Irvine, California 92619-7013 • Phone: 949-450-8700 • FAX: 949-450-8710

REVISION HISTORY

Revision	Date	Change Description
1.7	06/10/03	Updates to support Bulls features
1.6.5	03/27/03	Minor updates

Broadcom Corporation
16215 Alton Parkway
P.O. Box 57013
Irvine, California 92619-7013

INTRODUCTION

The device driver that supports the BCM430x 802.11 family of solutions provides mechanisms for advanced configuration above and beyond the normal 802.11 OS-specific configuration mechanisms. This advanced configuration is achieved by exporting a set of configuration I/O controls (IOCTLs) to query or set a number of different driver/chip operating parameters.

BROADCOM-SPECIFIC IOCTLS

In addition to the OIDs specified by the NDIS specification, we supply a complete set of OS-agnostic I/O control (IOCTL) objects. The IOCTLs listed in the table below exist in all builds of the device driver (NDIS, Linux and VxWorks.) Because of differences in the various calling mechanisms for each OS, the base where the IOCTLs start will vary depending on the particular OS in use.

Because of the Broadcom OneDriver_ model, a particular IOCTL may take different parameters depending on the underlying hardware. For instance, if the underlying hardware is an 802.11b chipset, then only the standard CCK rates can be specified. But if the hardware is 54g capable, then additional OFDM rates are allowed. Please read the description for each IOCTL to understand where it is applicable.

NDIS

All of the Broadcom-specific OIDs start at 0xFFE41420. The OIDs are accessed in the exact same manner that the NDIS-defined OIDs are.

Linux

For Linux, private IOCTLs must start at the Linux-defined location of SIOCDEVPRIVATE. Additionally, a device driver is allowed at most 16 private IOCTLs. Since the BCM4301 driver implements more IOCTLs than this, the IOCTLs listed below are encapsulated in a structure and passed to the driver using a single IOCTL command value at offset SIOCDEVPRIVATE. Please refer to the following code sample for an example of how to call Linux private IOCTLs.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <typedefs.h>
#include <error.h>
#include <rc.h>

static int
wl_ioctl(char *name, int ioctl, void *buf, int len)
{
        struct ifreq ifr;
        wl_ioctl_t ioc;
        int ret = 0;
        int s;

        /* open socket to kernel */
        if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
                perror("socket");
                exit(errno);
```

```
        }

        /* do it */
        ioc.cmd = ioctl;
        ioc.buf = buf;
        ioc.len = len;
        strncpy(ifr.ifr_name, name, IFNAMSIZ);
        ifr.ifr_data = (caddr_t) &ioc;
        if ((ret = ioctl(s, SIOCDEVPRIVATE, &ifr)) < 0)
                perror(ifr.ifr_name);

