BlazeNet

Wireless protocol for home automation

Air Protocol

Description of the wireless protocol used to communicate with devices

Overview

Wireless devices in the network communicate using this custom RF protocol layer. It guarantees reliable transmission of variable size packets to end devices, while being optimized to allow end devices to use as little power as possible. All the while, it ensures that the network remains secure at all times.

It’s primarily designed to work on Silicon Labs chips, atop the RAIL radio abstraction layer; but it should be supported on other devices as well.

General Operation

End devices can be either regular (powered,) which can receive data at any time; and sleepy devices, which turn off their radio receivers to save power. Sleepy devices work like regular device nodes, but may perform more aggressive power saving. This means the coordinator is responsible for buffering packets destined for the device until its next check-in period.

All packets sent over the air are encrypted using an unique session key, which is negotiated when a device associates to the network. Additionally, multicast and broadcast packets are encrypted as well with a shared network key, which is automatically re-generated during the lifetime of the network.

Topology

The network operates as a standard star topology; that is, all devices communicate directly with the coordinator, which sits at the center of the network. Nodes may directly communicate with one another, if they are in range of each other. At this time, relay mode through the AP is not supported.

In the future, the network may support a tree-like structure, with powered repeater nodes that forward packets to the central network coordinator.

There is no fixed limit on the maximum number of nodes, other than that the coordinator will need to keep track of state information for each node; and that operations such as group key ratcheting require communication with every single node to distribute new key material.

Node Addresses

Each device has an unique EUI-64 address it uses when communicating over the air. During association, the device receives a “short” 16-bit identifier instead, which is used as a shorthand in

the over the air protocol. These 16-bit addresses are the link layer addresses, which may change for the same device between associations; but they are guaranteed to remain constant for a given device for as long as it's associated with the network.

Addresses in the range of `0xFE00` - `0xFFFF` are reserved for use as multicast/broadcast addresses, and for internal network use. This still leaves roughly ~65K unique node addresses available for use.

Beacons

Coordinators regularly broadcast beacons, which indicate the availability of the network. Networks are identified by a unique 16 byte quantity (such as a UUID) which is programmed into nodes. The beacon in turn contains the short address of the coordinator, and information about supported features in the network.

Additionally, beacons contain notification flags (similar to 802.11 DTIM) for buffered packets for devices in deep sleep mode.

By default, the coordinator emits a beacon every 2.5 seconds.

Security

All messages passed over the network, with the exception of beacon frames, are encrypted. Unicast messages use per node keys, whereas multicast and broadcast keys use per network keys.

All coordinators are provisioned with a public/private keypair. When a device is provisioned onto the network, it receives hashes of all public keys in use by coordinators on the network, which is used so that the device may authenticate the coordinator (which provides its public key) before commencing with the key exchange.

When a node associates to a network, the coordinator and node perform key exchange to derive a unique session key for the node/coordinator. From this session key, separate keys are derived for packets sent from the coordinator to the node, and from the node to the coordinator. Additionally, random data is exchanged for use as an initialization vector for subsequently encrypted packets. Once these keys are collected, secure communication can take place over the network.

In addition to the unicast message keys, the coordinator will share the key used to encrypt multicast/broadcast packets. The coordinator can re-issue this key periodically (called the rekey interval) to improve the network's security.

Likewise, the node or coordinator can perform a new key exchange at any point during the association.

TODO: Investigate https://github.com/jedisct1/libhydrogen performance

Association

Nodes must associate with (join) a network before they're able to pass traffic. This is done in several stages:

1. Locate best coordinator

The node will scan all supported channels for beacons from coordinators. Beacons from all coordinators with matching network IDs will be stored, and sorted according to their received signal strength.

1. Connect to coordinator

Tune to the channel of the coordinator node that was declared "best" earlier, and wait to receive another beacon frame. If this times out, return to step 1.

