SpiderMesh IDE

Introduction

SpiderMesh IDE is a software that provides full-fledged access to the functionality of SpiderMesh cooperative mesh network. The features are: configuration of all radio modules parameters through a convenient interface, access to the serial port communication on a console, a scripting environment to program, compile and upload an embedded Virtual Machine. This document is intended for users who are new to the software.

Getting Started

In a SpiderMesh mesh network, nodes communicate with a gateway (the network coordinator). Therefore, one radio module has to be configured as a gateway. Note that the radio module is the same, no matter the purpose (end node or gateway). Radio modules are set to end-node operation at the factory.

When launching Spidermesh IDE (Figure 1), some of the features will be hidden/unavailable if no gateway is connected to the local serial port. As a result, the first step in setupping the radio modules is to connect one of them to the local serial port.

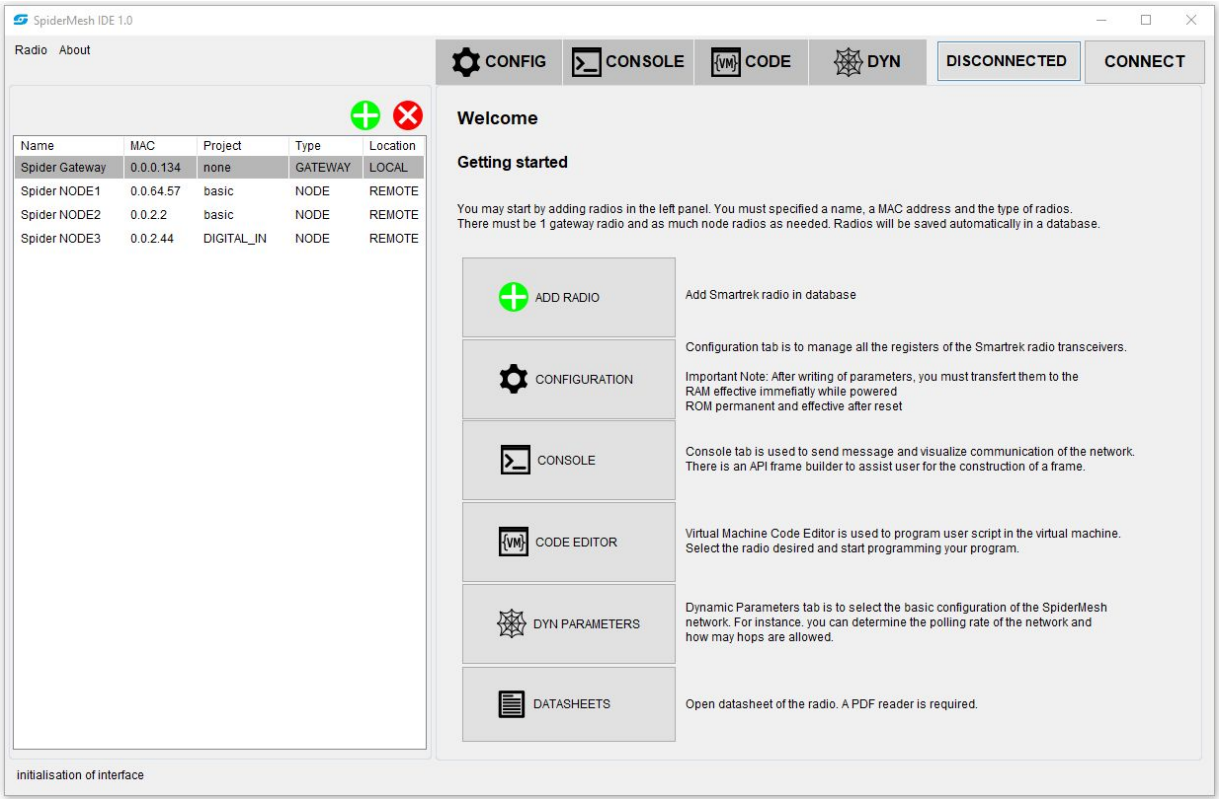


Figure 1. Getting started with SpiderMesh IDE

Connecting a Radio Module to the Local Serial Port

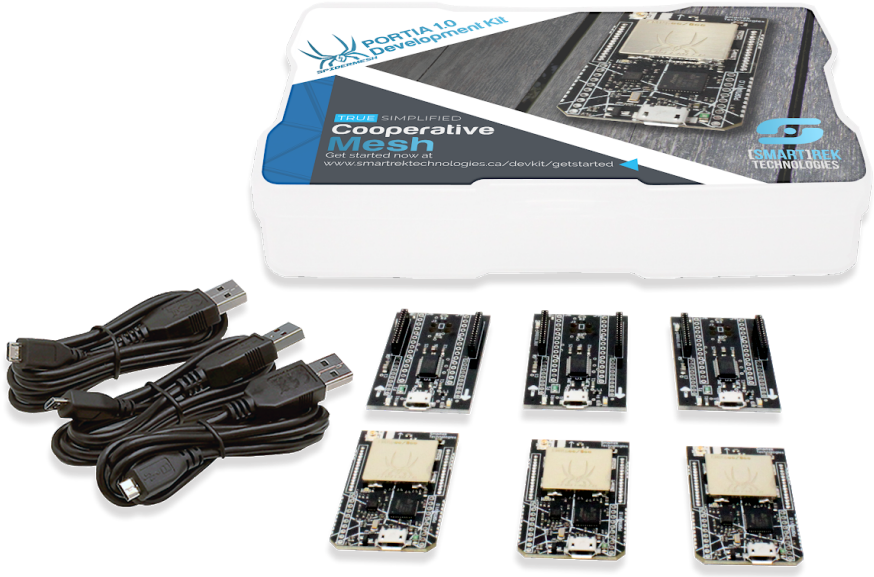
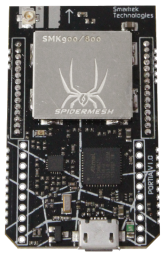


Figure 2. Portia radio module development kit

Table 1. Required equipment

Portia Radio Module	FTDI breakout board	Micro USB cable
		

Connect the Portia radio module to the FTDI breakout board with both USB connectors aligned (facing the same direction), as shown in Figure 3. Then, connect the breakout board to the PC running SpiderMesh IDE.

Figure 3. Connecting a radio module to the local serial port

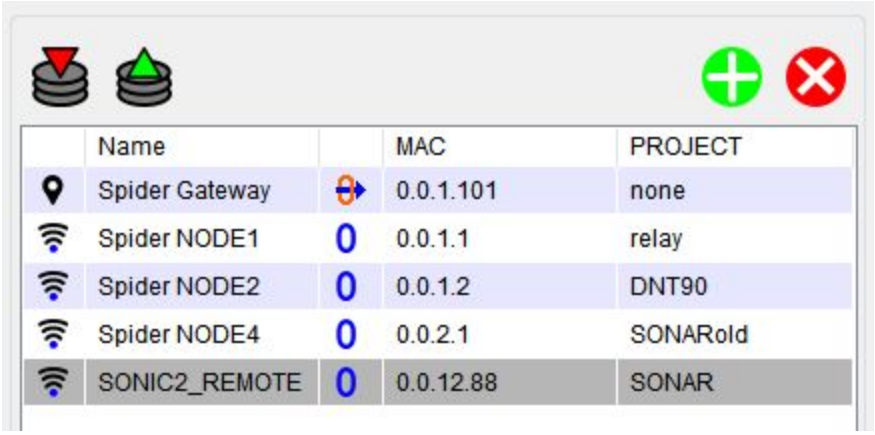
Radio Modules List

Once radio modules have been added to SpiderMesh IDE (see add a new radio button below, in Table 2), SpiderMesh IDE will allow local or OTA (over-the-air) commands to be executed.

By default, the software will save new radio modules in the local database.db database located in the root directory of the program. This file will be automatically opened if available. Figures 4 and 5 show the radio modules list view as well as the parameters to be entered when adding a new module. Table 3 describes these parameters.

