Codescape Debugger 8

Codescape Debugger is Imagination's bespoke debug environment for heterogeneous SoC development. It has recently undergone a major overhaul to include many new features for native debugging of MIPS targets and other IP from Imagination. From Codescape Debugger you can simultaneously debug MIPS CPU cores, as well as Ensigma communications RPU cores in a single environment, with more heterogeneous processing features coming soon.

Host environments

Linux

Microsoft Windows

Mac

Debug environments

MEOS by Imagination

Linux

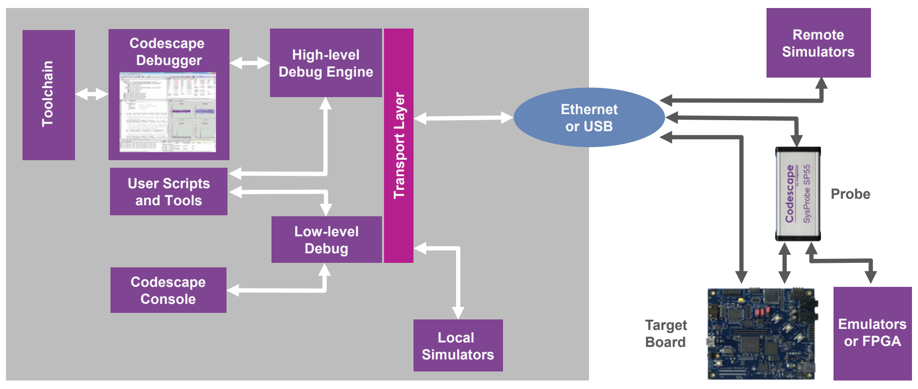
THREADX

NUCLEUS

freeRTOS

The Codescape Development System

Codescape Debugger forms the hub of a system that facilitates all stages of development alongside a low-level command-line console, built-in scripting, intelligent debug probes, emulators and simulators. For pre-hardware application development, Codescape Debugger works with the MIPS Instruction Accurate Simulator (IASim) and QEMU emulator. For silicon bring-up, application development, and testing on real hardware, Codescape Debugger can connect to Imagination's range of debug probes supporting JTAG, cJTAG and EJTAG-equipped targets. Host connection via USB or Ethernet enables remote debugging across networks.



Features

Simultaneous multiple debug adapter connections to multi-SoC, multi-core, multi-VPE with multi-OS task support

OS-aware debug support for Linux®, ThreadX®, Nucleus™, Free RTOS and MEOS™

Ultra-fast debugging performance using intelligent, low latency probes (<1 sec step with multiple connections, VPEs, TCs and threads. >1 MByte/sec binary load)

Configurable debug regions

Python scripting support

Run external tools such as make and user scripts at a key press

Ensigma RPU support – fully-featured MCP core debugging

Rogue Shader debug

Benefits

Powerful and cost-effective system for all members of SoC and application development teams

Single IDE for the entire development cycle

Mature solutions that minimize risk and reduce time to market

Proven technology used by major SoC manufacturers

Lifetime product support

Applications

IP evaluation

SoC design

SoC bring-up

Driver development

Application development

Code optimization

The Codescape Debugger User Interface

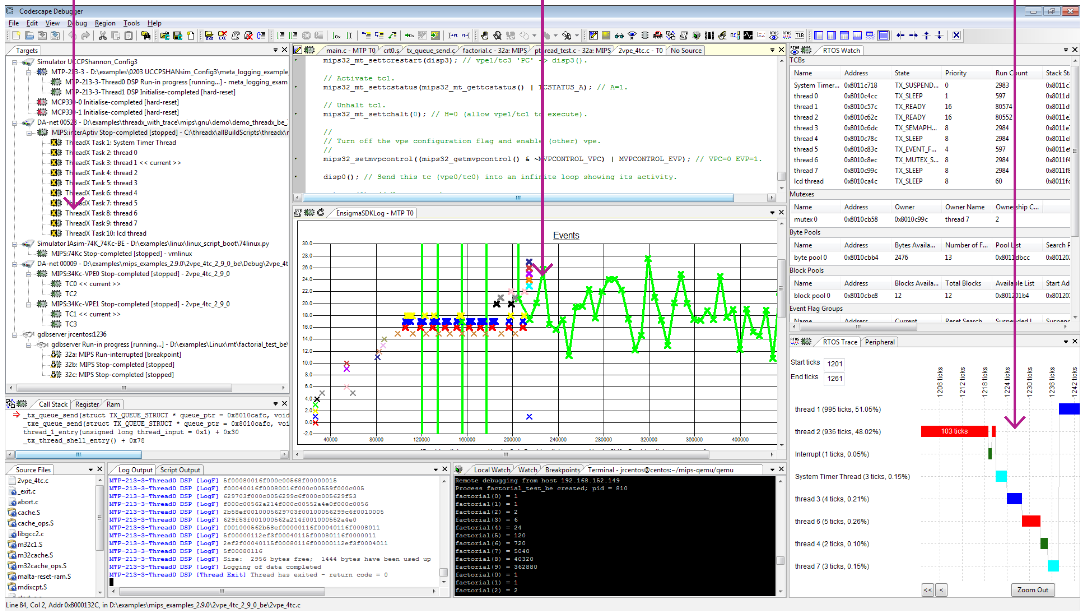
Codescape Debugger is driven from its own powerful GUI, running natively on 32-bit and 64-bit Windows, Linux and Mac OS hosts. For fast, efficient debugging, it provides a host of useful debug views and features such as editable memory, drag and drop between views and real-time OS-aware debugging.

