FarmShield Firmware Outline

The spears will run an nRF52840 microprocessor which will require the following hardware drivers to read and control the various peripherals.

Shield HAL Drivers

- SPI driver
- I2C Driver
- UART
- ADC
- EEPROM

Shield Peripheral Application Layer

All the peripherals on the shield MCU will require modules atop the hardware abstraction layer. These modules will be made modular so that the code may be reusable regardless of any hardware changes in the future.

The following are the modules required for the Shield.

- Battery Management stack
- Modem Stack
- Firmware OTA stack
- Configuration storage and retrieval stack
- SD card management stack
- Radio Stack
- Data over radio TX/RX stack
- Message Queue and encryption stack
- Relay Control Stack
- Motor Controller stack
- Water Flow Meter stack
- Valve Control Stack
- Light Level Sensor stack

Shield System Application Layer

This section covers how all the various stacks come together to follow a routine of how the device works according to a process flow. This includes receiving and bundling together data from the various spears paired to the shield, encrypting and transmitting data to the cloud. The system application layer will also handle how the device interacts with the cloud and users.

Messaging Protocol

Maximum length of data is 256 bytes over MQTT protocol

Start	length	Msg type	direction	s/n	msg id	payload	end
0x3C	1 byte	1 byte	0x55/0x44	12 bytes	1 byte	variable	0x3E

Msg type

value	Type
0x44	Sensor data msg
0x53	Settings msg
0x45	Error msg
0x41	Message ACK
0x00	Unknown type

direction

value	direction
0x55	Upload to Server
0x44	Download from server

if message is of type **Message ACK** , **payload** section will not be available

Example serial number (**s/n**) in hex
`e00fce689a754705e79a0e37` split into bytes like so
`0xe0,0x0f,0xce,0x68,0x9a,0x75,0x47,0x05,0xe7,0x9a,0x0e,0x37`

msg id increments and overflows to start over from zero to prevent multiple processing of the same message

payload

Message payload is packaged in form of a string of tuples in the followinf format

`<tuple_id><length><data_bytes><tuple_id><length><data_bytes>...`

tuple member	size
tuple id	1 byte
length	1 byte
data bytes	Specified by **length** above

Currently supported tuple ids and their length

Tuple Name	Tuple ID	Length	Data Type	Data Multiplier	Description
MSG_ACK	0x00	1	byte		

Tuple Name	Tuple ID	Length	Data Type	Data Multiplier	Description
SPEAR_ID	0x01	12	hex sn number	-	Serial number expressed in hex string
STORE_TIMESTAMP	0x02	4	unsigned int32	-	unix time stamp
SEND_TIMESTAMP	0x03	4	unsigned int32	-	unix time stamp
SOIL_MOISTURE	0x04	2	unisgned int16	-	soil moisture (Not in use)
AIR_HUMIDITY	0x05	2	unisgned int16	0.1	Relative humidity in %
SOIL_HUMIDITY	0x06	2	unisgned int16	-	See Topic --> calculating soil data
WATER_DISPENSED	0x07	4	float	-	floating point value of water dispensed in liters
CARBON_DIOXIDE	0x08	4	int32	-	reading in ppm
AIR_TEMPERATURE	0x09	2	unisgned int16	0.1	temperature in *C
SOIL_TEMPERATURE	0x0A	2	unisgned int16	-	See Topic --> calculating soil data
SOIL_NITROGEN	0x0B	2	unisgned int16	-	in ppm (mg/kg)
SOIL_PHOSPHOROUS	0x1D	2	unisgned int16	-	in ppm (mg/kg)
SOIL_POTASIUM	0x1C	2	unisgned int16	-	in ppm (mg/kg)
LIGHT_INTENSITY	0x0C	2	int16	0.83333	Light intensity in Lux (if raw = -1, invalid, if raw = -2, sensor error)
SHIELD_BATTERY_LEVEL	0x0D	4	float	-	floating point value of % state of charge in shield
SPEAR_BATTERY_LEVEL	0x0E	2	unisgned int16	1.6117	Spear battery in millivolts
VALVE_POSITION	0x0F	1	boolean	-	1 = OPEN, 0 = CLOSED

