锐客创新硬件操作命令，版本 1.0

延时 Delay

该硬件操作命令实现延时操作。延时操作通常和其他硬件操作命令组合使用，比如实现 GPIO 拉高 5 秒钟，然后拉低。

- 硬件操作命令：["delay", {DELAY_ID}, {TIME_UNIT}, {DELAY_VALUE}]

{DELAY_ID}	延时 ID，该参数固定为 0
{TIME_UNIT}	时间单位选择，可选 s(秒), ms(毫秒)或 us(微秒)
{DELAY_VALUE}	延时值，需要为正整数

- 结果返回：[{STATUS}]

{STATUS}	“SUCCESS”表示延时成功，”FAIL”表示失败

- 例子：

  以下的例子延时 5 秒：

  ["delay", 0, "s", 5]

  以下的例子为硬件操作组合使用，实现 GPIO 拉高 5 秒钟，然后拉低：

  ["gpio", 0, "output", 1], ["delay", 0, "s", 5], ["gpio", 0, "output", 0]

GPIO

设置 GPIO 为输入：

- 硬件操作命令：["gpio", {PIN_INDEX}, "input", {PULL_UP_RESISTOR_ENABLE}]

{PIN_INDEX}	配置为 GPIO 的引脚，可选 0-41
{PULL_UP_RESISTOR_ENABLE}	是否使用上拉电阻，1 使用，0 不使用

- 结果返回：[{PIN_STATE}]

{PIN_STATE}	1 表示引脚目前高电平，0 表示低电平

- 例子：

以下的例子设置 pin 10 为输入，并且使用上拉电阻：

["gpio", 10, "input", 1]

设置 GPIO 为输出：

- 硬件操作命令：["gpio", {PIN_INDEX}, "output", {MODE}]

{PIN_INDEX}	说明如上
{MODE}	0：输出低电平，1：输出高电平，2：高低电平反转

- 结果返回：[{PIN_STATE}]

{PIN_STATE}	引脚状态，1 表示引脚为高电平，0 表示引脚为低电平

- 例子：

以下的例子设置 pin 10 输出低电平

["gpio", 10, "output", 0]

OneWire

Onewire 命令实现在 GPIO 引脚上以给定的间隔进行采样，并返回采样的结果。

- 硬件操作命令：{"onewire", {PIN_INDEX }, {TIME_UNIT}, {DELAY_VALUE}, {SAMPLING_NUMBER}]

{PIN_INDEX }	进行 Onewire 操作的 GPIO 引脚，可选 0-41
{TIME_UNIT}	时间单位选择，可选 ms(毫秒)或 us(微秒)
{DELAY_VALUE}	延时值，需要为正整数
{SAMPLING_NUMBER}	采样的总个数

- 结果返回：[{SAMPLING_VALUE}]

{SAMPLING_VALUE}	采样结果，结果以字节形式返回，并先填充最高位

- 例子：

以下的例子以 10 微秒的间隔在 GPIO 引脚 0 上进行采样，采样个数为 10

["onewire", 0, "us", 10, 10]

返回值为 [201, 192]

201 的二进制为 11001001，192 的二进制为 11000000，组合在一起为 11001001,11000000。因为采样的个数为 10，即从第 0 微秒，10 微秒，20 微秒到第 100 微秒的采样为 1,1,0,0,1,0,0,1,1,1。

ADC

ADC 使用 10-bit 的采样精度(0 - 1023)，参考电压可以在 5V, 2.56V 和 1.1V 之间选择。

- 硬件操作命令：{"adc", {ADC_CHANNEL_ID}, {VOLTAGE_REFERENCE}]

{ADC_CHANNEL_ID}	选择 ADC 通道，0-15

{VOLTAGE_REFERENCE}	选择参考电压，可选 5v, 2.56v 或 1.1v (注意需要小写'v')

- 结果返回：[{ADC_VALUE}]

{ADC_VALUE}	ADC 采样值，通过计算 (ADC_VALUE / 1024) * VOLTAGE_REFERENCE 得到当前的电压值

- 例子：

以下的例子获得 ADC 通道 1 的值：

["adc", 1, "5v"]

PWM

PWM 可以在指定 Pin 脚输出特定频率和周期的方波，可以用来驱动电机，驱动蜂鸣器，调节 LED 灯的亮度等。该系统有两条 pwm 相关命令，分别用来开启或者关闭 PWM。系统支持对于一个周期，指定输出三个不同的频率，会有三个 pin 脚进行输出。

开启 PWM

- 硬件操作命令：["pwm", {TIMER_INDEX}, "enabled", {TIME_UNIT}, {PERIOD}, {DUTY_CYCLE_A}, {DUTY_CYCLE_B}, {DUTY_CYCLE_C}, {DURATION_IN_10MS}]