For more advanced product development, Codescape Debugger has many state-of-the-art features such as built-in graphical scripting, a fully-annotated memory mapped peripheral inspector, real-time event tracing and support for multi-SoC, multi-core, multi-VPE and multi-OS task development and debugging.

Simultaneously connect to a variety of multi-VPE and multi-core real and simulated targets debugging multiple OSs.

Use advanced in-built graphical scripting for data visualisation. This example shows event profiling on an Ensigma RPU.

View task swaps between threads with native RTOS aware debugging.



```
mips32_mt_settcrestart(disp3); // vpe1/tc3 'PC' -> disp3().

// Activate tc1.
mips32_mt_settcstatus(mips32_mt_gettcstatus() | TCSTATUS_A); // A=1.

// Unhalt tc1.
mips32_mt_settchalt(0); // H=0 (allow vpe1/tc1 to execute).

//
// Turn off the vpe configuration flag and enable (other) vpe.
//
mips32_setmvpcontrol((mips32_getmvpcontrol() & ~MVPCONTROL_VPC) | MVPCONTROL_EVP); // VPC=0 EVP=1.

disp0(); // Send this tc (vpe0/tc0) into an infinite loop showing its activity.
```

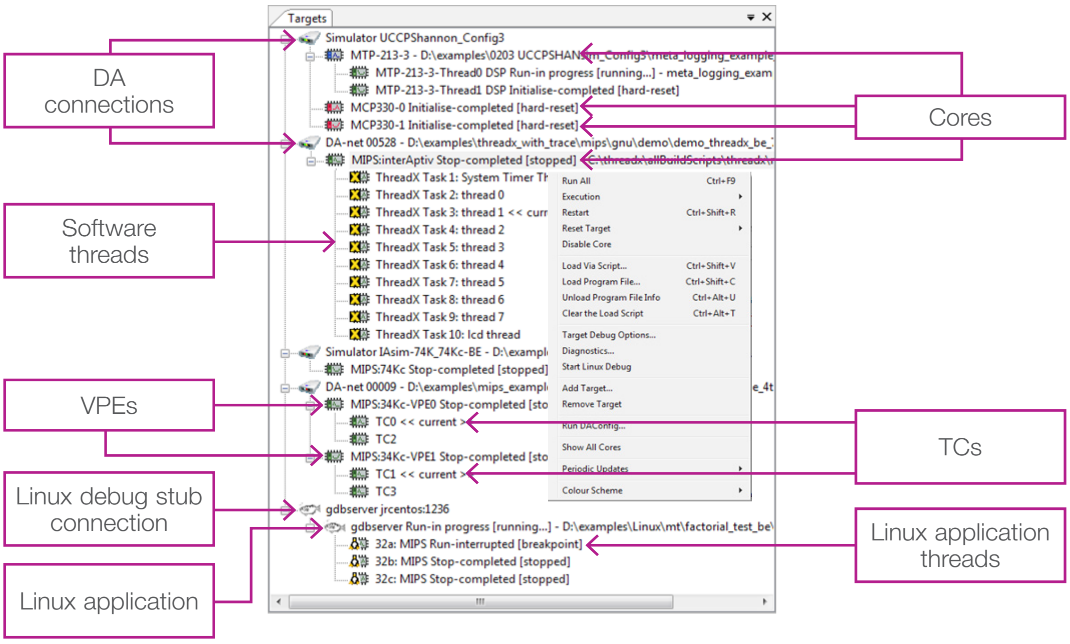
```
MTP-213-3-Thread0 DSP [LogF] 5f00080016f000c00568f00000015
MTP-213-3-Thread0 DSP [LogF] f00040016f00080016f000c00559f000c005
MTP-213-3-Thread0 DSP [LogF] 629703f000c0056299c6f000c005629f53
MTP-213-3-Thread0 DSP [LogF] f000c00562a214f000c00552a4e0f000c0056
MTP-213-3-Thread0 DSP [LogF] 2b58ef001000562970 3f00100056299c6f0010005
MTP-213-3-Thread0 DSP [LogF] 629f53f001000562a214f001000552a4e0
MTP-213-3-Thread0 DSP [LogF] f001000562b58ef00000116f00040116f0008011
MTP-213-3-Thread0 DSP [LogF] 5f00000112ef3f00040115f00080116f0000011
MTP-213-3-Thread0 DSP [LogF] 2ef2f00040115f00080116f00000112ef3f0004011
MTP-213-3-Thread0 DSP [LogF] 5f00080116
MTP-213-3-Thread0 DSP [LogF] Size:  2956 bytes free;  1444 bytes have been used up
MTP-213-3-Thread0 DSP [LogF] Logging of data completed
MTP-213-3-Thread0 DSP [Thread Exit] Thread has exited - return code = 0
```

```
Remote debugging from host 192.168.152.149
Process factorial_test_be created; pid = 810
factorial(0) = 1
factorial(1) = 1
factorial(2) = 2
factorial(3) = 6
factorial(4) = 24
factorial(5) = 120
factorial(6) = 720
factorial(7) = 5040
factorial(8) = 40320
factorial(9) = 362880
factorial(0) = 1
factorial(1) = 1
factorial(2) = 2
```

