



Morse Micro
reaching farther™

Linux Porting Guide

APPNOTE-24

Copyright © 2025

Table of Contents

1 Scope	3
2 Requirements	4
3 Applying Kernel Modifications	6
3.1 Existing git repository	6
3.2 Using patch files	6
4 Compiling the Driver	7
5 Firmware Binaries	8
6 User Space Software	9
5.1 hostapd and wpa_supplicant	9
5.1.1 Dependencies	9
5.1.2 Checkout	9
5.1.2 Configuration	10
5.1.3 Compilation	10
5.1.4 Installation	10
5.3 morse_cli/morsectrl	11
5.3.1 Dependencies	11
5.3.2 Compilation	11
5.3.3 Installation	11
6 Defining the Hardware for SDIO/SPI	12
6.1 Configuration	12
6.2 Compilation	14
6.2.1 Example 1 (FDT Blob in KBuild)	14
6.2.2 Example 2 (Out-of-Tree Compilation)	15
6.2.3 Example 3 (Devicetree Overlay)	15
7 Bring Up	16
7.1 Interface	16
7.2 Access Point	18
7.3 Station	19
8 Revision History	20

1 Scope

This guide will demonstrate the steps required to integrate Morse Micro Wi-Fi HaLow into an existing Linux based target platform. This document does not intend to be a tutorial for building Linux based firmware from scratch, nor will it discuss integration details into build systems such as Yocto, OpenWrt, or Buildroot.

Additionally, this document focuses on artefacts delivered by Morse Micro, and will not cover how to compile dependencies, and other, general, software utilities which may be required for wireless devices.

2 Requirements

Before integrating HaLow into a project, a developer should ensure they have the following resources prepared:

- a toolchain for the target platform;
- the Linux kernel source, checked out to a compatible version;
- the Linux kernel configuration, as required for their target;
- compiled user space libraries such as OpenSSL and libnl, required for the Morse Micro user space software utilities;
- compiled user space applications, such as `iw` and `ip` (BusyBox `ip` is sufficient).

Morse Micro will provide:

- the MM6108 driver source;
- kernel patches for a list targeting compatible kernel versions, below;
- `hostapd/wpa_supplicant` source modified for 802.11ah support;
- `morse_cli` source - the user space control utility for MM6108;
- MM6108 firmware binaries;
- BCF binaries for Morse Micro designed HaLow modules - to be used as a reference.

Note that HaLow modules from other manufacturers using MM6108 may require a different BCF. The developer should consult the manufacturer.

Devices directly incorporating MM6108 into their design will need to create their own BCF binaries.

This guide will use the 1.15.3 driver and associated files, available for download from <https://github.com/MorseMicro>.

The MorseMicro software releases are designed around the following kernels from <https://github.com/raspberrypi/linux>:

Version	git tag
4.9.80	raspberrypi-kernel_1.20180313-1
4.14.98	raspberrypi-kernel_1.20190401-1
4.19.66	raspberrypi-kernel_1.20190819-1
5.4.83	raspberrypi-kernel_1.20210108-1
5.10.11	raspberrypi-kernel_1.20210201-1
5.15.61	1.20220830
6.1.21	1.20230405
6.6.19	stable_20240529

However, to demonstrate Linux integration more generally, the remainder of this document will provide examples based on a 6.1.137 Linux Kernel and a 1.15.3 Morse Micro software release.

3 Applying Kernel Modifications

Before compiling the Linux kernel, a small number of modifications are required to enable various S1G features. These patches can be seen in the linux repository at:

<https://github.com/MorseMicro/linux/>

Note that there are different branches for different kernel versions, but unless your kernel version is exactly the same you may have to make minor adjustments when integrating them with your kernel. Therefore the easiest path is to build this kernel as you would normally for your platform. However, if you need to apply these changes to your existing kernel, follow one of the sections below.