        /* cleanup */
        close(s);
        return ret;
}
```

VxWorks

For VxWorks, the Broadcom-specific IOCTLs start at 0x180. Please refer to your VxWorks documentation for methods of issuing IOCTLs to the network driver.

IOCTL Table

The parameter column in the IOCTL table below indicates the argument type of the IOCTL in the information buffer. Any Broadcom-specific structures are defined in the **Data Structures** section.

IOCTL (HEX)	IOCTL Name	Query	Set	Description	Parameter
BASE+0x02	WLC_UP		X	Enables the driver after a WLC_DOWN or WLC_OUT command.	VOID
BASE+0x03	WLC_DOWN		X	Takes the driver out of the UP state. The driver must not be in the UP state when enabling testing modes (WLC_EVM, WLC_FREQ_ACCURACY). But be aware theat the DOWN state resets all driver calibration states. To take the driver out of the UP state but keep any run-time calibration information, use the WLC_OUT IOCTL.	VOID
BASE+0x06	WLC_DUMP	X		Dumps driver version information and limited configuration information. If the buffer size is too small, then an error will be returned.	char *Buffer
BASE+0x0C	WLC_GET_RATE	X		Returns the current rate override setting in units of 500 Kb/s.	int* Rate
BASE+0x0D	WLC_SET_RATE		X	Sets the rate to be used for transmit. Units are in multiples of 500Kb/s.	int *Rate

				For automatic rate negotiation, set to –1. For 802.11b solutions, possible values are 2, 4, 11 and 22. For a 54g solution, possible values are 2, 4, 11, 12, 18, 22, 36, 48, 72, 96 and 108.	
BASE+0x0F	WLC_GET_FRAG	X		Returns the current fragmentation threshold.	int *FragThreshold
BASE+0x10	WLC_SET_FRAG		X	Sets the fragmentation threshold. Value will be adjusted to within the range acceptable for the medium.	int *FragThreshold
BASE+0x11	WLC_GET_RTS	X		Returns the current RTS threshold.	int *RTSThreshold
BASE+0x12	WLC_SET_RTS		X	Sets the RTS threshold.	int *RTSThreshold
BASE+0x13	WLC_GET_INFRA	X		Returns 1 if in infrastructure mode, 0 otherwise.	int *Infra
BASE+0x14	WLC_SET_INFRA		X	Pass 0 to disable infrastructure mode, 1 to enable it.	int *Infra
BASE+0x17	WLC_GET_BSSID	X		Will return a byte array containing the current BSSID. If the station is not currently associated, will return –1.	char BSSID[6]
BASE+0x18	WLC_SET_BSSID		X	Sets the BSSID.	char BSSID[6]
BASE+0x19	WLC_GET_SSID	X		Returns the current SSID. Please note that once an SSID has been set, the BSS will be created. It will be created with the rate, channel, etc. setting in their current state. So this IOCTL should be called only after all other parameters have been set.	wlc_ssid_t *SSID
BASE+0x1A	WLC_SET_SSID		X	Sets the SSID.	wlc_ssid_t *SSID
BASE+0x1D	WLC_GET_CHANNEL	X		Returns pointer to a channel_info_t structure.	channel_info_t *Info
BASE+0x1E	WLC_SET_CHANNEL		X	Sets the default BSS channel. For 802.11b and 54g solutions, valid channels are generally 1-14. Please note that the exact valid channel set is dictated by the current locale setting.	int *Channel
BASE+0x1F	WLC_GET_SRL	X		Returns the short retry limit.	int *SRL
BASE+0x20	WLC_SET_SRL		X	Sets the long short limit. Valid values are 1-255.	int *SRL
BASE+0x21	WLC_GET_LRL	X		Returns the long retry limit.	int *LRL
BASE+0x22	WLC_SET_LRL		X	Sets the long retry limit. Valid values are 1-255.	int *LRL
BASE+0x23	WLC_GET_PLCPHDR	X		Returns the current setting for the PLCP	int *HeaderMode

				header mode. Refer to the PLCP Header Modes table below.	
BASE+0x24	WLC_SET_PLCPHDR		X	Sets the PLCP header mode. Refer to the PLCP Header Modes table below.	int *HeaderMode
BASE+0x25	WLC_GET_RADIO	X		Returns the radio state Bit 0 indicates that the radio is disabled via SW. Bit 1 indicates that the radio is disabled in HW.	int *Radio
BASE+0x26	WLC_SET_RADIO		X	Sets the SW radio state (0 for enabled or 1 for disabled.)	int *Radio
BASE+0x2A	WLC_GET_WEP	X		Returns whether WEP encryption is enabled. 0: WEP disabled. 1: WEP enabled	int *WEP
BASE+0x2B	WLC_SET_WEP		X	Sets whether WEP is to be used or not. 0: Do not use WEP. 1: Use WEP	int *WEP
BASE+0x32	WLC_SCAN		X	Initiates a scan across all channels. If no SSID is specified, then the scan will return all APs in range. If an SSID is specified, then the scan will return results for that SSID only. Call WLC_SCAN_RESULTS to get the results of the scan.	wlc_ssid_t *SSID
