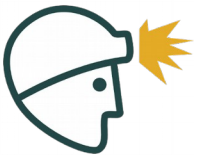




Lua for IoT



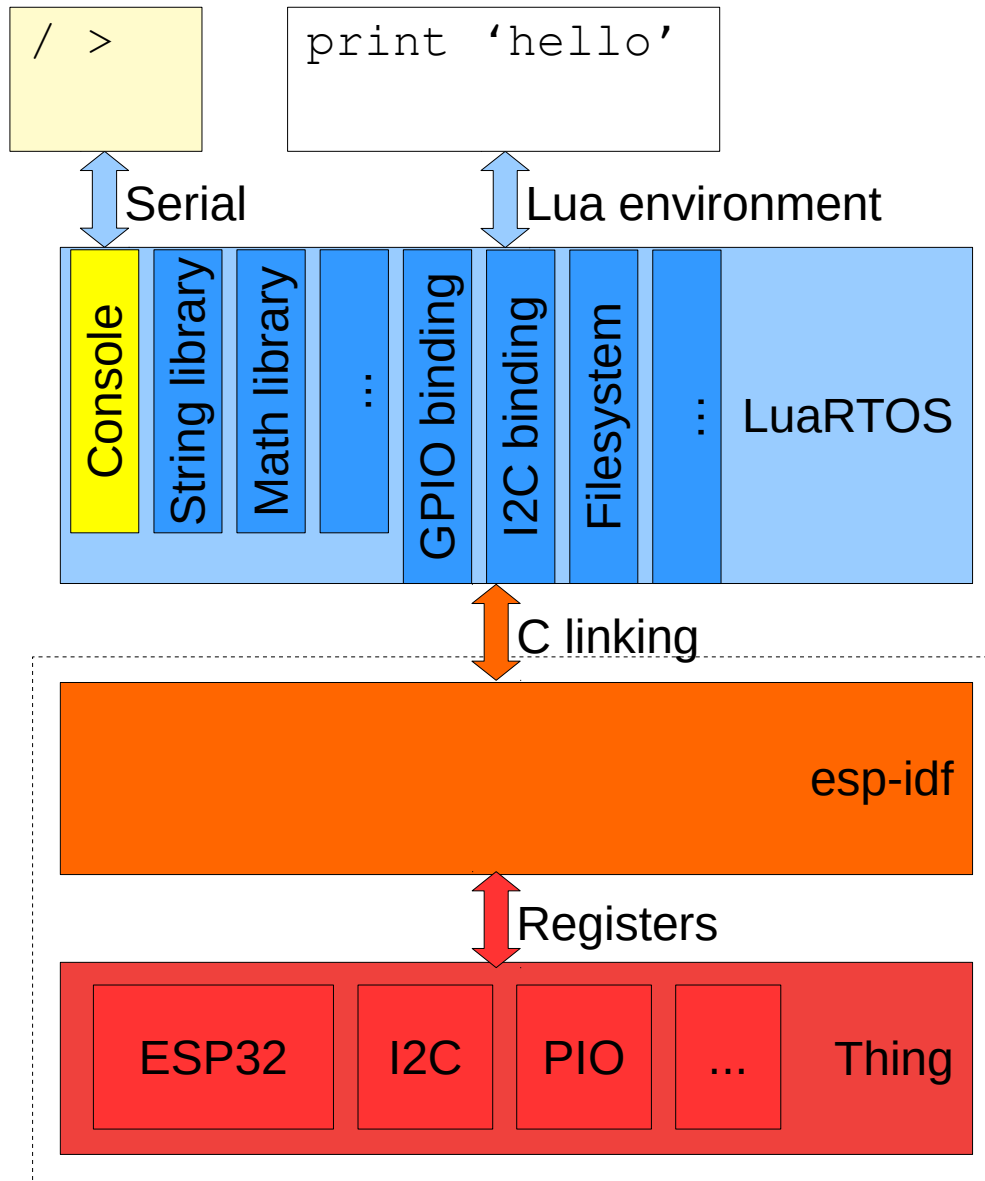
WSCF 2019
CYBER PHYSICAL SYSTEMS WORKSHOP



- The environment
- Micro tutorial
- The language



Environment



User control and scripts

RTOS + a Lua scripting engine
by Whitecat
<https://github.com/whitecatboard/Lua-RTOS-ESP32>

