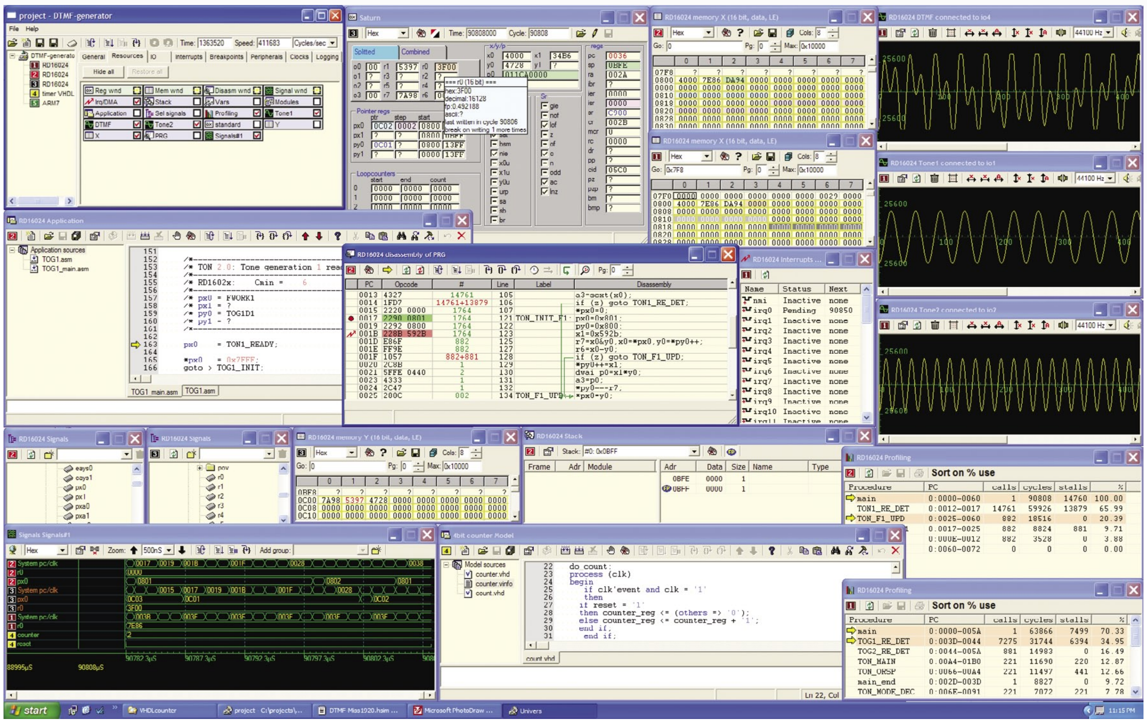
UNIVERS COVER

SW and HW verification in one hand!

GENERAL DESCRIPTION

Univers® COVER is a complete SW/HW co-verification tool for multi-processor SOC architectures, offering extensive debug capabilities, fast simulation and full speed target debugging support. It includes an RTL simulator as well as a multi-core IDE with most fascinating debug features and a configurable user-friendly graphical user interface. Adveda also provides fast cycle-accurate processor Instruction Set Simulators (ISS), which enable these features. Univers COVER is the Integrated System Co-verification tool of Adveda's UNIfied VERification Solutions.



Univers COVER offers one central cockpit from which the user can oversee his complete system-on-chip. It verifies the software running on multiple processors as well as the hardware blocks surrounding these processors. Univers COVER is the ideal solution to have the System Engineer, Software Engineer and the Hardware Engineer work together with one and the same tool. Here is a short summary of the main features:

- Fast cycle-accurate Instruction Set Simulators
- Combines multiple different ISS's
- Complete Integrated Development Environment
- Automated flow for multi-core SOC projects
- Ultra Fast Verilog/VHDL simulation
- Extensive and unique debug features
- Graphical static and dynamic profiling
- Open debug-API for integration
- Multi target (ISS, RTL, HW)
- Automatic embedded OS support

