RTEMS sample application suite

Summary

RTEMS comes with a set of small applications designed to test various features of a Board Support Package, which implements basic hardware "hook" functions needed by RTEMS. The DAT group implements two generic BSPs, one Virtex 4 FPGAs and one for Virtex 5 FPGAs. These BSPs in their turn have "application hooks" which allow for adjustment to application specific hardware variations, e.g., number of ethernet ports. For more information on the structure of BSPs look here.

The trouble with the extra layer of indirection is that the sample applications that come with RTEMS don't know about it. Most of the time this isn't a problem as the sample applications tend to deal with only the most basic features of RTEMS.

One seemingly basic thing that DAT BSPs fail to provide is a console allowing real-time interaction. Instead they offer a console driver that just writes output to a circular buffer and doesn't allow for input. Only when networking is up and the RTEMS shell is run do we have truly interactive sessions.

Setting up the RTEMS source tree

In general you should follow the instructions listed here, although you can take a shortcut and copy the right source tree from `/afs/slac/g/cci/package/rtems`. That will get you all our production BSPs and have all our patches installed.

Virtex4 BSP

This BSP is for Virtex 4 FPGAs with PowerPC 405 processors. The programmer's interface to the protocol plugins (I/O engines) is through Plugin Interface Code (PIC) firmware blocks. Two-way communication through a plugin requires four different types of blocks though there may be sharing of blocks amongst plugins. Each PIC block can raise external non-critical exceptions for four events that may occur during normal operation. (These exceptions are generally referred to as interrupts in code). There is a single control bit for each block which controls the enabling of these exceptions. Each block can also raise the critical external exception for various fault conditions.

RTEMS 4.10.2

The sample applications are found in `${RTEMS_ROOT}/testsuites/samples`.

PIC block resets

When we first tried running them on a Virtex 4 DPM, none would work, not even "hello". The Init task of the application was never called. The problem turned out to be a long-standing firmware bug which left PIC-block external interrupts enabled after a system reset. These interrupts are persistent and take precedence over most internal exceptions. The default exception handlers do nothing and so the system was snowed under by constant external exceptions as soon as RTEMS turned on multitasking and exception handling.

The right fix is to have the firmware bring up all PIC blocks with their "exception on event" bits cleared. For some reason that never happened; instead all our applications cleared these bits in the application pre-tasking hook. Comments in the BSP code complain of RTEMS enabling the exceptions "too soon". The sample apps are of course unaware of this hook. Also, one has to know how many of each kind of PIC block are on the board which varies from application to application; there is no "grand disable" bit one can set to silence them all. For the present we modify `bsp_start.c` so that `bsp_start()` calls the following function:

```
/* "se taire" is French for "to shut one's self up". The function will
   disable all interrupts from PPI's by clearing the appropriate mask
   bits in each PPI CSR on a COB mezzanine board with a PPC405.

   There are only two protocol plugins on a Gen 1 DPM (PPC405):
   one ethernet and one PGP. Each has one of each kind of PIC
   block: PIB, ECB, FLB and PIB.
*/

#define wdcr(dcrNum, value) \
  asm volatile("mtdcr %0,%1"::"i"(dcrNum), "r"(value))

#define MODIFY_EVENT_ENABLE 0x00080000

#define CLEAR_EVENT_ENABLE  0x00000000

void se_taire() {
  static const unsigned DISABLE_EVENTS = MODIFY_EVENT_ENABLE | CLEAR_EVENT_ENABLE;
  /* PEBs. */
  wdcr(0x300, DISABLE_EVENTS);
  wdcr(0x304, DISABLE_EVENTS);
  /*  ECBs */
  wdcr(0x340, DISABLE_EVENTS);
  wdcr(0x344, DISABLE_EVENTS);
  /* FLBs */
  wdcr(0x380, DISABLE_EVENTS);
  wdcr(0x384, DISABLE_EVENTS);
  /* PIBs */
  wdcr(0x3c0, DISABLE_EVENTS);
  wdcr(0x3c4, DISABLE_EVENTS);
}
```

Multitasking vs. console driver

Another problem, albeit a minor one, is that the console driver is neither re-entrant nor serializing. Output from multiple tasks gets scrambled as one task may overwrite what another has just written. Usually, though, it's possible to see whether the basic test failed.

Fixing this would be a bit tricky as the same circular buffer is also used for printk() before RTEMS is fully initialized. One would have to change the mechanism to re-entrancy or serialization in the pre-tasking hook.

Scorecard

Out of nine applicable tests the system passes eight; only nsecs fails.

Application	Pass?	Comments
hello	✅	
base_mp	n/a	Requires a true multiprocessor system with shared memory.
base_sp	✅	Output mangled by our console driver.
capture	n/a	Requires an interactive console.
cdtest	✅	The sample output is obsolete. The application now tells whether construction is local or global. No iostream test is performed but there is now an exception test.
fileio	n/a	Requires an interactive console.
iostream	✅	Like hello but uses C++ iostreams instead of printf().
loopback	✅	Test of network stack in software loopback mode.
minimum	n/a	We'd need a custom idle task for this.
nsecs	❌	It seems we have no working nanosecond clock.
paranoia	✅	Rating of arithmetic: Excellent.
pppd	n/a	Requires a working serial port.
ticker	✅	PIT interrupts evidently do occur since task TA1 gets woken up at 5-second intervals. The starting of the other two tasks was commented out, not by me.

unlimited	✓	Tests the feature allowing one to allocate as many RTEMS objects as working-space memory allows.

Virtex 5 BSP