

# ON Semiconductor

## Is Now

# onsemi™

To learn more about onsemi™, please visit our website at  
[www.onsemi.com](http://www.onsemi.com)

---

**onsemi** and **onsemi** and other names, marks, and brands are registered and/or common law trademarks of Semiconductor Components Industries, LLC dba "**onsemi**" or its affiliates and/or subsidiaries in the United States and/or other countries. **onsemi** owns the rights to a number of patents, trademarks, copyrights, trade secrets, and other intellectual property. A listing of **onsemi** product/patent coverage may be accessed at [www.onsemi.com/site/pdf/Patent-Marking.pdf](http://www.onsemi.com/site/pdf/Patent-Marking.pdf). **onsemi** reserves the right to make changes at any time to any products or information herein, without notice. The information herein is provided "as-is" and **onsemi** makes no warranty, representation or guarantee regarding the accuracy of the information, product features, availability, functionality, or suitability of its products for any particular purpose, nor does **onsemi** assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation special, consequential or incidental damages. Buyer is responsible for its products and applications using **onsemi** products, including compliance with all laws, regulations and safety requirements or standards, regardless of any support or applications information provided by **onsemi**. "Typical" parameters which may be provided in **onsemi** data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. **onsemi** does not convey any license under any of its intellectual property rights nor the rights of others. **onsemi** products are not designed, intended, or authorized for use as a critical component in life support systems or any FDA Class 3 medical devices or medical devices with a same or similar classification in a foreign jurisdiction or any devices intended for implantation in the human body. Should Buyer purchase or use **onsemi** products for any such unintended or unauthorized application, Buyer shall indemnify and hold **onsemi** and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that **onsemi** was negligent regarding the design or manufacture of the part. **onsemi** is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner. Other names and brands may be claimed as the property of others.



**ON Semiconductor®**

[www.onsemi.com](http://www.onsemi.com)

---

## AX5043 AXRadio V2 API Software Manual

### Introduction:

This document describes AXRadio. AXRadio is intended to be an easy to use “driver” for AXSEM Radio Chips, such as the AX5043 and AX5051, allowing the user to focus on his product functionality and not the details of the radio link.

The API consists of a header file, axradio.h, providing functions for transmitting and receiving packets, and switching the radio into different modes.

Most of the transceiver configuration is computed by AXRadioLab and stored in non-volatile memory; it will not be changeable by the firmware. These items include:

- Radio PHY configuration (except for channel)
- Radio MAC (Frame Format) configuration (except actual addresses)

Some items should be runtime configurable. Items include:

- Channel Number
- Actual MAC Addresses

### PREREQUISITIES

AXRadio V2 uses the Wakeup Timer facilities of LibMF (libmfwtimer.h). These facilities are a superset of the timing functions of AXRadio V1, but are much more flexible; they provide:

- 2 Timers, a slow but persistently running timer for events that need to survive sleep, and a fast but switched off during sleep timer for short intervals
- Multiple events per timer are supported
- Not only usable for AXRadio – also supports user events

- Mechanism for scheduling callbacks from interrupt context which run in main context
- Less overhead (assembly implementation where C compiler generated code was too inefficient)

In order for the LibMF Wakeup Timer to work properly, the user should not use `enter_sleep` and `enter_standby` directly, but should use `wtimer_idle` instead. Also, during long running computations, `wtimer_idle` should be called periodically to allow pending callbacks to run.