BASE+0x33	WLC_SCANRESULTS	X		Returns the results of the most recent scan in the passed structure.	wl_scan_results_t *Scan
BASE+0x3C	WLC_EVM		X	Sets EVM mode on the specified channel. To disable EVM mode, set to the channel in the wlc_evm_t structure to 0. Before EVM mode is entered, the driver must be “downed” by using the WLC_DOWN ioctl. After EVM testing is complete, use the WLC_UP ioctl to return the driver to normal operation. Please not that the EVM mode is only valid for 802.11b CCK rates. EVM will not work on OFDM rates.	wlc_evm_t *EVM
BASE+0x41	WLC_GET_TXPWR	X		Returns the current transmit power level in milliwatts.	int *TxPower
BASE+0x42	WLC_SET_TXPWR		X	Sets the transmit power level in milliwatts. The output power must comply with the limits based on the current locale. To override this,	int *TxPower

				the high-order bit must be set.	
BASE+0x45	WLC_GET_MACLIST	X		Returns the list of MAC addresses used for the accept/deny association list.	maclist *addresses
BASE+0x46	WLC_SET_MACLIST		X	Sets the list of MAC addresses used for the accept/deny association list.	maclist *addresses
BASE+0x47	WLC_GET_RATESET	X		Returns the default rate set for the wireless interface. Each individual rate is in 500Kb/s units. Also, if the most significant bit of the value is set, then the rate is included in the basic rate set.	rateset *rates
BASE+0x48	WLC_SET_RATESET		X	Sets the default rate set for the wireless interface. The default rate set controls which rates the wireless interface will communicate. Additionally, each rate can be set as a basic rate by OR'ing in the most significant bit of the value. Please see the section "54g™ Mode Settings" below for more information.	rateset *rates
BASE+0x49	WLC_GET_LOCALE	X		Returns the current locale in use by the driver.	wlc_locale_t *Locale
BASE+0x4A	WLC_SET_LOCALE		X	Sets the locale to be used by the driver. Before using this OID, a WLC_DOWN OID should be issued. And a WLC_UP OID should be issues once the locale has been set.	wlc_locale_t *Locale
BASE+0x4F	WLC_GET_SROM	X		Returns the contents of the SROM at the specified offset.	srom_rw_t *Srom
BASE+0x50	WLC_SET_SROM		X	Sets the contents of the SROM at the specified offset to the values passed in the srom_rw_t structure.	srom_rw_t *Srom
BASE+0x5C	WLC_FREQ_ACCURACY		X	Sets the board to transmit a continuous wave, single-tone carrier frequency. This is used to measure center channel frequency accuracy.	VOID
BASE+0x5D	WLC_CARRIER_SUPPRESS		X	Sets the board into a mode where a signal is generated in order to measure the RF carrier suppression.	VOID

BASE+0x69	WLC_GET_MAC_DENY	X		Indicates whether the mode of the MAC list. 0: MAC list is an accept list. 1: MAC list is a deny list.	int *Mode
BASE+0x6A	WLC_SET_MAC_DENY		X	Sets the mode of the MAC list.	int *Mode
BASE+0x75	WLC_GET_AP	X		Returns whether the driver is configured for AP mode. Note that the driver must have been compiled with both STA and AP options for this to be accurate. 0: driver in STA mode 1: driver in AP mode	int *Mode
BASE+0x76	WLC_SET_AP		X	Set the driver into either the STA operational mode or the AP operation mode.	int *Mode
BASE+0x77	WLC_GET_EAP_RESTRICT	X		Returns whether the EAP restrict mode is enabled or disabled. This mode must be enabled for proper 802.1x operation. Returns 0 if EAP restrict mode is disabled or 1 if it is enabled	int *Mode
BASE+0x78	WLC_SET_EAP_RESTRICT		X	Sets the EAP restrict mode. 1: EAP restricts is enabled, enabling 802.1x authentication. 2: EAP restrict mode is disabled	int *Mode
BASE+0x7B	WLC_GET_WDSLIST	X		Returns a structure containing all of the current WDS partners.	maclist *Partners
BASE+0x7C	WLC_SET_WDSLIST		X	Sets the list of WDS partners.	maclist *Partners
BASE+0x85	WLC_GET_WSEC	X		Returns a value containing a bit vector representing which wireless security modes are currently enabled. The vector is defined below: Bit 0: WEP_ENABLED Bit 1: TKIP_ENABLED Bit 2: AES_ENABLED	int *Vector
BASE+0x86	WLC_SET_WSEC		X	Sets the wireless security mode to be used.	int *Vector
BASE+0x8A	WLC_GET_LAZYWDS	X		Returns the value of the lazy WDS setting. 0: Lazy WDS is disabled. WDS partners must be set explicitly. 1: Lazy WDS is enabled and the AP will accept WDS partners from any MAC address	int *Mode