{TIMER_INDEX}	计时器选择，目前可选 0 或者 1
{TIME_UNIT}	时间单位选择，目前可选 ms 或者 us
{PERIOD}	时间周期
{DUTY_CYCLE_A}	PWM 输出 1 的脉宽
{DUTY_CYCLE_B}	PWM 输出 2 的脉宽
{DUTY_CYCLE_C}	PWM 输出 3 的脉宽
{DURATION_IN_10MS}	该 pwm 输出的时间(单位是 10ms)，如果输入负数如-1，则 pin 脚会持续输出 PWM。

- 结果返回：[{STATUS}]

{STATUS}	如果 pwm 设置成功，则为"success"; 否则会返回失败原因。

- 例子：

以下的例子使用定时器 0, 输出的周期为 20ms，三个输出 pin 脚分别输出脉宽为 8ms, 12ms, 15ms 的方波，PWM 输出持续 1 秒：

["pwm", 1, "enabled", "ms", 20, 8, 12, 15, 100]

关闭 PWM

- 硬件操作命令：["pwm", {TIMER_INDEX}, "disabled"]

{TIMER_INDEX}	计时器选择，目前可选 0 或者 1

- 结果返回：[{STATUS}]

{STATUS}	如果 pwm 设置成功，则为"success"; 否则会返回失败原因

SPI

系统提供 SPI 通信接口，由 4 条信号线组成，MISO, MOSI, SCK 和片选信号。用户可自己定义大多数 SPI 模块的参数，包括速率，片选信号引脚，发送数据的顺序等等。

- 硬件操作命令：["spi", {SPI_INDEX} {SPEED_LEVEL}, {CS_PIN}, {SAMPLE_MODE}, {MSB/LSB}, {RECEIVE_DATA_LENGTH}, {SEND_DATA_LENGTH}, {DATA_TO_SEND}]

{SPI_INDEX}	板子有一个 SPI 通道，该参数需固定为 0
{SPEED_LEVEL}	SPI 速率选择，可选 0，1，2。 0: 8Mbit/s 1: 4Mbit/s 2: 2Mbit/s
{CS_PIN}	参照 GPIO 部分，可选择任意引脚
{SAMPLE_MODE}	设置采样的时间点 "lr": 在第一个上升沿进行采样 "tf": 在跟随的下降沿进行采样 "lf": 在第一个下降沿进行采样 "tr": 在跟随的上升沿进行采样
{MSB/LSB}	Lsb：发送一个字节的数据是从低比特开始发送，msb：发送一个字节的数据是从高比特开始发送
{RECEIVE_DATA_LENGTH}	SPI 需要接收多少字节的数据
{SEND_DATA_LENGTH}	SPI 需要发送多少字节的数据
{DATA_TO_SEND}	SPI 发送的数据 (可选)

- 结果返回：[{DATA_RECEIVED}]

{DATA_RECEIVED}	根据在发送请求里指定的接收字节数量，返回接收到的数据

- 例子

以下例子展示用每秒 4 Mbit 的速率发送 0x1, 0x2, 0x3, 0x4 到接收端，并接收 10 字节，片选引脚是 15：

["spi", 0, 1, 15, "lr", "msb", 10, 4, 1, 2, 3, 4]

UART

系统为用户提供了两个 UART 模块，每个模块可以独立运行，有多个波特率可选。如果需要接收端发回数据，用户需要指定最长的等待时间，系统会在发送完数据后等待接收端返回指定个数的数据，当接收到足够数量的数据后，系统会自动返回并发回数据；否则，系统会一直等待，直到最长的等待时间结束。

- 硬件操作命令：["uart", {UART_INDEX}, {SPEED_SELECTION}, {PARITY_ENABLE}, {STOP_BIT},{DATA_SIZE}, {RECEIVE_TIME_OUT_SEC}, {RECEIVE_DATA_LENGTH}, {SEND_DATA_LENGTH}, {DATA_TO_SEND}]

{UART_INDEX}	目前板子上有 2 个 UART 模块，可以选择 0,1
{SPEED_SELECTION}	UART 速率选择，可选 9k/38k/115k “9k”对应的波特率：9600 “38k”对应的波特率：38400 “115k”对应的波特率：115200
{PARITY_ENABLE}	奇偶校验位 "disabled": 不使用校验 "even"： 使用偶校验 "odd"： 使用奇校验
{STOP_BIT}	停止位 1: 1 位停止位 2: 2 位停止位
{DATA_SIZE}	数据大小：可选 5/6/7/8
{RECEIVE_TIME_OUT_SEC}	等待接收数据的最大时间(单位：秒)
{RECEIVE_DATA_LENGTH}	期待接收端返回的数据长度
{SEND_DATA_LENGTH}	发送的数据长度
{DATA_TO_SEND}	发送的数据

- 例子：

以下例子用 9600 的波特率从模块 1 发送数据(1,2,3,4)到接收端，发送时不使用奇偶校验，1 位停止位，8bit 的数据包，并等待接收 10 字节的数据，最多等待 5 秒。

["uart", 1, "9k", "disabled", 1, 8, 5, 10, 4, 1, 2, 3, 4]