3.1 Existing git repository

If you have an existing git repository that is a fork of the normal Linux repository, you can add this repository as a remote and merge the appropriate branch. For instance, if you were using a 6.1 kernel:

```
git remote add mmlinux https://github.com/MorseMicro/linux.git
git fetch mmlinux
git merge mm/linux-6.1.21/1.15.x
```

Alternatively, you may wish to cherry-pick only the relevant commits, particularly if your kernel tree isn't based on the normal Linux one:

```
git cherry-pick v6.1.21..mmlinux/mm/linux-6.1.21/1.15.x
```

3.2 Using patch files

If you would prefer to apply patch files, you can either use the github web interface to generate them by adding .patch to commit URLs in the history - e.g. <https://github.com/MorseMicro/linux/commit/73969e73764a93bdffa28928c236ac5101ed25d0.patch> - or clone the repository and use `git format-patch`:

```
git clone https://github.com/MorseMicro/linux.git
cd linux
git checkout mm/linux-6.1.21/1.15.x
git format-patch v6.1.21
```

This will generate a number of patch files in the same directory which you can apply with `git am` or `patch`.

4 Compiling the Driver

Before compiling the driver, please ensure you've compiled your kernel as normal with appropriate `ARCH` and `CROSS_COMPILE` environment variables configured as required.

The Morse Micro driver produces two modules.

- `morse.ko` is the core driver for both SPI and SDIO attached devices. It provides core bus communication logic and network device registration.
- `dot11ah.ko` the shim layer, responsible for translating 802.11ah channel and rate information into 802.11ac, including 802.11ah specific frames such as S1G probe requests.

If the Morse Micro driver is recompiled, it is strongly recommended to update both modules in the system. Failure to do so can result in significantly reduced throughput or, possibly, a complete communication failure.

To compile the driver, execute the following commands

```
git clone https://github.com/MorseMicro/morse_driver.git
cd morse_driver
git checkout
git submodule update --init --recursive

make KERNEL_SRC=../linux \
    CONFIG_WLAN_VENDOR_MORSE=m \
    CONFIG_MORSE_USER_ACCESS=y CONFIG_MORSE_VENDOR_COMMAND=y
```

This will compile `morse.ko` and `dot11ah.ko` which can be installed into `/lib/modules/<kernel_version>/` of the target rootfs.

Additional compile time configuration options are available. For an up-to-date list, and detailed descriptions of the fields, inspect the `Kconfig` file in the source. Notably, to compile the driver for an SDIO or SPI attached device, one would need to use `CONFIG_MORSE_SDIO=y` or `CONFIG_MORSE_SPI=y`. To enable test modes set `CONFIG_MORSE_ENABLE_TEST_MODES=y`.

5 Firmware Binaries

Before the module can be inserted, in a running platform, two binary files need to be installed into `/lib/firmware/morse` of the rootfs:

- the chip firmware;
- a board configuration file (BCF) to provide calibration constants. You may need to acquire this from the manufacturer of the module separately, but be aware that if you have an old BCF you may be forced to use an old release of the driver; you should ask the module manufacturer for an up-to-date BCF.

```
mkdir $STAGING_DIR
curl -sL \
https://github.com/MorseMicro/firmware\_binaries/releases/download/v1.15.3/morsemicro-fw-rel\_1\_15\_3\_2025\_Apr\_16.tar | tar xv
curl -sL \
https://github.com/MorseMicro/bcf\_binaries/releases/download/v1.15.3/morsemicro-bcf-rel\_1\_15\_3\_2025\_Apr\_16.tar | tar xv
rm LICENSE
```

On insertion of the `morse.ko` kernel module, there are three paths the driver might take for selecting which BCF to use.

