OpenRTX: an open source firmware for ham radio devices

Silvano Seva - IU2KWO

4th February 2024
whoami

- Also known as Redman
- Born and living in Milan, Italy
- Ham radio operator since 2017 as IU2KWO
- Firmware developer by profession (and by passion)
- Co-founder and developer of OpenRTX
- Member of the M17 team since 2021
OpenRTX
OpenRTX

- An open-source firmware for ham radio devices
OpenRTX

• An open-source firmware for ham radio devices
• *Designed to be:*
OpenRTX

• An open-source firmware for ham radio devices
• Designed to be:
  • modular
OpenRTX

- An open-source firmware for ham radio devices
- Designed to be:
  - modular
  - *easily portable to new devices*
OpenRTX

- An open-source firmware for ham radio devices
- Designed to be:
  - modular
  - easily portable to new devices
  - *easily extendable to new protocols*
OpenRTX

• An open-source firmware for ham radio devices
• Designed to be:
  • modular
  • easily portable to new devices
  • easily extendable to new protocols

• Currently supporting FM and M17 modes
Timeline

- March 2020: project starts as a port of OpenGD77 to the TYT MD-380
- September 2020: original idea abandoned, “official” beginning of OpenRTX
- January 2021: first alpha release with working FM on the TYT MD-380
- February 2021: first TX tests of M17 mode on the MD-380
- April 2021: support for GD-77, DM-1801 and MD-UV380
- May 2022: release v0.3.3 brings full support for M17 voice transmission
- November 2022: implemented voice prompts for vision impaired operators
- October 2023: support for Lilygo T-TWR Plus (and various technical improvements)
- More to come...
Timeline

- **March 2020**: project starts as a port of OpenGD77 to the TYT MD-380
- **September 2020**: original idea abandoned, “official” beginning of OpenRTX
- **January 2021**: first alpha release with working FM on the TYT MD-380
- **February 2021**: first TX tests of M17 mode on the MD-380
- **April 2021**: support for GD-77, DM-1801 and MD-UV380
- **May 2022**: release v0.3.3 brings full support for M17 voice transmission
- **November 2022**: implemented voice prompts for vision impaired operators
- **October 2023**: support for Lilygo T-TWR Plus (and various technical improvements)
- **More to come ...**
Supported devices

- TYT MD-380/Retevis RT3 (FM, M17)
- TYT MD-UV380/Retevis RT3s (FM, M17)
- Radioditty GD-77 (FM)
- Baofeng DM-1801 (FM)
- Module17 (M17)
- Lilygo T-TWR Plus (FM)
Internals

- Core
  - Audio management
  - User settings
  - Voice prompts
- Graphics functions
- Codeplug
- GPS
- Operating modes
  - FM
  - M17
  - ...
- UI
- HW interface API
- (RTOS)
Internals

- Audio management
- Graphics functions
- User settings
- Codeplug
- Voice prompts
- GPS

UI

Operating modes
- FM
- M17
- ...

Core

HW interface API

(RT)OS

- Interface with the operating system:
  - thread management done using the Posix API
  - all the remaining parts use the standard C library
  - an RTOS is preferred on embedded devices

- Interface with the hardware:
  - APIs for display, keyboard, audio, radio and nonvolatile memory
  - "platform" API for device initialization and other stuff (LEDs, ...)
  - more devices can share a single API implementation (e.g. the display driver for MDx)
Internals

- **Interface with the operating system:**

- Interface with the operating system:
  - Interface with the operating system:
    - Thread management done using the Posix API
    - All the remaining parts use the standard C library
    - An RTOS is preferred on embedded devices

- Interface with the hardware:
  - APIs for display, keyboard, audio, radio and nonvolatile memory
  - "Platform" API for device initialization and other stuff (LEDs, ...)
  - More devices can share a single API implementation (e.g. the display driver for MDx)
• Interface with the operating system:
  • *thread management done using the Posix API*
Internals

- Interface with the operating system:
  - thread management done using the Posix API
  - all the remaining parts use the standard C library
Internals

• Interface with the operating system:
  • thread management done using the Posix API
  • all the remaining parts use the standard C library
  • an RTOS is preferred on embedded devices
Internals

- Interface with the operating system:
  - thread management done using the Posix API
  - all the remaining parts use the standard C library
  - an RTOS is preferred on embedded devices