CONSTANTS

Operation Modes

Table 1.

| Mode                          | Description   |
|-------------------------------|---|
| AXRADIO_MODE_OFF              | The radio is off, but in a configured state that allows it to be switched quickly to any other mode   |
| AXRADIO_MODE_DEEPSLEEP        | The radio is in deep sleep mode; this mode has the lowest possible power consumption, but may take some time to leave   |
| AXRADIO_MODE_CW_TRANSMIT      | The radio is setup for CW (constant carrier wave) transmit (used for basic tests)   |
| AXRADIO_MODE_ASYNC_TRANSMIT   | The radio is set up for asynchronous transmission. After a call to <code>axradio_transmit</code> , the transmitter is switched on and the packet is sent, afterwards the transmitter is switched off again.   |
| AXRADIO_MODE_WOR_TRANSMIT     | Same as <code>AXRADIO_MODE_ASYNC_TRANSMIT</code> , but starts transmission with a wakeup preamble   |
| AXRADIO_MODE_ACK_TRANSMIT     | The radio is set up for asynchronous transmission. After a call to <code>axradio_transmit</code> , the transmitter is switched on and the packet is sent, afterwards the radio waits for an acknowledge from the slave. If no acknowledge is received within a configured timespan, the packet is (optionally) retransmitted. |
| AXRADIO_MODE_WOR_ACK_TRANSMIT | Same as <code>AXRADIO_MODE_ACK_TRANSMIT</code> , but starts transmission with a wakeup preamble   |
| AXRADIO_MODE_ASYNC_RECEIVE    | The radio is setup for continuous asynchronous receive.   |
| AXRADIO_MODE_WOR_RECEIVE      | The radio is setup for wake-on-radio receive.   |

|                                    |   |
|------------------------------------|---|
| AXRADIO_MODE_ACK_RECEIVE           | The radio is setup for continuous asynchronous receive. After a packet is received, an acknowledge packet is sent back.   |
| AXRADIO_MODE_WOR_ACK_RECEIVE       | The radio is setup for wake-on-radio receive. After a packet is received, an acknowledge packet is sent back.   |
| AXRADIO_MODE_STREAM_TRANSMIT       | The radio is setup for streaming transmit (used for basic tests)  |
| AXRADIO_MODE_STREAM_TRANSMIT_UNENC | The radio is setup for unencoded streaming transmit (used for basic tests)  |
| AXRADIO_MODE_STREAM_TRANSMIT_SCRAM | The radio is setup for scrambled streaming transmit (used for basic tests)  |
| AXRADIO_MODE_STREAM_RECEIVE        | The radio is setup for streaming receive (used for basic tests)   |
| AXRADIO_MODE_STREAM_RECEIVE_UNENC  | The radio is setup for unencoded streaming receive (used for basic tests)   |
| AXRADIO_MODE_STREAM_RECEIVE_SCRAM  | The radio is setup for scrambled streaming receive (used for basic tests)   |
| AXRADIO_MODE_SYNC_MASTER           | The radio is setup as synchronous master.   |
| AXRADIO_MODE_SYNC_ACK_MASTER       | The radio is setup as synchronous master. The master expects an acknowledge from the slave, and reports the acknowledge or the absence of an acknowledge to the caller. |
| AXRADIO_MODE_SYNC_SLAVE            | The radio is setup as synchronous slave.  |
| AXRADIO_MODE_SYNC_ACK_SLAVE        | The radio is setup as synchronous slave. The slave sends an acknowledge whenever a packet is received.  |

**Error Codes**

**Table 2.**

| Error Code                 | Description   |
|----------------------------|---|
| AXRADIO_ERR_NOERROR        | No error occurred, operation completed successfully                         |
| AXRADIO_ERR_NOTSUPPORTED   | The operation is not supported  |
| AXRADIO_ERR_BUSY           | The operation could not be completed because the radio was busy             |
| AXRADIO_ERR_TIMEOUT        | The operation timed out (eg. The maximum number of retransmission exceeded) |
| AXRADIO_ERR_INVALID        | The operation failed because of an invalid parameter.                       |
| AXRADIO_ERR_NOCHIP         | No radio chip was found   |
| AXRADIO_ERR_RANGING        | The frequency could not be ranged.  |
| AXRADIO_ERR_LOCKLOST       | PLL lock was lost   |
| AXRADIO_ERR_RETRANSMISSION | This packet is a retransmission (due to no acknowledge received)            |
| AXRADIO_ERR_RESYNC         | The synchronous slave restarts resynchronization                            |
| AXRADIO_ERR_RESYNCTIMEOUT  | The synchronous slave restarts resynchronization.                           |
| AXRADIO_ERR_RECEIVESTART   | The synchronous slave powers up the receiver.                               |