Otherwise, send an association request (containing our full EUI-64 node address and supported protocol version and security modes) to the coordinator, and wait for a response. This response will indicate a temporary short node address to use for MAC frames during the association process. (A separate, smaller namespace for short node IDs during association helps guard against denial-of-service attacks against a single coordinator from exhausting node IDs shared across a larger network.)

1. Begin Key Exchange

Coordinator provides its public key, random data, and more network information. The public key is hashed and compared against the allowed list, and if it is allowed, the association continues. Otherwise, it's aborted, an error is logged, and another coordinator is attempted.

This message also contains information about the network's authentication mode. If no authentication is implemented, skip to (5).

1. Authenticate to network

If the network has security (as indicated by the coordinator's info message) the appropriate authentication is performed. This will consist of one or more round trips to the coordinator to complete some sort of challenge/response protocol. Devices will normally authenticate using their public/private keypair (which was previously provisioned onto the network) in a challenge/response protocol, possibly also including the node's EUI; however, for in-band pairing, an authentication mechanism using a pre-shared key (pairing code) based around ECJ-PAKE is available as well.

Once the authentication process is complete, the coordinator will send to the node either an "auth success" or "auth failure" message. On success, go to (5) to continue association; otherwise restart at (1).

1. Complete Association

When both sides have exchanged session encryption keys, secure communication can commence. The coordinator will respond to the client's key exchange message with an encrypted acceptance message, which provides the client information such as its final short node address (for use in MAC header,) supported power saving modes, and so forth.

The client then responds with an acknowledgement, using its new address, and makes requests for any desired power saving options such as request buffering.

At this stage, the node is considered associated to the network, even if any additional requests from the node are still outstanding (or fail down the line.) Devices remain associated for some time period N after the last successful packet exchange between the coordinator and node. If no packet is received from the node after a period of N, the coordinator will consider the node dead and forcibly disassociate it.

The interval N is configurable, and can vary per device, based on their desired power use.

Once associated, either the coordinator or end node may send a disassociation packet to terminate the session cleanly. Immediately after the disassociation is acknowledged by the coordinator (or device, for a coordinator-initiated disassociation) the disassociation is processed.

When a disassociation takes place, any encryption keys are zeroed and discarded, as well as any buffers and other information is also flushed. This means that packets destined for a sleeping node that missed its check-in window will be lost.

Device Types

Two device types exist: always on, and periodic devices. This difference has to do with the power saving abilities of the device: an always on device must *always* be listening, i.e. always able to accept and receive packets destined for it. By default, a device is assumed to be always on; if it wishes to operate in periodic mode, it must negotiate this once it's associated to the network.

This means that an always on device cannot make use of lower power states, where the radio is fully powered off for a period of time; therefore, this is more suited for permanently powered devices.

On the other hand are periodic devices. These are only able to accept direct messages during certain time windows. At all other times, any packets meant for the device are buffered by the coordinator, and will be retrieved by the device at a later time. The device will wake up periodically (at a multiple of the beacon interval, negotiated during association) to receive the coordinator beacon frames. These frames indicate which periodic devices have pending traffic: devices may then wake up, download buffered frames, and process them as appropriate.

Periodic devices may also wake up at any other time, and explicitly poll the coordinator for pending traffic. This gives the device explicit control over how it implements power saving modes, with the only constraint being a maximum sleep interval, after which the network coordinator assumes the

device has gone away.

Radio PHY

This page describes the physical radio configuration for use in sending packets over the air.

915MHz Mode

This is the primary (and currently, only) PHY configuration supported. It runs on 2MHz wide channels in the 915MHz ISM band, supporting data rates up to 250kbps.