I2C

系统提供了 I2C 模块，可以方便地向 I2C 外设发送数据。请注意，因为系统为 5V 系统，请确保外设支持 5V 的信号输入。

I2C 读操作

- 硬件操作命令：["i2c", {I2C_ID}, "read", {SPEED_IN_HUNDRED_KHZ}, {DEVICE_ADDRESS}, {REGISTER_ADDRESS}, {RECEIVE_DATA_LENGTH}]

{I2C_ID}	I2C 模块选择，板子上有一个 I2C 模块，该参选只支持 0
{SPEED_IN_HUNDRED_KHZ}	I2C 速度选择，可选 1/2/3/4 1: 100kHz 2: 200kHz 3: 300kHz 4: 400kHz
{DEVICE_ADDRESS}	外设地址，请参考外设的技术手册
{REGISTER_ADDRESS}	如果需要发送外设的寄存器地址，可以在这里指定。如果不需要发送外设的寄存器地址，请在这里指定-1。
{RECEIVE_DATA_LENGTH}	需要接收多少字节的数据

- 例子

比如，要用 400kHz 的速率读取 I2C 外设(地址 0x21 或十进制 33)中的 10 个字节的数据(无寄存器地址)：

["i2c", 0, "read", 4, 33, -1, 10]

I2C 写操作

- 硬件操作命令：["i2c", {I2C_ID}, "write", {SPEED_IN_KHZ}, {DEVICE_ADDRESS}, {REGISTER_ADDRESS}, {SEND_DATA_LENGTH}, {DATA_TO_SEND}]

{I2C_ID}	I2C 模块选择，板子上有一个 I2C 模块，该参选只支持 0
{SPEED_IN_KHZ}	I2C 速度选择，可选 1/2/3/4 1: 100kHz 2: 200kHz 3: 300kHz 4: 400kHz
{DEVICE_ADDRESS}	外设地址，请参考外设的技术手册
{REGISTER_ADDRESS}	如果需要发送外设的寄存器地址，可以在这里指定。如果不需要发送外设的寄存器地址，请在这里指定-1。
{SEND_DATA_LENGTH}	发送信息的长度
{DATA_TO_SEND}	发送的信息

- 例子

比如，要发送 10 个字节的数据(从 0 到 9),到地址 0x21 或十进制 33,使用相同的参数：

["i2c", 0, "write", 4, 33, -1, 10, 0, 1, 2, 3, 4, 5, 6, ,7 ,8, 9]

File

系统提供文件系统，可进行文件的读写操作。

进行文件的读操作

- 硬件操作命令：["file", {FILE_ID}, "read", {FILE_NAME}]

{FILE_ID}	文件 ID, 该参数固定为 0
{FILE_NAME}	文件名称

- 结果返回：[{FILE_CONTENT}]

{FILE_CONTENT}	文件的内容

- 例子：

以下例子读取 test.txt 文件：

["file", 0, "read", "test.txt"]

进行文件的写操作

- 硬件操作命令：["file", {FILE_ID}, "read", {FILE_NAME}, {FILE_CONTENT}]

{FILE_ID}	文件 ID, 该参数固定为 0
{FILE_NAME}	文件名称
{FILE_CONTENT}	文件内容

- 结果返回：[{WRITE_LENGTH}]

{WRITE_LENGTH}	返回写入文件中字节的数量

- 例子：

以下例子将 hellotest! 写入文件 test.txt:

["file", "write", "test.txt", "hellotest!"]

RTC (时间模块)

平台集成了 RTC 时钟单元帮助系统得到精确的时间信息。用户可以通过读命令得到当前的时钟信息，或通过写命令修改时钟信息。

系统的 RTC 模块需要有额外的纽扣电池供电，以确保在系统断电后，时钟也可以正常计时。请确保正确安装好纽扣电池。

时钟读取命令

- 硬件操作命令：["rtc", {RTC_ID}, "read"]

{RTC_ID}	RTC ID，该参数固定为 0

- 结果返回：[{YEAR}, {MONTH}, {DATE}, {HOUR}, {MINUTE}, {SECOND}]

{YEAR}	数值是基于 2000 年
{MONTH}	月份
{DATE}	日期
{HOUR}	时：24 小时制
{MINUTE}	分
{SECOND}	秒

- 例子

比如，发送如下的 ACTION 读取 RTC 的值：

["rtc", 0, "read"]

时钟修改命令

- 硬件操作命令：["rtc", {RTC_ID}, "write", {YEAR}, {MONTH}, {DATE}, {HOUR}, {MINUTE}, {SECOND}]

{RTC_ID}	RTC ID，该参数固定为 0
{YEAR}	数值是基于 2000 年
{MONTH}	月份
{DATE}	日期
{HOUR}	时：24 小时制
{MINUTE}	分
{SECOND}	秒

- 结果返回：返回空。即[]
- 例子：

比如，将 RTC 的时间修改为 2015 年，3 月 6 号，8 点 48 分 21 秒:

["rtc", 0， "write", 15, 3, 6, 8, 48, 21]