In Codescape Debugger's intuitive flexible, real estate conscious GUI, regions can be docked, floated or tabbed.

What you can debug…

Codescape Debugger has been designed from the ground up to have multiple connections to multiple, heterogeneous, multi-core SoCs with hardware threads, virtual processors (VPEs), hardware thread contexts (TCs), and multiple software threads.



```
Simulator UCCPShannon_Config3
  MTP-213-3 - D:\examples\0203 UCCPSHANsim_Config3\meta_logging_example
    MTP-213-3-Thread0 DSP Run-in progress [running...] - meta_logging_exam
    MTP-213-3-Thread1 DSP Initialise-completed [hard-reset]
  MCP330-0 Initialise-completed [hard-reset]
  MCP330-1 Initialise-completed [hard-reset]
DA-net 00528 - D:\examples\threadx_with_trace\mips\gnu\demo\demo_threadx_be_
  MIPS:interAptiv Stop-completed [stopped] - C:\threadx\allBuildScripts\threadx\
    ThreadX Task 1: System Timer Th
    ThreadX Task 2: thread 0
    ThreadX Task 3: thread 1 << curr
    ThreadX Task 4: thread 2
    ThreadX Task 5: thread 3
    ThreadX Task 6: thread 4
    ThreadX Task 7: thread 5
    ThreadX Task 8: thread 6
    ThreadX Task 9: thread 7
    ThreadX Task 10: lcd thread
Simulator IAsim-74K_74Kc-BE - D:\exampl
  MIPS:74Kc Stop-completed [stopped]
DA-net 00009 - D:\examples\mips_exampl                     e_4t
  MIPS:34Kc-VPE0 Stop-completed [sto
    TC0 << current >>
    TC2
  MIPS:34Kc-VPE1 Stop-completed [sto
    TC1 << current >>
    TC3
gdbserver jrcentos:1236
  gdbserver Run-in progress [running...] - D:\examples\Linux\mt\factorial_test_be\
    32a: MIPS Run-interrupted [breakpoint]
    32b: MIPS Stop-completed [stopped]
    32c: MIPS Stop-completed [stopped]
```

Debug Regions

Codescape Debugger provides a host of useful debugging regions to display data, and using the extensive scripting support you can create bespoke regions and plug-ins to display your data in unique ways. Regions can be tied to specific threads, or the current thread, and can be docked, floating or stacked in tabbed groups. Data can be dragged between regions and targets using intelligent, contextual drag-and-drop.

Source	Fully featured syntax highlighted editor.
Hex Editor	Edit and display binaries in many formats.
Disassembly	Can show interleaved source and disassembly.
Register	Layout is user definable and can display in different radices/formats.
Callstack	Can unwind through interrupt handlers. Uses code reading and debug info.
Memory	Shows all types of memory (RAM, DSP, CORE etc) in many different formats.
Breakpoint	Shows breakpoint state for all, current, or specific threads.
Watch	Watch and edit values of variables or complex expressions.
Local Watch	Automatically populated with variables in the current scope.
Peripheral Watch	Populated with the peripheral registers. Bit fields shown/edited as mnemonics.
RTOS Watch	Automatically populated with RTOS data e.g. threads, mutexes, block pools etc.
RTOS Trace	Graphically displays the Task execution captured by the RTOS.
TLB	Displays all TLB in Raw or Decoded format.
ICache	Displays the ICache in a human readable format.
DCache	Displays the DCache in a human readable format.
Script	Create your own region using wxWidgets & Python.
Terminal	VT100 emulator. Stream output to file.
Profiler	Low impact PC capture. Data shown next to code.
Realtime Trace	Graphical representation of Real Time Trace data.
Trace Results	Setup and display data from the MIPS PDTrace system.
Overlay	Shows the current status of Overlays.

Linux Application Debug

Linux applications can be debugged via gdbserver running on Linux on the target. Connecting to gdbserver over a specified port, Codescape Debugger displays gdbserver as a target, and when debugging Linux user code with multiple threads, each pthread can be debugged just like a core.