Table 2. The Buttons in the Radio Modules List

Button	Description
	Import database
	Export database
	Add new radio module
	Remove radio module



Name		MAC	PROJECT
Spider Gateway		0.0.1.101	none
Spider NODE1	0	0.0.1.1	relay
Spider NODE2	0	0.0.1.2	DNT90
Spider NODE4	0	0.0.2.1	SONARold
SONIC2_REMOTE	0	0.0.12.88	SONAR

Figure 4. The Radio Modules List

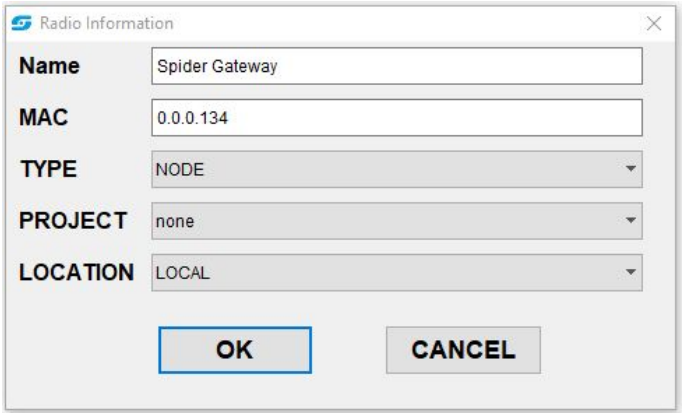


Figure 5. The required information when adding a new module

Tableau 3. Radio module informations

Settings	Description
Name	Radio module description
MAC	The MAC address of the radio module can be found either on the label affixed to the radio itself or in the manual provided with the device. A valid MAC address must follow this format (0-255 . 0-255 . 0-255 . 0-255)
Type	This field will not affect the configuration in the register. This information will only be used to sort your radio modules by type.
Project	Although optional, a VM project can be assigned to a specific radio module. This project can be edited and compiled in the provided VM Code Editor.
Location	The location determines whether the radio module connects locally or remotely on the serial port.

Connecting to the Serial Port

With a connected and powered radio module, the CONNECT button can be activated. Once pressed, the menu (Figure 6) prompts the user to configure the serial port.

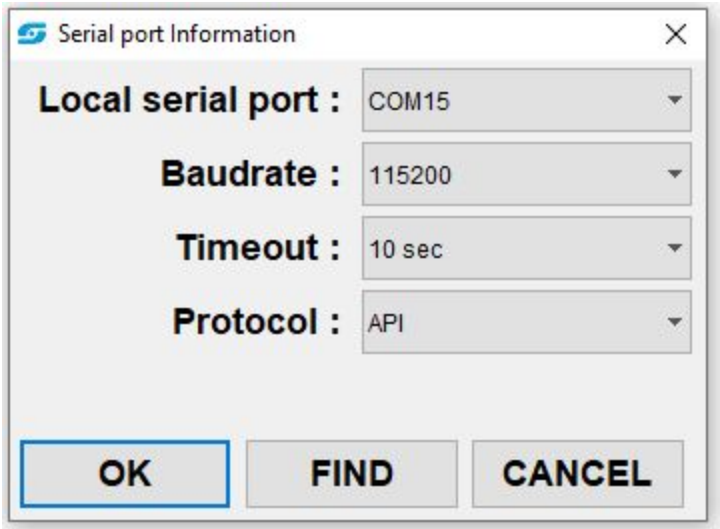


Figure 6. Configuring the serial port

Tableau 4. Serial port settings

Settings	Description
Local Serial port	Select the desired port. If the port to be used is unknown, then use the FIND button* to launch a scan to automatically configure the port and baudrate.
Baudrate	Default baudrate: **115200**.
Timeout	Maximum time allowed for the reception of an expected response. The timer starts from the time the command requiring an answer is sent.
Protocol	Communication protocol selection: - Transparent - API

*Note: When a gateway is on and tweeting near the nodes (radio modules) to be configured, then the FIND button is available and allows automatic search for the serial port connected to

the radio module (locally connected). However, it should be noted that this search will only be functional if the node (or radio module) is within reach of the gateway.

Communication Protocol

The SpiderMesh IDE software allows the user to choose between two communication protocols. For more information on these two modes, please refer to the technical datasheet of the Portia radio module.

Transparent

This protocol allows message broadcasting to all radios in the network. The gateway coordinates information exchanges in an invisible way while also acting as a node to send and receive messages.

Note: When powering up the radio, the communication protocol is in transparent mode. Then the software modifies this mode once the serial port is connected.

API

This protocol allows access to all the features of the SpiderMesh network. Communication between two points or to all nodes is then made possible. In this mode, radio modules can be configured and Virtual Machine code can be uploaded.

Radio Module Configuration

Radio modules are configured as end-nodes by default. Therefore, one radio module must be reconfigured into gateway mode in order to coordinate the network that will be deployed. This configuration is done in the Radio Configuration tab (Figure 7). Several other settings are also made available to the user so that the mesh network can be tailored to any specific needs.

Table 5 presents a summary of the basic settings for the configuration of a SpiderMesh network.

Table 5. Mesh network settings

Settings	**Description**
Node Type	*Gateway or end-node* (default: end-node)
Network ID	Channel to filter system data on the same hop table. It acts as a basic filter. (range => 0 to 7 - defaut 0)
Hop Table	Frequency hopping sequences (range => 0 to 5 - defaut 0)
Uart Baudrate	Communication speed between the local serial port and the radio module. (defaut 115200)

Changes to the registers can be made in the Configuration tab.

The configuration tab features (shown at Figure 7):

1. Selected radio module information
2. Save settings into database
3. Write into temporary memory
4. Read EEPROM memory
5. Settings status LED indicator
6. Settings status details
7. Read all registers
8. Reset factory settings
9. Reset software
10. Transfer temporary memory written data to the selected memory type
11. Memory type selection

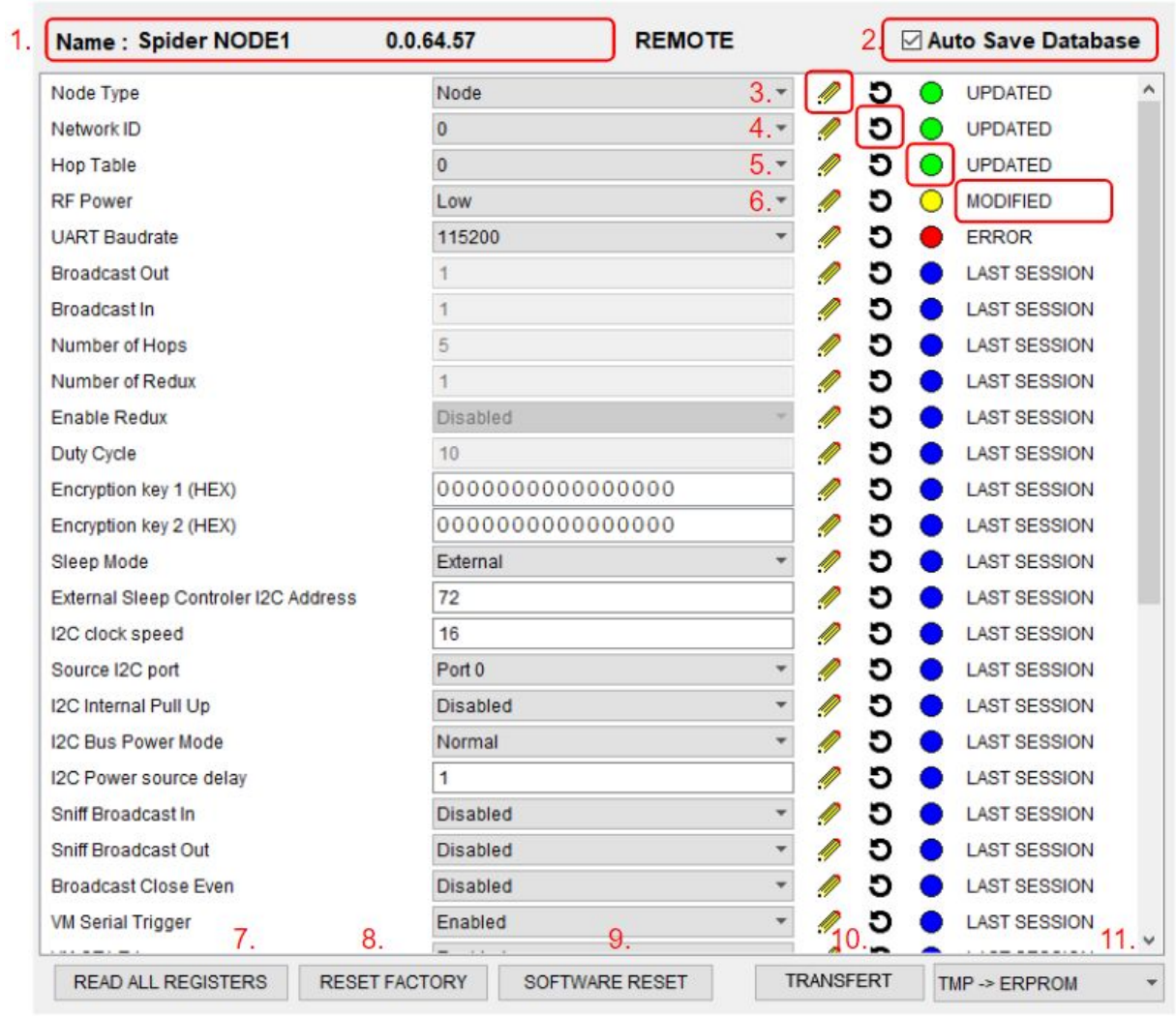


Figure 7. Configuring registers in the configuration tab

Console

The console (Figure 8) is a tool to view the received communication frames. It also offers a frame builder. Communication can be displayed in hexadecimal, ASCII or decimal format. All exchanges on the serial port between the radio module and the software are made visible in the console tab.