Tuple Name	Tuple ID	Length	Data Type	Data Multiplier	Description
IGH_SEND_SETTINGS	0x10	Variable	byte stream	-	Update device settings using this tuple ID
IGH_READ_SETTINGS	0x11	Variable	byte stream	-	Request settings on device using this tuple ID
SPEAR_DATA	0x12	Variable	byte stream	-	Data from spear is wrapped using this overal tuple ID
SPEAR_RF_ID	0x13	2	unisgned int16	-	Tuple ID to change the Spear RF ID
SHIELD_RF_ID	0x14	2	unisgned int16	-	Tuple ID to change the Shield RF ID
SEND_INTERVAL	0x15	4	unisgned int16	-	Tuple ID to change the sendig interval
OP_STATE	0x16	1	byte	-	Tuple ID to change the Op state of the device
SHIELD_ID	0x17	12	hex sn number	-	Serial number expressed in hex string
SPEAR_BATT_LOW_THRESHOLD	0x18	2	unisgned int16	-	Tuple ID to change the spear battery low threshold
SHIELD_BATT_LOW_THRESHOLD	0x19	2	unisgned int16	-	Tuple ID to change the shield battery low threshold
BUTTON_PRESS	0x1A	1	boolean	-	How long button was pressed in seconds
SHIELD_FW_VERSION	0x1B	3	byte stream	-	FW version on each shield
SPEAR_FW_VERSION	0x1B	3	byte stream	-	FW version on each spear
SPEAR_SERIAL_SENSOR_TYPE	0x1E	1	byte	-	Tuple to change the serial sensor type (CO2 or NPK)
EVENT	0xFC	1	byte	-	Event data, see supported events
RESTART	0xFD	1	boolean	-	Restart event

Tuple Name	Tuple ID	Length	Data Type	Data Multiplier	Description
DATA_PKT	0xFE	Variable	byte stream	-	Start of data packet
END_OF_PKT_ID	0xFF	-	byte	-	End of packet

CALCULATING SOIL DATA

To calculate soil data, use the following equations

```
temperature = D1 + D2 * raw_temperature_reading;
linearHumidity = C1 + C2 * raw_humidity_reading + C3 * raw_humidity_reading *
raw_humidity_reading;
correctedHumidity = (temperature - ROOM_TEMPERATURE) * (T1 + T2 *
raw_humidity_reading) + linearHumidity;

Where:
D1                      = -39.66
D2                      = 0.01
raw_temperature_reading = the integer value after processing the payload bytes
C1                      = -2.0468
C2                      = 0.0367
C3                      = -1.5955e-6
T1                      = 0.01
T2                      = 0.00008
ROOM_TEMPERATURE        = 25
raw_humidity_reading    = the integer value after processing the payload bytes

Temperature is in *C while corrected humidity is in %

Any readings calculated above 98 should be considered as 100% humidity
```

SPEAR_SERIAL_SENSOR_TYPE

To change the serial sensor type to either NPK or CO2 sensor, you will need to send this tuple to the spear. The default sensor type is the CO2 sensor.

```
SERIAL_SENSOR_NPK = 0x01
SERIAL_SENSOR_CO2 = 0x02
```

Example tuple 1E0102 ---> will change the sensor type to CO2 type

changing the serial sensor type requires a device reset to take effect

EVENTS

Here is a list of supported events and their values. Events are packaged in tuples with the `EVENT` tuple ID, a length of `one` followed by the event id that occured.

Event Name	Event ID
EVENT_DEVICE_RESTART	0x50
EVENT_SD_CARD_ERROR	0x51
EVANT_BUTTON_IRRIGATION_ON	0x52
EVANT_BUTTON_IRRIGATION_OFF	0x53
EVENT_VAVLE_OPENED	0x54
EVENT_VAVLE_CLOSED	0x55
EVENT_IRRIGATION_SUSPENDED	0x56
EVENT_IRRIGATION_RESUMED	0x57
EVENT_IRRIGATION_ENABLED	0x58
EVENT_IRRIGATION_DISABLED	0x59
EVENT_MQTT_ERROR	0x5A
EVENT_UNKNOWN_MQTT_CMD	0x5B
EVENT_CMD_SENT_TO_WRONG_DEVICE	0x5C
EVENT_SETTINGS_UPDATE_SUCCESS	0x5D
EVENT_SETTINGS_UPDATE_FAIL	0x5E
EVENT_CALL_HOME	0x5F
EVENT_RESET_IRRIGATION	0x60
EVENT_SYSTEM_RESET	0x61
EVENT_INVALID_SOIL_DAT	0x62