1. Use the file specified in the `bcf` module parameter - e.g. `bcf=bcf_mf08651.bin`
2. For modules with the board type ID programmed into OTP, use the file with name `bcf_boardtype_<board type id>.bin`
3. Use `bcf_default.bin`

Some modules, including MM6108-MF08651-US, are shipped with the board type ID programmed into OTP. To support this it is recommended the developer adds the following symbolic link to their rootfs/target platform.

```
ln -s /lib/firmware/morse/bcf_mf08651.bin \
    $STAGING_DIR/lib/firmware/morse/bcf_boardtype_0801.bin
```

For other modules the developer should run the following command

```
ln -s /lib/firmware/morse/$BCF.bin \
    $STAGING_DIR/lib/firmware/morse/bcf_default.bin
```

where `$BCF` matches the appropriate BCF for the target platform.

6 User Space Software

A device integrating HaLow requires more than just the driver.

- **wpa_supplicant** - user space wireless client, including 802.11s mesh points
- **hostapd** - user space daemon for access points
- **morse_cli** - Morse Micro's user space control utility. It is required that this is installed on the target system for the correct function of hostapd and wpa_supplicant provided by Morse Micro.
- **morsectrl** - Morse Micro's advanced user space control utility. This utility provides extra commands useful for testing radio performance. The installation of **morsectrl** is not recommended for image builds targeting end users, and is not distributed as open source.

The above utilities have dependencies on additional software libraries which must be compiled for the target platform. This guide will not provide instructions on how to prepare these dependencies.

For hostapd to function correctly at run-time, two cryptographic APIs need to be enabled in the Linux kernel configuration. Also, please also ensure the kernel includes the following configuration.

- CONFIG_CRYPTO_CCM=y
- CONFIG_CRYPTO_GCM=y

Additionally, standard networking utilities such as `iw` or `ip` may be required during bring up. This guide won't discuss the compilation and installation of these utilities, but a developer should add them to their platform.

5.1 hostapd and wpa_supplicant

5.1.1 Dependencies

Please ensure the following dependencies are compiled, and if required, installed into the staging directory of your build environment.

- libnl-3 (tested with 3.11.0), including libnl-route-3 and libnl-genl-3
- openssl (tested with 3.5.0)

5.1.2 Checkout

Clone the repository and checkout the appropriate branch using git:

```
git clone https://github.com/MorseMicro/hostap.git
cd hostap
git checkout 1.15.3
```

5.1.2 Configuration

The Morse Micro hostap repository provides a default configuration for hostapd and wpa_supplicant for a minimal release.

```
cp wpa_supplicant/defconfig wpa_supplicant/.config
cp hostapd/defconfig hostapd/.config
```

Depending on the application, additional configuration options may need to be enabled. The options available are extensive, and out of scope for this document. Consult the .config file for more information on the available configuration options

5.1.3 Compilation

Compiling wpa_supplicant and/or hostapd requires the above mentioned dependencies be made accessible to the build. In this example, libnl-3 and openssl were “installed” into the staging directory with `make install`.

The below command also assumes your `cc` flag is set appropriately for your target. If additional `CFLAGS`, `LDFLAGS`, or other environment variables are required for your platform, add them as required.

Compiling wpa_supplicant:

```
CFLAGS="-Wno-error=sign-compare
-Wno-error=deprecated-declarations -I
$STAGING_DIR/include/libnl3 -I $STAGING_DIR/include/" \
LDFLAGS="-L $STAGING_DIR/lib/ -L $STAGING_DIR/lib64/" \
DESTDIR="$STAGING_DIR" BINDIR=/usr/sbin \
make MORSE_VERSION=rel_1_15_3_2025_Apr_16 -C wpa_supplicant/
```

Compiling hostapd:

```
CFLAGS="-Wno-error=sign-compare -I $STAGING_DIR/include/libnl3
-I $STAGING_DIR/include/" \
LDFLAGS="-L $STAGING_DIR/lib/ -L $STAGING_DIR/lib64/" \
DESTDIR="$STAGING_DIR" BINDIR=/usr/sbin \
make MORSE_VERSION=rel_1_15_3_2025_Apr_16 -C hostapd/
```

5.1.4 Installation

Installation with a staging directory appropriately set is done by adding the `install` as a target to the above make commands.