**Status Change Codes**

**Table 3.**

| Status Change Code         | Description   |
|----------------------------|---|
| AXRADIO_STAT_RECEIVE       | Receive Packet arrived  |
| AXRADIO_STAT_TRANSMITSTART | Transmitter start notification  |
| AXRADIO_STAT_TRANSMITDATA  | Transmitter new data needed notification for streaming transmit modes |
| AXRADIO_STAT_TRANSMITEND   | Transmitter end notification  |
| AXRADIO_STAT_RECEIVESFD    | Receiver SFD detected notification                                    |
| AXRADIO_STAT_CHANNELSTATE  | Channel state update  |

In the acknowledge modes, at the start of transmission, TRANSMITSTART is called either with NOERROR or RETRANSMISSION, depending on whether it is the first or subsequent transmission of a packet. At the end of the packet transmission, TRANSMITEND is called with BUSY. When an acknowledge is received, TRANSMITEND is called again with NOERROR. If no acknowledge is received after a timeout, and the number of retransmissions is used up, TRANSMITEND is called with TIMEOUT.

The synchronous master first calls TRANSMITDATA approximately 1ms before turning on the transmitter. This call may be used to prepare the transmit packet and call AXRadio\_transmit.

**FUNCTIONS**

**UINT8\_T AXRADIO\_INIT (VOID)**

Initialize the driver and the chip. This routine must be called before any other AXRadio routine is called. Returns one of the following error codes:

**Table 4.**

| Error Code          | Description                       |
|---------------------|-----------------------------------|
| AXRADIO_ERR_NOERROR | No error occurred.                |
| AXRADIO_ERR_NOCHIP  | No radio chip was found           |
| AXRADIO_ERR_RANGING | The frequency could not be ranged |

**UINT8\_T AXRADIO\_CANSLEEP (VOID)**

This function should be used as follows:

```
wtimer_runcallbacks( );
uint8_t flags = WTFLAG_CANSTANDBY;
if (axradio_cansleep( ) )
    flags |= WTFLAG_CANSLEEP;
wtimer_idle(flags);
```

**UINT8\_T AXRADIO\_SET\_MODE(UINT8\_T MODE)**

This function sets the mode of the radio. Supply one of the AXRADIO\_MODE\_\* constants. Not all modes may be supported, depending on the configuration set in AXRadioLab.

It returns one of the following error codes:

**Table 5.**

| Error Code               | Description   |
|--------------------------|---|
| AXRADIO_ERR_NOERROR      | No error occurred, operation completed successfully |
| AXRADIO_ERR_NOTSUPPORTED | The operation is not supported                      |
| AXRADIO_ERR_NOCHIP       | No radio chip was found                             |
| AXRADIO_ERR_RANGING      | The frequency could not be ranged                   |

**UINT8\_T AXRADIO\_GET\_MODE(VOID)**

This function returns the current chip operating mode. See the AXRADIO\_MODE \* constants.

**UINT8\_T AXRADIO\_SET\_CHANNEL(UINT8\_T CHNUM)**

This function sets the channel number to be used. The mapping between channel number and frequency is configured in AXRadioLab. This function returns one of the following error codes:

**Table 6.**

| Error Code          | Description   |
|---------------------|---|
| AXRADIO_ERR_NOERROR | No error occurred, operation completed successfully             |
| AXRADIO_ERR_BUSY    | The operation could not be completed because the radio was busy |
| AXRADIO_ERR_INVALID | The operation failed because of an invalid parameter            |
| AXRADIO_ERR_RANGING | The frequency could not be ranged                               |

**UINT8\_T AXRADIO\_GET\_CHANNEL(VOID)**

This function returns the currently used channel number.