Manufacturer's C SDK

Hardware



Remote Console

```
xopxe@timur:~$ picocom --baud 115200 /dev/ttyUSB0  
picocom v2.2
```

```
port is          : /dev/ttyUSB0  
flowcontrol      : none  
baudrate is      : 115200  
parity is        : none  
databits are     : 8  
stopbits are     : 1  
escape is        : C-a  
local echo is    : no  
noinit is        : no  
noreset is       : no  
nolock is        : no  
send_cmd is      : sz -vv  
receive_cmd is   : rz -vv -E  
imap is          :  
omap is          :  
emap is          : crcrlf,delbs,
```

Type [C-a] [C-h] to see available commands

Terminal ready

```
/ > print 'hello world'  
hello world  
/ >  
Thanks for using picocom  
xopxe@timur:~#
```

← Ctrl + A + Q



Running Scripts

```
xopxe@timur:~# wcc -p /dev/ttyUSB0 -up hello.lua hello.lua
xopxe@timur:~$ picocom --baud 115200 /dev/ttyUSB0
picocom v2.2
```

...

Type [C-a] [C-h] to see available commands

Terminal ready

```
/ > os.ls()
f          2446
d          -
f          2407
d          -
f          39
/ >
/ > os.cat('hello.lua')
for i=1,3 do
  print 'hello world'
end

/ > dofile('hello.lua')
hello world
hello world
hello world
/ >
```

hello.lua

```
for i=1,3 do
  print 'hello world'
end
```

```
system.lua
examples
config.lua
lib
hello.lua
```



Autorunning Scripts

```
xopxe@timur:~# wcc -p /dev/ttyUSB0 -up hello.lua autorun.lua
xopxe@timur:~$ picocom --baud 115200 /dev/ttyUSB0
picocom v2.2
```

...

Type [C-a] [C-h] to see available commands

Terminal ready

```
/ > os.ls()
f      2446      system.lua
d      -        examples
f      2407      config.lua
d      -        lib
f      39        hello.lua
f      39        autorun.lua
/ >
```

...

Booting Lua RTOS...

Lua RTOS beta 0.1 powered by Lua 5.3.4

Executing /system.lua ...

Executing /autorun.lua ...

hello world

hello world

hello world

/ >

RESET



Lua Design

Lua is a powerful, fast, lightweight, embeddable scripting language.

Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode for a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.

lua.org/about.html



Lua

```
-- a factorial function
function fact (n)
    if n == 0 then
        return 1
    else
        return n * fact(n - 1)
    end
end

print( 'Type a number:' )
local a=io.read( "*number" ) -- reads console
print( 'The factorial is:', fact(a) )
```




Lua: Dynamic Typing

- string
- number
- boolean
- table
- function
- nil
- coroutine
- userdata



Lua: string

```
a = " text"
```

```
b = 'more text'
```

```
a_c = a .. " concat to " .. c
```

```
length = #d
```



Lua: number

- A single type for all numbers. Integer and float representation selected automatically

```
n = 5
```

```
nn = n + 1
```

```
pi = 3.14159
```

```
half = 1 / 2
```

```
two = 5 // 2
```

```
three = math.floor(pi)
```



Lua: boolean

- **v = true**
f = false
hmm = v or not f and true
- **10 or 20** → 10
10 and 20 → 20
nil or "a" → "a"
nil and 10 → nil
if 5.0 then print 'x' end → x
if nil then print 'x' end →
false or nil → nil
- **val = parameter or 0.0** --a default



Lua: tables (1)

- The only datastructure available, used for everything
- ```
t = {}
t[1] = "first"
t["first"] = true
```
- Value `nil` signals absence  

```
t["first"] = nil --delete entry
```
- Keys and values can be of any type (except `nil` as key)  

```
t[print] = {}
```
- ```
for k, v in pairs(t) do  
    print(k, v)  
end
```



Lua: tables (2)

- `t.first` is syntactic sugar for `t["first"]`

```
o = {}
```

```
o.pi = 3.14
```

```
o.circ = function(r) return 2*r*o.pi end
```

- Special support for arrays in tables (keys 1,2,3...n):
 - Can appear in any table
 - Can be iterated with `ipairs()`
 - `pairs()` iterates all entries, with no order
 - `ipairs()` iterates only array part, in order
 - Special operators:
 - `#` (array length) append to array is `t[#t+1]="a"`
 - `table.insert()`, `table.remove()`, `table.sort()`



Lua: table constructors

- `days = {"monday", "tuesday"}`
 - `days={}; days[1]="monday"; days[2]="tuesday"`
- `point = {x=0, y=0}`
 - `point={}; point.x=0; point.y=0`
- `a = {"a truth"]=true, ["lie!"]=false}`
 - `a={}; a["a truth"]=true; a["lie!"]=false`
- `triangle = {
 {x=0,y=0},
 {x=0,y=1},
 {x=1,y=0},
}`



