Radio Collar Tracker Technical Reference Manual

Nathan Hui, Project Lead
Engineers for Exploration, UC San Diego

March 7, 2019
v0.3

Contents

List of Figures

List of Tables

Listings

Chapter 1

Payload

1.1 UP Core

1.1.1 Locating the UP Core

The UP Core is currently configured to automatically connect to either the Sealab or UCSD-Protected networks. It has a hostname of e4e-UPCORE-1.

At current, the Sealab network is configured as a bridge to the UCSD network, so all devices can be accessed via hostname.dynamic.ucsd.edu.

The UP Core can always be accessed (on UCSD campus) at e4e-upcore-1.dynamic.ucsd.edu.

Due to this network configuration, the UP Core is accessible only via SSH public key authentication. To add a public key, append the appropriate key to `/home/e4e/.ssh/authorized_keys`.

1.2 UI Board

1.2.1 Flashing the UI Board

The UI Board currently uses an Atmel ATmega32u4. The best way to flash this is to use the Dragon ISP (In System Programmer) to load the program into flash. The recommended way to interact with the Dragon ISP for Atmel microcontrollers is by using `avrdude`. See Listing 1.2 for details on specific command line options.

For the UI Board, use the following options:

- `partno`: `m32u4`
- `bitclock`: `10`
- `programmer`: `dragon_isp`
- `port`: `usb`
- `memtype`: `flash`

For example, if we were to flash the file `foo.hex` to the UI Board, we would use the command in Listing 1.1.

Listing 1.1: Example avrdude command

```
avrdude -p m32u4 -c dragon_isp -B 10 -P usb -U flash:w:foo.hex
```

Listing 1.2: `avrdude` usage

```
Usage: avrdude [options]
Options:
  -p <partno>                 Required. Specify AVR device.
  -b <baudrate>               Override RS-232 baud rate.
  -B <bitclock>               Specify JTAG/STK500v2 bit clock period (us).
  -C <config-file>            Specify location of configuration file.
  -c <programmer>             Specify programmer type.
  -D                          Disable auto erase for flash memory
  -i <delay>                  ISP Clock Delay [in microseconds]
  -P <port>                   Specify connection port.
  -F                          Override invalid signature check.
  -e                          Perform a chip erase.
  -O                          Perform RC oscillator calibration (see
                              AVR053).
  -U <memtype>:r|w|v:<filename>[:format]
                              Memory operation specification.
                              Multiple -U options are allowed, each request
                              is performed in the order specified.
  -n                          Do not write anything to the device.
  -V                          Do not verify.
  -u                          Disable safemode, default when running from a
                              script.
  -s                          Silent safemode operation, will not ask you
                              if fuses should be changed back.
  -t                          Enter terminal mode.
  -E <exitspec>[,<exitspec>] List programmer exit specifications.
  -x <extended_param>         Pass <extended_param> to programmer.
  -y                          Count # erase cycles in EEPROM.
  -Y <number>                 Initialize erase cycle # in EEPROM.
  -v                          Verbose output. -v -v for more.
  -q                          Quell progress output. -q -q for less.
  -l logfile                  Use logfile rather than stderr for
                              diagnostics.
  -?                          Display this usage.
```

1.2.2 Compiling Programs for the UI Board

Arduino IDE

To use the Arduino IDE to compile programs, load the code into the Arduino IDE. In the IDE, set the `Board` option to `Arduino Leonardo`, and enable verbose output for compilation.

Compile the code by clicking the `Verify` button, or by selecting `Verify/Compile` from the `Sketch` menu. In the output area, look for the line that invokes `avr-objcopy` and outputs a file that hax the `.hex` extension. This is the file that needs to be uploaded to the UI Board using the instructions in Subsection 1.2.1.

avr-gcc

If the Arduino IDE is not available, or not an option, we can use the AVR compilation toolchain. This is a version of `gcc` that compiles specifically to the AVR instruction set.

In general, use `avr-g++` or `avr-gcc` to compile the individual object files as appropriate. Then use `avr-g++` to link all the object files and libraries into an `.elf` (Executable and Linkable Format). Finally, use

`avr-objcopy` to copy the text and data portions of the program into the appropriate locations in the `.hex` file (Intel Hexcode).

An example compilation may look like the following:

Listing 1.3: Example avr-gcc compilation

```
avr-g++ -c test.cpp -o test.o
avr-g++ -o test.elf test.o
avr-objcopy -j .text -j .data -O ihex test.elf test.hex
```

See the `ui_board` branch of the Radio Collar Tracker Github[1] for a makefile containing the rules to compile, link, and format the program.

1.2.3 UI Board Functionality

The UI Board shall gather data from the following sensors: GPS, compass, and user switch. It must transmit the data and state information to the UP Core as specified in Section 1.4.2. Simultaneously, the UI Board shall also receive status messages from the UP Core as specified in Section 1.4.1, and switch LEDs in the appropriate response. See Tables 1.1, 1.2, 1.3, and 1.4 for the state tables.