The Console tab features (as shown in Figure 8):

1. Selected radio module information
2. Frame builder
3. Transmissions in hexadecimal format
4. Send transmission on the serial port button
5. Address information
6. Type of displayed data
7. Auto-scroll console
8. Clear console
9. Console box

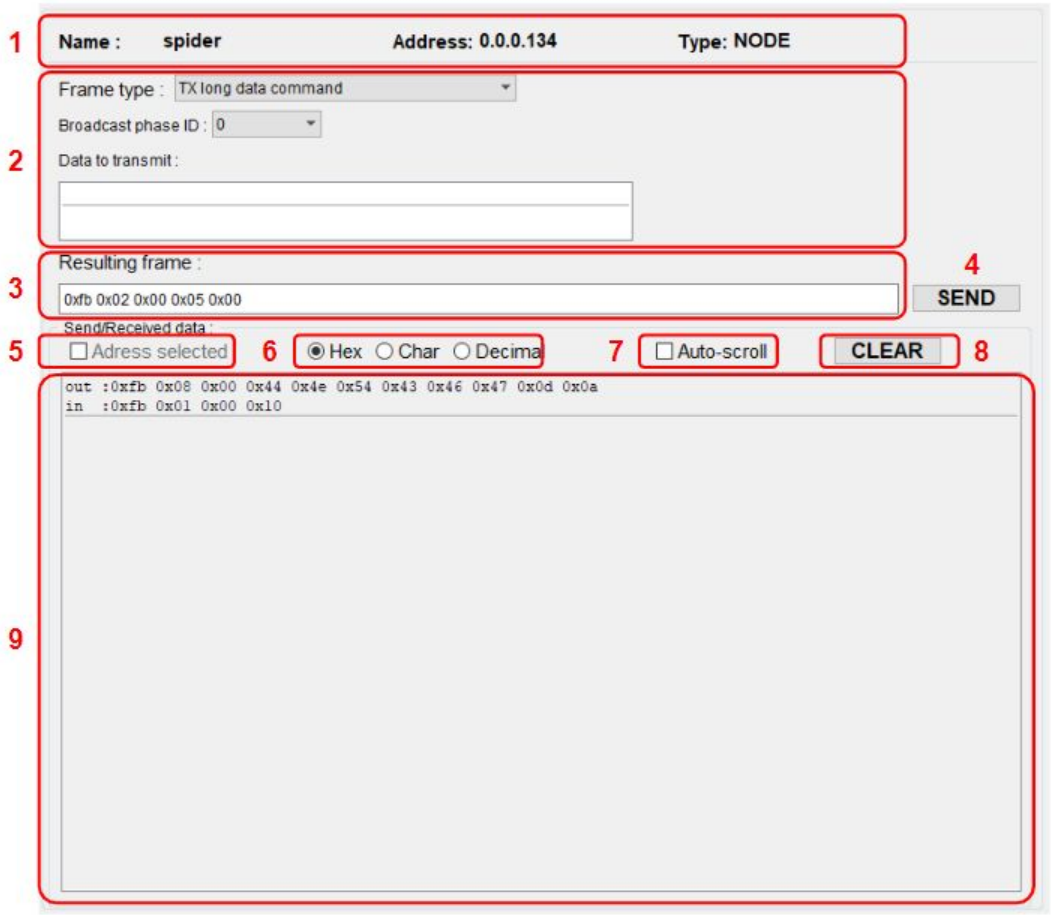


Figure 8. The Console tab is a tool to visualize exchanges between devices

VM Code Editor

This VM Code Editor tab (Figure 9) is a programming and compiling tool for the generation of virtual machine code. To access peripherals, the following functions are made available. Note that all peripherals have to respect the min/max = 0V - 3.3V requirement.1x TTL Port Série, 13x GPIOs, 1x SPI, 1xI2C, 2x canaux ADC

GPIO	13x input or output, with option pull up, pull down or disconnected.
Port série	1x Uart TTL
I2C	1x Two-wire interface
SPI	1x Serial Peripheral Interface
ADC	2x Analog to Digital Converter
DAC	1x Digital to Analog Converter

The SMK900.evi file contains the allowed functions. The latter are described in the VM functions section.

The VM Code Editor tab features (shown in Figure 9):

1. Script compiling
2. Script compiling and upload to the radio module
3. Open existing project
4. Create new project
5. Remove existing project
6. Import files
7. Delete files
8. Selected radio module information
9. Script editor
10. Console (compiler messages)

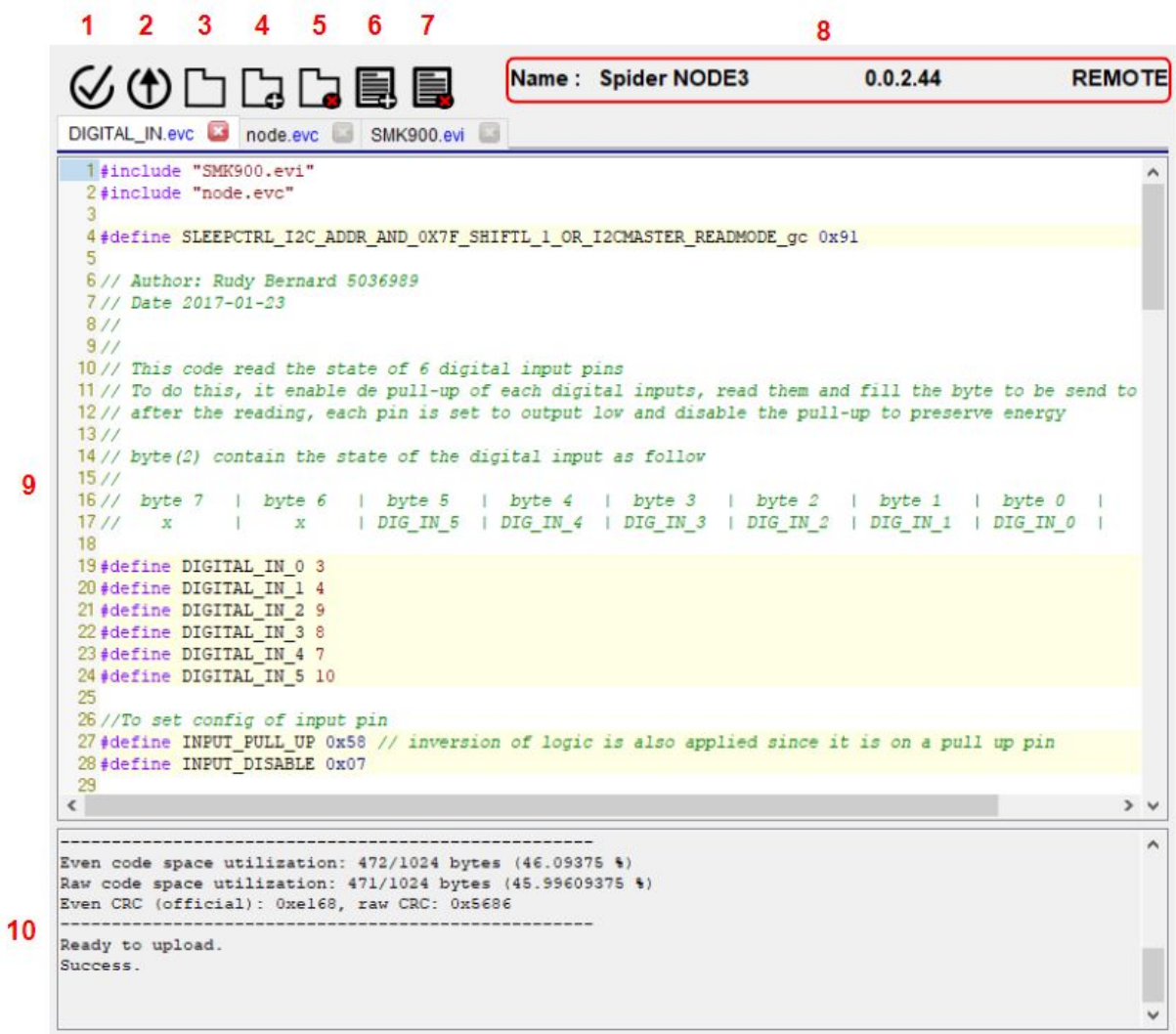


Figure 9. The VM Code Editor tab

Creating a VM project

When a new VM project is created (Figure 10), then it will be automatically made available for upload to all radio modules. The same project can hold several files. To add a new file, simply click on the ADD button in the toolbar. This file will then open in a new tab.