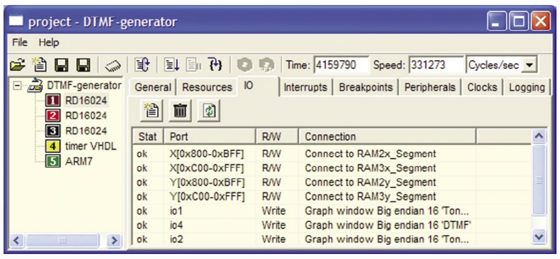
UNIVERS FAMILY

Meet the family

Univers COVER is the top-line product of Adveda's Univers family and combines Univers RTSIM a standalone RTL simulator and Univers ISSIM, a multi-core software debugger, into one unique SW/HW co-verification tool, which can run a complete system-level cycle-accurate simulation of both hardware and software at an unsurpassed speed.
Univers COVER provides a multi-window interface with which developers can activate desired functionality and select system visibility as the debugging requirements evolve. It also integrates the development tools (C-compiler, assembler, linker) for each processor and manages the flow between these tools. The flow for hardware compilation is identical to the software approach and many debugging features of both the SW and HW domain can be interwoven. All usual run-control functions are provided through menus and icon-buttons within the context of different windows. All settings of your project are saved in a XML-formatted project file.

PROJECT WINDOW

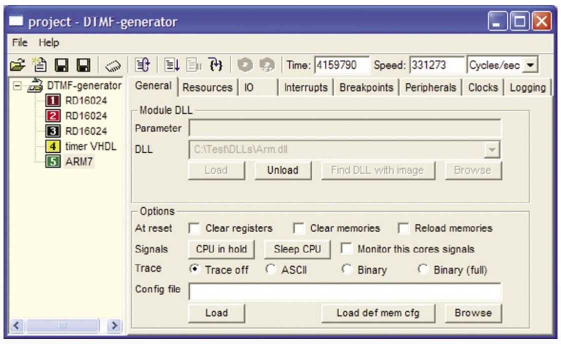
The Project Window instantiates the project and controls the overall debug process.
Every processor core and RTL peripheral in the project is instantiated in this window.
Each processor or peripheral may have its own list of features and relevant debugging windows.



Time: 4159790 Speed: 331273 Cycles/sec

Stat	Port	R/W	Connection
ok	X[0x800-0xBFF]	R/W	Connect to RAM2x_Segment
ok	X[0xC00-0xFFF]	R/W	Connect to RAM3x_Segment
ok	Y[0x800-0xBFF]	R/W	Connect to RAM2y_Segment
ok	Y[0xC00-0xFFF]	R/W	Connect to RAM3y_Segment
ok	io1	Write	Graph window Big endian 16 'Ton...
ok	io4	Write	Graph window Big endian 16 'DTMF'
ok	io2	Write	Graph window Big endian 16 'Ton...

The project window also contains a logging field as well as a command prompt. Furthermore,clocks, IO, DMA, breakpoints as well as inter-processing communication can be setup via this window.

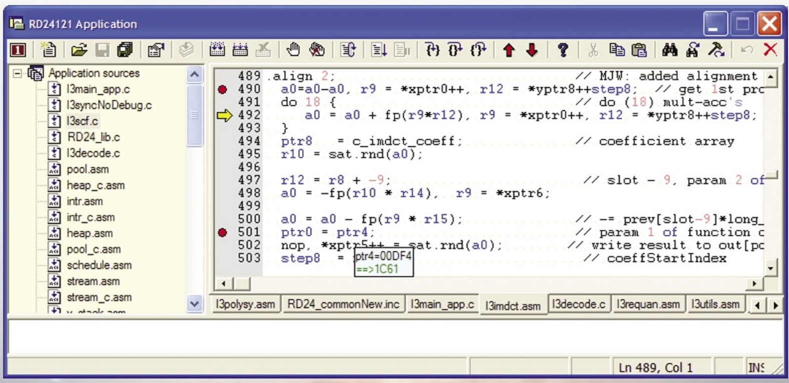


Time: 4159790 Speed: 331273 Cycles/sec

Parameter:
DLL: C:\Test\DLLs\Arm.dll
At reset: ☐ Clear registers ☐ Clear memories ☐ Reload memories
Signals: ☐ Monitor this cores signals
Trace: [✓ Trace off] ASCII Binary Binary (full)
Config file:

