1. Introduction

With the help of this simulator a micropython program becomes testable without being uploaded to real ESP32 hardware. The simulator has not implemented full hardware functionality, instead, the tool can be primarily used to check whether the control program is working properly. Hence we do not deal with the implementation of physical layer simulation of high level protocols (e.g. I2C, OWI, etc). Furthermore, it is not the goal to achieve the running speed of the real hardware. You can perform many tests even without real running speed. For example, you have a chance to find most exceptions that arise at run time only without upload to real hardware.

2. Short description

Some python libraries are available to the simulator that run at the PC, and collect and convert IO basic commands of GPIO pins and higher level commands of complex devices (e.g. I2C slaves, built in DHT, RTC, WIFI radio) from your micropython program to UDP packets. These packets have been sent to an independent GUI written in c#. The GUI processes UDP packets and shows the states of controlled peripherals on its own window. The simulated peripherals send their state changes back to your micropython program through python libraries using UDP packets as well. After your micropython program has been finished, you can upload it to ESP32 and most probably it will work properly. The simulator allows testing two or more programs running at different ESP32 devices at the same time, therefore you can examine the cooperation of simulated devices.

Simulator built in components:
WDT and Reset handling
WIFI RADIO with limited functionality (In reality, no traffic goes through it, but you can check it's setup and the connection parameters. Socket connections between simulated ESPs and/or the PC work on your local network of course)
RTC with real time, incremental and hand setup mode
LED handles PWM, too of course
PUSHBUTTON with latch (can use it for touchpad sensor, too) and Pin IRQ handling on machine side
POTENTIOMETER for analog inputs
BARGRAPH DISPLAY for analog outputs, and other display functions
DS18X20 temp sensor on non-scannable OWI bus with external socket control
DHT11, DHT22 temp & humidity sensors with external socket control

It is also possible to write custom components for the simulator. The basic GPIO and I2CSLV containers allow the integration of your component into the simulator. The GPIO container manages devices with digital and analog IO pins. The I2CSLV container manages I2C devices. You can use the high level commands (e.g. scan, write, read) and of course additional digital IO pins of device (e.g. chip select, interrupt) as well.

Custom components that have been completed so far:
ADXL345light accelerometer sensor with limited functionality
7SEGMENT DISPLAY

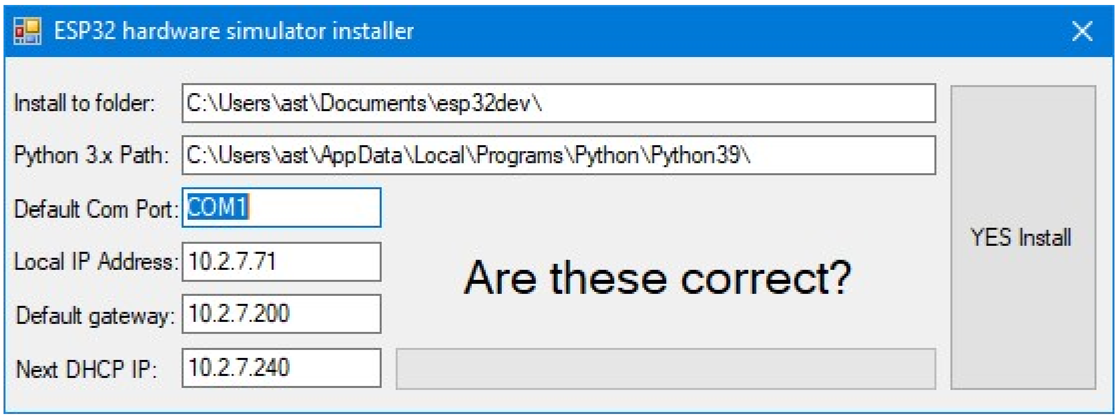
3. Preconditions

Python 3.x availability on computer. If you want to upload your projects to real ESP32 hardware, too, then **esptool** and **mpy-repl-tool** are necessary. Check them with the „pip list" command:

```
c:\Users\ast\AppData\Local\Programs\Python\Python39>pip list
Package       Version
------------- -------
bitstring     3.1.7
cffi          1.14.4
colorama      0.4.4
cryptography  3.3.1
ecdsa         0.16.1
esptool       3.0
mpy-repl-tool 0.13
pip           21.1.1
pycparser     2.20
pyserial      3.5
reedsolo      1.5.4
setuptools    49.2.1
six           1.15.0
```

4. Installation

Download E32SimInstaller.exe and start it. The following window will appear:



ESP32 hardware simulator installer

Install to folder: C:\Users\ast\Documents\esp32dev\
Python 3.x Path: C:\Users\ast\AppData\Local\Programs\Python\Python39\
Default Com Port: COM1
Local IP Address: 10.2.7.71
Default gateway: 10.2.7.200
Next DHCP IP: 10.2.7.240

Are these correct?