If performing a manual installation on an existing device, you can install the following binaries into `/usr/sbin`:

- `wpa_supplicant_s1g`

- wpa_cli_s1g
- wpa_passphrase_s1g
- hostapd_s1g
- hostapd_cli_s1g

5.3 morse_cli/morsectrl

5.3.1 Dependencies

Please ensure the following dependencies are compiled, and if required, installed into the staging directory of your build environment.

- libnl-3 (tested with 3.11.0)
- libusb (tested with 1.0.28; this may also require libudev)

5.3.2 Compilation

Compiling morse_cli requires the above mentioned dependencies be made accessible to the build. In this example, libnl-3 and libusb were installed into the staging directory with `make install`.

The below command also assumes your `cc` flag is set appropriately for your target. If additional `CFLAGS`, `LDFLAGS`, or other environment variables are required for your platform, add them as required.

```
git clone https://github.com/MorseMicro/morse\_cli.git
cd morse_cli
git checkout 1.15.3

CFLAGS="-I$STAGING_DIR/include/libnl3/
-I$STAGING_DIR/include/libusb-1.0" \
LDFLAGS="-L $STAGING_DIR/lib/" \
make CONFIG_MORSE_TRANS_NL80211=1
```

If you have the morsectrl source, it can be compiled in a similar way, and will also generate a copy of morse_cli (morse_cli contains a subset of morsectrl commands).

5.3.3 Installation

To install morse_cli, copy the compiled binary to `/usr/bin` of your device or target rootfs.

6 Defining the Hardware for SDIO/SPI

6.1 Configuration

In order for the driver to detect and communicate with the chip via SDIO or SPI, the hardware must be described to the kernel. This ensures all necessary clocks, pins, power regulators, and necessary host bus peripherals are correctly configured for the target platform.

This hardware configuration is described to the kernel using devicetree, which is compiled and passed to the kernel as a binary blob.

A developer may choose to modify an existing board devicetree file (`.dts`), compiling the changes into a single Flattened Device Tree (FDT) binary (`.dtb`). Alternatively, a device tree “fragment” can be composed and a device tree overlay compiled (`.dtbo`) allowing that hardware configuration to be changed on boot.

Below is a small snippet from the Morse Micro EKH03 devicetree (`.dts`) file, redacted for brevity and to highlight the required properties and nodes to describe an SDIO connected MM6108.

```
...
&sdhci {
    ...
    cap-sd-highspeed;
    cap-mmc-highspeed;
    non-removable;
    disable-wp;
    cap-sdio-irq;
    ...
    mm6108_sdio: mm6108_sdio@0 {
        compatible = "morse,mm610x";
        reset-gpios = <&gpio 12 0>;
        power-gpios = <&gpio 7 0>,
                      <&gpio 8 0>;
        status = "okay";
        reg = <2>;
        bus-width = <4>;
    };
};
...
```

For SPI devices, the MM6108 device tree node looks slightly different. The snippet below, taken from the MorseMicro EKH01 EVK, demonstrates how a developer might prepare a devicetree overlay to describe an MM6108 connected to the first SPI bus of

the host processor. Of particular note is the different `compatible`, and the `spi-irq-gpios` parameter.

```
/dts-v1/;
/plugin/;

/ {
    compatible = "brcm,bcm2835", "brcm,bcm2836", "brcm,bcm2708",
    "brcm,bcm2709", "brcm,bcm2711";

    fragment@0 {
        target = <&spi0>;
        frag0: __overlay__ {
            pinctrl-0 = <&spi0_pins &spi0_cs_pins>;
            cs-gpios = <&gpio 8 1>;
            #address-cells = <1>;
            #size-cells = <0>;
            status = "okay";

            mm6108: mm6108@0 {
                compatible = "morse,mm610x-spi";
                reg = <0>; /* CE0 */
                reset-gpios = <&gpio 5 0>;
                power-gpios = <&gpio 3 0>,
                    <&gpio 7 0>;
                spi-irq-gpios = <&gpio 25 0>;
                spi-max-frequency = <50000000>;
                status = "okay";
            };
        };
    };
};
```

