LAUTERBACH
DEVELOPMENT TOOLS

Integrated Run & Stop Mode Debugging for Embedded Linux

Until now it has been common practice to use two different debuggers in the development of embedded Linux applications. To start up the target hardware a JTAG debugger is normally used. As soon as the essential components of embedded Linux are running on the target the process debugging continues with GDB.

At Embedded Systems Conference 2007, Lauterbach will be presenting an integrated Linux debugger that combines these two debugging concepts. Development times for embedded Linux applications can then be reduced considerably since this allows the strengths of both methods to be used in a uniform user interface.

The following shows the concepts of the integrated Linux debugger using the ARM architecture as an example.

Stop Mode Debugging

A JTAG debugger works with so-called *Stop Mode Debugging*: The processor and thus the whole system are stopped at a breakpoint. Information about the state of the processor or the target hardware can now be read out via the JTAG interface (see Fig. 1).

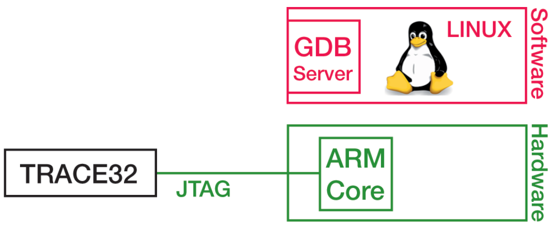


Fig. 1: In Stop Mode Debugging, the processor and thus the whole system is stopped via the JTAG interface.

Some advantages of *Stop Mode Debugging*:

- The only requirement for *Stop Mode Debugging* is a functioning JTAG interface. This enables debugging from the reset vector.

Integrated Run & Stop Mode Debugging

Embedded Linux with GDB as debug agent

ARM
Run mode debugging via DCC available

ARM
Run mode debugging via Ethernet planned for Q2/2007

PowerPC
Run mode debugging via Ethernet planned for Q2/2007

Symbian OS with TRK as debug agent

ARM
Run mode debugging via DCC available

- Debugging of the kernel and beyond process boundaries is possible using a debugger that offers both Linux and MMU support.
- If the software ceases to react, the processor can be stopped to find out the point in the code where the processor hung up mistakenly.
- If the processor is stopped neither the kernel or any other process can cause any disturbing effects.

However, *Stop Mode Debugging* has a serious disadvantage:

As soon as the processor stops, all communication interfaces are also stopped. The usual result of this is that external devices that communicate with the Linux application via Ethernet, Bluetooth or CAN will cut the connection as the application is no longer responding. Therefore, stopping at a breakpoint can change the state of the overall system. Trying to continue debugging may then be meaningless.

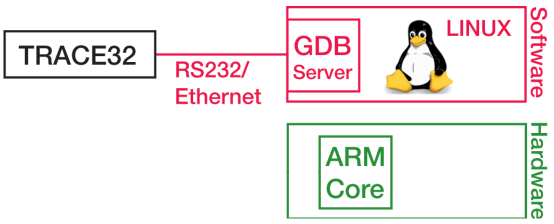


Fig. 2: In Run Mode Debugging, the selected process is stopped while the overall system continues to run.

Run Mode Debugging

GDB works in so-called *Run Mode Debugging*: At a breakpoint, only the selected process is stopped, the kernel and all other processes continue to run.

However, GDB is purely a software debugger. The following is required for debugging:

- GDB server running as a Linux process on the target hardware
- debugger software – here TRACE32 – on the host (see Fig. 2)

TRACE32 communicates with the GDB server via an RS232 or Ethernet interface to query information about the currently stopped process.

Run Mode Debugging is always ideal:

- if startup of the target hardware has been completed.
- if the GDB server can always be activated – that is, the communication interface is running properly and the processor has not mistakenly hung up at a code point.

Quite clearly, both debugging methods have great strengths and weaknesses. For this reason, Lauterbach now offers a debugger that combines the two methods in such a way that their strengths are fully put to use, while their weaknesses disappear completely.

Integrated Run & Stop Mode Debugging

The TRACE32 debugger with *Integrated Run & Stop Mode Debugging* for embedded Linux works as follows (see Fig. 3):

1. The TRACE32 debugger is first started via the JTAG interface in *Stop Mode Debugging*. In a first step, the target hardware and the *Run Mode Debugging* (GDB) must be configured.
2. If the startup of the target hardware is the focus, *Stop Mode Debugging* (JTAG) is used.