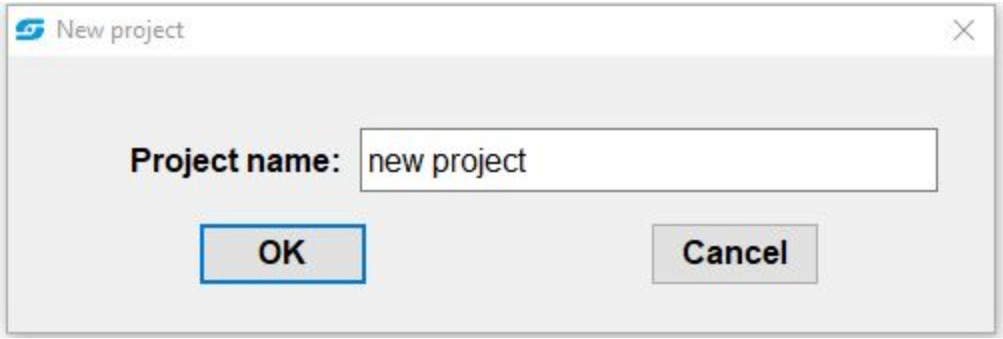


Figure 10. Creating a new VM project

DYN Parameters

The DYN parameters (Figure 11) will determine how the SpiderMesh mesh network behave. The Dyn Parameters tab let the user play with the various settings of the network. The software predicts the performance of the network from these configurations. For more information on DYN settings, please refer to the Portia radio module data sheet.

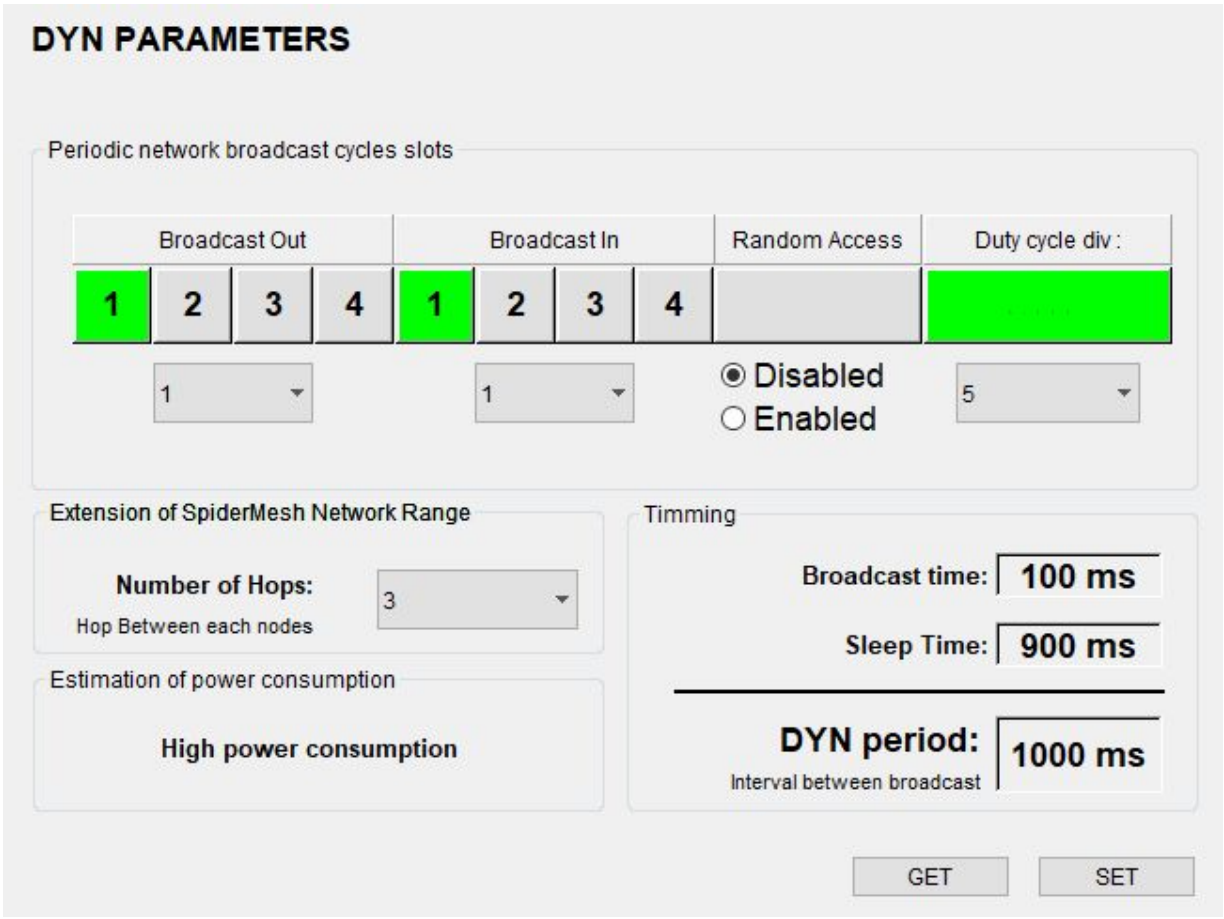


Figure 11. DYN parameters

VM Implementation Example

Example of a VM code of a temperature (RTD probe) using:
Communication via Serial Peripheral Interface (SPI).
An implementation of BITBANG (spi_write() and spi_read()).
A main register to store, perform operations and send data.