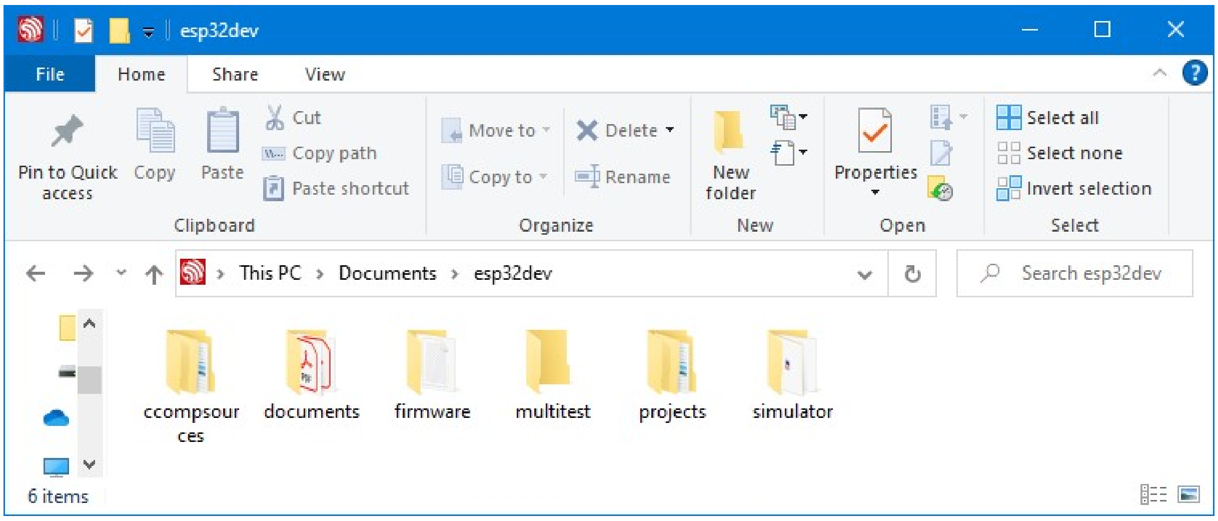
Set up your serial com port if you want to upload your projects to real ESP32 hardware as well. Push „YES Install" button. Don't panic! The simulator installer will run its own project generator four times. When the „Installation was successful" text appears, push the „Close" button.

5. Folders

After installation the following folders are created under simulator root path (in the given "Install to folder"):

ccompsources	-	source codes of custom components
documents	-	descriptions of the simulator

firmware	-	ESP32 mpy custom firmwares
multitest	-	starter links and configurations for multitesting
projects	-	project folders for simulation and upload
simulator	-	executables and configs, custom components



6. Project folders

The project folder contains the files of your individual ESP32 micropython project:

boot.py: runs at every boot on real hardware. The simulator does not run it, therefore it contains only debug information code and sets up WIFI radio to base state. If you upload it to real hardware you will able to use it to get system information in debug mode.

main.py: the control code that runs on the simulated machine and on real hardware.

config.py, config-live.py: configurations for your projects. Python codes that run on real hardware and on the PC are not necessarily the same. Although the differences are not too large, they must be chosen separately. There are flags and variables in different configurations for this. Your main.py has to import only config.py, because config-live.py will be uploaded as config.py to real hardware.

esp32simsetup.py: this file contains the setup of network socket ports and machine clock frequency. Simulated machine object in python uses these socket ports to connect to simulator GUI.

serconfig.py: tool for making extra serialized configuration, if you want use it.

SimGui.cfg: configuration file for simulator functions and GUI.

SimWifi.cfg: configuration file for simulated WIFI radios.