Semihosting

The MIPS Toolkit and Codescape Debugger provide support for semihosting functions from a target via a built-in API in the toolkit. The debugger allows you to set a root directory for semi-hosting operations so that programs running on a target can use relative address pathing for file operations. No additional libraries or function calls are required. Semihosting operations supported include file operations such as fopen, fwrite, fread and fclose and outputs such as stdout, stderr and printf.

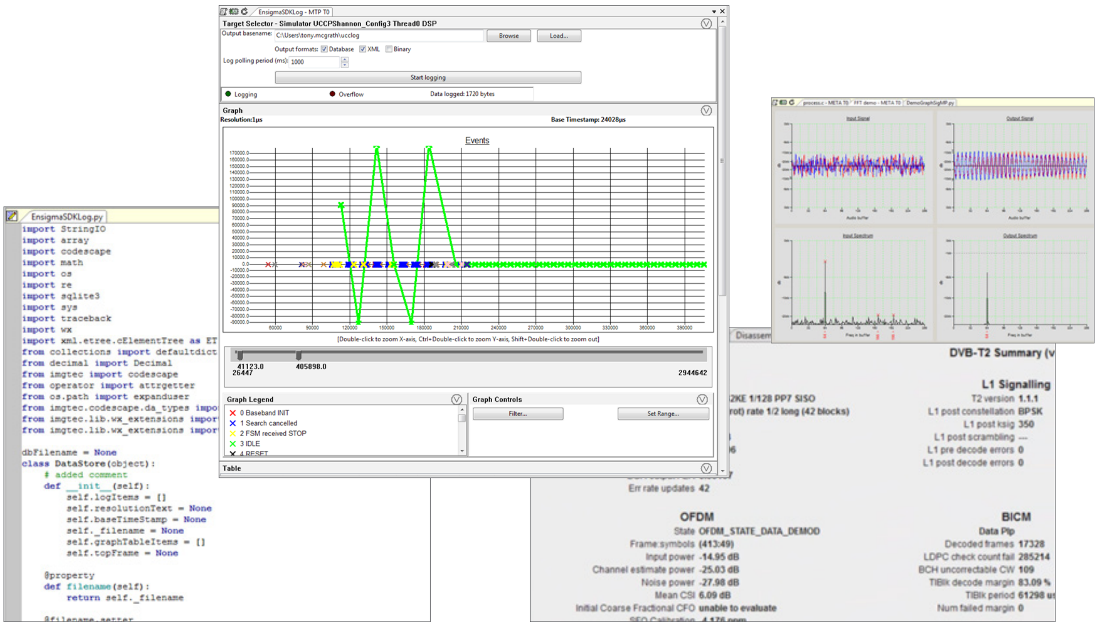
Make Manager

Make Manager provides a quick and simple way to call 'make' and see your build log without exiting the debugger. Multiple configurations for 'make' can be specified with individually-specified parameters for each configuration.

Scripting and Customization

Imagination supplies an advanced, Python-based, scripting interface and command-line console, Codescape Console, that enables direct access to Imagination's debug probes from your host PC without using the Codescape Debugger user interface. This provides an ultra-low-level, non-intrusive, scripting layer that is ideal for target bring-up and allows you to perform such tasks as read/write memory and registers, or manually control JTAG signals, with very predictable impact on the target.

In addition to these external scripting environments, Codescape Debugger has its own internal, fully-configurable, script region that supports standard wxPython to enable advanced graphical scripting for data visualisation, input/output device emulation, and the creation of bespoke debug regions.



```
import StringIO
import array
import codescape
import math
import os
import re
import sqlite3
import sys
import traceback
import wx
import xml.etree.cElementTree as ET
from collections import defaultdict
from decimal import Decimal
from imgtec import codescape
from operator import attrgetter
from os.path import expanduser
from imgtec.codescape.da_types impo
from imgtec.lib.wx_extensions impor
from imgtec.lib.wx_extensions impor

dbFilename = None
class DataStore(object):
    # added comment
    def __init__(self):
        self.logItems = []
        self.resolutionText = None
        self.baseTimeStamp = None
        self._filename = None
        self.graphTableItems = []
        self.topFrame = None

    @property
    def filename(self):
        return self._filename

    @filename.setter
```

Target Selector - Simulator UCCPShannon_Config3 Thread0 DSP
Output basename: C:\Users\tony.mcgrath\ucclog
Output formats: ☑ Database ☑ XML ☐ Binary
Log polling period (ms): 1000
● Logging ● Overflow Data logged: 1720 bytes
Resolution:1µs Base Timestamp: 24028µs
[Double-click to zoom X-axis, Ctrl+Double-click to zoom Y-axis, Shift+Double-click to zoom out]
41123.0 26447 405898.0 2944642
✗ 0 Baseband INIT
✗ 1 Search cancelled
✗ 2 FSM received STOP
✗ 3 IDLE
✗ 4 RESET