Table 1.1: LED 1 (System) State Table

System State	LED State
Init	Blinking (5 Hz)
Wait for Start	Off
Wait for End	Blinking (1 Hz)
Finish	Solid
Fail	Off

Table 1.2: LED 2 (Storage) State Table

Storage State	LED State
Init	Blinking (5 Hz)
Ready	Solid
Fail	Off
Retry	Blinking (1 Hz)

Table 1.3: LED 3 (SDR) State Table

SDR State	LED State
Init	Blinking (5 Hz)
Ready	Solid
Fail	Off
Retry	Blinking (1 Hz)

LED 5 shall be solid in and only in the following states: GPS is Ready, SDR is Ready, STR is Ready, and Sys is wait_start.

[1] https://github.com/UCSD-E4E/radio_collar_tracker_drone/blob/ui_board/src/arduino/Makefile

Table 1.4: LED 4 (GPS) State Table

GPS State	LED State
Init	Blinking (5 Hz)
Ready	Solid
Fail	Off
Retry	Blinking (1 Hz)

1.3 UI Board to GPS/Compass Protocol Specification

The GPS/Compass module is a DroTek Ublox GPS and Compass Module, configured with the XL patch antenna.

1.3.1 GPS

The GPS is a UBlox M8N. This has a 5V TTL UART configured with 8N1 frames and 9600 baud. Data is transmitted as a NMEA 0183 stream, with a nominal message rate of 1 Hz.

1.3.2 Compass

The compass is a Honeywell HMC5983 magnetometer[2]. This has an I2C bus, with an address of `0x3C`. Reference the datasheet for communication protocols.

1.4 UI Board to UP Core Protocol Specification

Data transmission shall take place as a software serial port over USB. This connection shall go from the CN7 connector on the UP Core to the P2 connector on the UI Board. This physical link shall comply with USB 2.0 standards.

The virtual serial port shall be configured as 8N1 with a baud rate of 9600.

Data shall be formatted as JSON dictionaries.

1.4.1 UP Core to UI Board

The UP Core shall transmit a status message at 1 Hz nominal frequency. This message shall be formatted as a JSON dictionary with the following keys: `STR`, `SDR`, `SYS`. See Listing 1.4 for an example message. See Table 1.5 for the status values. These keys are not guaranteed to come in any particular order.

Listing 1.4: UP Core Status Message

```
{"STR": 3, "SYS": 3, "SDR": 4}
```

Each module may enter any state at any time.

1.4.2 UI Board to UP Core

The UI Board shall act as a data aggregator for the UP Core, collecting telemetry and state information from the GPS, compass, and user switch. The UI Board shall then pass these data on to the UP Core as a JSON dictionary with the following keys: `lat`, `lon`, `hdg`, `tme`, `run`, `fix`, `sat`. These keys are not guaranteed to come in any particular order. See Table 1.6 for a description of the individual fields, and Listing 1.5 for an example message.

[2] https://aerocontent.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5983_3_Axis_Compass_IC.pdf

Table 1.5: Status Message Lookup Tables

Storage State	State Name	STR value
Init	STR_INIT	0
Ready	STR_READY	1
Fail	STR_FAIL	2
Retry	STR_RETRY	3

SDR State	State Name	SDR value
Init	SDR_INIT	0
Ready	SDR_READY	1
Fail	SDR_FAIL	2
Retry	SDR_RETRY	3

GPS State	State Name	GPS value
Init	GPS_INIT	0
Ready	GPS_READY	1
Fail	GPS_FAIL	2
Retry	GPS_RETRY	3

System State	State Name	SYS value
Init	SYS_INIT	0
Wait for Start	SYS_WAIT_FOR_START	1
Wait for End	SYS_WAIT_FOR_END	2
Finish	SYS_FINISH	3
Fail	SYS_FAIL	4

Listing 1.5: UI Board Sensor Message

```
{"lat": 327054113, "hdg": 270, "lon": -1171710165, "tme": 1530025076, "run
    ": true, "fix": 1, "sat": 14}
```

Table 1.6: Sensor Message Field Descriptions

Field	Key	Data Type	Description
GPS Latitude	lat	int	GPS Latitude in WGS84 datum, formatted as lat × 1e7
GPS Longitude	lon	int	GPS Longitude in WGS84 datum, formatted as lon × 1e7
Vehicle Heading	hdg	int	Vehicle Heading in degree East of Magnetic North
GPS Time	tme	int	GPS Timestamp in seconds since GPS epoch (06 Jan 1980)
User Switch	run	bool	True if the switch is in the ON position, False otherwise
GPS Fix Type	fix	int	GPS Fix, Differential GPS Fix, RTK Fix
GPS Satellite Count	sat	int	Number of satellites used in current fix

This sensor data shall be transmitted at a nominal rate of 1 Hz.

1.5 Payload Configuration

All configuration options are set in the file `/usr/local/etc/rct_config`. This file is owned by `root`, and should have permissions set to `644`.

1.5.1 Center Frequency

The center frequency for the SDR is set to CENTER_FREQ by the line `freq=CENTER_FREQ`. This is the center frequency the SDR is recording at, in Hz. Due to physical limitations of the SDR, set this to be at least 1 kHz away from the nearest frequency that needs to be measured.