APPLICATION WINDOW

Each processor has an Application Window, which contains the software source codes of the project for that processor. The object code can be built with all related files or can be individually compiled and assembled, under menu or button control.



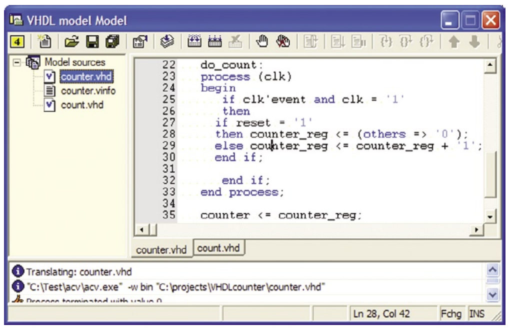
```
.align 2;                                           // MJW: added alignment
 a0=a0-a0, r9 = *xptr0++, r12 = *yptr8++step8;   // get 1st prc
 do 18 {                                            // do (18) mult-acc's
    a0 = a0 + fp(r9*r12), r9 = *xptr0++, r12 = *yptr8++step8;
 }
 ptr8   = c_imdct_coeff;                            // coefficient array
 r10 = sat.rnd(a0);

 r12 = r8 + -9;                                      // slot - 9, param 2 of
 a0 = -fp(r10 * r14),  r9 = *xptr6;

 a0 = a0 - fp(r9 * r15);                            // -= prev[slot-9]*long_
 ptr0 = ptr4;                                       // param 1 of function c
 nop, *xptr5++ = sat.rnd(a0);                      // write result to out[pc
 step8   =                                           // coeffStartIndex
```

In the Application Window, the source code can be viewed, edited and (re-) compiled. The source code is context-sensitively colored for ease of reading. For easy debugging, the values of variables pop-up when the mouse is moved over the associated source code text. The Application Window also allows a walk through a function's backtrace, while simultaneously updating the Variables Windows, Stack Windows and Application Window itself, thus facilitating easy tracing of function calls. Obviously, breakpoints in your source-code text can be placed as well.

SOC verification at unsurpassed speed

RTL MODEL WINDOW

Every HDL design has a Model Window, which contains the RTL source codes of such a peripheral. In the Model Window, the RTL code can be viewed, edited and (re-) compiled. The RTL code is context-sensitively colored for ease of reading. For easy debugging, the values of signals pop-up when the mouse is moved over the associated source code text. Breakpoints in your RTL code can be placed as well.


VHDL model Model

Model sources: counter.vhd, counter.vinfo, count.vhd

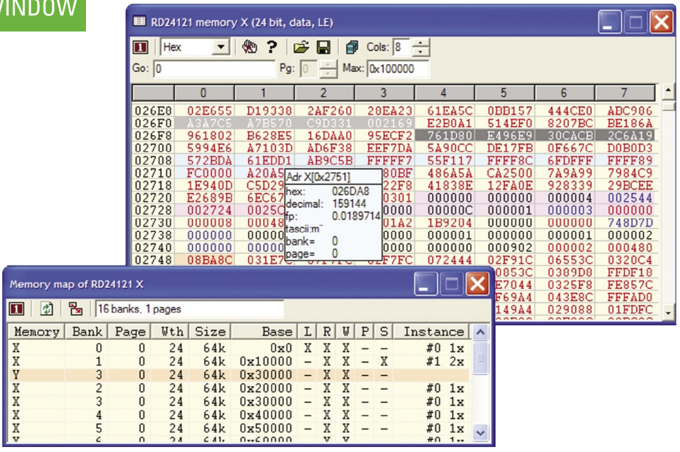
```
    do_count:
    process (clk)
    begin
       if clk'event and clk = '1'
       then
      if reset = '1'
      then counter_reg <= (others => '0');
      else counter_reg <= counter_reg + '1';
      end if;

       end if;
    end process;

    counter <= counter_reg;
```

