Instructions for using the OpenVSM library v0.2

The library was written by me initially for purely personal purposes - to collect models in the Linux OS. Since the g ++ compiler was used, there were problems with the fact that VC++, which Proteus assembled, had different calling conventions than g++.

Among all the ways to get around problems, a method was chosen that was exactly workable on a huge number of compilers - to rewrite the code in C, since there could be no discrepancy. In the process, it became quickly clear that debugging the compiled code was not very convenient. It would be more convenient to use something like scripts at least for the time being. No sooner said than done. The language Lua was chosen, as extremely flexible and, importantly, fast.

Principle of Operation

All simulated devices use the same library, only scripts must be unique. Of course, you can use different versions of the library, but this only increases the occupied space on the disk. When creating a model, you only need to create the following parameters in the text model script:

```
{MODDLL = "MY DEVICE", HIDDEN STRING}
{LUA = "LUA", STRING}
{MODDLL = openvsm.dll}
{LUA = my_script.lua}
```

In this case , The Lua model file is called ***my_script.lua***.

The script itself should be located in the directory specified in the LUAVSM environment variable. In the most general case, it is necessary:

1. Put the openvsm.dll library in the directory with the Proteus models, or next to the DSN file (For Proteus 8 +).
2. Create a directory for scripts, or use any available
3. Create an environment variable with the path to this directory (runs in cmd.exe): **setx LUAVSM "C: \ script"**
4. Create a script file and get started.

Building a Library

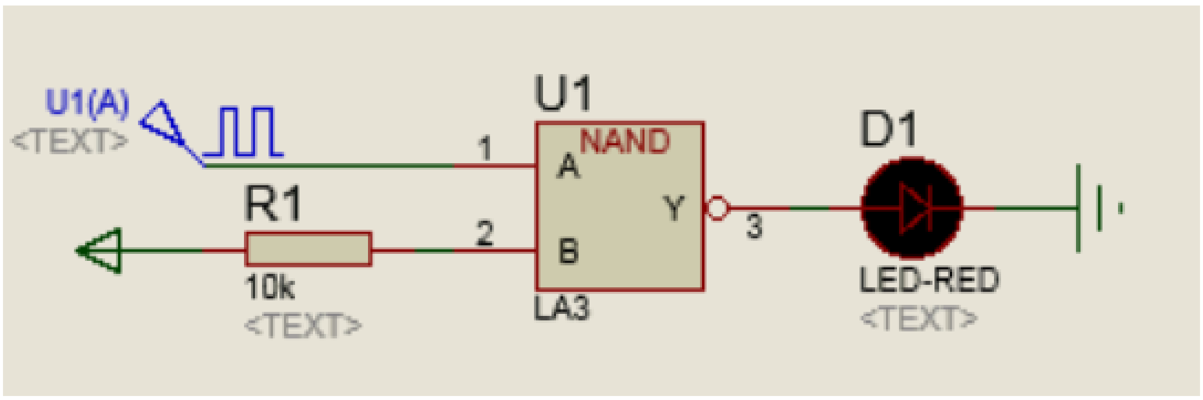
Assembly of the library is extremely rare. Or if you want to fix / improve the code, add features if you want to write a new module to the engine. Well, or simply because of the love of art.

Linux

Windows

Modeling on Lua

Let's create the simplest model and on its example we will analyze the basis of the work (All examples are available in the Sample section of the repository)



A text-based script for the model:

```
{*DEVICE}
NAME=LA3
{PREFIX=U}
{*PROPDEFS}
{PRIMITIVE="Primitive Type",HIDDEN STRING}
{MODDLL="NAND Gate",HIDDEN STRING}
{PACKAGE="PCB Package",HIDDEN PACKAGE}
{LUA="LUA",STRING}
{*INDEX}
{CAT=TTL 155 series}
{SUBCAT=Gates & Inverters}
{ITFMOD=}
{MFR=USSR}
{DESC=NAND}
{*COMPONENT}
{PRIMITIVE=DIGITAL}
{LUA=nand.lua}
{MODDLL=openvsm.dll}
{PACKAGE=DIL16}
```

And actually the script that simulates the AND gate:

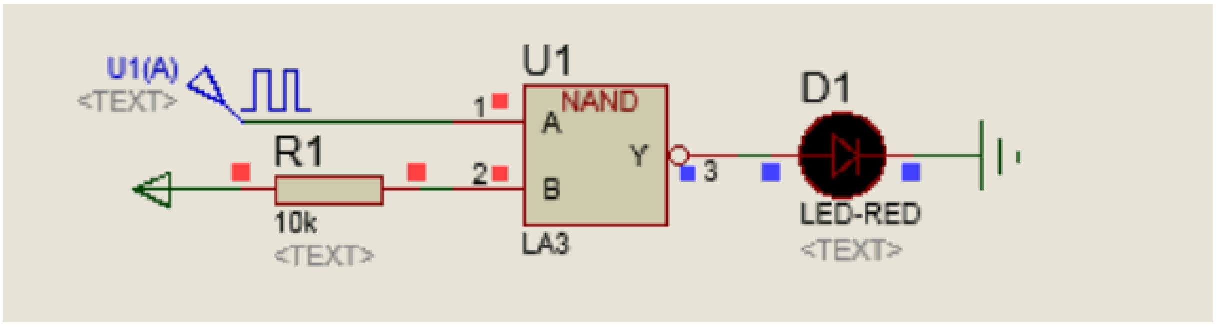
```
--NAND gate

SAFE_MODE=true
LOGIC_TYPE=TTL

device_pins =
{
   {is_digital=true, name = "A"},
   {is_digital=true, name = "B"},
   {is_digital=true, name = "Y"},
}

function device_simulate ()
   Y:set(1-(A:get() * B:get()))
End
```

```
That's all, start the simulation
```


U1(A)
<TEXT>
R1
10k
<TEXT>
U1
NAND
A
Y
B
1
2
3
LA3
D1
LED-RED
<TEXT>