DVB-T2 Summary (v
2KE 1/128 PP7 SISO
rot) rate 1/2 long (42 blocks)
Err rate updates 42

L1 Signalling
T2 version 1.1.1
L1 post constellation BPSK
L1 post ksig 350
L1 post scrambling ---
L1 pre decode errors 0
L1 post decode errors 0

OFDM
State OFDM_STATE_DATA_DEMOD
Frame:symbols (413:49)
Input power -14.95 dB
Channel estimate power -25.03 dB
Noise power -27.98 dB
Mean CSI 6.09 dB
Initial Coarse Fractional CFO unable to evaluate
SFO Calibration -4.176 ppm

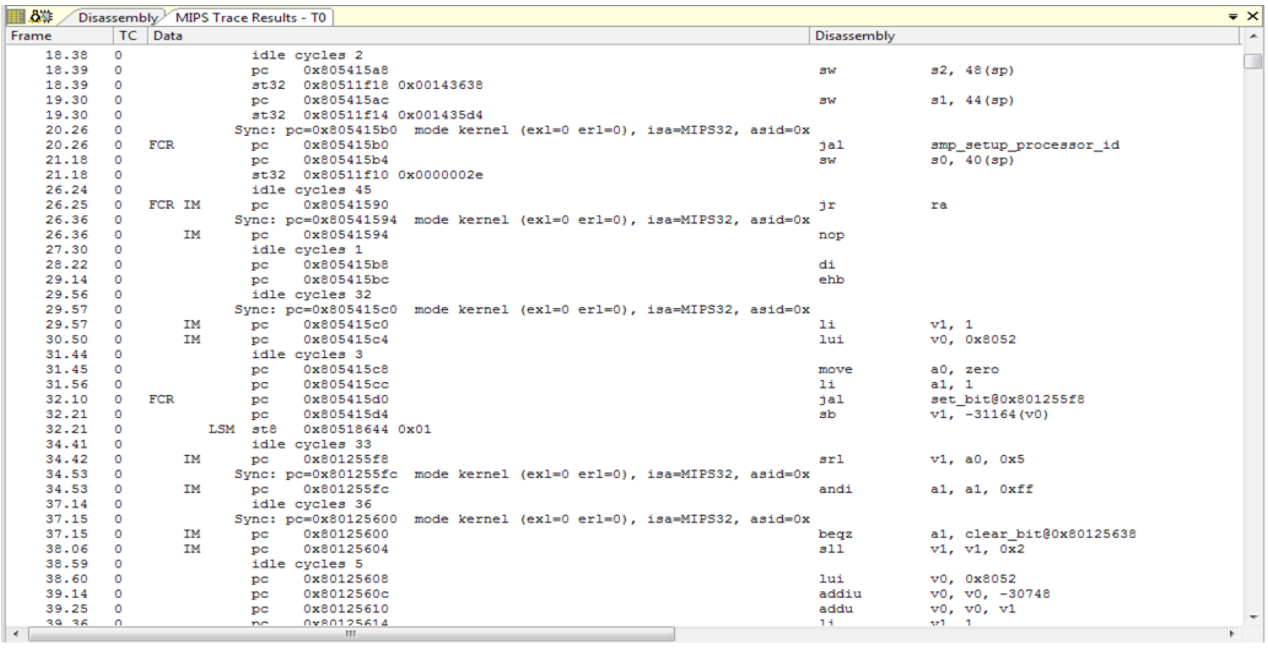
BICM
Data Plp
Decoded frames 17328
LDPC check count fail 285214
BCH uncorrectable CW 109
TIBlk decode margin 83.09 %
TIBlk period 61298 us
Num failed margin 0

Profiling and Tracing

Statistical profiling data, with variable sample rate, can be displayed alongside the standard Source and Disassembly regions.