Tabs: counter.vhd | count.vhd

```
Translating: counter.vhd
"C:\Test\acv\acv.exe"  -w bin "C:\projects\VHDLcounter\counter.vhd"
```

Ln 28, Col 42 | Fchg | INS

MEMORY WINDOW


RD24121 memory X (24 bit, data, LE)

Hex | Cols: 8 | Go: 0 | Pg: 0 | Max: 0x100000

```
         0       1       2       3       4       5       6       7
026E8  02E655  D19338  2AF260  28EA23  61EA5C  0BB157  444CE0  ABC986
026F0  A3A7C5  A7B570  C9D331  002169  E2B0A1  514EF0  8207BC  BE186A
026F8  961802  B628E5  16DAA0  95ECF2  761D80  E496E9  30CACB  2C6A19
02700  5994E6  A7103D  AD6F38  EEF7DA  5A90CC  DE17FB  0F667C  D0B0D3
02708  572BDA  61EDD1  AB9C5B  FFFFF7  55F117  FFFF8C  6FDFFF  FFFF89
02710  FC0000  A20A5             80BF  486A5A  CA2500  7A9A99  7984C9
02718  1E940D  C5D29             22F8  41838E  12FA0E  928339  29BCEE
02720  E2689B  6EC67             0301  000000  000000  000004  002544
02728  002724  0025C             0000  00000C  000001  000003  000000
02730  000008  00048             01A2  1B9204  000000  000000  748D7D
02738  000000  00000             0000  000001  000000  000001  000002
02740  000000  00000             0000  000000  000902  000002  000480
02748  08BA8C  031E7              F7FC  072444  02F91C  06553C  0320C4
                                        0853C  0389D8  FFDF18
                                        E7044  0325F8  FE857C
                                        F69A4  043E8C  FFFAD0
                                        149A4  029088  01FDFC
```

```
Adr X[0x2751]
hex:      026DA8
decimal:  159144
fp:       0.0189714
tascii:m"
bank=     0
page=     0
```

Memory map of RD24121 X

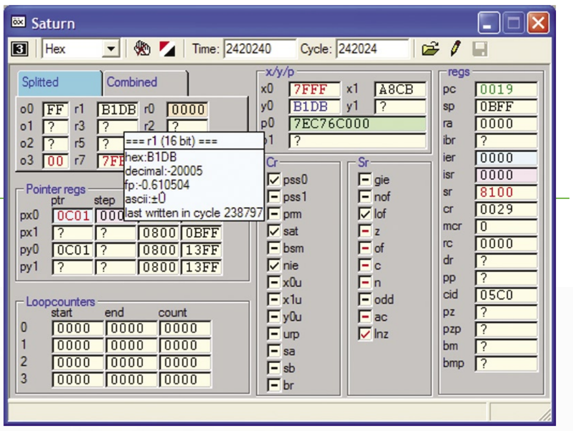
16 banks, 1 pages

Memory	Bank	Page	Wth	Size	Base	L	R	W	P	S	Instance
X	0	0	24	64k	0x0	X	X	X	–	–	#0 1x
X	1	0	24	64k	0x10000	–	X	X	–	X	#1 2x
Y	3	0	24	64k	0x30000	–	X	X	–	–	
X	2	0	24	64k	0x20000	–	X	X	–	–	#0 1x
X	3	0	24	64k	0x30000	–	X	X	–	–	#0 1x
X	4	0	24	64k	0x40000	–	X	X	–	–	#0 1x
X	5	0	24	64k	0x50000	–	X	X	–	–	#0 1x

Every memory space may be made visible in multiple windows. In such a window one may set all types of breakpoints at every individual location or simultaneously at a range of locations. In every Memory Window one can define how a memory is instantiated and/or shared with other processors or peripheral blocks.