- Interface with the hardware:
Internals

• Interface with the operating system:
  • thread management done using the Posix API
  • all the remaining parts use the standard C library
  • an RTOS is preferred on embedded devices

• Interface with the hardware:
  • APIs for display, keyboard, audio, radio and nonvolatile memory
Internals

• Interface with the operating system:
  • thread management done using the Posix API
  • all the remaining parts use the standard C library
  • an RTOS is preferred on embedded devices

• Interface with the hardware:
  • APIs for display, keyboard, audio, radio and nonvolatile memory
  • “platform” API for device initialization and other stuff (LEDs, …)
Internals

- Interface with the operating system:
  - thread management done using the Posix API
  - all the remaining parts use the standard C library
  - an RTOS is preferred on embedded devices

- Interface with the hardware:
  - APIs for display, keyboard, audio, radio and nonvolatile memory
  - “platform” API for device initialization and other stuff (LEDs, ...)
  - more devices can share a single API implementation (e.g. the display driver for MDx)
Internals

- Core
  - Audio management
  - Graphics functions
  - User settings
  - Codeplug
  - Voice prompts
  - GPS
  - UI

- Operating modes
  - FM
  - M17

- HW interface API
  - (RT)OS

User interface:
- Currently a "standard" GUI + an ad-hoc GUI for Module17
- You can write your own from scratch, if you want
- Future plans to make the standard GUI scriptable/expandable

Operating modes/protocols:
- C++ here, but simple
- All the operating modes are subclasses of a generic "OpMode" class
- Pre-defined functions: enable, disable, periodic update (33Hz), squelch status
- Still some work to do: functions to get/set mode-specific data (e.g. configuration)
Internals

• **User interface:**

- Currently a “standard” GUI + an ad-hoc GUI for Module17
- You can write your own from scratch, if you want
- Future plans to make the standard GUI scriptable/expandable

• Operating modes/protocols:
  - C++ here, but simple
  - All the operating modes are subclasses of a generic “OpMode” class
  - Pre-defined functions: enable, disable, periodic update (33Hz), squelch status
  - Still some work to do: functions to get/set mode-specific data (e.g., configuration)
Internals

- User interface:
  - currently a “standard” GUI + an ad-hoc GUI for Module17
Internals

• User interface:
  • currently a “standard” GUI + an ad-hoc GUI for Module17
  • you can write your own from scratch, if you want
Internals

- User interface:
  - currently a “standard” GUI + an ad-hoc GUI for Module17
  - you can write your own from scratch, if you want
  - future plans to make the standard GUI scriptable/expandable
Internals

- User interface:
  - currently a “standard” GUI + an ad-hoc GUI for Module17
  - you can write your own from scratch, if you want
  - future plans to make the standard GUI scriptable/expandable

- Operating modes/protocols:
Internals

- **User interface:**
  - currently a “standard” GUI + an ad-hoc GUI for Module17
  - you can write your own from scratch, if you want
  - future plans to make the standard GUI scriptable/expandable

- **Operating modes/protocols:**
  - *C++ here, but simple*
Internals

• User interface:
  • currently a “standard” GUI + an ad-hoc GUI for Module17
  • you can write your own from scratch, if you want
  • future plans to make the standard GUI scriptable/expandable

• Operating modes/protocols:
  • C++ here, but simple
  • all the operating modes are subclasses of a generic “OpMode” class
Internals

• User interface:
  • currently a “standard” GUI + an ad-hoc GUI for Module17
  • you can write your own from scratch, if you want
  • future plans to make the standard GUI scriptable/expandable

• Operating modes/protocols:
  • C++ here, but simple
  • all the operating modes are subclasses of a generic “OpMode” class
  • pre-defined functions: enable, disable, periodic update (33Hz), squelch status
• User interface:
  • currently a “standard” GUI + an ad-hoc GUI for Module17
  • you can write your own from scratch, if you want
  • future plans to make the standard GUI scriptable/expandable

• Operating modes/protocols:
  • C++ here, but simple
  • all the operating modes are subclasses of a generic “OpMode” class
  • pre-defined functions: enable, disable, periodic update (33Hz), squelch status
  • still some work to do: functions to get/set mode-specific data (e.g. configuration)
M17 support