1.5.2 Sampling Frequency

The sampling frequency for the SDR is set to SAMPLING_FREQ by the line `sampling_freq=SAMPLING_FREQ`. This is the sampling frequency that the SDR is recording at, in samples per second. Due to physical limitations of the SDR, this value can be only be set to between 200 kHz and 56 MHz in steps of 1 Hz. Ensure that all frequencies to be recorded are within half the sampling frequency away from the center frequency.

1.5.3 RF Gain

The RF gain for the SDR receive chan is set to RF_GAIN by the line `gain="RF_GAIN"`. This is the gain that is applied to the RF signal in the LNA stage in the SDR, in dB. Due to physical limitations of the SDR, this value can only be set to between 0 dB and 76 dB, in steps of 1 dB. Ensure that all recorded signals are not clipping with any new gain setting. Ideally, the loudest signal should result in an amplitude no greater than 80% of the dyanmic range of the SDR.

1.5.4 Initializing the USRP B200-mini(-i)

The USRP B200-mini and USRP B200-mini-i are not capable of retaining an FPGA image between boots. This image needs to be flashed every boot. This can be accomplished via command line.

```
sudo uhd_usrp_probe --args="type=b200" --init-only
```

Images can be downloaded on Linux systems via command line. This requires at least 200 MB of disk space.

```
sudo uhd_images_downloader
```

1.6 Payload Autostart

The payload is configured to automatically initialize itself on boot, and conduct sanity checks of its internal environment. In addition, the payload provides a simple switch interface to enable the user to start and stop recording data without needing to SSH into the payload. This system is comprised of two components - the User Control Board, and the payload autostart software.

1.6.1 User Control Board

The User Control Board is located at the front of the aircraft, just below the fuselage. The User Control Board has 5 LEDs. From left to right (when looking at the aircraft), the LEDs are Payload Status (Green), SDR Status (Yellow), Directory Status (Orange), GPS Status (Red), and Payload Ready (Blue). On the right of the board is the Payload Control Switch.

Status LEDs

The Status LEDs provide status information to the user. On boot, the payload conducts an internal check of the connected hardware to ensure proper operation. The status of these checks are denoted by the SDR Status LED, Directory Status LED, and GPS Status LED.

SDR Status LED The SDR Status LED denotes the initialization and standby status of the SDR. Upon boot, the payload checks for the presence and state of a USRP B200 series SDR. Upon detecting the SDR, the payload loads the firmware for the SDR. During this time, the SDR Status LED will be blinking, approximately once a second. Upon loading the firmware for the SDR, the payload will place the SDR in standby, whereupon the SDR Status LED will become solid. Should the payload fail to detect an SDR, or fail to upload the firmware for the SDR, the LED will turn off.

Directory Status LED The Directory Status LED denotes the detection and status of the output directory for data for the payload. Upon boot, the payload checks for the presence of the micro SD card. Unfortunately, due to hardware issues, the payload cannot detect a micro SD card that has been left in during boot, so the micro SD card must be remounted after boot. Upon detecting the micro SD card (device will enumerate as /dev/mmcblk0), the payload uses udev rules to mount the micro SD card to /mnt/RAW_DATA. After this, the payload will check the mounted filesystem for free space, to ensure that there is enough space for the next run. The minimum required space is 9 GB, which is enough for a 20 minute run. Should the payload fail to detect the micro SD card, or fail to detect enough free space, the LED will turn off. The payload will attempt to detect the micro SD card again in 10 seconds, trying for 60 seconds.

GPS Status LED The GPS Status LED denotes the detection and state of the GPS. Upon boot, the payload searches for and connects to the GPS serial device. The GPS device is expected to supply NMEA data at a baud rate of 9600. The payload then waits for the GPS device to gain a lock (defined as a 3D fix with greater than 6 satellites). Should the payload fail to find the GPS device, or fail to receive a fix message, the LED will turn off. If the payload is communicating with the GPS device and receiving fix messages, but the fix is not yet good enough, the LED will blink. When the payload has found the GPS device and has determined that the GPS has a good GPS fix, then the LED will stay on.

Payload Status LED The Payload Status LED denotes the recording state of the payload. The LED is blinking while the payload is actively recording data, and solid while the payload is finishing writing data to the SD card. The LED is off during standby and initialization.

Payload Ready LED The Payload Ready LED denotes the ready state of the payload. It is on only when all initialization checks have completed and the payload is standing by for the start command.

Control Switch

The Control Switch controls the recording state of the payload. The switch must be in the OFF position during boot. Flipping the switch to the ON position sets the payload to the recording state. The payload will begin recording immediately if the initialization is complete, or as soon as the initialization completes if not complete already. Flipping the switch to the OFF position stops the payload recording, at which point the payload finishes writing all recorded data to disk. When the data is recorded, the payload reinitializes the system, then goes into standby mode waiting for the ON position.

Payload Autostart Software

The payload autostart is controlled by the rctstart Linux service, and the autostart flag in `/usr/local/etc/rct\_config`. If the line in `/usr/local/etc/rct\_config` is autostart=false, then the payload service will not start. The service can also be stopped once started by calling sudo service rctstart stop.