Trace Results Region

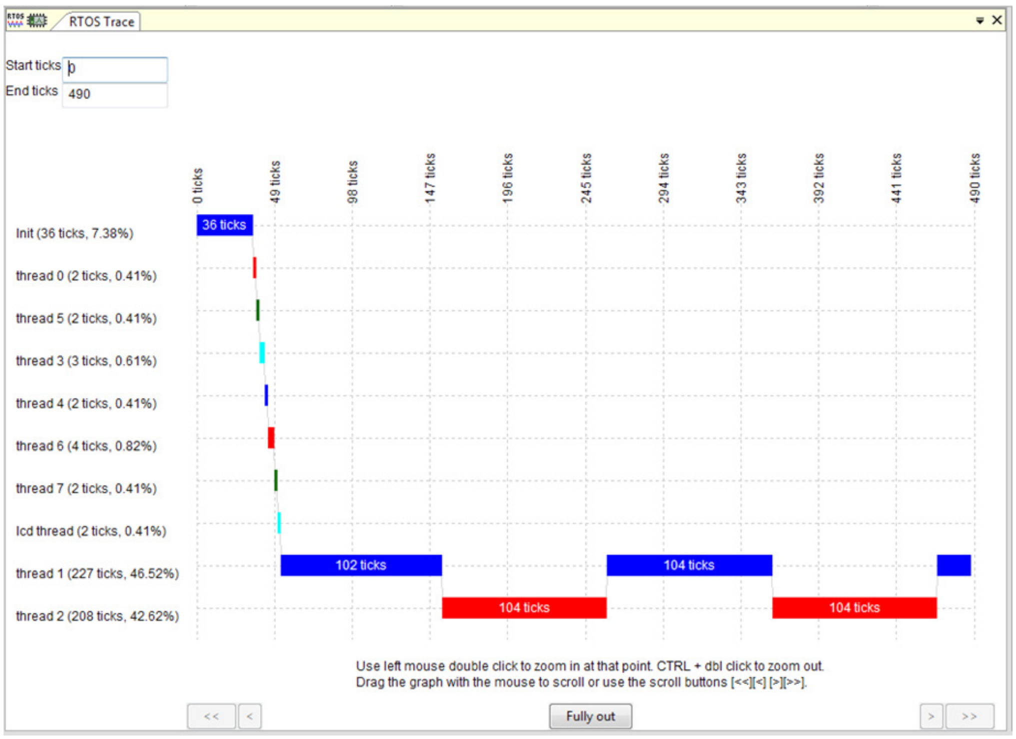
The Trace Results Region reads trace data saved by MIPS' PDTrace on-chip tracing hardware. The data captured is configurable and includes processor-specific information captured from each pipeline and from non-processor-specific blocks such as the Coherence Manager block in a Coherent Multi-Processor system.



```
Frame     TC  Data                                                                  Disassembly
  18.38   0              idle cycles 2
  18.39   0              pc     0x805415a8                                          sw      s2, 48(sp)
  18.39   0              st32   0x80511f18 0x00143638
  19.30   0              pc     0x805415ac                                          sw      s1, 44(sp)
  19.30   0              st32   0x80511f14 0x001435d4
  20.26   0            Sync: pc=0x805415b0  mode kernel (exl=0 erl=0), isa=MIPS32, asid=0x
  20.26   0   FCR        pc     0x805415b0                                          jal     smp_setup_processor_id
  21.18   0              pc     0x805415b4                                          sw      s0, 40(sp)
  21.18   0              st32   0x80511f10 0x0000002e
  26.24   0              idle cycles 45
  26.25   0   FCR IM     pc     0x80541590                                          jr      ra
  26.36   0            Sync: pc=0x80541594  mode kernel (exl=0 erl=0), isa=MIPS32, asid=0x
  26.36   0       IM     pc     0x80541594                                          nop
  27.30   0              idle cycles 1
  28.22   0              pc     0x805415b8                                          di
  29.14   0              pc     0x805415bc                                          ehb
  29.56   0              idle cycles 32
  29.57   0            Sync: pc=0x805415c0  mode kernel (exl=0 erl=0), isa=MIPS32, asid=0x
  29.57   0       IM     pc     0x805415c0                                          li      v1, 1
  30.50   0       IM     pc     0x805415c4                                          lui     v0, 0x8052
  31.44   0              idle cycles 3
  31.45   0              pc     0x805415c8                                          move    a0, zero
  31.56   0              pc     0x805415cc                                          li      a1, 1
  32.10   0   FCR        pc     0x805415d0                                          jal     set_bit@0x801255f8
  32.21   0              pc     0x805415d4                                          sb      v1, -31164(v0)
  32.21   0          LSM st8    0x80518644 0x01
  34.41   0              idle cycles 33
  34.42   0       IM     pc     0x801255f8                                          srl     v1, a0, 0x5
  34.53   0            Sync: pc=0x801255fc  mode kernel (exl=0 erl=0), isa=MIPS32, asid=0x
  34.53   0       IM     pc     0x801255fc                                          andi    a1, a1, 0xff
  37.14   0              idle cycles 36
  37.15   0            Sync: pc=0x80125600  mode kernel (exl=0 erl=0), isa=MIPS32, asid=0x
  37.15   0       IM     pc     0x80125600                                          beqz    a1, clear_bit@0x80125638
  38.06   0       IM     pc     0x80125604                                          sll     v1, v1, 0x2
  38.59   0              idle cycles 5
  38.60   0              pc     0x80125608                                          lui     v0, 0x8052
  39.14   0              pc     0x8012560c                                          addiu   v0, v0, -30748
  39.25   0              pc     0x80125610                                          addu    v0, v0, v1
  39.36   0              pc     0x80125614                                          li      v1, 1
```

RTOS Trace Region

Codescape Debugger supports debug and trace of several common realtime operating systems, including ThreadX, FreeRTOS and MEOS. The RTOS Watch region reads, formats, and displays system variables in use by real-time operating systems, and the RTOS Trace region reads, processes, and displays thread execution order and duration.


