LAUTERBACH
DEVELOPMENT TOOLS

TRACE32 Lua Library

Release 02.2025

MANUAL

TRACE32 Lua Library

Version 13-Feb-2025

TRACE32 Lua Library

This manual describes the TRACE32 specific extension to the standard Lua language. The functions described below can be directly used in your Lua scripts.

For a description of the **LUA** command group, refer to **"General Commands Reference Guide L"** (general_ref_l.pdf).

In this manual:

- **Lua Functions for Jtag access**

  Library to access the low-level JTAG interface (similar to the **JTAG** command group)

- **Lua Input-Output Functions**

  Library to access the input and output buffer used for communication between the Lua script and TRACE32 (see **LUA.Program.RUN** for details)

- **Example**

Functions for JTAG Access

Function Name	Description
t32jtag.writeIR(*<buf>, <len>***)**	Standard jtag function to shift to the IR register. • *<buf>*: a BYTE buffer containing data to be shifted. • *<len>*: the length in BIT to be shifted. • Return: none.
t32jtag.writeDR(*<buf>, <len>***)**	Standard jtag function to shift to the DR register. • *<buf>*: a BYTE buffer containing data to be shifted. • *<len>*: the length in BIT to be shifted. • Return: none.
t32jtag.readIR(*<len>***)**	Standard jtag function to read from the IR register. • *<len>*: the length in BIT to be read. • Return: a BYTE array containing data read from IR.
t32jtag.readDR(*<len>***)**	Standard jtag function to read from the DR register. • *<len>*: the length in BIT to be read. • Return: a BYTE array containing data read from DR.
t32jtag. readWriteIR(*<buf>, <len>***)**	Standard jtag function to simultaneously write to and read from the IR register. • *<buf>*: a BYTE buffer containing data to be shifted. • *<len>*: the length in BIT to be shifted. • Return: a BYTE array containing data read from IR.
t32jtag. readWriteDR(*<buf>, <len>***)**	Standard jtag function to simultaneously write to and read from the DR register. • *<buf>*: a BYTE buffer containing data to be shifted. • *<len>*: the length in BIT to be shifted. • Return: a BYTE array containing data read from DR.
t32jtag. WriteRaw(*<tms>, <tdi>, <len>***)**	Raw access of the JTAG interface. • *<tms>*: a BYTE buffer containing TMS data. • *<tdi>*: a BYTE buffer containing TDI data. • *<len>*: number of BITs to be shifted. • Return: none.

Function Name	Description
t32jtag. readWriteRaw(*<tms>, <tdi>, <len>***)**	Raw access of the JTAG interface. • *<tms>*: a BYTE buffer containing TMS data. • *<tdi>*: a BYTE buffer containing TDI data. • *<len>*: number of BITs to be shifted. • Return: a BYTE butter containing TDO data.
t32jtag. resetWithTMS()	Reset the JTAG interface to the part state by shifting continuous ones to TMS. • Parameter: none. • Return: none.
t32jtag.setPin(*<pin>, <value>***)**	Low-level function to set a JTAG pin (see also **JTAG.PIN**). The name of the pin must exactly match one of the following supported pins. If the name does not match, nothing is written to the signal lines. • *<pin>*: name of the pin. The following pins are supported: - TCK, TMS, TDO, TDI, NTRST, NRESET. - VTREF, EN, DIS. • *<value>*: value to be set. • Return: none.
t32jtag.getPin(*<pin>***)**	Low-level function to read a JTAG pin. • *<pin>*: name of the pin. The name of the pin must exactly match one of the following supported pins. If the given name does not match, 0 is returned. The following pins are supported: - TCK, TMS, TDO, TDI, TRESET, CTST, RTCK. - DBGACK. • Return: the value read from pin.

NOTE:	**JTAG.LOCK** command should be executed before running a Lua script that uses TRACE32 JTAG access functions, in order to avoid conflicts on the JTAG port. After running the Lua script, **JTAG.UNLOCK** command can be executed to hand back control to the debugger.

Function Name	Description
t32io.getInputBuffer(*<index>, <length>***)**	Read from the input buffer. • *<index>*: BYTE position of the input buffer to be read (up to 0x1000 bytes). • *<length>*: length in BYTE to be read.
t32io.setOutputBuffer(*<data>, <index>, <length>***)**	Writes to the output buffer. • *<data>*: lua array containing data to be written to the output buffer (up to 0x1000 bytes). • *<index>*: BYTE position of output buffer to be written. • *<length>*: length in BYTE to be written.
t32io.resetOutputBuffer()	Clear everything in the output buffer.

Example

```
-- IR and DR write
-- The following shift corresponds to command “data.set 0x3000 0xbeef”
-- On Teaklite 3/4 architecture
ir = {0,0,0,0x83}
dr = {0,0,0,0,0,0,0xef,0xbe}
t32jtag.writeIR(ir,32)
t32jtag.writeDR(dr,64)
ir = {0,0,0,0x82}
dr = {0x00,0x0c,0x00,0x80,0x0}
t32jtag.writeIR(ir,32)
t32jtag.writeDR(dr,33)

-- Read DR
dr=t32jtag.readDR(64)
-- The return value dr is a BYTE array
-- The following Lua code prints them to the console
for key,value in pairs(dr) do print(key,value) end

-- Raw shift and reset
tms = {0x00,0x00,0x00,0x03,0x00}
tdi = {0x00,0x30,0x00,0x01,0x00}
dr=t32jtag.readWriteRaw(tms,tdi,5*8);
t32jtag.resetWithTMS()

-- Read input buffer position 0
-- Use LUA.SET command to set the input buffer from host SW
test=t32io.getInputBuffer(0,4)
print("result = ",string.format("0x%08x",test)
-- Write the output buffer
t32io.setOutputBuffer(test,8,4)

-- Poll on the BRKOUT pin using t32jtag.getPin()
while (t32jtag.getPin("BRKOUT")~=1) do
end
-- Set output pin
a= {1,0,0,0,0,1,0,1,0,1,0,1,0}
b= {1,0,0,0,0,0,1,0,0,1,0,0,1}
i=0
while (i<10 and t32jtag.getPin("TDO")~=1) do
t32jtag.setPin("TDI", a[i], "TCK", b[i]);
i=i+1;
end
```