Property	Note
<code>cap-sdio-irq</code>	Enable SDIO IRQ signalling on this bus
<code>compatible</code>	Must be "morse,mm610x" in the MM6108 node to correctly probe the SDIO driver Must be "morse,mm610x-spi" in the MM6108 node to correctly probe the SPI driver.
<code>reset-gpios</code>	GPIO descriptor connected to the MM6108 RESET line.
<code>power-gpios</code>	GPIO descriptor list connected to the MM6108 WAKE and BUSY pins, respectively
<code>spi-irq-gpios</code>	GPIO descriptor connected to the MM6108 SPI_INT pin.
<code>status</code>	Must be "okay" to enable the device!

Due to the target specific requirements of booting a device, there is no one specific approach a developer should use to compile and deploy changes to the device tree. The next 3 subsections describe 3 common methods for which device tree changes may be incorporated into a target platform.

6.2 Compilation

6.2.1 Example 1 (FDT Blob in KBuild)

In some cases, for some targets, the Linux kernel can ease the compilation of and "installation" of the FDT Blob by specifying the device tree file to load in the kernel config. This is useful for targets with limited bootloaders which are unable to hand the FDT blob to the kernel at boot time.

As an example, if we wanted to add HaLow support to the VoCore 2 module, we could add the EKH03 device tree change above to `arch/mips/boot/dts/ralink/vocore2.dts`. A kernel checkout configured for a VoCore 2 will have the following kernel options set:

- `CONFIG_BUILTIN_DTB`
- `CONFIG_DTB_VOCORE2`

The Kbuild Makefile `arch/mips/boot/dts/ralink/Makefile` handles compilation of the FDT blob.

For the ambitious reader, it's worth noting again that the EKH03 snippet above is incomplete and will not function as-is when applied to the `vocore2.dts` file in the kernel tree. Morse Micro recommends consulting the OpenWrt SDK for more complete examples.

6.2.2 Example 2 (Out-of-Tree Compilation)

Devicetree files may also be composed outside of the kernel build tree. As above, the EKH03 device tree change should be added to an out-of-tree `board.dts` file.

Compilation of the FDT blob will follow steps similar to below

```
# preprocess #includes and #defines
$(CPP) -nostdinc -x assembler-with-cpp $(DTS_CPPFLAGS) \
    -I$(LINUX_DTS_DIR) -I$(LINUX_DTS_DIR)/include \
    -I$(LINUX_DIR)/include -undef -D__DTS__ \
    -o board.dtb.tmp board.dts

# compile FDT
$(DTC) -O dtb $(FLAGS) -o board.dtb board.dtb.tmp
```

The FDT blob can simply be appended to the kernel image for “installation”, prior to any image compression.

```
cat board.dtb >> kernel.bin
```

For devices with an FDT aware bootloader, the compiled FDT blob can be handed to the kernel at boot time by the bootloader. The specific details of how this is carried out are platform specific.

6.2.3 Example 3 (Devicetree Overlay)

If flexibility in the hardware configuration is required, devicetree overlay fragments, similar to the fragment used for the EKH01 shown above, can be loaded by FDT aware bootloaders to modify the behaviour of the device FDT. Steps to compile an overlay fragment into a `.dtbo` mirror the compilation of the FDT blob.

```
# compile FDT overlay
$(DTC) -@ -O dtbo $(FLAGS) -o overlay.dtbo overlay.dts
```

7 Bring Up

The configuration options for `hostapd` and `wpa_supplicant` are extensive. Certain options may also depend on compile time configuration. This guide aims to provide a minimal bring up of the HaLow interface, and recommend a developer look to the `hostapd` documentation for detailed descriptions of the configuration available.