RTOS Trace
0 ticks
49 ticks
98 ticks
147 ticks
196 ticks
245 ticks
294 ticks
343 ticks
392 ticks
441 ticks
490 ticks
Init (36 ticks, 7.38%)
thread 0 (2 ticks, 0.41%)
thread 5 (2 ticks, 0.41%)
thread 3 (3 ticks, 0.61%)
thread 4 (2 ticks, 0.41%)
thread 6 (4 ticks, 0.82%)
thread 7 (2 ticks, 0.41%)
lcd thread (2 ticks, 0.41%)
thread 1 (227 ticks, 46.52%)
thread 2 (208 ticks, 42.62%)
36 ticks
102 ticks
104 ticks
104 ticks
104 ticks
<<
<
Fully out
>
>>

Start ticks 0
End ticks 490

Use left mouse double click to zoom in at that point. CTRL + dbl click to zoom out.
Drag the graph with the mouse to scroll or use the scroll buttons [<<][<] [>][>>].

Codescape Console

As part of the debug tools suite, Imagination supplies a stand-alone, command-line, interface called Codescape Console that enables direct access to Imagination's debug probes from your host PC without using the Codescape Debugger user interface. It is supplied with comprehensive command reference documentation and examples to enable development of functionally-complex scripts for non-intrusive low-level debug and testing.

It is an extension of the cross-platform Python interpreter in which extensive task-specific tab completion and command history has been added. Commands are automatically available in the global namespace and are optimised to be easy to type. No knowledge of Python is required, but familiarity will help you to become more productive and maximise the potential of Codescape Console. Customers familiar with MIPS System Navigator Console (NavCon) will find that Codescape Console is very similar with equivalent commands.

Low-level Debug

Codescape Console is especially useful for target bring-up and low-level debug because only commands that are submitted by the user are performed. This allows you to execute such tasks as read/write memory and registers, or manually controlling IR and DR JTAG lines with very predictable impact on the target where using Codescape Debugger would result in a more intrusive target interaction. To support low-level bring-up, Imagination supplies comprehensive documentation and example scripts for validation of EJTAG implementation, target detect, and low-level troubleshooting.

Debugging Extensions

Codescape Console provides high levels of logging for contents of registers, memory, disassembly and caches/TLB. For example, registers are displayed with field information, memory read results are displayed with address and ascii representation, and where no better display is available integer expressions are displayed in hexadecimal.

The asm assembler supplied with the Codescape MIPS toolchain can be invoked from Codescape Console and assembled instruction sequences displayed in the console. ELF program files with symbols can be loaded onto your target directly from Codescape Console. Target definition commands enable target configuration and TAP layout to be supplied to the debug probe to allow for a minimally-intrusive connection and to accommodate non-standard targets.