REGISTER WINDOW

Every processor has its own set of registers, which are displayed in a separate processor Register Window. It is also possible to display the actual registers of the RTL hardware peripherals in the same or another register window. Besides just viewing the content of registers, it is possible to overwrite these or set various types of breakpoints for each register. The register contents are color-coded indicating the type of usage of the registers since last execution.


Saturn

Hex | Time: 2420240 | Cycle: 242024

Splitted | Combined

```
o0 FF    r1 B1DB   r0 0000
o1 ?     r3 ?      r2 ?
o2 ?     r5 ?
o3 00    r7 7FF
```

```
=== r1 (16 bit) ===
hex:B1DB
decimal:-20005
fp:-0.610504
ascii:±Û
last written in cycle 238797
```

Pointer regs

```
      ptr    step
px0   0C01   000
px1   ?      ?      0800  0BFF
py0   0C01   ?      0800  13FF
py1   ?      ?      0800  13FF
```

Loopcounters

```
    start  end   count
0   0000   0000  0000
1   0000   0000  0000
2   0000   0000  0000
3   0000   0000  0000
```

x/y/p

```
x0  7FFF        x1  A8CB
y0  B1DB        y1  ?
p0  7EC76C000
p1  ?
```

Cr: [✓] pss0, [–] pss1, [–] prm, [✓] sat, [–] bsm, [✓] nie, [–] x0u, [–] x1u, [–] y0u, [–] urp, [–] sa, [–] sb, [–] br

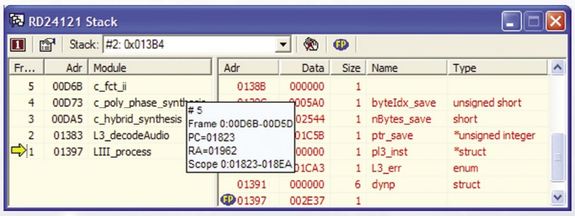
Sr: [–] gie, [–] nof, [✓] lof, [–] z, [–] of, [–] c, [–] n, [–] odd, [–] ac, [✓] lnz

regs

```
pc   0019
sp   0BFF
ra   0000
ibr  ?
ier  0000
isr  0000
sr   8100
cr   0029
mcr  0
rc   0000
dr   ?
pp   ?
cid  05C0
pz   ?
pzp  ?
bm   ?
bmp  ?
```

STACK WINDOW

Univers COVER automatically detects stack operations as well as different thread actions and will store both into the Stack Window. This allows Univers COVER to support any OS on any type of processor. This is one of the unique features of this integrated development environment.


RD24121 Stack

Stack: #2: 0x013B4

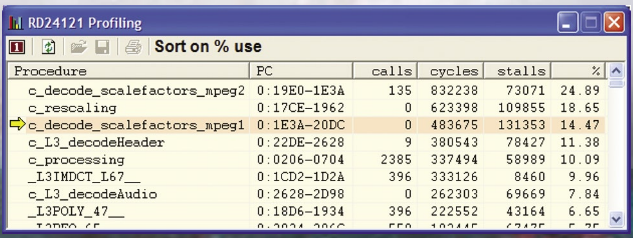
Fr...	Adr	Module
5	00D6B	c_fct_ii
4	00D73	c_poly_phase_synth
3	00DA5	c_hybrid_synthesis
2	01383	L3_decodeAudio
⇨1	01397	LIII_process

```
# 5
Frame 0:00D6B-00D5D
PC=01823
RA=01962
Scope 0:01823-018EA
```