- *First work done on the TYT MD-380, then extended to the MD-UV380*
M17 support

- First work done on the TYT MD-380, then extended to the MD-UV380
- *Everything is handled in the MCU*
M17 support

• First work done on the TYT MD-380, then extended to the MD-UV380

• Everything is handled in the MCU

• **Hardware must have the following connections:**

  - mic to MCU
  - RF stage to MCU, DC to ∼3kHz
  - MCU to speaker
  - MCU to RF stage, DC to ∼3kHz
M17 support

• First work done on the TYT MD-380, then extended to the MD-UV380

• Everything is handled in the MCU

• Hardware must have the following connections:
  - mic to MCU

  - RF stage to MCU, DC to ∼3kHz
  - MCU to speaker
  - MCU to RF stage, DC to ∼3kHz

• Current limitations:
  - you need to mod the radio hardware
  - the MCU has to be powerful enough
  - Codec2 uses floating point math
M17 support

- First work done on the TYT MD-380, then extended to the MD-UV380
- Everything is handled in the MCU
- Hardware must have the following connections:
  - mic to MCU
  - RF stage to MCU, DC to ~3kHz
- Current limitations:
  - you need to mod the radio hardware
  - the MCU has to be powerful enough
  - Codec2 uses floating point math
M17 support

- First work done on the TYT MD-380, then extended to the MD-UV380
- Everything is handled in the MCU
- Hardware must have the following connections:
  - mic to MCU
  - RF stage to MCU, DC to \(~3\text{kHz}\)
  - MCU to speaker

Current limitations:
- you need to mod the radio hardware
- the MCU has to be powerful enough
- Codec2 uses floating point math
M17 support

- First work done on the TYT MD-380, then extended to the MD-UV380
- Everything is handled in the MCU
- Hardware must have the following connections:
  - mic to MCU
  - RF stage to MCU, DC to $\sim 3$kHz
  - MCU to speaker
  - $MCU$ to $RF$ stage, $DC$ to $\sim 3$kHz

Current limitations:
- you need to mod the radio hardware
- the MCU has to be powerful enough
- Codec2 uses floating point math
M17 support

• First work done on the TYT MD-380, then extended to the MD-UV380

• Everything is handled in the MCU

• Hardware must have the following connections:
  - mic to MCU
  - RF stage to MCU, DC to $\sim$3kHz
  - MCU to speaker
  - MCU to RF stage, DC to $\sim$3kHz

• **Current limitations:**
M17 support

• First work done on the TYT MD-380, then extended to the MD-UV380

• Everything is handled in the MCU

• Hardware must have the following connections:
  - mic to MCU
  - RF stage to MCU, DC to $\sim$3kHz
  - MCU to speaker
  - MCU to RF stage, DC to $\sim$3kHz

• Current limitations:
  - you need to mod the radio hardware
M17 support

- First work done on the TYT MD-380, then extended to the MD-UV380
- Everything is handled in the MCU
- Hardware must have the following connections:
  - mic to MCU
  - RF stage to MCU, DC to $\sim$3kHz
  - MCU to speaker
  - MCU to RF stage, DC to $\sim$3kHz
- Current limitations:
  - you need to mod the radio hardware
  - *the MCU has to be powerful enough*
M17 support

- First work done on the TYT MD-380, then extended to the MD-UV380

- Everything is handled in the MCU

- Hardware must have the following connections:
  - mic to MCU
  - RF stage to MCU, DC to $\sim3$kHz
  - MCU to speaker
  - MCU to RF stage, DC to $\sim3$kHz

- Current limitations:
  - you need to mod the radio hardware
  - the MCU has to be powerful enough
  - Codec2 uses floating point math
Codeplug

- **Trying to make something which:**

- is open and free
- supports common ham radio needs (direct comm, repeaters, hotspots)
- is portable across devices, both for end users and developers

Currently WIP, an RFC open at https://github.com/OpenRTX/openrtx.github.io/pull/32

Technical details:
- binary format
- up to 65,535 channels, contacts and banks (aka “zones”)
- currently supporting FM, DMR and M17 operating modes
- May become a separate entity from the firmware
Codeplug

- Trying to make something which:
  - *is open and free*
Codeplug

- Trying to make something which:
  - is open and free
  - supports common ham radio needs (*direct comm, repeaters, hotspots*)
Codeplug