At boot-up, the node (in addition to the standard boot-up) initializes the direction of the PINs and their initial states.
Following an air command from the gateway, the node fetches the voltage value via I2C and configures the probe (SPI). Once ready (PIN DTRDY high), data is read (SPI). Data is then processed (error code is fetched and a Resistance to Temperature conversion is made) using the main register buffer (R) in order to perform the various mathematical operations. Finally, the final data is saved in the register buffer and sent.

```
#include "refdes_trx.evi"

#define SLEEPCTRL_I2C_ADDR_AND_0X7F_SHIFTL_1_OR_I2CMASTER_READMODE_gc 0x91
//#define USE_BITBANG

// Defining the PINs used to communication with the probe
#define ONOFF 5
#define DTRDY 6
#define CS 7
#define SDI 8
#define SDO 9
#define SCK 10

// Defining constants containing the data location in the main register buffer//
// Is used for calculation
#define R_TEMPERATURE 0
#define R_RESISTANCE_TMP 4
#define R_TMP 8
#define R_TMP_RC 12
#define R_TMP_1RC 16
#define R_TMP_2RC 20

// Defining SPI control functions
#define spi_start() SetPinOut(CS,0)
#define spi_stop() SetPinOut(CS,1)
#define spi_dly() Delay(2)

// Function to save contiguous data
function transferBytesManuallyToReg(r, lsbyte0, byte1, byte2, msbyte3){
        SetBuffer(r,lsbyte0&0xFF,1);
        SetBuffer(r+1,byte1&0xFF,1);
        SetBuffer(r+2,byte2&0xFF,1);
        SetBuffer(r+3,msbyte3&0xFF,1);
}
```

```
// SPI write function
// Uses bitbang or register, depending on configuration
function spi_write(value){
#ifdef USE_BITBANG
	local i;
	for(i=0;i<8;i++){
		if(value & 0x80){
			SetPinOut(SDI,1);
		}
		else{
			SetPinOut(SDI,0);
		}
		SetPinOut(SCK,0);
		value <<=1;

		spi_dly();
		SetPinOut(SCK,1);
	}
#else
	SetPerReg(_SPIC_DATA, value);
#endif
	SetPinOut(SDI,1);
}

function spi_write2(value1,value2){
	spi_start();
	spi_write(value1&0xFF);
	spi_write(value2&0xFF);
	spi_stop();
}

// SPI read function
// Uses bitbang or register depending on configuration
function spi_read(){
#ifdef USE_BITBANG
	local i;
	local out;
	out=0;
	for(i=0;i<8;i++){
		SetPinOut(SCK,0);

		spi_dly();
		SetPinOut(SCK,1);
		out <<=1;
		out += GetPinIn(SDO);
	}
	return out;
#else
	SetPerReg(_SPIC_DATA, 0xFF);
	return GetPerReg(_SPIC_DATA);
#endif
}

// Pin status function
function turnPINOut(stateSPI, stateHead){
	// turn spi ON/OFF
	SetPinOut(CS,stateSPI);
	SetPinOut(SDI,stateSPI);
	SetPinOut(SCK,stateSPI);

	// turn sensor head ON/OFF
	SetPinOut(ONOFF,stateHead);
}
```

```
// PIN initialization function (for SPI)
function spi_init(){
	// Pin Initialization (Input/Output)
	SetPinDir(ONOFF,1);
	SetPinDir(CS,1);
	SetPinDir(SCK,1);
	SetPinDir(SDI,1);
	SetPinDir(SDO,0);
	SetPinDir(DTRDY,0);

	// Pin status initialization
	turnPINOut(1,0);

#ifdef USE_BITBANG
#else
	SetPerReg(_SPIC_CTRL,0x5F);

	spi_start();
	spi_read();
	spi_stop();
#endif
}

// Operation simplification function
// Divides the content of the main register at offset R_TEMPERATURE by the content of the
main register at offset R_TMP and then adds the value tmp
function DivAddBuffer_32(tmp){
	DivBuffer_S32(R_TEMPERATURE,R_TMP);
	AddBuffer_32(R_TEMPERATURE,tmp);
}

// Bootup specific function
// Using a standard bootup function as defined in "Example VM User Script using
// I2C Functions"
function exec_bootup_extended(){
	exec_bootup_std();
	spi_init();
}

// Main function called at the reception of an aircmd
// Triggers a read operation
// For the specific temperature-rtd-probe application
//  - Ports/pins initialization
//  - Fetch voltage
//  - Probe configuration (SPI write)
//  - Wait for a probe ready signal. This is done by checking the DTRDY status
//  - Acknowledge (spi_write(0x01)) then read data (spi_read())
//  - Separation of the error code from the rest of the data
//  - Conversion of resistance to temperature (using the formula below and the main register
//buffer (function [opération]Buffer_[type]) )
//  - write to buffer
//  - send data
function exec_aircmd(){
	local V,R,T1,T2,tmp;

	//Variable pour serie de TAYLOR
	local lRC;		// resistance value for the center of the Taylor series
	local lT3RC;

	// 0x10=PORT_OPC_PULLDOWN_gc 0x00=PORT_OPC_TOTEM_gc
	SetPerReg(_PORTB_PIN2CTRL,0x10);//ONOFF:PIN9 ->IO_5->B2
	SetPerReg(_PORTB_PIN3CTRL,0x10);//DTRDY:PIN13 ->IO_6->B3
```

```
        SetPerReg(_PORTC_PIN4CTRL,0x10);//CS:PIN22 ->IO_7->C4
        SetPerReg(_PORTC_PIN5CTRL,0x10);//SDI:PIN21 ->IO_8->C5
        SetPerReg(_PORTC_PIN6CTRL,0x10);//SDO:PIN20 ->IO_9->C6
        SetPerReg(_PORTC_PIN7CTRL,0x10);//SCK:PIN23 ->IO_10->C7

        //turn PIN Out ON
        turnPINOut(1,1);

        //time to read external sleep controller voltage using I2C bus
        if(I2C_Start( SLEEPCTRL_I2C_ADDR_AND_0X7F_SHIFTL_1_OR_I2CMASTER_READMODE_gc ) ){
//failure
                V=0;
        }
        else{
                V = I2C_ReadNak();
        }
        I2C_Stop();

 // Probe configuration using SPI
        spi_write2(0x80,0x92);
        spi_write2(0x80,0x90);
        spi_write2(0x80,0xB0);

        // wait for probe to be ready
        tmp=0;
        while(tmp<1000 && GetPinIn(DTRDY)==1){
                tmp += 1;
                Delay(1);
        }

        if(tmp>=1000){
                tmp=0x01;
                T1=0x0000;
                T2=0x0000;
        }else{
                //Read RTD Register -> response : [MSB] [LSB][FAULT]
                spi_start();
                spi_write(0x01);
                R = (spi_read() << 8)+spi_read();// = [RTD MSB REGISTER] 8 bit + [RTD LSB
REGISTER] 7 bit + [FAULT] 1bit
                spi_stop();

                tmp = (R & 0x01); //  D0 of the RTD LSBs register is a Fault bit that
indicates whether any RTD faults have been detected.
                R >>=1; //  Resistance to convert in Temperature

                // Measured resistance R conversion to temperature T //
                //initial algorithm:
                //R = rtd;
                //RREF = 430;
                //RTD_A = 3.9083e-3;
                //RTD_B = -5.775e-7;
                //RTDnominal = 100;
                //Rtmp = R*RREF/(2^15);
                //Z1 = -RTD_A;
                //Z2 = RTD_A^2 - 4*RTD_B;
                //Z3 = 4*RTD_B / RTDnominal;
                //Z4 = 2*RTD_B;
                //T = (sqrt(Z2 + Z3 * Rtmp) + Z1) / Z4

                //A = RREF / (RTDnominal * RTD_B * 2^15)
                //B = (RTD_A / (2*RTD_B)) * (RTD_A / (2*RTD_B)) - (1/lRTD_B);
                //C = -RTD_A / (2*RTD_B)
```

```
               //T = -sqrt(A*R + B) + C
               //T' = -A / (2*sqrt(A*R+B))
               //T'' = A^2 / (4 * (A*R+B)^(3/2))
               //T''' = -3*A^3 / (8 * (A*R+B)^(5/2))

               // TAYLOR
               // T(r) = T(rC) + (r-rC) * (T'(rC) + (r-rC) * (T''(rC)/2 + (r-rC) *
(T'''(rC)/6)));

               //int32_t lCoeffMultiTemp = 1000; // Multiplier coefficient of the output
temperature (Tréel=12.34 -> Talgo=12.34*Coeff=12340)
               // used to get the decimal part of the temperature
               //int32_t lCoefPre  = 10000000;     // to increase algo precision (global)
               //int32_t lCoefPre2 = 200000;       // to increase algo precision (for const
lT2RC and lT3RC)

               //R_TMP_RC     = T(rc) temperature of the center of the taylor series
               //R_TMP_1RC    = T'(rC) * lCoefPre
               //R_TMP_2RC    = T''(rC)/2 * lCoefPre * lCoefPre2
               //lT3RC               = T'''(rC)/6 * lCoefPre * lCoefPre2

               //tmp3  = R_RESISTANCE_TMP * T3RC;
               //tmp2  = R_RESISTANCE_TMP * (R_TMP_2RC + tmp3);
               //tmp1  = R_RESISTANCE_TMP * (R_TMP_1RC + tmp2/lCoefPre2);
               //lTemp = R_TMP_RC + tmp1/lCoefPre;

               // intervals : [-80;50] - ]50;180]
               // centres     :   -15        115
               if(R<=9099){
                       // intervalle : [-80;50] -> centre en -15
                       lRC    = 7173;
                                       // R for T=-15
                       transferBytesManuallyToReg(R_TMP_RC, 0x98, 0x3A, 0x00, 0x00);    //
15000
                       InvBuffer_S32(R_TMP_RC);
                               // -15000 = -15*lCoeffMultiTemp to get lCoeffMultiTemp*Output
Temperature
                       transferBytesManuallyToReg(R_TMP_1RC, 0xC8, 0x19, 0x05, 0x00);   //
334280
                       transferBytesManuallyToReg(R_TMP_2RC, 0x45, 0x04, 0x05, 0x00);   //
328773
                       lT3RC  = 3;
                                       // <- 3.23356
               }
               else{
                       // interval: ]50;180] -> center at 115
                       lRC    = 10987;
                                       // R for T=115
                       transferBytesManuallyToReg(R_TMP_RC, 0x38, 0xC1, 0x01, 0x00);    //
115000 = 115*lCoeffMultiTemp to get lCoeffMultiTemp*Output Temperature
                       transferBytesManuallyToReg(R_TMP_1RC, 0xB5, 0x4D, 0x05, 0x00);   //
347573
                       transferBytesManuallyToReg(R_TMP_2RC, 0xA5, 0xA3, 0x05, 0x00);   //
369573
                       lT3RC  = 4;
                                       // 3.92965
               }

               transferBytesManuallyToReg(R_RESISTANCE_TMP, R&0xFF, (R>>8)&0xFF, 0x00,
0x00);
               transferBytesManuallyToReg(R_TMP, lRC&0xFF, (lRC>>8)&0xFF, 0x00, 0x00);  //
Center of the Taylor Serie
               InvBuffer_S32(R_TMP);
```

```
               AddBuffer_32(R_RESISTANCE_TMP,R_TMP);
                              // difference between the resistance and the center of the
Taylor series

               // With double precision coefficient + multiplier coefficient to get 3
decimals
               transferBytesManuallyToReg(R_TEMPERATURE, lT3RC&0xFF,0x00,0x00,0x00);    //
T3RC
               MulBuffer_S32(R_TEMPERATURE,R_RESISTANCE_TMP);

                                                     // tmp3
               AddBuffer_32(R_TEMPERATURE,R_TMP_2RC);
                              // R_TMP_2RC + tmp3
               MulBuffer_S32(R_TEMPERATURE,R_RESISTANCE_TMP);
                       // tmp2
               transferBytesManuallyToReg(R_TMP, 0x40,0x0D,0x03,0x00);
       // lCoefPre2 (=200000)
                DivAddBuffer_32(R_TMP_1RC);// tmp2/coeffPre2
                                           // R_TMP_1RC + tmp2/coeffPre2
               MulBuffer_S32(R_TEMPERATURE,R_RESISTANCE_TMP);
                       // tmp1
               transferBytesManuallyToReg(R_TMP, 0x10,0x27,0x00,0x00);
       // 10 000 (lCoefPre/lCoeffMultiTemp) to get 1000*Output Temperature
               DivAddBuffer_32(R_TMP_RC);// tmp1/10000
                                        // R_TMP_RC + tmp/10000 = lCoeffMultiTemp *
Temperature
               T1 = GetBuffer_16(R_TEMPERATURE);
               T2 = GetBuffer_16(R_TEMPERATURE+2);
       }

       //turn PIN Out OFF
       turnPINOut(0,0);

       // Writing data to register
       SetBuffer(0,GetRSSI(),1); // set byte0 as the RSSI of the last known good packet
received
       SetBuffer(1,V,1); // set byte1 as voltage fetched via I2C bus
       SetBuffer(2,tmp,1); // set byte2 as error from header fetched via spi
       SetBuffer(3,T1,1); // set byte3 as 1rst byte of temperature
       SetBuffer(4,((T1 >> 8) & 0xFF),1); // set byte4 as 2nd byte of temperature
       SetBuffer(5,T2,1);// set byte5 as 3rd byte of temperature
       SetBuffer(6,((T2 >> 8) & 0xFF),1);// set byte6 as 4th byte of temperature
       // Envoi des données précédement enregistrées
       Send(7);
}

function main()
{
       local execType;

       execType = GetExecType();
       if(execType==MESHEXECTYPE_BOOTUP_bm){
               exec_bootup_extended();
       }
       else if(execType==MESHEXECTYPE_AIRCMD_bm){
               exec_aircmd();
       }
}
```