Adr	Data	Size	Name	Type
0138B	000000	1		
[illegible]	005A0	1	byteIdx_save	unsigned short
[illegible]	02544	1	nBytes_save	short
[illegible]	01C5B	1	ptr_save	*unsigned integer
[illegible]	00000	1	pl3_inst	*struct
[illegible]	01CA3	1	L3_err	enum
01391	000000	6	dynp	struct
FP 01397	002E37	1		

PROFILE WINDOW

The Profile Window collects all data at run-time and it may be viewed at each moment in time. The profiler displays execution cycles, stall cycles and types of stalls for each function in a very convenient way, highlighting the options for optimizing the application programs. The profiler supports graphical static profiling as well as graphical dynamic profiling.


RD24121 Profiling

Sort on % use

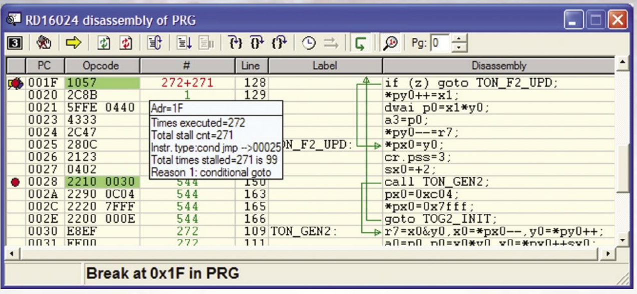
Procedure	PC	calls	cycles	stalls	%
c_decode_scalefactors_mpeg2	0:19E0-1E3A	135	832238	73071	24.89
c_rescaling	0:17CE-1962	0	623398	109855	18.65
⇨ c_decode_scalefactors_mpeg1	0:1E3A-20DC	0	483675	131353	14.47
c_L3_decodeHeader	0:22DE-2628	9	380543	78427	11.38
c_processing	0:0206-0704	2385	337494	58989	10.09
_L3IMDCT_L67__	0:1CD2-1D2A	396	333126	8460	9.96
c_L3_decodeAudio	0:2628-2D98	0	262303	69669	7.84
_L3POLY_47__	0:18D6-1934	396	222552	43164	6.65

UNIVERS COVER

UNIVERS COVER

DISASSEMBLY WINDOW

Every processor may have its own Disassembly Window containing a lot of information about the execution of the program. It doesn't only show the disassembled code, but also provides information how many times instructions are executed, how many stalls are introduced by each instruction and which effect caused the stall. This provides all basic information, which can be used to improve the overall performance of your application.