Radio Configuration

- Modulation scheme: ~~2GFSK, 250kbps data rate~~ OQPSK, 250ksym/sec (500kbps), 500kHz deviation
- DSSS: Code length 32, spreading factor 8 (4 bits/chip), chip base 74 4A C3 9B
- Data whitening: Bit 0, PN9 polynomial, seed 0x01F0
- Preamble: 32 bits
  - The precise value of the preamble is one of the chip codes, so it’s not specified
- Sync word: ??
  - TODO: can this be used as address filtering for devices?
- Forward error correction: None
- Checksum: CCITT_16, seed 0xffff, includes packet header

https://community.silabs.com/s/article/understanding-dsss-encoding-and-decoding-on-efr32-devices?language=en_US

Device Pairing

To add a device to an existing BlazeNet network, it will need to be paired to the network. This pairing process can take place over both an in-band mechanism (over the air) or out of band (via additional interfaces) but the general scheme of operation is the same regardless.

Overview

Pairing is the process of registering a device's authentication key (which is the public part of a public/private keypair unique and internal to the device) with the network control element. This allows the network control element to issue authentication challenges (and more importantly, verify the responses) to the device when it attempts to join the network.

Additionally, during the pairing process, the end device will receive a list of all public keys used by coordinators in the network. This allows the device to verify the identity of the network when joining. (This list can be updated over the air whenever the device associates to the network, though these changed key lists must be cross-signed with the key of a coordinator the end device already trusts.)

Depending on the pairing mechanism, additional data (such as the network identity, channel, regulatory region, etc.) may be provided to the device as well. Regardless of the way pairing takes place, the secondary stage of pairing is the same.

Pairing Exchange

This describes the exchange of messages to perform pairing; it's assumed that by this point, we've established some sort of communications channel between the control plane of the BlazeNet network, and the device through another means.

1. Request pairing: Device notifies the coordinator control plane it wishes to pair to the network.
    1. Depending on the current security policy and network configuration, this request may be declined: for example, a network may not allow new devices at all, or may not permit devices paired in-band.
    2. The network responds with supported pairing methods and algorithms
2. Perform key exchange: Perform a supported key exchange mechanism to derive encryption keys for the rest of the pairing communications
    1. Currently, only Ed25519 is implemented
    2. These keys will encrypt all subsequent packets during the pairing exchange. This provides confidentiality over potentially non-secure pairing channels. (Note that the

device is unable to verify the coordinator's keys, and vice versa at this point)
3. Request pairing challenge: The coordinator provides random data to the device to sign
4. Submit device challenge: Sign the provided random data, concatenated with a constant
    1. Constant serves to guard against using this as a signing oracle
    2. If pairing is in band, the pairing code is concatenated as well
5. Receive network information: Coordinator provides various information about the network, such as the public keys of all coordinator nodes.
    1. At this point, the device's key is registered and it may join the network normally

At the end of the pairing process, the node should (re)associate to the network. It is not required to do so: pairing is considered successful as long as the pairing exchange completes successfully.

Out-of-band Pairing

In this section is a description of how devices are paired to the network using out-of-band communication mechanisms.

NFC

The end device exposes an NFC "dynamic tag," which can be read and written by a separate user device (such as a mobile phone,) or by specifically outfitted coordinator devices. This tag contains a fixed region, which holds information about the device type, as well as its public key.

During the pairing process, the end device and user device exchange messages:

1. Read out information: User device queries the read-only area of the NFC tag, which holds the device's public key, and stores it for later reference.
    1. 1. The reader device SHOULD upload the public key (which it read out from the EEPROM earlier) to the coordinator, to allow it to join the network
2. Upload network configuration: This consists of the network name (UUID,) channel, regulatory information, as well as the public keys of all coordinator nodes
3. Associate to network (using device key)

Once this is complete, the end device will attempt to associate to the network. During this process, the user device may periodically poll the end device as to its association status to determine when it's completed the process. The pairing process is considered complete when the end device either successfully associates to the network, or the association fails (due to invalid credentials, inability to find a coordinator, timeout, or cancelled by user device.) If the pairing process fails at any stage prior to successful association, the network information is not saved and the end device will continue to use its previous network settings, if any.