7.1 Interface

Before bringing up an access point, ensure the `dot11ah` and `morse` kernel modules are inserted.

```
modprobe mac80211
modprobe crc7
insmod dot11ah/dot11ah.ko
insmod morse.ko country=AU
```

Confirm that your kernel logs show the device being correctly registered. A new phy and associated network device (eg `wlan0`) should now be registered with the kernel.

You can now bring up the interface. If you're not using a DHCP client daemon, you should also set an IP address on the interface using `ip`:

```
ip addr add 192.168.1.1/24 dev wlan0
ip link set wlan0 up
```

Remember to choose a different IP in the same subnet for the AP and the Station.

There are a number of module parameters supported by the Morse Micro driver. The full list can be found by querying the driver with `modinfo` or by inspecting the files in `sysfs`. However, the following module parameters are often beneficial for testing.

- **country** - adjust the default region for which the device should operate;
- **spi_clock_speed** - controls the SPI clock speed requested by the driver;
- **test_mode** - execute one of the test modes in the table below

test_mode	Identifier	Description
0	DISABLED	normal boot
1	DOWNLOAD_ONLY	downloads the firmware

2	DOWNLOAD_AND_GET_HOST_TBL_PTR	downloads the firmware and gets the location of the host table
3	GET_HOST_TBL_PTR_ONLY	only get the location of the host table
4	RESET	chip reset
5	BUS	perform block reads and writes to profile the communication bus

7.2 Access Point

Make sure the Morse Micro driver is inserted as described in [7.1 Interface](#), and make sure `morse_cli` is in your path. Then start `hostapd_slg` with the minimal config file below. Make sure to change the name of `wlan0` if necessary.

```
hostapd_slg hostapd.conf
```

hostapd.conf:

```
ctrl_interface=/var/run/hostapd_slg
interface=wlan0
driver=nl80211
hw_mode=a
ieee80211ah=1
channel=44
op_class=71
country_code=AU
slg_prim_chwidth=1
slg_prim_1mhz_chan_index=0
ssid=MorseMicro
beacon_int=100
dtim_period=1
ap_max_inactivity=65536
slg_capab=[SHORT-GI-ALL]
wpa=2
wpa_key_mgmt=SAE
rsn_pairwise=CCMP
sae_password=12345678
ieee80211w=2
wnm_sleep_mode=1
sae_pwe=1
```

You may find the `-B` (daemonise) and `-s` (output logs to syslog) options useful once you've confirmed that association is working.

7.3 Station

Make sure the Morse Micro driver is inserted as described in [7.1 Interface](#). Then start `wpa_supplicant_s1g` with the minimal config file below. Make sure to change the name of `wlan0` if necessary.

```
wpa_supplicant_s1g -D nl80211 -i wlan0 \  
-c wpa_supplicant.conf
```

`wpa_supplicant.conf`

```
ctrl_interface=/var/run/wpa_supplicant_s1g  
#update_config=1  
  
pmf=2  
sae_pwe=1  
network={  
    ssid="MorseMicro"  
    key_mgmt=SAE  
    pairwise=CCMP  
    psk="12345678"  
}
```

You may find the `-B` (daemonise) and `-s` (output logs to syslog) options useful once you've confirmed that association is working.

8 Revision History

Release Number	Release Date	Release Notes
01	20/08/2024	Initial release
02	30/05/2025	Update for 1.15.3 and github release



Morse Micro Pty. Ltd. - HQ

Level 8, 10-14 Waterloo Street,
Surry Hills, NSW 2010, AUSTRALIA

Morse Micro Inc. – USA

40 Waterworks Way
Irvine, CA 92618, UNITED STATES

Office Locations:

Bangalore, India
Cambridge, UK
Hangzhou, China
Irvine, CA - USA
Picton, NSW - Australia
Portola Valley, CA - USA
Shenzhen, China
Taipei, Taiwan

sales@morsemicro.com

www.morsemicro.com

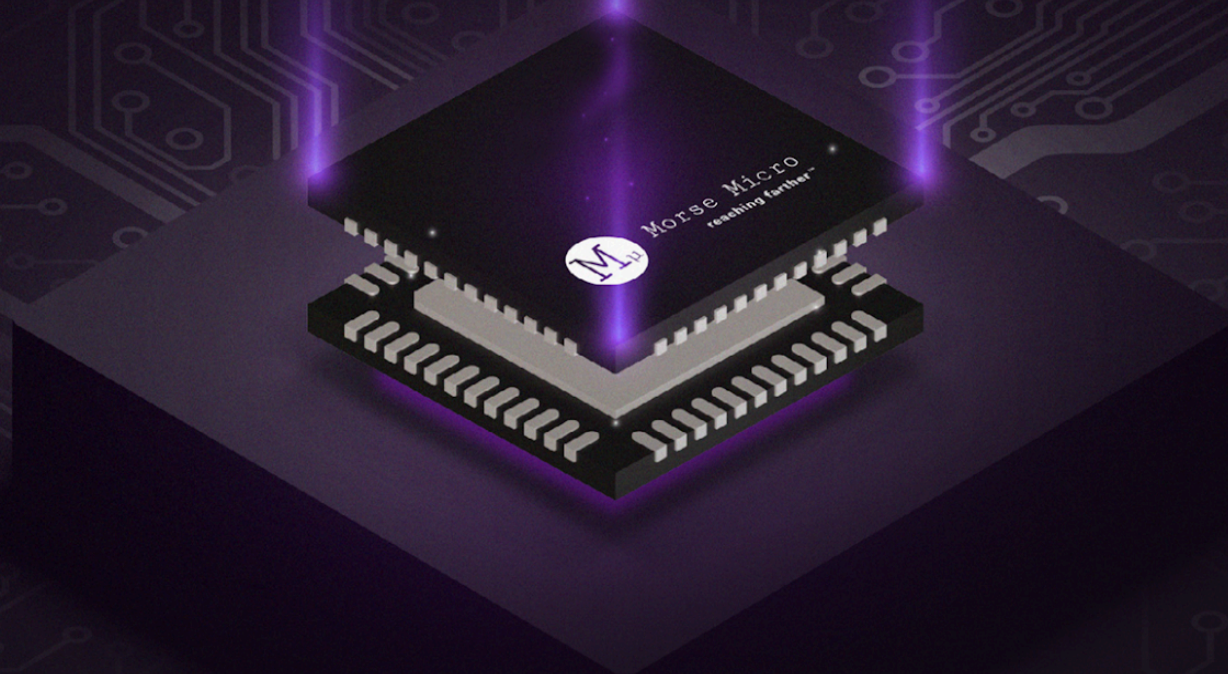
© 2016–2025 Morse Micro Pty. Ltd.

All rights reserved. Morse Micro and the Morse Micro logo are trademarks of Morse Micro Pty. Ltd. All other trademarks and service marks are the property of their respective owners.

Morse Micro makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Morse Micro assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Morse Micro have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Morse Micro. It is the Buyer's responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Morse Micro hereunder is provided "as is, where is" and with all faults, and the entire risk associated with such information is entirely with the Buyer. Morse Micro does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Morse Micro, and Morse Micro reserves the right to make any changes to the information in this document or to any products and services at any time without notice.

About Morse Micro

Morse Micro is a leading Wi-Fi HaLow fabless semiconductor company based in Sydney, with global offices. As the world's premier Wi-Fi HaLow company, we pioneer next-gen IoT wireless connectivity solutions. Morse Micro is now sampling its Wi-Fi CERTIFIED HaLow MM6108 production silicon: the fastest, smallest, lowest power and longest-range Wi-Fi HaLow chip available in the market.



Morse Micro
reaching farther™