Features

Python-based command line console

Use stand-alone or in conjunction with Codescape Debugger

High levels of logging

In-line assembler

Elf load with symbols

Tab complete plus history and command/parameter help

Full command reference documentation with examples

Benefits

Non-intrusive debug

Predictable impact on target

Cross-platform

Customizable and extensible

Applications

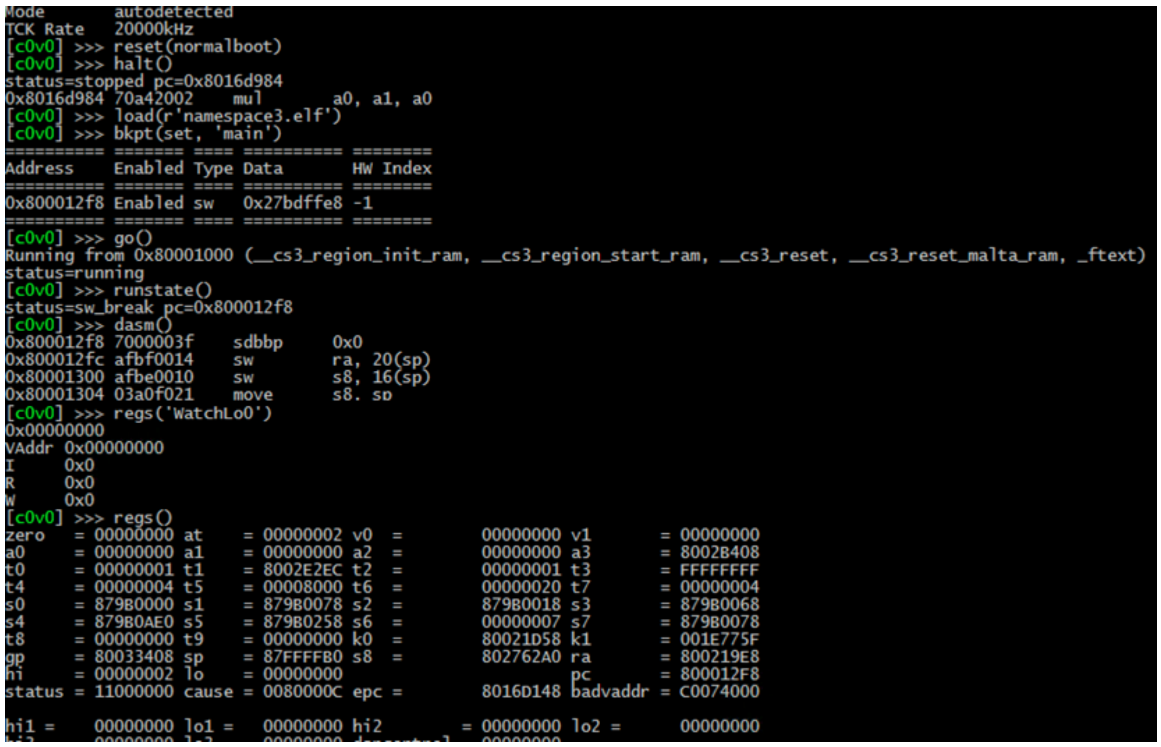
Silicon bring-up

Debugging and working around core and cache bugs

TAP configuration and validation

Test automation

Debugging unstable or non-standard targets



```
Mode       autodetected
TCK Rate   20000kHz
[c0v0] >>> reset(normalboot)
[c0v0] >>> halt()
status=stopped pc=0x8016d984
0x8016d984 70a42002    mul        a0, a1, a0
[c0v0] >>> load(r'namespace3.elf')
[c0v0] >>> bkpt(set, 'main')
========== ======= ==== ========== ========
Address    Enabled Type Data       HW Index
========== ======= ==== ========== ========
0x800012f8 Enabled sw   0x27bdffe8 -1
========== ======= ==== ========== ========
[c0v0] >>> go()
Running from 0x80001000 (__cs3_region_init_ram, __cs3_region_start_ram, __cs3_reset, __cs3_reset_malta_ram, _ftext)
status=running
[c0v0] >>> runstate()
status=sw_break pc=0x800012f8
[c0v0] >>> dasm()
0x800012f8 7000003f    sdbbp      0x0
0x800012fc afbf0014    sw         ra, 20(sp)
0x80001300 afbe0010    sw         s8, 16(sp)
0x80001304 03a0f021    move       s8, sp
[c0v0] >>> regs('WatchLo0')
0x00000000
VAddr 0x00000000
I     0x0
R     0x0
W     0x0
[c0v0] >>> regs()
zero   = 00000000 at    = 00000002 v0  =       00000000 v1       = 00000000
a0     = 00000000 a1    = 00000000 a2  =       00000000 a3       = 8002B408
t0     = 00000001 t1    = 8002E2EC t2  =       00000001 t3       = FFFFFFFF
t4     = 00000004 t5    = 00008000 t6  =       00000020 t7       = 00000004
s0     = 879B0000 s1    = 879B0078 s2  =       879B0018 s3       = 879B0068
s4     = 879B0AE0 s5    = 879B0258 s6  =       00000007 s7       = 879B0078
t8     = 00000000 t9    = 00000000 k0  =       80021D58 k1       = 001E775F
gp     = 80033408 sp    = 87FFFFB0 s8  =       802762A0 ra       = 800219E8
hi     = 00000002 lo    = 00000000                      pc       = 800012F8
status = 11000000 cause = 0080000C epc =       8016D148 badvaddr = C0074000

hi1 =    00000000 lo1 =   00000000 hi2        = 00000000 lo2 =      00000000
```

Writing Your Own Commands

Because Codescape Console is written in Python, new commands can be easily written to automate common tasks. Assistance for writing and documenting extension commands is provided in the Codescape Console documentation.

Integrating with Codescape Debugger

Just as with any other high-level debugger, Codescape Debugger makes many demands on a connected target and problems can be encountered such as lock-up on illegal memory accesses or sensitivity to memory accesses performed shortly after reset. By using Codescape Console to initialise parts of the system you can enable Codescape Debugger to work more reliably with sensitive targets.

Connect scripts can be invoked from Codescape Debugger so a console script runs to perform connection tasks before Codescape Debugger connects to the target. Examples are provided for common tasks such as enabling the JTAG port on connection, initialising SPRAM, or dynamically selecting a hardware support package.

Once connection has been established, scripts that run with an active connection to Codescape Debugger can be invoked to run on various events such as target reset or breakpoints. Scripts can also be run from Codescape Debugger's Run Script pane following user input or from the Script Region where graphical scripts can receive target events.

Where to buy

Information and download details for the Codescape MIPS SDK, debug probes, customer support and user forums can be found at **community.imgtec.com/developers/mips**

Codescape Debugger can run natively on all variants of Microsoft Windows from XP onwards, Linux 2.6 kernels and Mac OS X. Floating licenses for multiple users or single-user node-locked licenses are available.

A range of approved development boards and vendor details is available at **store.imgtec.com/product-category/mips**

UK	t: +44 1923 260511	enquiries@imgtec.com
USA	t: +1 408 530 5000	www.imgtec.com