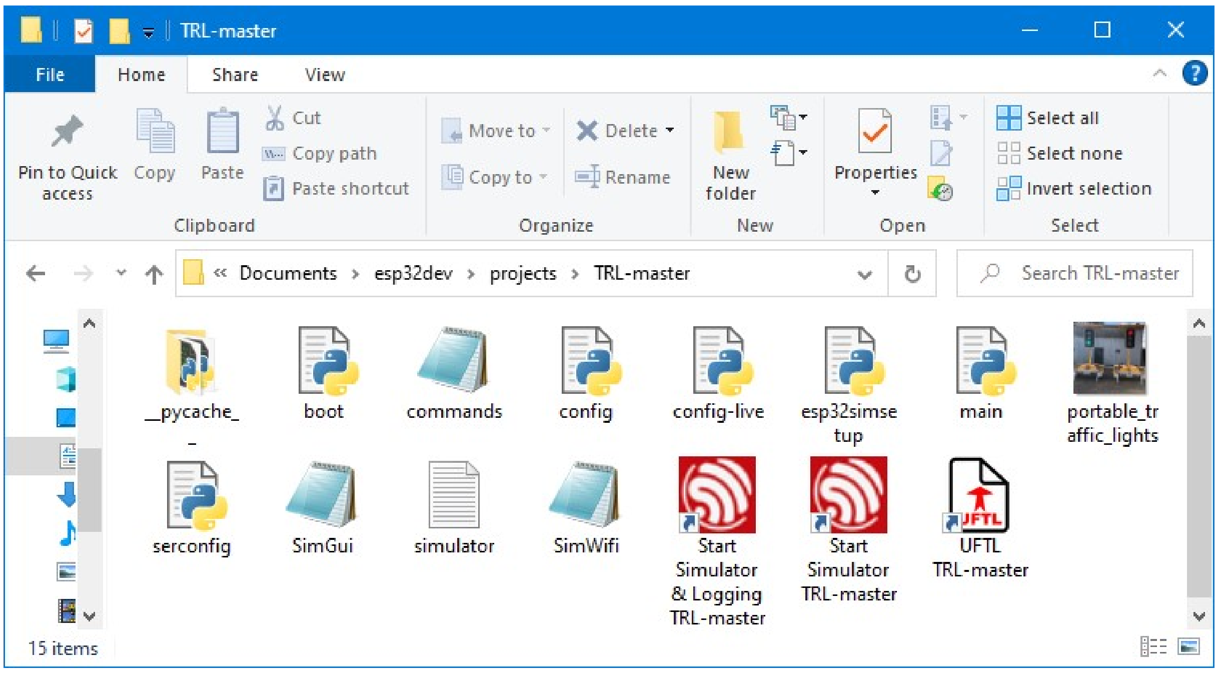
commands.cfg: configuration file for USEFULTOOL.

simulator.log: log file if you start simulator with logging option

Start Simulator Projectname.lnk: click on the link runs the simulator. Link contains the parameters necessary for the simulator.

Start Simulator & Logging Projectname.lnk: click on the link runs the simulator with logger function. Link contains the parameters necessary for the simulator.

UFTL Projectname.lnk: link for running USEFULTOOL. UFTL reads its commands.cfg and creates GUI for the complicated and long commands. For example, if you want to upload main.py to real hardware you have to push the suitable button only, and the command will be usable on clipboard immediately. Paste it to the cmd window and start upload. The commands that can be picked up are customizable. You have to rewrite the commands.cfg file only.

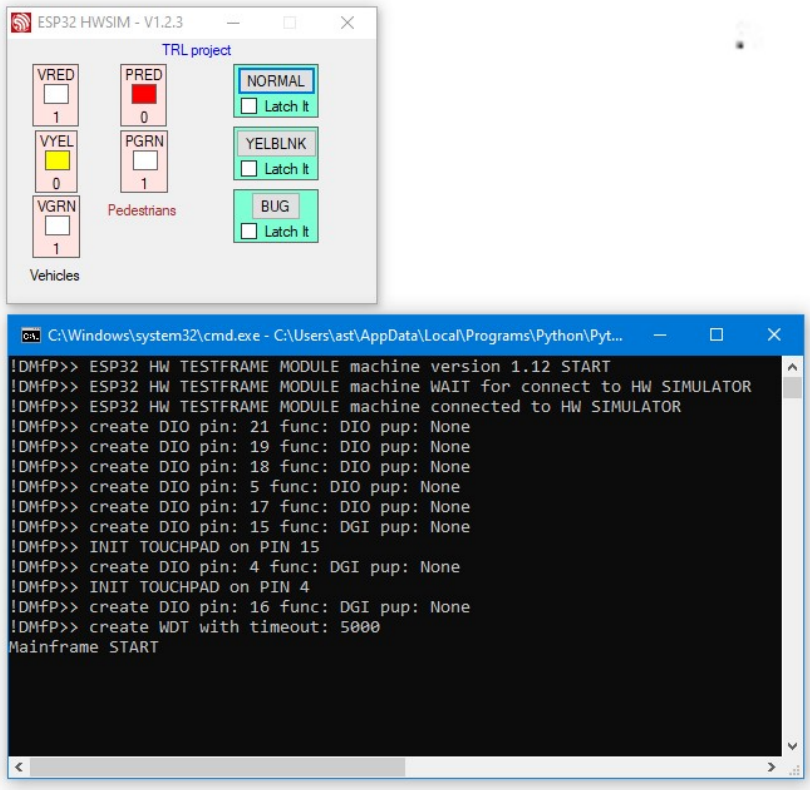


7. Template folder in projects folder

When you generate a new project, the project generator clones the content of Template folder. It replaces the project name, the path in config files and clones the file commands.cfg. All files are modifiable in Template folder, but afterwards these modified files will be the base of new cloning.