- Trying to make something which:
  - is open and free
  - supports common ham radio needs (direct comm, repeaters, hotspots)
  - *is portable across devices, both for end users and developers*
Codeplug

• Trying to make something which:
  • is open and free
  • supports common ham radio needs (direct comm, repeaters, hotspots)
  • is portable across devices, both for end users and developers

• Currently WIP, an RFC open at
  https://github.com/OpenRTX/openrtx.github.io/pull/32
Codeplug

- Trying to make something which:
  - is open and free
  - supports common ham radio needs (direct comm, repeaters, hotspots)
  - is portable across devices, both for end users and developers

- Currently WIP, an RFC open at
  https://github.com/OpenRTX/openrtx.github.io/pull/32

- Technical details:
Codeplug

• Trying to make something which:
  • is open and free
  • supports common ham radio needs (direct comm, repeaters, hotspots)
  • is portable across devices, both for end users and developers

• Currently WIP, an RFC open at
  https://github.com/OpenRTX/openrtx.github.io/pull/32

• Technical details:
  • binary format
Codeplug

• Trying to make something which:
  • is open and free
  • supports common ham radio needs (direct comm, repeaters, hotspots)
  • is portable across devices, both for end users and developers

• Currently WIP, an RFC open at
  https://github.com/OpenRTX/openrtx.github.io/pull/32

• Technical details:
  • binary format
  • *up to 65,535 channels, contacts and banks (aka “zones”)*
Codeplug

• Trying to make something which:
  • is open and free
  • supports common ham radio needs (direct comm, repeaters, hotspots)
  • is portable across devices, both for end users and developers

• Currently WIP, an RFC open at
  https://github.com/OpenRTX/openrtx.github.io/pull/32

• Technical details:
  • binary format
  • up to 65'535 channels, contacts and banks (aka “zones”)
  • currently supporting FM, DMR and M17 operating modes
Codeplug

- Trying to make something which:
  - is open and free
  - supports common ham radio needs (direct comm, repeaters, hotspots)
  - is portable across devices, both for end users and developers

- Currently WIP, an RFC open at https://github.com/OpenRTX/openrtx.github.io/pull/32

- Technical details:
  - binary format
  - up to 65,535 channels, contacts and banks (aka “zones”)
  - currently supporting FM, DMR and M17 operating modes

- May become a separate entity from the firmware
What’s next

- **Firmware side:**
- Codeplug!
- Event system
- APRS support
- ...

- **Hardware side:**
- M17’s OpenHT
- Baofeng DM-1701 (can do M17!)
- Yaesu FT-70 and “sisters”
- ...

S. Seva OpenRTX: an open source firmware for ham radio devices
What’s next

- Firmware side:
  - *Codeplug!*
What’s next

• Firmware side:
  • Codeplug!
  • *Event system*
What’s next

- Firmware side:
  - Codeplug!
  - Event system
  - *APRS support*
What’s next

- Firmware side:
  - Codeplug!
  - Event system
  - APRS support
  - ...
What’s next

• Firmware side:
  • Codeplug!
  • Event system
  • APRS support
  • ...

• Hardware side:
What’s next

- Firmware side:
  - Codeplug!
  - Event system
  - APRS support
  - ...

- Hardware side:
  - *M17’s OpenHT*
What’s next

• Firmware side:
  • Codeplug!
  • Event system
  • APRS support
  • ...

• Hardware side:
  • M17’s OpenHT
  • *Baofeng DM-1701 (can do M17!)*
What’s next

• Firmware side:
  • Codeplug!
  • Event system
  • APRS support
  • ...

• Hardware side:
  • M17’s OpenHT
  • Baofeng DM-1701 (can do M17!)
  • *Yaesu FT-70 and “sisters”*
What’s next

- **Firmware side:**
  - Codeplug!
  - Event system
  - APRS support
  - ...

- **Hardware side:**
  - M17’s OpenHT
  - Baofeng DM-1701 (can do M17!)
  - Yaesu FT-70 and “sisters”
  - ...

S. Seva
OpenRTX: an open source firmware for ham radio devices
Happy hacking!

https://openrtx.org
https://mastodon.radio/@openrtx
https://matrix.to/#/#openrtx:matrix.org