VIRTUAL MACHINE INTEGRATION GUIDE

PORTIA Module

April 10, 2019

Contents

1 VM Operations

1.1 UserCall functions

The virtual machine compiler interacts with native code through UserCall functions, which use the following syntax:

```
$ufX(Y, param1, param2, param3, ...)
```

where, `X` is the function number `fn#`, and `Y` is the `ALIAS#`. Alias names are also disponible for more readable code.

1.2 Internal buffer

Most VM operations access the internal byte-aligned general-purpose buffer. This buffer, refered to as the buffer in this document, can be used for multi-byte operations, in which case the index will refer to the index to the first byte of the data.

1.3 Operations Summary

Table 1: Buffer commands

ALIAS NAME	fn#	ALIAS#	argc	argv
`GetBuffer_S8(r)`	0	0x20	2	aliasID, r
`GetBuffer_U8(r)`	0	0x21	2	aliasID, r
`GetBuffer_16(r)`	0	0x22	2	aliasID, r
`InvBuffer_S32(r)`	0	0x25	2	aliasID, r
`GetRegisterRAMBUF(r, regOffset)`	0	0x50	3	aliasID, r, regOffset
`GetRegisterRAM(r, regOffset)`	0	0x51	3	aliasID, r, regOffset
`GetRegisterEEPROM(r, regOffset)`	0	0x52	3	aliasID, r, regOffset
`SetRegisterRAMBUF(r, regOffset)`	0	0x54	3	aliasID, r, regoffset
`AssertBcastPinFromLastBcast10Us(r)`	0	0x26	2	aliasID, r
`AssertBcastPinAtSystemTime10Us(r)`	0	0x27	2	aliasID, r

Continued on next page

Continued from previous page

ALIAS NAME	fn#	ALIAS#	argc	argv
`CompBuffer_64(r1, r2)`	0	0x42	3	aliasID, r1, r2
`AddBuffer_32(r1, r2)`	0	0x43	3	aliasID, r1, r2
`SubBuffer_32(r1, r2)`	0	0x44	3	aliasID, r1, r2
`MulBuffer_S32(r1, r2)`	0	0x45	3	aliasID, r1, r2
`DivBuffer_S32(r1, r2)`	0	0x46	3	aliasID, r1, r2
`CompBuffer_F32(r1, r2)`	0	0x4C	3	aliasID, r1, r2
`ShiftLeftBuffer_I32(r, shLeft)`	0	0x56	3	aliasID, r, shLeft
`ShiftLeftBuffer_U32(r, shLeft)`	0	0x57	3	aliasID, r, shLeft
`SetBuffer_16(r, val)`	0	0x58	3	aliasID, r, val
`Math1Var_F32(r, funcID)`	0	0x59	3	aliasID, r, funcID
`ShiftLeftBuffer_I64(r, shLeft)`	0	0x5A	3	aliasID, r, shLeft
`ShiftLeftBuffer_U64(r, shLeft)`	0	0x5B	3	aliasID, r, shLeft
`ConvertI32ToFloat(r, exp10)`	0	0x5E	3	aliasID, r, exp10
`ConvertFloatToI32(r, exp10)`	0	0x5F	3	aliasID, r, exp10
`CopyBuffer(r1, r2, len)`	0	0x60	4	aliasID, r1, r2, len
`Math2Var_I32(r1, r2, funcID)`	0	0x62	4	aliasID, r1, r2, funcID
`Math2Var_I64(r1, r2, funcID)`	0	0x63	4	aliasID, r1, r2, funcID
`Math2Var_F32(r1, r2, funcID)`	0	0x64	4	aliasID, r1, r2, funcID
`Math1VarExt_F32(r, rIn, funcID)`	0	0x65	4	aliasID, r, rIn, funcID
`Lookup_I16(rStart, rEnd, index)`	0	0x66	4	aliasID, rStart, rEnd, index
`LinInterpol_I16(rStart, rEnd, val)`	0	0x67	4	aliasID, rStart, rEnd, val
`LinInterpol_I32(r, rStart, rEnd)`	0	0x68	4	aliasID, r, rStart, rEnd
`LinInterpol_F32(r, rStart, rEnd)`	0	0x69	4	aliasID, r, rStart, rEnd
`McLaurin_F32(r, rStart, rEnd)`	0	0x6A	4	aliasID, r, rStart, rEnd
`SetBuffer(r, val, len)`	0	0x70	4	aliasID, r, val, len
`GetAirBuf(r, i, len)`	0	0x74	4	aliasID, r, i, len
`GetTxBuf(r, i, len)`	0	0x75	4	aliasID, r, i, len
`GetRegisterIndexedRAMBUF(r, regOffset, index)`	0	0x7A	4	aliasID, r, regOffset, index
`InitFloat10_F32(r, mant10, exp10)`	0	0x90	4	aliasID, r, mant10, exp10
`GetRegisterIndexedExtRAMBUF`	0	0xA0	5	aliasID, r, regOffset, index, indexsz

Table 2: Misc module

ALIAS NAME	fn#	ALIAS#	argc	argv
`GetExecType()`	1	0x00	1	aliasID
`GetRSSI()`	1	0x01	1	aliasID
`ResetWatchdog()`	1	0x02	1	aliasID
`TransferConfigRAMBUF()`	1	0x08	1	aliasID
`TransferConfigRAM()`	1	0x09	1	aliasID
`TransferConfigEEPROM()`	1	0x0A	1	aliasID
`ResetFactoryDefaults()`	1	0x0B	1	aliasID
`Send(count)`	1	0x20	2	aliasID, count
`SetBufferAtZeroTo_16(val)`	1	0x21	2	aliasID, val
`Delay(delay)`	1	0x24	2	aliasID, delay
`EasyGetRegisterIndexedRAMBUF`	1	0x40	3	aliasID, regOffset, index
`I2CPowerBus_Activate()`	2	0x00	1	aliasID
`I2CPowerBus_Deactivate()`	2	0x01	1	aliasID
`I2C_Stop()`	2	0x04	1	aliasID
`I2C_ReadAck()`	2	0x05	1	aliasID
`I2C_ReadNak()`	2	0x06	1	aliasID

Continued on next page

Continued from previous page

ALIAS NAME	fn#	ALIAS#	argc	argv
`I2C_Start(addr_rw)`	2	0x20	2	aliasID, addr_rw
`I2C_Write(val)`	2	0x21	2	aliasID, val
`GetPinIn(pinID)`	3	0x00..	1	aliasID&0x0F=pinID
`GetPinDir(pinID)`	3	0x10..	1	aliasID&0x0F=pinID
`AnalogRead(pinID)`	3	0x40..	1	aliasID&0x0F=pinID
`SetPinOut(pinID)(val)`	3	0x00..	2	aliasID&0x0F=pinID , val
`SetPinDir(pinID)(dir)`	3	0x10..	2	aliasID&0x0F=pinID , dir
`SetPinPull(pinID)(val)`	3	0x20..	2	aliasID&0x0F=pinID , val
`AnalogWrite(pinID)(val)`	3	0x40..	2	aliasID&0x0F=pinID , val
`GetPerReg(reg)`	4	0x00..	1	aliasID=reg(16bit!)
`SetPerReg(reg)(val)`	4	0x00..	2	aliasID=reg(16bit!), val

1.4 API documentation

`GetBuffer_S8`

- Description

  Get buffer byte at `r` as (`int8`), and return cast to `int16`

Parameter	Type	Description
`r`	`int16`	Buffer index

- Return

  Return the byte at index `r` of the buffer.