8. Example projects TRL, TRL-master, TRL-slave, ALLCompTest

<u>TRL</u> is a simple traffic light controller with lights for vehicles and pedestrians. 5 LEDs at outputs, 2 touchpad sensors and a pushbutton at inputs. To run the simulation go to project folder TRL and click on link **Start Simulator TRL**. Two windows will appear:

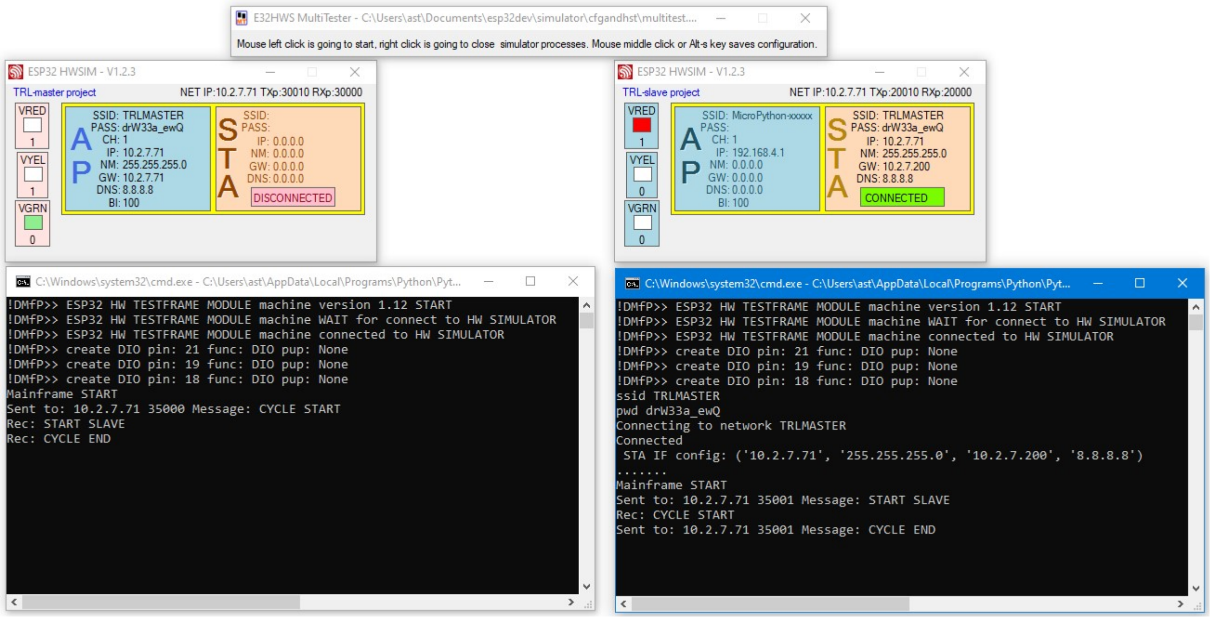


```
!DMfP>> ESP32 HW TESTFRAME MODULE machine version 1.12 START
!DMfP>> ESP32 HW TESTFRAME MODULE machine WAIT for connect to HW SIMULATOR
!DMfP>> ESP32 HW TESTFRAME MODULE machine connected to HW SIMULATOR
!DMfP>> create DIO pin: 21 func: DIO pup: None
!DMfP>> create DIO pin: 19 func: DIO pup: None
!DMfP>> create DIO pin: 18 func: DIO pup: None
!DMfP>> create DIO pin: 5 func: DIO pup: None
!DMfP>> create DIO pin: 17 func: DIO pup: None
!DMfP>> create DIO pin: 15 func: DGI pup: None
!DMfP>> INIT TOUCHPAD on PIN 15
!DMfP>> create DIO pin: 4 func: DGI pup: None
!DMfP>> INIT TOUCHPAD on PIN 4
!DMfP>> create DIO pin: 16 func: DGI pup: None
!DMfP>> create WDT with timeout: 5000
Mainframe START
```

GUI shows up in the top window, and the main.py of TRL project is running in the lower window. If you push the „Yelblink" button the pedestrians' light is going to switch off. The yellow light in the middle is going to blink on the other traffic light. If you push the „Normal" button the traffic light goes back to the normal function and plays the Red, Red-Yellow, Green – wait – Yellow, Red – wait sequence in a never ending cycle. The „Bug" button calls a function which divides by zero. The exception is going to interrupt the run of main.py. This is an excellent test for watchdog timer. The machine restarts your main.py 5 seconds after the exception.

TRL-master and TRL-slave are working together, and represent two portable traffic lights used on roadworks. Two stations are connecting together using WIFI.