BASE+0x8B	WLC_SET_LAZYWDS		X	Sets the lazy WDS setting.	int *Mode
BASE+0x8C	WLC_GET_BANDLIST	X		Returns a structure detailing the number of radio bands supported. The values for the	Bandlist *Bands;

				supported band are: 1: WLC_BAND_A 2: WLC_BAND_B	
BASE+0x8D	WLC_GET_BAND	X		Returns the radio band in use: 0: Automatically choose radio band 1: Use only 802.11a band 2: Use only 802.11b band	int *Band
BASE+0x8E	WLC_SET_BAND		X	Sets the radio band to be used.	int *Band
BASE+0x90	WLC_GET_SHORTSLOT	X		Returns whether 54g short slot timing is currently being used. 0: short slot timing is NOT in use 1: short slot timing is in use	int *Mode
BASE+0x91	WLC_GET_SHORTSLOT_OVERRIDE	X		Returns the 54g short slot mode. -1: Automatic mode. Short slot timing is used by default as long as no non-short slot capable STAs are associated to the AP. 0: Short slot disabled. Only long slot timing will be used and advertised. 1: Short slot enabled. Only short slot timing will be used and advertised. 802.11b and other solutions not providing short slot timing will not be able to associate.	int *Mode
BASE+0x92	WLC_SET_SHORTSLOT_OVERRIDE		X	Sets the 54g short slot mode. Please see the section “54g™ Mode Settings” below for more information.	int *Mode
BASE+0x93	WLC_GET_SHORTSLOT_RESTRICT	X		Returns whether the AP is accepting short slot-only or short/long slot capable STAs. 0: Accept both short and long slot capable STAs 1: Accept only short slot capable STAs	
BASE+0x94	WLC_SET_SHORTSLOT_RESTRICT		X	Sets the short slot restriction mode. Please see the section “54g™ Mode Settings” below for more information.	int *Mode

BASE+0x95	WLC_GET_GMODE_PROTECTION	X		Used to determine whether protection mechanisms are *currently* being used. 0: protection mechanisms are not currently being used 1: protection mechanisms are currently being used	int *Enabled
BASE+0x96	WLC_GET_GMODE_PROTECTION_OVERRIDE	X		Returns the current setting of the gmode protection mode. 0: protection mechanisms will NEVER be used 1: protection mechanisms will ALWAYS be used -1: protection mechanism auto mode. Protection will be used if either a) an 11b node joins the BSS or b) the AP detects an 11b BSS on the same channel	int *Mode
BASE+0x97	WLC_SET_GMODE_PROTECTION_OVERRIDE		X	Sets the gmode protection mode, as described in the previous IOCTL	int *Mode
BASE+0x9B	WLC_GET_IGNORE_BEACONS	X		Returns whether the AP will ignore beacons from other BSS's on the same channel. If ignore beacons is disabled and the AP sees another BSS on the same channel, it will drop to long slot timing if the other BSS only supports long slot timing. Per the 802.11g draft specification, an 802.11g AP can ignore these. 0: Beacons will not be ignored. 1: Beacons will be ignored	int *Mode
BASE+0x9C	WLC_SET_IGNORE_BEACONS		X	Sets the ignore beacon mode.	int *Mode
BASE+0x9F	WLC_GET_ASSOCLIST	X		When in AP mode, returns a structure containing the list of MAC addresses of associated clients to the AP.	maclist *AssocList;
BASE+0xA2	WLC_GET_UP	X		Returns the operating state of the driver. 0: driver is in the DOWN or OUT state 1: driver is in the UP state	int *State
BASE+0xA3	WLC_OUT		X	Sets the driver in the OUT state. This	VOID

				removes the driver from the UP state but preserves all driver calibration settings.	
BASE+0xA4	WLC_GET_WPA_AUTH	X		Returns the WPA authentication mode. 0: None 1: 802.1x 2: PSK 255: Disabled	int *Mode
BASE+0xA5	WLC_SET_WPA_AUTH		X	Sets the WPA authentication mode.	int *Mode
BASE+0xB2	WLC_GET_GMODE_PROTECTION_CONTROL	X		Returns the scope of gmode protection. 0: off 1: local BSS only 2: overlapping BSS	int *Mode
BASE+0xB3	WLC_SET_GMODE_PROTECTION_CONTROL		X	Sets the scope of the gmode protection.	int *Mode
BASE+0xC6	WLC_GET_GMODE_PROTECTION_CTS	X		Returns whether the gmode protection mechanism is CTS-to-self. 0: use RTS/CTS 1: use CTS-to-self	int *Mode
BASE+0xC7	WLC_SET_GMODE_PROTECTION_CTS		X	Sets whether to use CTS-to-self gmode protection.	int *Mode
BASE+0xDA	WLC_GET_FRAMEBURST	X		Returns whether frame bursting is being used or not. 0: do not use frame bursting 1: use frame bursting	int *Mode
BASE+0xDB	WLC_SET_FRAMEBURST		X	Sets whether to use frame bursting or not.	int *Mode

Data Structures