**UINT8\_T AXRADIO\_GET\_PLLRANGE(VOID)**

**UINT8\_T AXRADIO\_GET\_PLLRANGE\_TX(VOID)**

These functions return the current PLL ranges for the currently set frequency. This is mainly for debugging.

**VOID AXRADIO\_SET\_LOCAL\_ADDRESS(CONSTRUCT AXRADIO\_ADDRESS\_MASK ADDR)**

This function sets the MAC address of the local radio node. The length of a MAC address is configured by AXRadioLab.

The AXRadio\_address\_mask structure has the following definition:

```
struct axradio_address_mask {
    uint8_t addr[4];
    uint8_t mask[4];
};
```

**VOID AXRADIO\_GET\_LOCAL\_ADDRESS(STRUCT AXRADIO\_ADDRESS\_MASK ADDR)**

This function returns the currently configured local radio node MAC address. A pointer to a memory space where the address can be stored into must be provided.

**UINT8\_T AXRADIO\_SET\_PREQOFFSET(INT32\_T OFFS)**

AXRadio allows the user to shift the transceiver somewhat from the channel center frequency. This can be useful in a master-slave setup, where the slaves adjust their frequency upon reception from the master (AXRadio measures the frequency offset of every packet received), to compensate for drifting crystals.

This routine sets the frequency offset from the channel center frequency that should be used. The offset remains when channels are switched.

The unit is a driver internal one. It can be converted to and from Hz using the axradio\_conv\_freq\_\* routines.

**INT32\_T AXRADIO\_GET\_FREQOFFSET(VOID)**

This routine returns the current frequency offset.

**INT32\_T AXRADIO\_CONV\_FREQ\_TOHZ(INT32\_T F)**

This routine converts internal unit frequency offsets to Hz.

**INT32\_T AXRADIO\_CONV\_FREQ\_FROMHZ(INT32\_T F)**

This routine converts frequency offsets in Hz into internal units for frequency offset.

**INT32\_T AXRADIO\_CONV\_TIMEINTERVAL\_TOTIMERO (INT32\_T DT)**

This function converts a time interval, such as a difference of two status callback st -> time values, into wakeup timer 0 units.

**UINT32\_T AXRADIO\_CONV\_TIME\_TOTIMERO (UINT32\_T DT)**

This function converts an absolute time, such as a status callback st -> time value, from internal units to wakeup timer 0 units. Note that status callback st -> time values are generally only valid during the status callback, as the relationship between the internal timer and the wakeup timer 0 may change, for example when the radio chip is powered down.

**UINT8\_T AXRADIO\_TRANSMIT(CONST STRUCT AXRADIO\_ADDRESS \*ADDR, CONST UINT8\_T \*PKT, UINT16\_T LEN)**

Calling this function transfers the user packet data pointed to by pkt and having length len to AXRadio for transmission. Only one packet may be in the process of being transmitted at any time. If a second packet transmission is attempted, a busy error is returned.

The semantics of this routine slightly differs depending on whether the driver is in an asynchronous or a synchronous mode.

In an asynchronous mode, calling this routine queues the packet and immediately starts transmission.

In a synchronous mode, the data is stored for transmission in the next time slot. If this routine is called a second time before the next time slot, the old data is replaced by the data passed in the second call. This may be used to record default data early in the cycle, and possibly update the data if something happens.

```

struct axradio_status {
    uint8_t status; // one of the AXRADIO_STAT_* constants
    uint8_t error; // one of the AXRADIO_ERR_* constants
    uint32_t time; // timestamp of the event
    //
    union {
        // status == AXRADIO_STAT_RECEIVE
        struct {
            struct {
                struct {
                    int8_t rssi; // RSSI, dBm
                    int32_t offset; //frequency offset,internal units
                } phy;
                struct {
                    uint8_t remoteaddr[4];
                    uint8_t localaddr[4];
                    const __xdata uint8_t *raw;
                } mac;
                const __xdata uint8_t *pktdata;
                uint16_t pktlen;
            } rx;
            // status == AXRADIO_STAT_CHANNELSTATE
            struct {
                int8_t rssi; // RSSI, dBm
                uint8_t busy; // 1=over the LBT threshold
            } cs;
        } u;
    };
};

```

The addr argument specifies the address of the remote station this packet is destined to.