The master is the WIFI access point and the slave is the WIFI station. The two stations cooperate together. When the green cycle has finished in one, it is going to message to the other. The other is doing same. To test their behavior at the same time you have to go to the simulator folder and start E32MT.exe multitester. The multitester appears at the top of the screen. You have to left click on its surface.
The multitester is going to run the TRL-master and TRL-slave projects in separate simulators. Four new windows will appear on the screen:



```
E32HWS MultiTester - C:\Users\ast\Documents\esp32dev\simulator\cfgandhst\multitest....
Mouse left click is going to start, right click is going to close  simulator processes. Mouse middle click or Alt-s key saves configuration.
```

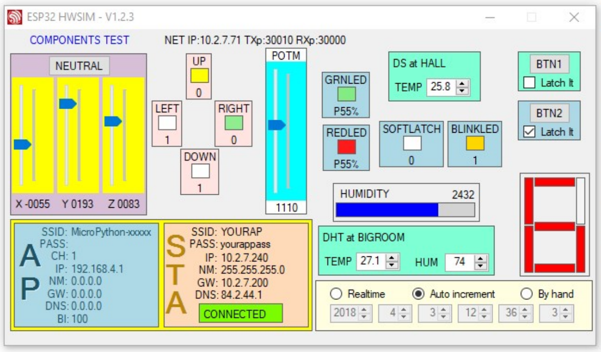
```
ESP32 HWSIM - V1.2.3
TRL-master project          NET IP:10.2.7.71 TXp:30010 RXp:30000
VRED 1   VYEL 1   VGRN 0
AP   SSID: TRLMASTER        STA  SSID:
     PASS: drW33a_ewQ            PASS:
     CH: 1                       IP: 0.0.0.0
     IP: 10.2.7.71               NM: 0.0.0.0
     NM: 255.255.255.0           GW: 0.0.0.0
     GW: 10.2.7.71               DNS: 0.0.0.0
     DNS: 8.8.8.8                DISCONNECTED
     BI: 100
```

```
ESP32 HWSIM - V1.2.3
TRL-slave project           NET IP:10.2.7.71 TXp:20010 RXp:20000
VRED 1   VYEL 0   VGRN 0
AP   SSID: MicroPython-xxxxx    STA  SSID: TRLMASTER
     PASS:                           PASS: drW33a_ewQ
     CH: 1                           IP: 10.2.7.71
     IP: 192.168.4.1                 NM: 255.255.255.0
     NM: 0.0.0.0                     GW: 10.2.7.200
     GW: 0.0.0.0                     DNS: 8.8.8.8
     DNS: 0.0.0.0                    CONNECTED
     BI: 100
```

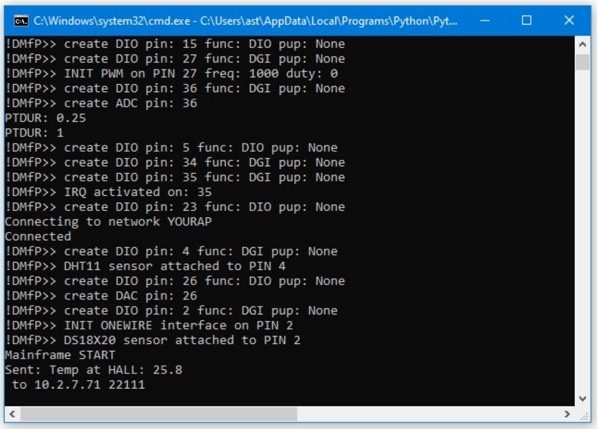
```
C:\Windows\system32\cmd.exe - C:\Users\ast\AppData\Local\Programs\Python\Pyt...
!DMfP>> ESP32 HW TESTFRAME MODULE machine version 1.12 START
!DMfP>> ESP32 HW TESTFRAME MODULE machine WAIT for connect to HW SIMULATOR
!DMfP>> ESP32 HW TESTFRAME MODULE machine connected to HW SIMULATOR
!DMfP>> create DIO pin: 21 func: DIO pup: None
!DMfP>> create DIO pin: 19 func: DIO pup: None
!DMfP>> create DIO pin: 18 func: DIO pup: None
Mainframe START
Sent to: 10.2.7.71 35000 Message: CYCLE START
Rec: START SLAVE
Rec: CYCLE END
```

```
C:\Windows\system32\cmd.exe - C:\Users\ast\AppData\Local\Programs\Python\Pyt...
!DMfP>> ESP32 HW TESTFRAME MODULE machine version 1.12 START
!DMfP>> ESP32 HW TESTFRAME MODULE machine WAIT for connect to HW SIMULATOR
!DMfP>> ESP32 HW TESTFRAME MODULE machine connected to HW SIMULATOR
!DMfP>> create DIO pin: 21 func: DIO pup: None
!DMfP>> create DIO pin: 19 func: DIO pup: None
!DMfP>> create DIO pin: 18 func: DIO pup: None
ssid TRLMASTER
pwd drW33a_ewQ
Connecting to network TRLMASTER
Connected
 STA IF config: ('10.2.7.71', '255.255.255.0', '10.2.7.200', '8.8.8.8')
.......
Mainframe START
Sent to: 10.2.7.71 35001 Message: START SLAVE
Rec: CYCLE START
Sent to: 10.2.7.71 35001 Message: CYCLE END
```