Lua: functions

- First order members. Just as any variable.
- ```
average = function (a, b)
 local v = (a+b)/2
 return v
end
```
- ```
oldprint = print
print = function(...)
    oldprint(os.time(), ...)
end
print("ping!", "pong!")    → 1302201902 ping! pong!
```

```
local function f(x)
    return x
end
```

```
local f = function(x)
    return x
end
```




Lua: casting

```
pi = tonumber("3.14159")  
print("pi=" .. tostring(pi))  
print(type(pi))
```



Lua: Scoping

- Variables are *global* or *local*

```
a = 10          -- global
```

```
local b = 10    -- local
```

- Globals stored in the environment, available everywhere
- Locals use lexical scoping: visible from declaration and inside function, control structure or block
 - In case of doubt, use locals
- Undeclared variables are `nil`
- Why not locals by default?
 - Cue: nested functions
<http://lua-users.org/wiki/LocalByDefault>

```
local x
for i=1,5 do
    local y=1
    x=i
end
print(x,y)
```

```
→
5  nil
```

```
local a = 1
function f()
    local a = 2
    print(a)
end
f()
print(a)
```

```
→
2
1
```



Lua: multiple assignments

- `local a, b, c = 1, 0, "Hi!"`
`a, b = b, a` --swap values
- `local function sum_mult (a,b)`
 `return a+b, a*b`
`end`
`local s, m = sum_mult(2,3)`
- `local function div (a, b)`
 `if b~=0 then`
 `return a/b`
 `else`
 `return nil, "divide by zero"`
 `end`
`end`
`local q=assert(div(2,0))`



Lua: Garbage Collector

Just works, mostly leave it alone.

- When all references to a value are unreachable, memory is freed
- Release: assign **nil**

```
data = string.rep('x', math.random(1000000))  
function f ()  
    local s = string.rep('y',math.random(1000000))  
    return #(data..s)  
end  
print( f() )  
data = nil
```



Lua: standard libraries included

- Math
 - `math.cos`, `math.random`, `math.log`...
- Table
 - `table.insert`, `table.concat`, `table.sort`...
- String
 - `string.upper`, `string.match`, `string.gsub`...
- I/O
 - `io.open`, `io.write`, `io.read`...
- OS
 - `os.time`, `os.execute`, `os.getenv`...
- Debug



Lua: my own libraries

- mymodule.lua:
local M = {}
local n=1 *--privado al modulo*
M.inc = **function**(x) **return** x+n **end**
return M
- Main program:
local m = **require** ("mimodulo")
print(m.inc(10))
- **require** vs **dofile**
- **M:inc()** is the same as **M.inc(M)**
Used for OO-like access.



Lua: beware

- Variables are globals by default
 - Use `local` as needed
- Arrays
 - Start from 1
 - When there are “holes” in the numeration, `#` may behave weirdly
- `m:f(...)` is the same that `m.f(m,...)`. Check which to use in documentation or examples.
- Boolean
 - The only *falsy* values are `false` y `nil` (`0` is *truthy*)



Examples

- Power up a LED

```
led_pin = pio.GPIO14  
pio.pin.setdir(pio.OUTPUT, led_pin)  
pio.pin.setval(1, led_pin)  
tmr.sleepms(500)  
pio.pin.setval(0, led_pin)
```


Examples

- Read an analog sensor

```
channel = adc.attach(adc.ADC1, pio.GPIO32)
while true do
  local v = channel:read()
  print("reading:", v)
  tmr.sleepms(500)
end
```



- Use a I2C data bus

```
temp_sensor = i2c.attach(i2c.I2C0, i2c.MASTER, 400000)
function read_temp(sensor)
    sensor:start() -- write the register we want to obtain
    sensor:address(0x40)
    sensor:write(0xE3) -- temperature register address
    sensor:stop()
    tmr.sleepms(50) -- wait for the reading
    sensor:start() -- read the register
    sensor:address(0x40, true)
    local msb = temp_sensor:read() -- read a byte
    local lsb = temp_sensor:read()
    local chck = temp_sensor:read()
    sensor:stop()
    return msb*256+lsb
end
print('temperature:', read_temp(temp_sensor) )
```

Thanks!

References

LuaRTOS Wiki:

<https://github.com/whitecatboard/Lua-RTOS-ESP32/wiki>

Lua 5.3 Reference:

<https://www.lua.org/manual/5.3/manual.html>

Programming in Lua:

<http://www.lua.org/pil/>