In-band Pairing

This section describes the process of pairing a new device to a network in-band. It exists as a fall-back for other pairing mechanisms, but should be used as a last resort as it is less secure, less convenient, and much slower than other supported out-of-band mechanisms. Due to its nature, the end device will have to blindly trust the network it is associating to, and must already know the regulatory region the network is operating in.

The basic premise of in-band pairing is that every device will have an unique “pairing code” associated with it. It’s usually derived from its serial number, and should be printed on the device itself. The device will use this pairing code to authenticate to the network during association, instead of its device key.

First Stage

A network will need to be made “joinable,” which sets a flag in its beacon. This also activates an additional authentication mechanism during association, allowing the device to attempt to authenticate with its pairing code instead of its key. Coordinator nodes *should* have a timeout that disables the joinable mode (and purges all permitted pairing codes) automatically after some time.

During this time, the end device will scan all radio channels in the currently active bandplan to find active networks broadcasting beacon frames. It will attempt to authenticate with the pairing code against all networks that are currently joinable. (Note that this does *not* allow the radio to be updated with a new regulatory region, as with out-of-band pairing.)

Second Stage

If the device can associate successfully with its pairing code, it should perform the standard pairing exchange immediately after. It should *not* attempt to negotiate group encryption keys, power saving modes, or attempt to pass any other traffic, as it will be dropped by the coordinator for all end devices authenticated with pairing codes.

Once complete, the end device is expected to re-associate to the network using its device key. The coordinator will invalidate the device’s association at the conclusion of the pairing process, or if any of the steps during the second stage fail.

Security Schemes

This page describes the security/authentication mechanisms in use by the protocol.

Background

Each coordinator and device are required to have a public/private key pair, using an elliptic curve. These are used to authenticate devices and coordinators, as as deriving encryption keys.

Rekeying

Devices *must* support rekeying both unicast and multicast keys at arbitrary intervals. Rekeying can be initiated by the coordinator or the device. In the unicast case, the new key is used immediately for all subsequent packets, once the rekeying process has completed.

For multicast rekeying, the new key is provided via the existing multicast group, with a new key id. Clients are expected to acknowledge receipt of the new key, and packets will switch to using the new key once all clients have confirmed the key change, or after a pre-defined timeout period (which will cause an error message to be logged.)

Unicast Keys

There are several key derivation schemes supported, which are used to derive the unicast keys that protect direct communication between the coordinator and end device node.

Key Based

In this scheme, the device will authenticate using its device key. It is the default scheme for devices that wish to pass actual BlazeNet traffic. The coordinator and device negotiate a shared session key used to encrypt and authenticate all subsequent messages.

This key exchange takes place in two stages. In the first stage, a shared secret is calculated, using ECDH (with their known and previously exchanged public keys.) This shared secret is then appended with two nonces (one provided by the coordinator, the other by the device) and hashed using Poly1305 to produce the final session key. Hashing the key with nonces is used to further diversify the key, and hides some potential underlying biases in ECDH (relating to the resulting keys not being evenly distributed in the key space.)

As part of the association, the coordinator issues a challenge (a random 32-byte blob) the device must sign with its device private key (using EdDSA/Ed25519) and return to the coordinator before being allowed on the network.

Passphrase Based

This scheme is used when performing over-the-air pairing for a peripheral.

A session key is negotiated with the coordinator by using ECJ-PAKE, with the pairing code as the input. This modes does *not* mutually authenticate either the device or coordinator based on previously exchanged public keys; it’s intended to allow a device to easily join a network, based on a fixed pairing code.

Coordinators may place additional requirements on nodes that associated with this scheme.

Multicast Keys

All multicast (and by extension, non-beacon broadcast packets) are encrypted using one of several pre-defined keys. These keys are generated and managed entirely by the coordinator, and provided to client devices through the key management interface exposed by the coordinator.

Packets

This chapter outlines the encoding/format of various packets.

MAC/PHY Packet Structure

All messages sent over the air take the form of a packet. Packets encapsulate the actual user data payload with information required for the network to operate. They are similar to their 802.15.4 counterparts, but extended to allow larger payloads.