AllCompTest allows you to test all existing components of the simulator (as of Sept 2021). It includes an accelerometer sensor, four LEDs for tilt position indication, a potentiometer and two LEDs for PWM testing, an RTC and a seven segment display for showing the last digit of minutes of time, a DHT11 sensor and a bargraph display for showing humidity, a DS1820 sensor and a WIFI radio for sending the sensor's temperature value, a button and LED for blink control, and another button and LED for interrupt test in the surface of the simulator GUI.



```
ESP32 HWSIM - V1.2.3
COMPONENTS TEST             NET IP:10.2.7.71 TXp:30010 RXp:30000
NEUTRAL   X -0055   Y 0193   Z 0083
UP 0   LEFT 1   RIGHT 0   DOWN 1
POTM 1110
GRNLED P55%   REDLED P55%
DS at HALL  TEMP 25.8
SOFTLATCH 0   BLINKLED 1
BTN1 [ ] Latch It   BTN2 [✓] Latch It
HUMIDITY 2432
DHT at BIGROOM  TEMP 27.1  HUM 74
( ) Realtime   (•) Auto increment   ( ) By hand
2018  4  3  12  36  3
AP   SSID: MicroPython-xxxxx    STA  SSID: YOURAP
     PASS:                           PASS: yourappass
     CH: 1                           IP: 10.2.7.240
     IP: 192.168.4.1                 NM: 255.255.255.0
     NM: 0.0.0.0                     GW: 10.2.7.200
     GW: 0.0.0.0                     DNS: 84.2.44.1
     DNS: 0.0.0.0                    CONNECTED
     BI: 100
```



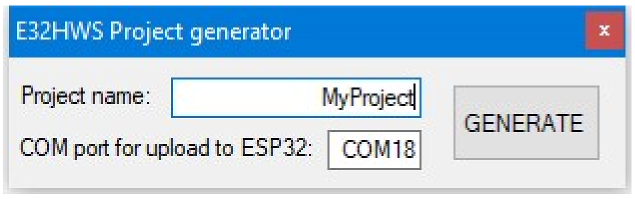
```
C:\Windows\system32\cmd.exe - C:\Users\ast\AppData\Local\Programs\Python\Pyt...
!DMfP>> create DIO pin: 15 func: DIO pup: None
!DMfP>> create DIO pin: 27 func: DGI pup: None
!DMfP>> INIT PWM on PIN 27 freq: 1000 duty: 0
!DMfP>> create DIO pin: 36 func: DGI pup: None
!DMfP>> create ADC pin: 36
PTDUR: 0.25
PTDUR: 1
!DMfP>> create DIO pin: 5 func: DIO pup: None
!DMfP>> create DIO pin: 34 func: DGI pup: None
!DMfP>> create DIO pin: 35 func: DGI pup: None
!DMfP>> IRQ activated on: 35
!DMfP>> create DIO pin: 23 func: DIO pup: None
Connecting to network YOURAP
Connected
!DMfP>> create DIO pin: 4 func: DGI pup: None
!DMfP>> DHT11 sensor attached to PIN 4
!DMfP>> create DIO pin: 26 func: DIO pup: None
!DMfP>> create DAC pin: 26
!DMfP>> create DIO pin: 2 func: DGI pup: None
!DMfP>> INIT ONEWIRE interface on PIN 2
!DMfP>> DS18X20 sensor attached to PIN 2
Mainframe START
Sent: Temp at HALL: 25.8
 to 10.2.7.71 22111
```

If you change the values of the X and Y axis with potentiometers of adxl sensor, then UP, DOWN, LEFT, RIGHT LEDs will show the tilt directions. REDLED is a traditional LED. GRNLED is a white clear LED. If you change the POTM value then GRNLED and

REDLED will emit as much light as the PWM value they received. If temperature changes at the DS sensor it will send the temperature value to the local host's 22111 port. Look at this with SckTool (for a short description see later). If SckTool is already running, you can send values to the DHT (on port 9923) and DS (on port 9924) sensors. (See instructions in ESP32_hwsim_configuration.pdf.) While BTN2 is in the pushed position BLINKLED is blinking. If you click on BTN1 then SOFTLACHLED changes its state.