The axradio\_address structure has the following definition:

```

struct axradio_address {
    uint8_t addr[4];
};

```

**UINT8\_T AXRADIO\_AGC\_FREEZE(VOID)  
 UINT8\_T AXRADIO\_AGC\_THAW(VOID)**

axradio\_agc\_freeze/axradio\_agc\_thaw may be used during the streaming receive modes to freeze or thaw the automatic gain control

**VOID AXRADIO\_STATUSCHANGE (CONST XDATA STRUCT AXRADIO \*ST)**

This function must be provided by the user code. It is called by AXRadio whenever an event that needs to be notified happens.

STATIC CONFIGURATION ITEMS

These static configuration constants are computed by AXRadioLab

**EXTERN CONST \_\_CODE\_UINT8\_T AXRADIO\_MACLEN;**

This constant contains the length of the MAC header in front of the user packet data.

**EXTERN CONST \_\_CODE\_UINT8\_T**

**AXRADIO\_ADDRLEN;**

This constant contains the length of a MAC address, and may be in the range of 1–4.

EXAMPLE USAGE CODE

This section lists simplified code to illustrate the usage of the API

**SIMPLE ASYNCHRONOUS**

This example shows the skeleton for the simplest possible asynchronous transmitter to receiver case.

*TRANSMITTER*

```
#include "ax8052.h"
#include "libmftypes.h"
#include "libmfwtimer.h"
#include "libmfflash.h"
#include "libmfradio.h"
#include "axradio.h"

static const __code struct axradio_address_mask localaddr = {
    { 0x12, 0x34, 0x56, 0x78 },
    { 0xFF, 0xFF, 0xFF, 0xFF }
};

static const __code struct axradio_address remoteaddr = {
    { 0xCA, 0xFE, 0xBA, 0xBE }
};

void axradio_statuschange(const __xdata struct axradio_status *st)
{
}

uint8_t _sdcc_external_startup(void)
{
    // initialize GPIO, peripherals
    if (PCON & 0x40)
        return 1;
    return 0;
}

#ifdef SDCC
extern uint8_t _start__stack[];
#endif

void main (void)
{
#ifdef !defined(SDCC)
    _sdcc_external_startup();
#else
    __asm
    G$_start__stack$0$0 = __start__stack
    globl G$_start__stack$0$0
    __endasm;
#endif
}
```

```

#endif
    flash_apply_calibration(); // check for non-existing calibration
    CLKCON = 0x00;
    wtimer_init(CLKSRC_LPOSC, 1, CLKSRC_FRCOSC, 7);
    EA = 1;
    if (!(PCON & 0x40)) {
        axradio_init(); // check for error
        axradio_set_local_addr(&localaddr);
    } else {
        AX5051_commsleepexit();
    }
    for (;;) {
        uint8_t flg;
        if (key is pressed) {
            __xdata uint8_t userpkt[..];
            // fill userpkt
            axradio_transmit(&remoteaddr, userpkt, sizeof(userpkt));
        }
        wtimer_runcallbacks();
        flg = WTFLAG_CANSTANDBY;
        if (axradio_cansleep())
            flg |= WTFLAG_CANSLEEP;
        wtimer_idle(flg);
    }
}

```

## RECEIVER

```

#include "ax8052.h"
#include "libmftypes.h"
#include "libmfwtimer.h"
#include "libmfflash.h"
#include "libmfradio.h"
#include "axradio.h"
static const __code struct axradio_address_mask localaddr = {
    { 0xCA, 0xFE, 0xBA, 0xBE },
    { 0xFF, 0xFF, 0xFF, 0xFF }
};
void axradio_statuschange(const __xdata struct axradio_status *st)
{
    switch (st->status) {
    case AXRADIO_STAT_RECEIVE:
        // check st->error
        // display st->u.rx.pktdata / st->u.rx.pktlen
        break;
    default:
        break;
    }
}