This page specifically describes the lowest level PHY and MAC packet headers, which encapsulate the actual BlazeNet protocol data.

Headers

These go at the start of the packet, before the user payload. Going from outermost to innermost header:

Synchronization

While this is not technically part of the data, it's used by the underlying radio subsystem to recognize the frame. The sync header is the very first part of the packet to be transmitted, and is always the same. It's first a 32-bit preamble sequence (generated and specified by the radio PHY) followed by a fixed synchronization word – this is specified by the network, with a default value of `0xF6`.

PHY

A single byte PHY header is added to the start of the packet. This consists of a the entire 8 bits as a length counter for the rest of the packet; a packet length of up to 255 is permitted.

Byte 0							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Length Counter							

MAC

Next, a variable length header is used to indicate the source and destination address of the packet, as well as some additional flags and information about encryption, if used. At a minimum, the header will be 6 bytes in length, and contains the following information:

0	1	2	3	4	5
Flags	Sequence number (tag)	Source Address		Destination Address	

Source and destination address are short addresses, which are assigned to the nodes when they associate to the network. These addresses may change between associations, but are guaranteed to be constant for a node as long as it's associated to the network. (Higher level protocols should implement a form of address resolution to convert more convenient, permanent logical addresses to these network addresses, similar to IPv4 ARP.)

The sequence number is a monotonically increasing counter. It is used to identify the packet for acknowledgement, and may be used as an input to security schemes (but may NOT be used as a key.) Devices can use this to drop duplicate packets, though they must pay attention to correctly implement counter roll-over.

Flags

7	6	5	4	3	2	1	0
Reserved (must be set to 0)	Fragment	Endpoint			Ack request	Data pending	Security Enabled

When "ack request" is set, the recipient of the message should generate an acknowledgement response, and send it to the source of this message, once it's been processed. Data pending is set if the source device has additional buffered packets for the recipient. Security enabled indicates the MAC header is followed by a security header, containing information on how to authenticate and decrypt the packet. Packets can be larger than an underlying PHY maximum packet size.

Lastly, the endpoint value specifies how the payload of the packet is to be interpreted. The following values are assigned:

- 0b000: Network control (used for network management, such as association requests)
- 0b001: Acknowledgement response
- 0b010: User data

All other endpoint values are reserved, and packets containing such values will be discarded.

Security

If the MAC header indicates the packet has security, this variable length (5 byte minimum) security header will be added immediately after the MAC header, but before the payload. The first part of the header indicates the type of security scheme used, as well as a unique nonce for the message. (TODO: define nonce overflow handling better)

0	1	2	3	4
Security Type	Frame Counter			

The frame counter is a unique monotonic counter, used as a nonce for message authentication, encryption and anti-replay protection.

Security type is an enumeration that defines if there are any additional security header data, as well as the crypto scheme in use to protect the message. The following values are defined; messages containing any other security type will be discarded by the protocol layer:

- 0x00: No encryption or security, only anti-replay via frame counter
- 0x01: AES-CCM-128
  - Authentication + encryption (hardware accelerated)
  - Packet is followed with a 16-byte MAC trailer
- 0x02: AES-CTR-128
  - Encryption only (hardware accelerated)
  - No additional header data (existing frame counter is used)
- 0x03: ChaCha20-Poly1305
  - Authentication + encryption (hardware accelerated)
  - Packet followed with a 16-byte tag trailer (for authentication)

Key Source

For all encryption types that perform encryption, the security header is followed up with a key header. This identifies the key index, and an optional key source value.

Byte 0								Byte 1-4
7	6	5	4	3	2	1	0	
Has Source?	Key Index							Key Source

This structure will always have the first byte. If the key index is in the "default" realm (that is, the keys negotiated during association via key exchange) the "has source" flag will be cleared, and the key index corresponds directly to such a "well known" key.