9. Create your first project

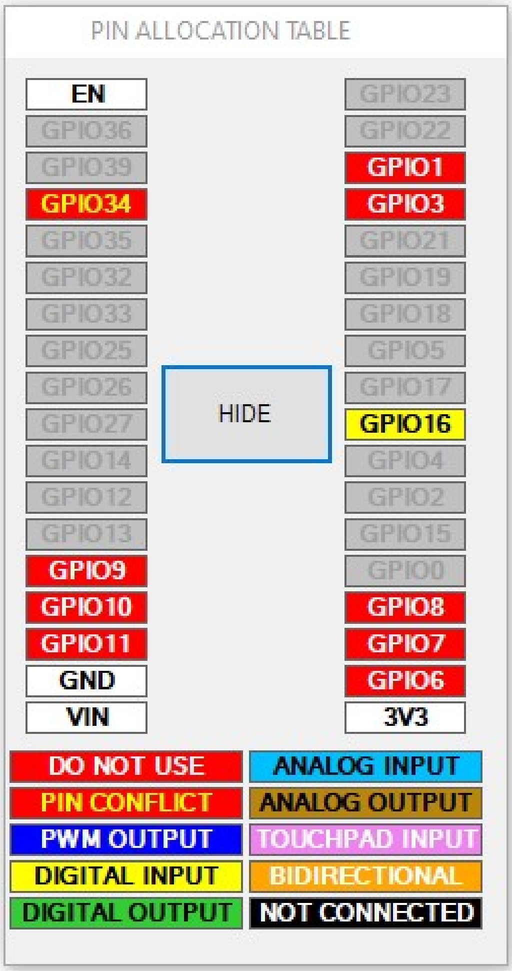
Go to the simulator folder, click on executable genE32simproj.exe and add the name of your project.


E32HWS Project generator

Project name: MyProject

COM port for upload to ESP32: COM18

MyProject is created in the projects folder. Afterwards, push „**Generate**". First you have to write the SimGui.cfg file. Customize your GUI, choose the components, and ESP32 GPIO pins what you want to use. (See instructions in ESP32_hwsim_configuration.pdf.) We are aware that it would be good to have a graphical GUI editor, but unfortunately it does not exist yet. A little help: left click on surface of simulator's GUI and position of cursor will be available on the clipboard in "posx=X,posy=Y" format. Then it can be pasted easily to file SimGui.cfg. Finally, comment out the line starting with CMD=cmdwin.... with the # character. Click on the „**Start Simulator MyProject**" icon or link if you use total commander. The simulator will start and show you the GUI with your components. If PIN ALLOCATION TABLE appears and it has a blinking item, then some kind of pin conflict occured (e.g. you want to use pin34 (GPIO34) for output).
Exit from the simulator and correct the configuration until the PIN ALLOCATION TABLE does not show itself. (The simulator reads configuration at start only.)
After that, remove the comment character in the line #CMD=cmdwin... in the SimGui.cfg file. When you start the simulator again it shows the GUI and a command line window. The simulator starts your project's main.py, which is going to run in the command line window.
You can stop the run of main.py with Ctrl-C. The main.py from „Template" contains a MAINFRAME class and a PASSIVETIMER class. PASSIVETIMER is usable for non-blocking timings in MAINFRAME's never ending loop „run(self)". When your main.py is running on PC, and if the bug in your program causes an exception then the traceback info will appear. When your main.py is running in the real hardware, then the traceback info appends to traceback.log file. After the upload, any remaining bug can be tracked. If you allocate a GPIO pin in your program that is not connected anywhere in SimGui.cfg, then PIN ALLOCATION TABLE shows itself again. If GPIO pin is blinking with white-black colors it is hanging in the air. After you corrected your main.py code you do not need to restart the whole simulator. Right clicking on the simulator GUI surface (not on the components) will copy the command line code to the clipboard. Paste it to the command line window and press enter. Your main.py is going to run again until the next bug or until pressing Ctrl-C. Before restarting your main.py code you can switch any component to base state by double clicking on the component's surface. Double clicking on the surface of the simulator switches all components to base state. If you want to show the PIN ALLOCATION state press key „p".



There are two hotkeys when you are using logger function. Press key "c" when simulator is the active window. Then the logger shows the log lines changes only. This is usable when you watch the log with tail.exe for example, and read any peripherals continuously. Return to normal logger function press key "n".

10. Configure multitester

If E32MT.exe is started without parameters, then it will read default configuration from simulator\cfgandhst\multitest.cfg. The multitest.cfg file contains some setup opportunities:

selfwinpos	-	window position on the screen
start	-	start specified simulator
delay	-	waiting between two simulator starts

selwinpos parameters:
left
top

start parameters:
project_name
[log]