IGH_READ_SETTINGS

In order to get the current settings in a device, the cloud platform must send a message with and **IGH_READ_SETTINGS** tuple in the payload.
The read settings tuple should be in the following format:
`<IGH_READ_SETTINGS><length><Settings_subid><Settings_subid>...`
The settings subid are listed in the table below.

The device will respond with the requested settings in the next payload in the following format:
`<IGH_READ_SETTINGS><total_length><Settings_subid><length><data><Settings_subid><length><data>...`

SETTINGS SUBID TUPLES

Subid Name	Tuple ID	Length
SUBID_OPSTATE	0x01	1
SUBID_REPORTING_INTERVAL	0x02	4
SUBID_DATA_RESOLUTION	0x03	4
SUBID_SET_SERIAL_NUMBER	0x04	12
SUBID_MQTT_BROKER	0x05	Variable
SUBID_MQTT_BROKER_PORT	0x06	2
SUBID_TIMEZONE	0x07	1
SUBID_IRRIGATION_HR	0x08	1
SUBID_WATER_DISP_PERIOD	0x09	4
SUBID_MQTT_USERNAME	0x0A	Variable
SUBID_MQTT_PASSWORD	0x0B	Variable
SUBID_WATER_AMOUNT_BY_BUTTON	0x0C	4
SUBID_AUTO_IRRIGATION_TYPE	0x0D	1
SUBID_CLOCK_IRRIGATION_INTERVAL	0x0E	4
SUBID_SOIL_MOISTURE_LOW	0x10	2
SUBID_AIR_HUMIDITY_LOW	0x11	2
SUBID_SOIL_HUMIDITY_LOW	0x12	2
SUBID_CARBON_DIOXIDE_LOW	0x13	2
SUBID_AIR_TEMPERATURE_LOW	0x14	2
SUBID_SOIL_TEMPERATURE_LOW	0x15	2
SUBID_SOIL_NPK_LOW	0x16	2
SUBID_LIGHT_INTENSITY_LOW	0x17	2
SUBID_SHIELD_BATTERY_LEVEL_LOW	0x18	2
SUBID_SPEAR_BATTERY_LEVEL_LOW	0x19	2
SUBID_DAILY_WATER_DISPENSED_MIN	0x1A	4
SUBID_SOIL_MOISTURE_HIGH	0x30	2
SUBID_AIR_HUMIDITY_HIGH	0x31	2
SUBID_SOIL_HUMIDITY_HIGH	0x32	2
SUBID_CARBON_DIOXIDE_HIGH	0x33	2
SUBID_AIR_TEMPERATURE_HIGH	0x34	2

Subid Name	Tuple ID	Length
SUBID_SOIL_TEMPERATURE_HIGH	0x35	2
SUBID_SOIL_NPK_HIGH	0x36	2
SUBID_LIGHT_INTENSITY_HIGH	0x37	2
SUBID_SHIELD_BATTERY_LEVEL_HIGH	0x38	2
SUBID_SPEAR_BATTERY_LEVEL_HIGH	0x39	2
SUBID_DAILY_WATER_DISPENSED_MAX	0x3A	4

IGH_SETTINGS

The **IGH_SETTINGS** tuple is used to send new settings down to the device in the following byte stream format using the SUBIDs listed above.

```
<IGH_SEND_SETTINGS><total_length><Settings_subid><length><data><Settings_subid>
<length><data>...
```

SUBID_AUTO_IRRIGATION_TYPE

This tuple can be used to set a device to either irrigate by the hour or using data from the sensors. If hourly irrigation is chosen, the device will dispense the same amount of water dispensed by a button press or a sensor reading once every hour. When irrigating by sensor readings, the device will irrigate each time it gets valid soil humidity sensor data. there is a one hour cool down between sensor readings to give dispensed water enough time to sip into the soil before adding more water. Water will stop flowing if the total amount of water hits the upper limit of water to dispense in a day.