`GetBuffer_U8`

- Description

  Get buffer byte at `r` as (`uint8`), and return cast to `int16`

Parameter	Type	Description
`r`	`int16`	Buffer index

- Return

  Return the byte at index `r` of the buffer.

`GetBuffer_16`

- Description

  Get buffer `r`, 2 first bytes, as `int16` / `uint16`, and return it verbatim (as `int16`)

Parameter	Type	Description
`r`	`int16`	Buffer index

- Return

  Return the word at index `r` of the buffer.

`InvBuffer_S32`

- Description

```
WriteI32ToBuffer(rBuffer+r, -ReadBufferToI32(rBuffer+r));
```

Invert the `int32` at index `r` of the buffer.

Parameter	Type	Description
`r`	`int16`	Buffer index

- Return

  Nothing

`GetRegisterRAMBUF`

- Description

  Fetch register at `regOffset`, put it in `r` (RAMBUF)

Parameter	Type	Description
`r`	`int16`	Buffer index
`regOffset`	`int16`	Register index

- Return

  Nothing

`GetRegisterRAM`

- Description

  Fetch register at `regOffset`, put it in `r` (RAM)

Parameter	Type	Description
`r`	`int16`	Buffer index
`regOffset`	`int16`	Register index

- Return

  Nothing

`GetRegisterEEPROM`

- Description

  Fetch register at `regOffset`, put it in `r` (EEPROM)

Parameter	Type	Description
`r`	`int16`	Buffer index
`regOffset`	`int16`	Register index

- Return

  Nothing

`SetRegisterRAMBUF`

- Description

  Put content of `r` in register at regOffset (RAMBUF)

Parameter	Type	Description
`r`	`int16`	Buffer index
`regOffset`	`int16`	Register index

- Return

  Nothing

`AssertBcastPinFromLastBcast10Us`

- Description

  Assert broadcast LED at a time $10\mu s\times$ `(int32)buffer[r]`, from the beginning of the last broadcast cycle.

Parameter	Type	Description
`r`	`int16`	Buffer index

- Return

  Nothing

`AssertBcastPinAtSystemTime10Us`

- Description

  Assert bcast LED at `buffer[r]` where `buffer[r]` is in $10\mu s$ (64 bit) increments, and is the time since beginning of last bcast cycle. The reference used is the system clock.

Parameter	Type	Description
`r`	`int16`	Buffer index

- Return

  Nothing

`CompBuffer_64`

- Description

  Compare the values of two `int64` stored in the buffer.

Parameter	Type	Description
`r1`	`int16`	Buffer index
`r2`	`int16`	Buffer index

- Return

```
return buffer[r1]>buffer[r2]?1 : (buffer[r1] == buffer[r2]?0 : -1 )
```

`AddBuffer_32`

- Description

  Add the values of two `int32` stored in the buffer.

```
buffer[r1] += buffer[r2];
```

Parameter	Type	Description
`r1`	`int16`	Buffer index
`r2`	`int16`	Buffer index

- Return

  Nothing

`SubBuffer_32`

- Description

  Subtract the values of two `int32` stored in the buffer.

  ```
  buffer[r1] -= buffer[r2];
  ```

Parameter	Type	Description
`r1`	`int16`	Buffer index
`r2`	`int16`	Buffer index

- Return

  Nothing

`MulBuffer_S32`

- Description

  Multiply the values of two `int32` stored in the buffer.

  ```
  buffer[r1] *= buffer[r2];
  ```

Parameter	Type	Description
`r1`	`int16`	Buffer index
`r2`	`int16`	Buffer index

- Return

  Nothing

`DivBuffer_S32`

- Description

  Divide the values of two `int32` stored in the buffer.

  ```
  buffer[r1] /= buffer[r2];
  ```

Parameter	Type	Description
`r1`	`int16`	Buffer index
`r2`	`int16`	Buffer index

- Return

  Nothing

`CompBuffer_F32`

- Description

  Compare the values of two `float` stored in the buffer.

Parameter	Type	Description
`r1`	`int16`	Buffer index
`r2`	`int16`	Buffer index

- Return

```
return buffer[r1]>buffer[r2]?1 : (buffer[r1] == buffer[r2]?0 : -1 )
```

`ShiftLeftBuffer_I32`

- Description

  `r «= arithShLeft` (assumes `int32`); note that shLeft CAN be <0 in which case it shifts `r` right

```
buffer[r] <<= r2;
```

Parameter	Type	Description
`r`	`int16`	Buffer index
`shLeft`	`int16`	Shift amound

- Return

  Nothing

`ShiftLeftBuffer_U32`

- Description

  `r «= arithShLeft` (assumes `uint32`); note that shLeft CAN be <0 in which case it shifts `r` right

```
buffer[r] <<= r2;
```

Parameter	Type	Description
`r`	`int16`	Buffer index
`shLeft`	`int16`	Shift amound

- Return

  Nothing

`SetBuffer_16`

- Description

  Write (`int16`)val to buffer `r`, 2 first bytes, as `int16` / `uint16`

Parameter	Type	Description
`r`	`int16`	Buffer index
`val`	`int16` or `uint16`	Value

- Return

  Nothing

`Math1Var_F32`

- Description

  Math on a `float` variable.

  ```
  r = operation(r);
  ```

Parameter	Type	Description
`r`	`int16`	Buffer index
`funcId`	`uint8`	The function

- Return

  Nothing

`ShiftLeftBuffer_I64`

- Description

  `r «= arithShLeft` (assumes `int64`); note that shLeft CAN be <0 in which case it shifts `r` right

  ```
  buffer[r] <<= r2;
  ```

Parameter	Type	Description
`r`	`int16`	Buffer index
`shLeft`	`int16`	Shift amound

- Return

  Nothing

`ShiftLeftBuffer_U64`

- Description

  `r «= arithShLeft` (assumes `uint64`); note that shLeft CAN be <0 in which case it shifts `r` right

  ```
  buffer[r] <<= r2;
  ```

Parameter	Type	Description
`r`	`int16`	Buffer index
`shLeft`	`int16`	Shift amound

- Return

  Nothing

`ConvertI32ToFloat`

- Description

Inplace cast of `int32` to a `float`. A scaling of `10^exp10` is applyed to the produced value

Parameter	Type	Description
`r`	`int16`	Buffer index
`exp10`	`int16`	Scaling

- Return

  Nothing

`ConvertFloatToI32`

- Description

  Inplace cast of `float` to a `int32`. A scaling of `1/10^exp10` is applyed to the produced value

Parameter	Type	Description
`r`	`int16`	Buffer index
`exp10`	`int16`	Scaling

- Return

  Nothing

`CopyBuffer`

- Description

  Copy buffer address `r2` to `r1`, `len` bytes (max of 256)

Parameter	Type	Description
`r1`	`int16`	Destination buffer index
`r2`	`int16`	Source buffer index
`len`	`uint8`	Number of bytes to copy

- Return

  Nothing

`Math2Var_I32`

- Description

  Math on two `int32` variables.

  ```
  r1 = operation(r1, r2);
  ```

Parameter	Type	Description
`r1`	`int16`	Buffer index
`r2`	`int16`	Buffer index
`funcId`	`uint8`	The function

- Return

  Nothing

`Math2Var_I64`

- Description

  Math on two `int64` variables.

  ```
  r1 = operation(r1, r2);
  ```

Parameter	Type	Description
`r1`	`int16`	Buffer index
`r2`	`int16`	Buffer index
`funcId`	`uint8`	The function

- Return

  Nothing

`Math2Var_F32`

float advanced math fns 2 vars Custom function call (math sin/cos/etc)

- Description

  Math on two `float` variables.

  ```
  r1 = operation(r1, r2);
  ```

Parameter	Type	Description
`r1`	`int16`	Buffer index
`r2`	`int16`	Buffer index
`funcId`	`uint8`	The function

- Return

  Nothing

`Math1VarExt_F32`

- Description

  Math on a `float` variables.

  ```
  r = operation(rIn);
  ```

Parameter	Type	Description
`r`	`int16`	Destination buffer index
`rIn`	`int16`	Source buffer index
`funcId`	`uint8`	The function

- Return

  Nothing

`Lookup_I16`

- Description

  Perform a lookup in a table of `int16`. The first element of the table is located at `rStart`, while the last is located at `rEnd` inclusively.

Parameter	Type	Description
`rStart`	`int16`	Position of the first element
`rEnd`	`int16`	Position of the last element
`index`	`int16`	The index

- Return

  The result of the lookup

`LinInterpol_I16`

interpol 16 bit: val is input, returns output (rEnd inclusive)

- Description

  Linear interpolation. The array of points (4 bytes for each point, two `int16`, x and y) is determined by the positions `rStart` and `rEnd` (inclusive).

  The x coords must be monotonically increasing and $x_1 \ldots x_n$ must never be the same values.

  `val` is the input value.