```

```

uint8_t _sdcc_external_startup(void)
{
    // initialize GPIO, peripherals
    if (PCON & 0x40)
        return 1;
    return 0;
}
#ifdef SDCC
extern uint8_t __start__stack[];
#endif

void main (void)
{
#ifdef !SDCC
    _sdcc_external_startup();
#else
    __asm
    G$__start__stack$0$0 = __start__stack
    .globl G$__start__stack$0$0
    __endasm;
#endif
    flash_apply_calibration(); // check for non-existing calibration
    CLKCON = 0x00;
    wtimer_init(CLKSRC_LPOSC, 1, CLKSRC_FRCOSC, 7);
    EA = 1;
    if (!(PCON & 0x40)) {
        axradio_init(); // check for error
        axradio_set_local_addr(&localaddr);
    } else {
        AX5051_commsleepexit();
    }
    for (;;) {
        uint8_t flg;
        wtimer_runcallbacks();
        flg = WTFLAG_CANSTANDBY;
        if (axradio_cansleep())
            flg |= WTFLAG_CANSLEEP;
        wtimer_idle(flg);
    }
}

```

FILES

This section lists the files that need to be included in an AXRadio project and what their purpose is.

Table 7.

| File         | Description                               |
|--------------|---|
| axradio.h    | AXRadio API declaration                   |
| easyax5043.h | AXRadio AX5043 private header—AX5043 only |
| easyax5043.c | AXRadio AX5043 main code—AX5043 only      |
| easyax5051.h | AXRadio AX5051 private header—AX5051 only |
| easyax5051.c | AXRadio AX5051 main code—AX5051 only      |
| config.c     | Parameters generated by AXRadioLab        |

ON Semiconductor and  are trademarks of Semiconductor Components Industries, LLC dba ON Semiconductor or its subsidiaries in the United States and/or other countries. ON Semiconductor owns the rights to a number of patents, trademarks, copyrights, trade secrets, and other intellectual property. A listing of ON Semiconductor's product/patent coverage may be accessed at [www.onsemi.com/site/pdf/Patent-Marking.pdf](http://www.onsemi.com/site/pdf/Patent-Marking.pdf). ON Semiconductor reserves the right to make changes without further notice to any products herein. ON Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does ON Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation special, consequential or incidental damages. Buyer is responsible for its products and applications using ON Semiconductor products, including compliance with all laws, regulations and safety requirements or standards, regardless of any support or applications information provided by ON Semiconductor. "Typical" parameters which may be provided in ON Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. ON Semiconductor does not convey any license under its patent rights nor the rights of others. ON Semiconductor products are not designed, intended, or authorized for use as a critical component in life support systems or any FDA Class 3 medical devices or medical devices with a same or similar classification in a foreign jurisdiction or any devices intended for implantation in the human body. Should Buyer purchase or use ON Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold ON Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that ON Semiconductor was negligent regarding the design or manufacture of the part. ON Semiconductor is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner.

**PUBLICATION ORDERING INFORMATION**

**LITERATURE FULFILLMENT:**

Literature Distribution Center for ON Semiconductor  
 19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA  
**Phone:** 303-675-2175 or 800-344-3860 Toll Free USA/Canada  
**Fax:** 303-675-2176 or 800-344-3867 Toll Free USA/Canada  
**Email:** [orderlit@onsemi.com](mailto:orderlit@onsemi.com)

**N. American Technical Support:** 800-282-9855 Toll Free  
 USA/Canada  
**Europe, Middle East and Africa Technical Support:**  
 Phone: 421 33 790 2910  
**Japan Customer Focus Center**  
 Phone: 81-3-5817-1050

**ON Semiconductor Website:** [www.onsemi.com](http://www.onsemi.com)  
**Order Literature:** <http://www.onsemi.com/orderlit>

For additional information, please contact your local Sales Representative