```
typedef struct channel_info {
        int hw_channel;         /* The channel that the radio */
                                /* currently set to.          */

        int target_channel;     /* The channel of the current */
                                /* (I)BSS.                    */

        int scan_channel;       /* If a scan is in progress,  */
                                /* the channel that is cur-   */
                                /* rently being scanned.      */
} channel_info_t;

typedef enum _wlc_locale {
      WLC_WW = 0,               /* WorldWide locale */
      WLC_THA,                  /* Thailand */
      WLC_ISR,                  /* Israel */
      WLC_JDN,                  /* Jordan */
      WLC_PRC,                  /* China */
      WLC_JPN,                  /* Japan */
      WLC_FCC,                  /* USA */
} wlc_locale_t;
```

```
typedef struct wlc_ssid {
        uint32          SSID_len;
        uchar           SSID[32];
} wlc_ssid_t;

typedef struct wlc_evm {
      uint32            Channel;
      uint32            Rate;       /* In 500Kb/s increments */
} wlc_evm_t;

typedef struct srom_rw {
        uint   byteoff;             /* byte offset */
        uint   nbytes;              /* number of bytes */
        uint16 buf[];
} srom_rw_t;

struct maclist {
        uint count;                 /* number of MAC addresses */
        struct ether_addr ea[1];    /* var length array of addrs */
};

struct rateset {
      uint  count;                  /* Number of rates in the set */
      uint8 rates[12];              /* The rates to be supported in */
                                    /* 500Kb/s increments */
};

struct bandlist {
      uint  count;                  /* Number of supported bands */
      uint  bands[];                /* The bands supported by the card */
};


typedef struct wl_bss_info {
      uint32  version;              /* version field */
      uint32  length;               /* byte length of data in this */
                                    /* record, starting at version and */
                                    /* including IEs */
      struct ether_addr BSSID;
      uint16 beacon_period;         /* units are Kusec */
      uint16 capability;            /* Capability information */
      uint8  SSID_len;
      uint8  SSID[32];
      struct {
            uint    count;          /* # rates in this set */
            uint8   rates[16];      /* rates in 500kbps units w/hi bit */
                                    /* set if basic */
      } rateset;                    /* supported rates */
      uint8   channel;              /* Channel no. */
      uint16  atim_window;          /* units are Kusec */
      uint8   dtim_period;          /* DTIM period */
      int16   RSSI;                 /* receive signal strength (in dBm) */
      int8    phy_noise;            /* noise (in dBm) */
      uint32  ie_length;            /* byte length of Information Elements */
} wl_bss_info_t;
```

```
typedef struct wl_scan_results {
      uint32 buflen;
      uint32 version;
      uint32 count;
      wl_bss_info_t bss_info[1];
} wl_scan_results_t;
```