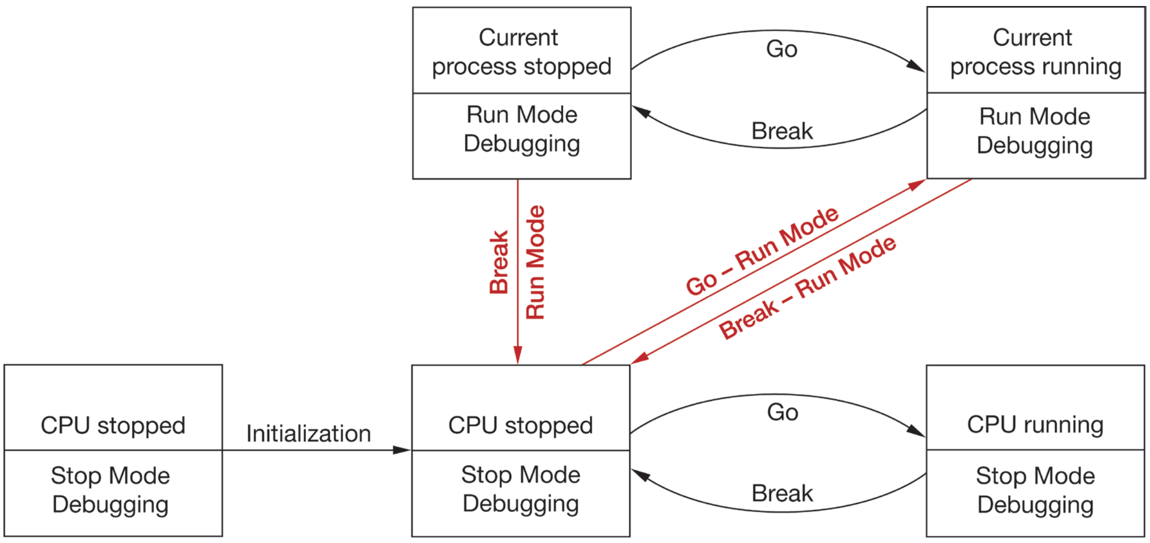


Fig. 3: To test, Run Mode or Stop Mode Debugging may be used depending on requirements.

3. After the hardware startup, TRACE32 can switch over to *Run Mode Debugging* (GDB) for application debugging. Individual processes can now be tested while the complete system is running.
4. If the connection to the GDB server is cut during *Run Mode Debugging*, you can switch back to *Stop Mode Debugging* at any time to find the cause of the problem.

Simultaneously with the implementation of *Integrated Run & Stop Mode Debugging*, the following functions have been added to *Run Mode Debugging*:

- For the ARM architecture, the *Debug Communications Channel* (DCC) can be used as the communication interface in addition to ethernet and RS232. In this way, *Run & Stop Mode Debugging* can function with JTAG as the only interface (headless target).
- If required, it is possible to have simultaneous debugging of two or more processes.

DCC as communication interface

The JTAG interface for the ARM architecture includes a so-called *Debug Communications Channel* (DCC). In principle, information exchange via DCC should be possible between

- debugger software on the host (TRACE32)
- any application on the target system – here, with the GDB server

while the application is running on the processor.

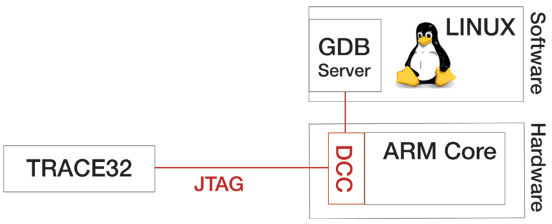


Fig. 4: Instead of an external communication interface, you can use the DCC function of the JTAG interface as a communication channel to the GDB server.

Therefore, if TRACE32 uses the DCC function of the JTAG interface to query the GDB server for information about the currently stopped process, no external communication interface is needed for *Run Mode Debugging* (see Fig. 4).

Simultaneous debugging of several processes

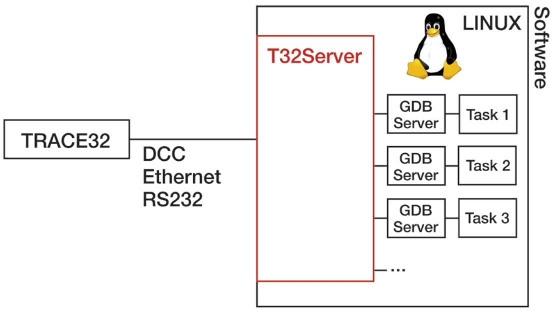


Fig. 5: Using T32Server, a separate GDB server can be assigned to each process, enabling the simultaneous debugging of processes.

In some cases, it is necessary to debug several processes simultaneously. To be able to offer this feature, Lauterbach now provides the *T32Server* for *Run Mode Debugging*.

After *T32Server* has been started as a Linux process from the terminal window, the following is possible using TRACE32 commands:

- Starting processes (TASK.RUN)
- Attaching to running processes (TASK.SELect)
- Ending processes (TASK.KILL)

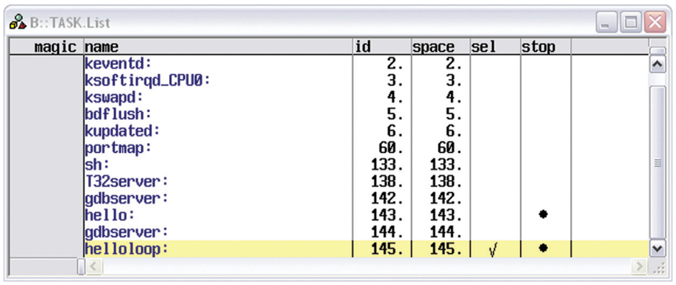
When a process is started/attached, a separate GDB server is assigned to each process by *T32Server* (see Fig. 5).
Fig. 6 on the next page shows TRACE32 *Run Mode Debugging* using the example of a TASK.List window.

Summary

Integrated Run & Stop Mode Debugging offers an optimum basis for the efficient development of embedded Linux applications enabling the user to find complex hardware and software errors quickly using a single development tool and a universal user interface. In addition no modifications are needed to either the application or Linux.

Integrated Run & Stop Mode Debugging has been supported for the ARM architecture since November 2006 and can be used at no extra cost with every TRACE32 JTAG debugger for an ARM processor.

An implementation for PowerPC architectures is planned for May 2007.



magic	name	id	space	sel	stop
	keventd:	2.	2.		
	ksoftirqd_CPU0:	3.	3.		
	kswapd:	4.	4.		
	bdflush:	5.	5.		
	kupdated:	6.	6.		
	portmap:	60.	60.		
	sh:	133.	133.		
	T32server:	138.	138.		
	gdbserver:	142.	142.		
	hello:	143.	143.		●
	gdbserver:	144.	144.		
	helloloop:	145.	145.	√	●

Fig. 6: The hello and helloloop processes are stopped. The helloloop process is the process currently selected for debugging.