Otherwise, the 4 byte key source field follows the key index to qualify the key index. (Currently, the only implemented values are a 16-bit node id, with the high 16 bits set to all 0's. Additionally, a value of all 1's indicates that a network shared key will be used.)

Fragmentation

If the “fragmented” bit in the MAC header flag field is set, the packet is transmitted in multiple pieces over the air. This is useful to emulate larger minimum packet sizes (such as a 1280 byte MTU for IPv6) for upper protocol layers without any extra work by simply passing large packets to the network stack.

All such fragments have an additional byte header:

7	6	5	4	3	2	1	0
Reserved (must be 0)				Fragment Index			

The fragment index specifies the offset into the actual packet data, that is, the offset of the first byte of this packet’s payload in the logical packet’s payload buffer. The last packet in a fragmented transmission is indicated by a payload length smaller than the PHY maximum.

Footers

Additional information needed to validate packets goes after the payload data.

Checksum

Contains a 16-bit checksum over all headers (starting with the PHY header) and payload data.

This checksum is to be a 16-bit CRC, using the CCITT polynomial with a seed of `0xFFFF`. CRCs are calculated LSB first, and output most significant byte (and bit in the byte) first.

The radio layer shall discard all received packets where the checksum footer's value does not match the computed packet checksum. Packet headers shall not be interpreted until the checksum has been validated, as they may have been corrupted otherwise.

Beacon Packet Structure

This page describes the format of beacon frames.

Fixed Part

Every beacon frame starts out with these fields:

Optional Part

More optional data can be specified as part of the beacon. These optional fields are each specified as tag/length/value tuples. Both the tag and length are a byte; the length does not include these two bytes of a header, but only the payload. Zero length values are allowed.

Client devices should ignore any tags that it doesn’t understand how to decode.

Buffered Traffic Map

- Type: `0x01`
- Length: 2 bytes min

If one or more periodic devices have buffered traffic pending in the coordinator, it will insert a buffered traffic map (BTM) optional tag into the beacon. The BTM has as its first (and only mandatory) value a 16-bit value N that contains the periodic device id of the first (lowest) device with buffered traffic.

If there are any more devices with pending traffic, the message will be larger than this 16-bit quantity; the rest of the message is interpreted as a bitmap, where bit 0 of byte 0 corresponds to device N+1. If a bit is set, that device has pending messages.

Coordinator Rev1

Notes on the first revision coordinator hardware and software

Hardware Errata

Collection of some notes on the rev1 coordinator hardware

Assembly

- Combine 100nF capacitors on BoM
  - C804 (100V) and others (logic level) can be served by the same item
- 0402 size for bottom indicator LEDs is ass
  - They ought to be larger, 0603 or 0805
  - Alternatively, use reverse-mount LEDs that are easier to work with (and can be reflowed)
- Ethernet PHY is not recognized
  - Appears to just be a soldering issue with the pads on the SoM not making proper connection
  - Next rev should have larger SoM pads with more cream?
- Fix footprint for C802, C803
  - They are actually 3225, not 3216

Operation

- Wifi seems to be busted
  - This likely is a software issue; the module initializes correctly and is recognized on boot (with the appropriate drivers being loaded)
  - ```
    [   19.308901] mwifiex_sdio mmc0:0001:1: info: FW download over, size 616840
    bytes
    [   19.834461] mwifiex_sdio mmc0:0001:1: WLAN FW is active
    [   20.254606] mwifiex_sdio mmc0:0001:1: info: MWIFIEX VERSION: mwifiex 1.0
    (15.68.7.p189)
    [   20.261290] mwifiex_sdio mmc0:0001:1: driver_version = mwifiex 1.0
    (15.68.7.p189)
    [   32.322732] fixed-3v3: disabling
    [   32.324575] vdd_sd: disabling
    [  317.632606] mwifiex_sdio mmc0:0001:1: Firmware wakeup failed
    [  317.637160] mwifiex_sdio mmc0:0001:1: PREP_CMD: FW in reset state
    [  317.653566] mwifiex_sdio mmc0:0001:1: info: shutdown mwifiex...
    [  317.672972] mwifiex_sdio mmc0:0001:1: PREP_CMD: card is removed
    [  317.775670] mwifiex_sdio mmc0:0001:1: WLAN FW already running! Skip FW dnld
    [  317.781226] mwifiex_sdio mmc0:0001:1: WLAN FW is active
    [  327.952520] mwifiex_sdio mmc0:0001:1: mwifiex_cmd_timeout_func: Timeout cmd
    id = 0xa9, act = 0x0
    ```
    ```