delay parameter:
delay_in_seconds

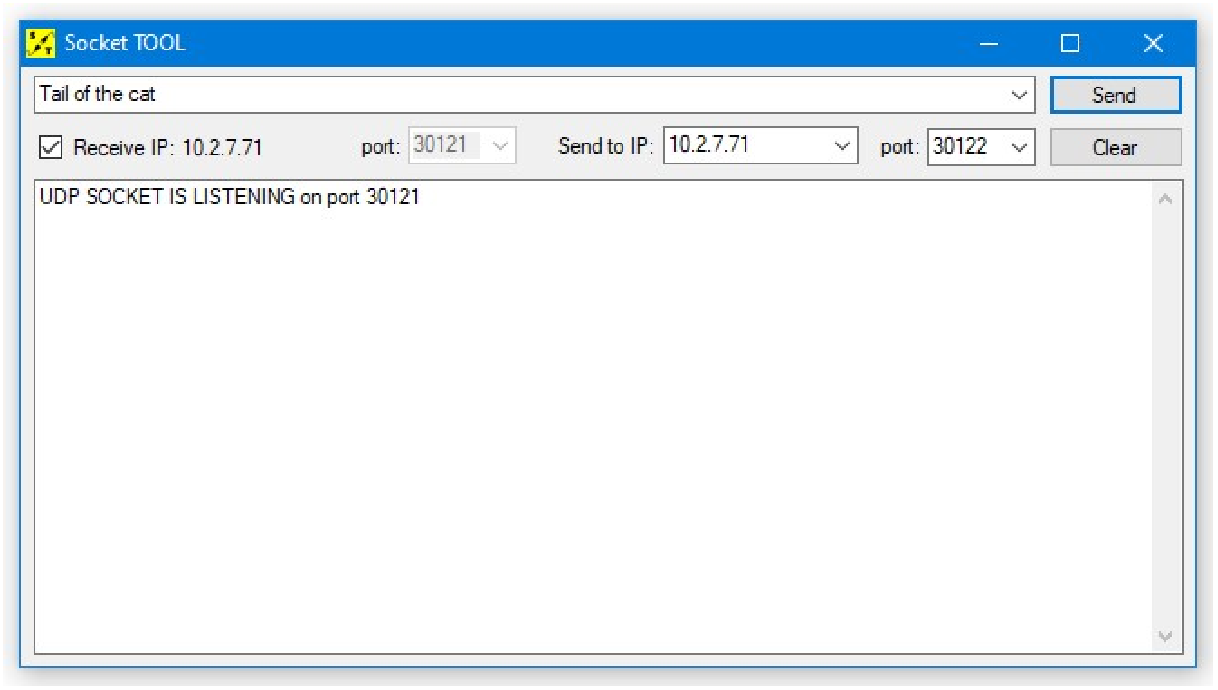
for example:

```
selfwinpos,left=300;top=4
start,TRL-master,log
delay,1
start,TRL-slave
```

It means that the E32MT window appears on screen at left=300 and top=4 position. If you click on the left button of the mouse it will start the simulator with project „TRL-master" and the started simulator will write a log. After that, wait for 1 second. Next, start another simulator with project „TRL-slave".
After you have written the suitable configuration of multitesting your projects, the configuration can be saved to the multitesting folder. Click on the middle button of the mouse on the surface of multitester's window or press Alt-s keys. Name your config and press the OK button. The multitester will write the configuration into the multitest folder and create a link with the given name that points to it. After that, the multitester can be started by clicking on the created link.
If you use the multitester, do not forget to change commport numbers in the SimGui.cfg and esp32simsetup.py files in each project's folder. Individual machine objects and simulated GUIs must be connected on different TX and RX ports. See how to do it in the TRL-master and TRL-slave folders.

11. Socket tool

Socket tool is a useful gadget for testing the socket communications of running main.py and the external environment.



Tail of the cat

☑ Receive IP: 10.2.7.71 port: 30121 Send to IP: 10.2.7.71 port: 30122

UDP SOCKET IS LISTENING on port 30121

It is receiving UDP packets on adjusted port and can send UDP packets to given IP address and port. All combo boxes save individual history, which can be cleared by clicking on the middle button of the mouse when it points to the surface of the suitable combo box.

12. Custom components

As you have already read in the second chapter, you can write custom components for the simulator. For example, you can create a power LED component controlled by PWM. Your virtual device can connect to existing virtual DS or DHT sensors and send its temperature value to sensors. You can test your PID controller program by using these.
When you want to write a custom component you have to create a dynamic link library. Look at how to do it in ccompsources/template, ccompsources/disp7seg and ccompsources/adxl345light folders, where you will find c# source codes with detailed descriptions. Put the finished dll to simulator/contandccomp folder. Start simulator with logging mode. If dll loaded successfuly, then it is ready to use.

13. ESP32 mpy custom firmwares

Why does the simulator contain custom firmwares? Because the simulated WIFI radio contains a custom parameter „apbcinterval" in ap_if.config() and the official firmware does not contain that. The default value of apbcinterval (BI in simulator's surface) is 100 (in milliseconds). This is the beacon interval of access point and causes pervasive clicking noise on DAC outputs of the real hardware. It is advisable to switch this to a longer period if you want noiseless DAC outputs and there is no possibility to switch off access point. Of course, someone who does not use DAC or is not bothered by clicking it does not have to use these firmwares.
These firmwares also contain a possibility to switch WIFI radios to long range mode. Use network.MODE_LR constant in function network.phy_mode(if, mode) for this. There are two version of modified 1.15-83 firmwares in the firmware folder. One is the standard and the other is for SPI RAM version.