Parameter	Type	Description
`rStart`	`int16`	Position of the first element
`rEnd`	`int16`	Position of the last element
`val`	`int16`	Input Value

- Return

  Interpolation result.

`LinInterpol_I32`

- Description

  Linear interpolation. The array of points (8 bytes for each point, two `int32`, x and y) is determined by the positions `rStart` and `rEnd` (inclusive).

  The x coords must be monotonically increasing and $x_1 \ldots x_n$ must never be the same values.

  `r` is the buffer position used for the interpolation location value and for the result.

Parameter	Type	Description
`r`	`int16`	Buffer index to the value
`rStart`	`int16`	Position of the first element
`rEnd`	`int16`	Position of the last element

- Return

  Nothing

`LinInterpol_F32`

- Description

  Linear interpolation. The array of points (8 bytes for each point, two `floats`, x and y) is determined by the positions `rStart` and `rEnd` (inclusive).

  The x coords must be monotonically increasing and $x_1 \ldots x_n$ must never be the same values.

  `r` is the buffer position used for the interpolation location value and for the result.

Parameter	Type	Description
`r`	`int16`	Buffer index to the value
`rStart`	`int16`	Position of the first element
`rEnd`	`int16`	Position of the last element

- Return

  Nothing

`McLaurin_F32`

- Description

  McLaurin series calc: `rStart` and `rEnd` (inclusive) defines the start float and the end floats, say you have three `r0`, `r1`, `r2`, then calculation is: `r = r0*r*r + r1*r + r2`.

  `r` is the buffer position used for the indeterminate value and for the result.

Parameter	Type	Description
`r`	`int16`	Buffer index to the indeterminate value
`rStart`	`int16`	Position of the first element
`rEnd`	`int16`	Position of the last element

- Return

  Nothing

`SetBuffer`

- Description

  Set buffer from location `r`, `len` bytes, to (`int16`) `val` (latter being casted as `int8=/u=int8`)

Parameter	Type	Description
`r`	`int16`	Buffer index
`val`	`int16`	Value
`len`	`int16`	Number of bytes

- Return

  Nothing

`InitFloat10_F32`

- Description

  Write a float on a buffer location.

  ```
  buffer[r] = mant10 * pow(10, exp10);
  ```

Parameter	Type	Description
`r`	`int16`	Buffer index
`mant10`	`int16`	Mantissa
`exp10`	`uint8`	Exponent (between -38 and 38)

- Return

  Nothing

`GetAirBuf`

- Description

  Copy `mid(airbuf, i, len)` into buffer at index `r`: returns number of valid bytes in `airbuf`

Parameter	Type	Description
`r`	`int16`	Buffer index
`i`	`int16`	Buffer index
`len`	`uint8`	

- Return

  Returns number of valid bytes in `airbuf`

`GetTxBuf`

- Description

  Copy `mid(airbuf, i, len)` into buffer at index `r`: returns number of valid bytes in `txbuf`

Parameter	Type	Description
`r`	`int16`	Buffer index
`i`	`int16`	Buffer index
`len`	`uint8`	

- Return

  Returns number of valid bytes in `airbuf`

`GetRegisterIndexedRAMBUF`

- Description

  GetRegisterRAMBUF, with last var containing `int16 index`

Parameter	Type	Description
`r`	`int16`	Buffer index
`regOffset`	`int16`	Register index
`index`	`uint8`	Index

- Return

  Nothing

`GetRegisterIndexedExtRAMBUF`

- Description

GetRegisterRAMBUF, with last var containing `index`, with size `indexsz`

Parameter	Type	Description
`r`	`int16`	Buffer index
`regOffset`	`int16`	Register index
`index`	`uint8`	Index
`indexsz`	`uint8`	Index Byte Size

- Return

  Nothing

`GetExecType`

- Description

  Get the current execution type

Parameter	Type	Description

- Return

  The EVM execute type (0= normal packet, 1=air, 2=bootup, 3=enterSeekMode, 4=leaveSeekMode, 5=enterBroadcast, 6=leaveBroadcast)

`GetRSSI`

- Description

Parameter	Type	Description

- Return

  The last RSSI of received packet

`ResetWatchdog`

- Description

  Reset the watchdog timer

Parameter	Type	Description

- Return

  Nothing

`TransferConfigRAMBUF`

- Description

  Tranfer config to RAMBUF at next reset.

Parameter	Type	Description

- Return

Nothing

`TransferConfigRAM`

- Description

  Tranfer config to RAM at next reset.

Parameter	Type	Description

- Return

  Nothing

`TransferConfigEEPROM`

- Description

  Tranfer config to EEPROM at next reset.

Parameter	Type	Description

- Return

  Nothing

`ResetFactoryDefaults`

- Description

  Reset config to factory defaults at next reset.

Parameter	Type	Description

- Return

  Nothing

`Send`

- Description

  Trigger transfer of data on next broadcast. Will send `count` bytes (bytes taken from buffer at `r` = 0)

Parameter	Type	Description
`count`	`int16`	The number of bytes

- Return

  Nothing

`SetBufferAtZeroTo_16`

- Description

  Write an (`int16`)val to buffer at index 0

  ```
  buffer[0] = val;
  ```

Parameter	Type	Description
`val`	`int16`	The value

- Return

  Nothing

`Delay`

- Description

  Delay thread `delay` ticks (100 us = 1 tick) (approximative). Max allowed value of `delay` is 127 (12.7 msec)

  ```
  r1 = operation(r1, r2);
  ```

Parameter	Type	Description
`delay`	`uint8`	Number of ticks to delay

- Return

  Nothing

`EasyGetRegisterIndexedRAMBUF`

- Description

  GetRegisterRAMBUF, with last var containing `int16 index`.

Parameter	Type	Description
`regOffset`	`int16`	Register index
`index`	`uint8`	Index

- Return

  The value of the `GetRegister` operation.

`I2CPowerBus_Activate`

- Description

  Enable power to the I2C bus.

Parameter	Type	Description

- Return

  Nothing

`I2CPowerBus_Deactivate`

- Description

  Disable power to the I2C bus.

Parameter	Type	Description

- Return

Nothing

`I2C_Stop`

- Description

  Stop I2C command.

Parameter	Type	Description

- Return

  Nothing

`I2C_ReadAck`

- Description

  I2C send ack.

Parameter	Type	Description

- Return

  The value (`uint8`) read while the ack is sent.

`I2C_ReadNak`

- Description

  I2C send nak.

Parameter	Type	Description

- Return

  The value (`uint8`) read while the nak is sent.

`I2C_Start`

- Description

  Math on two `int32` variables.

  ```
  r1 = operation(r1, r2);
  ```

Parameter	Type	Description
`addr_rw`	`int16`	Address

- Return

  Returns 1 if it fails, 0 if it succeeds.

`I2C_Write`

- Description

  Write a value on the I2C bus.

Parameter	Type	Description
`val`	`uint8`	The value to send

- Return

  The byte read while sending.

`GetPinIn`

- Description

  Read a pin state

Parameter	Type	Description
`pinID`	`int16`	Pin number

- Return

  The current state of the pin (1 = HIGH, 0 = LOW)

`GetPinDir`

- Description

Parameter	Type	Description
`pinId`	`int16`	Pin number

- Return

  The pin direction

`AnalogRead`

- Description

Parameter	Type	Description
`pinId`	`int16`	Pin number

- Return

  The ADC value of the pin.

`SetPinOut`

- Description

  Set the value of a pin.

Parameter	Type	Description
`pinId`	`int16`	Pin number
`val`	`bool`	Value

- Return

  Nothing

`SetPinDir`

- Description

  Set the direction of a pin.

Parameter	Type	Description
`pinId`	`int16`	Pin number
`dir`	`bool`	Direction

- Return

  Nothing

`SetPinPull`

- Description

  Enable or disable the internal pull-up/pull-down resistors

Parameter	Type	Description
`pinId`	`int16`	Pin number
`val`	`bool`	State

- Return

  Nothing

`AnalogWrite`

- Description

  Write an analog value to a pin.

Parameter	Type	Description
`pinId`	`int16`	Pin number
`val`	`int16`	Value

- Return

  Nothing

`GetPerReg`

- Description

  Get peripheral register reg (8 bit)

Parameter	Type	Description
`reg`	`int16`	Register

- Return

  The value of the register

`SetPerReg`

- Description

  Set peripheral register reg (8 bit)

Parameter	Type	Description
`reg`	`int16`	Register
`val`	`int16`	Value

- Return

  Nothing

1.5 Math functions

Table 70: One variable math functions

funcId	Function
0	Invert
1	Sqrt
2	SqrtRount
3	ln
4	log base 2
5	log base 10
6	exp
7	exp base 2
8	exp base 10
9	sin
10	cos
11	tan

Table 71: Two variable math functions

funcId	Function
0	add
1	subtract
2	multiply
3	divide
4	modulo
5	shift left
6	atan2