To change the auto irrigation type, send the following hex bytes array; **10,03,0D,01,01** for *Sensor irrigation* **10,03,0D,01,02** for *Hourly irrigation*

hourly irrigation is set as the default type of auto irrigation

MQTT Protocol

Each IGH device shall connect to the MQTT broker `farmshield.illuminumgreenhouses.com` at port `1883`. The following are credentials for this broker:

```
Username = shields
Password = 940610b43b1
```

The devices will use their serial number in hex string as their IDs and subscribe to a download topic but publish to a data topic.

The topics shall be the device serial number with either `44` for download or `55` for upload appended at the beginning of the string.

For example;

Downlaod Topic
`44e00fce689a754705e79a0e37`

Upload Topic
`55e00fce689a754705e79a0e37`

for device with serial number `e00fce689a754705e79a0e37`.

`(All the parameters can be updated via settings except for the Download and Upload topics which are automatically generated by the device.)`

All messages sent to the device must be channeled throu the respective download topic and the device must send any payload through its respective
Upload topic.
The Messages published over MQTT must be in the format expressed above.

WATER IRRIGATION LOGIC

Button Control Logic

The button will now open or close the valve if pressed for 3 seconds or more. Pressing and holding the button for longer than ten seconds will disable or enable auto irrigation.

NOTES:
100 litres of water - *Default value, can be changed using settings* ***SUBID_WATER_AMOUNT_BY_BUTTON***
30 minutes - *Default value, can be changed using settings* ***SUBID_WATER_DISP_PERIOD***
300 litres of water - *Default value, can be changed using settings* ***SUBID_DAILY_WATER_DISPENSED_MIN***
800 litres of water - *Default value, can be changed using settings* ***SUBID_DAILY_WATER_DISPENSED_MAX***
0600Hrs - *Default value, can be changed using settings* ***SUBID_IRRIGATION_HR***

OTA Firmware Update for the IGH Shield

1. Open PowerShell or CMD from the folder with the new binary image
2. Type in `particle flash device_name igh_shield_v17.bin` then press enter

Note: *The device name can be found on the device cosole as part of the device information next to the ID. This is the name given to the device during registration process of the Boron. Before sending an OTA image, check that the device is online from the device console. Sending this command when the device is offline may sometimes be unreliable as the image is not stored permanently on the server.*

Running the InitialState Application

Prerequisites

- Python3 installed on machine
- run `pip install -r requirements.txt` on your command line

1. Download mqtt_snooper.py from the scripts colder of this repositofy
2. Run the python script, running it on a dedicated machine like a raspberry pi and get it to run every time on bootup would be ideal.
3. Log onto initialstate using company credentials to view data.

NOTE: every new device will automatically start sending data to initialstate each time a new message is published to MQTT

EXTERNAL 3rd PARTY SIM Vs INTERNAL PARTICLE SIM

It is not possible to dinamically switch between the internal and external sim cards on run time. Choosing between the two sim cards can be done by editing the `simcard.inc` file in the `src` folder of this application by applying either the constant variable INTERNAL_SIM or EXTERNAL_SIM and triggering a build of the system. Switching between the two sims may require a full power cycle of the devices.

Upgrading to later pre-release versions

Settings between pre-release version before and after V0.0.30 are different in structure and may conflict. In order to switch from an older version to V0.0.30 and above, you must clear out the settings in memory manually You can clear the memory settings by doing the following:

1. Hold down the irrigation button
2. Reset the Boron via the reset button while still holding down the irrigation button
3. The following prompt will show `HOLD DOWN BUTTON FOR 5 MORE SECONDS TO CLEAR EEPROM` followed by fullstops to help count the seconds
4. After holding the button down for at least 5 seconds after the prompt, release the button.
5. There will be a prompt stating `EEPROM CLEARED` if successful

This will result in the systm defaults being set as the running settings and will require sending of the required settings to update them. This means that the Device serial number will be lost along with connectivity settings for MQTT and RF module.