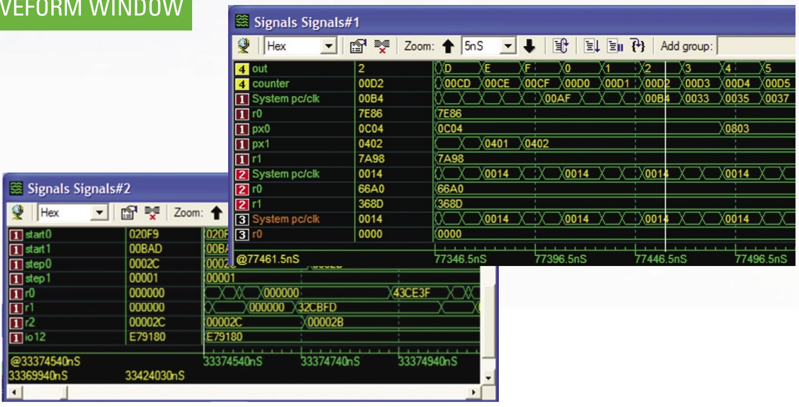

RD16024 disassembly of PRG

```
PC    Opcode      #         Line  Label          Disassembly
001F  1057        272+271   128                  if (z) goto TON_F2_UPD;
0020  2C8B        1         129                  *py0++=x1;
0021  5FFE 0440                                  dwai p0=x1*y0;
0023  4333                                       a3=p0;
0024  2C47                                       *py0--=r7;
0025  280C                        ON_F2_UPD:     *px0=y0;
0026  2123                                       cr.pss=3;
0027  0402                                       sx0=+2;
0028  2210 0030   544       150                  call TON_GEN2;
002A  2290 0C04   544       163                  px0=0xc04;
002C  2220 7FFF   544       165                  *px0=0x7fff;
002E  2200 000E   544       166                  goto TOG2_INIT;
0030  E8EF        272       109   TON_GEN2:      r7=x0&y0,x0=*px0--,y0=*py0++;
```

```
Adr=1F
Times executed=272
Total stall cnt=271
Instr. type:cond jmp -->00025
Total times stalled=271 is 99
Reason 1: conditional goto
```

Break at 0x1F in PRG

WAVEFORM WINDOW


Signals Signals#1

```
4 out              2
4 counter          00D2
1 System pc/clk    00B4
1 r0               7E86
1 px0              0C04
1 px1              0402
1 r1               7A98
2 System pc/clk    0014
2 r0               66A0
2 r1               368D
3 System pc/clk    0014
3 r0               0000
@77461.5nS   77346.5nS   77396.5nS   77446.5nS   77496.5nS
```

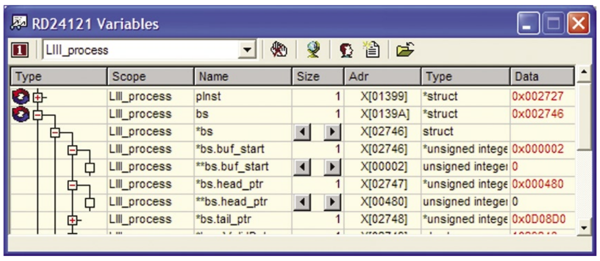
Signals Signals#2

```
1 start0   020F9
1 start1   00BAD
1 step0    0002C
1 step1    00001
1 r0       000000
1 r1       000000
1 r2       00002C
1 io12     E79180
@33374540nS   33374540nS   33374740nS   33374940nS
33369940nS    33424030nS
```

Within the Waveform Window every signal of the RTL code can be viewed. Also any value within the ISS of each processor or each peripheral can be traced in the same or another Waveform Window. This enables the display of the history of selected signals and may give a very good understanding of detailed cycle aspects. It allows to debug and fine-tune the application code together with RTL signals.

VARIABLES WINDOW

All variables are displayed in the Variables Window. Variables may be selected per procedure or from a user-defined list. The window shows the value of a variable as well as his absolute address in one of the associated memory spaces. For RTL signals there is a similar Signal Window.


RD24121 Variables — LIII_process

Type	Scope	Name	Size	Adr	Type	Data
	LIII_process	pInst	1	X[01399]	*struct	0x002727
	LIII_process	bs	1	X[0139A]	*struct	0x002746
	LIII_process	*bs		X[02746]	struct	
	LIII_process	*bs.buf_start	1	X[02746]	*unsigned intege	0x000002
	LIII_process	**bs.buf_start		X[00002]	unsigned intege	0
	LIII_process	*bs.head_ptr	1	X[02747]	*unsigned intege	0x000480
	LIII_process	**bs.head_ptr		X[00480]	unsigned intege	0
	LIII_process	*bs.tail_ptr	1	X[02748]	*unsigned intege	0x0D08D0

ISS AND HW-TARGET

Univers COVER is a complete solution for hardware and software co-design and verification. When used together with the fast Instruction Set Simulators from Adveda, all debug features can be used. Other Instruction Set Simulators may be integrated too, but might have less debug capabilities. Univers COVER also supports interfaces to real hardware implementations of the processors. Some debug features may not be available in such mode since not all data, available from the ISS models, can be obtained from real hardware implementations. The amount of debug features will vary per processor implementation.

ADVEDA

VERIFICATION SOLUTIONS FOR CHIP DESIGN