```
[  327.959898] mwifiex_sdio mmc0:0001:1: num_data_h2c_failure = 0
[  327.965752] mwifiex_sdio mmc0:0001:1: num_cmd_h2c_failure = 0
[  327.971442] mwifiex_sdio mmc0:0001:1: is_cmd_timedout = 1
[  327.976846] mwifiex_sdio mmc0:0001:1: num_tx_timeout = 0
[  327.982131] mwifiex_sdio mmc0:0001:1: last_cmd_index = 0
[  327.987447] mwifiex_sdio mmc0:0001:1: last_cmd_id: a9 00 28 00 16 00 cd 00 1e
00
[  327.994853] mwifiex_sdio mmc0:0001:1: last_cmd_act: 00 00 13 00 01 00 01 00
00 00
[  328.002295] mwifiex_sdio mmc0:0001:1: last_cmd_resp_index = 4
[  328.008046] mwifiex_sdio mmc0:0001:1: last_cmd_resp_id: df 80 28 80 16 80 cd
80 1e 80
[  328.015874] mwifiex_sdio mmc0:0001:1: last_event_index = 1
[  328.021326] mwifiex_sdio mmc0:0001:1: last_event: 00 00 0b 00 00 00 00 00 00
00
[  328.028643] mwifiex_sdio mmc0:0001:1: data_sent=1 cmd_sent=1
[  328.034285] mwifiex_sdio mmc0:0001:1: ps_mode=0 ps_state=0
[  328.043571] mwifiex_sdio mmc0:0001:1: info: _mwifiex_fw_dpc: unregister
device
```

- No pull-up on /RF_RESET line
  - This means that every time the SoC resets, the RF part also resets. A hardware pull-up prevents this when the GPIOs go tristate during reset
- Ethernet port LEDs are ~ bright ~
  - Should be increased from 330Ω, they’re a little on the bright side. 1k is probably fine, as they needn’t be super bright

Firmware Notes

This page collects some notes about the host-driven RF firmware, running on the EFR32FG23 chip on the board.

- RAIL initialization fails with custom build system
  - This is because a LDMA transfer is set up, but something gets wonky with the initialization and the destination address mode is set to *decrement.*
  - Most likely caused due to an ABI mismatch between the compiled RAIL library, and our code (or some headers?) compiled for the drivers, specifically the LDMA driver; current workaround is to monkeypatch the LDMA driver to never set the direction bits (thus completely ignoring the "broken" transfer struct)
- Temperature sensor driver has cast to double
  - In `TEMPDRV_GetTemp()`, the 0.5 constant needs to have a `f` suffix to make it float, rather than double to compile with the enhanced warnings about upcasts
- Host irq system is busted
  - There should be a separate “irq acknowledge” register, instead of making the interrupt levels be dependent on doing some action
  - Currently there’s a possible race between an irq handler and a packet receive, which wedges irq’s

Host Firmware Notes

Some notes about the Linux host firmware (see this GitHub repo).

- UARTs don't work in DMA mode
  - All UARTs (including the high speed RF TTY/serial spew) can only operate in interrupt driven mode. This is pretty inefficient
- GPIO IRQs don't work
  - Kernel drivers (such as buttons) fail to get external interrupts
  - Buttons need to be operated in polling mode