**Should we spend time porting older controller code to DEBROS?**

Original Version: 2012-08-21, by Mark Davis,

Last updated: 2012-08-22, by Mark Davis,

**Ongoing issues with controllers running old Dynamic-C code:**

* Use of the older auto-discovery/configuration process which is ***slow and unreliable***, *particularly when replacing an existing controller with a spare*. It also generates ***excessive amounts of ARP traffic****, which has been known to cause problems for other devices* (such as VME crate IOCs hanging prior to being moved to a separate sub-net).
* Almost non-existent diagnostic capabilities. This means we often don’t have ANY useful information to work with when problems do occur. The result: ***Problems with the older controllers rarely get resolved***.
* Saving changes to a controller’s configuration literally***causes the application code to stop*** while writing the new configuration to flash. In the worst known case, the effect lasts for *more than 10 seconds*!
* Little or no program memory left to make changes. There are several known issues and unresolved ones that ***cannot BE resolved***in part because we cannot make any significant changes to the code.
* ***Memory corruption goes undetected*:** Controllers in the RF/ECR area often suffer from memory corruption whenever the cyclotron is running (root cause never determined). The Dynamic-C code has no means at all to detect this or its effect (one *possible* explanation for some of the unresolved problems that still occur).
* Updates can only be done while the controllers are offline

**Immediate advantages gained by porting a controller application to DEBROS:**

* Use of the newer auto-discovery/configuration process which **is *fast, reliable****,* and ***does* *not generate excess ARP traffic.*** Each application ported to DEBROS means a reduction in this troublesome ARP traffic.
* Numerous diagnostic capabilities and reliability improvements including:
  + An event log that is ***continuously backed-up and withstands reboots*** (and power cycling with a battery). This means a lack of diagnostic information is unusual.
  + Automatically ***archives all saved changes to configuration data***
  + On-demand near real-time plotting of run-time values (a very powerful diagnostic and debugging tool, providing far more information than is available via EPICS).
  + Support for ***hardware enforced protection mechanisms*** that helps detect bugs sooner rather than later. Used to trigger a dump of the OS and CPU states that includes a stack trace, making it possible to track down very subtle, hard-to-find problems that cannot be easily reproduced.
  + ***Detection of corruption*** in read-only memory segments before it results in weird, unexplainable behavior. Coupled with a “soft reset” approach that restarts the code without so much as a glitch on any of the outputs, controllers can almost always ***recover from this without disrupting normal operations***.
* Disruptions caused by writing to flash are ***reduced from seconds to milliseconds***, so operations such as saving configuration changes don’t result in a temporarily unresponsive controller.
* Updates are fast and simple and are done while the controller is running the previous version (which is still available after the update). A 30 second reboot switches to the new version. Where supported, the “soft reset” mode makes a “live update” possible (i.e. no disruption to operations).
* Access to more data and via more tools via the ***EPICS CA protocol***. The DEBROS CA service was completed and extended to handle values that the Modbus protocol cannot (such as strings and waveform data).
* ***The ability to add to or change the code when needed to fix problems:*** DEBROS code is much smaller and more efficient, which means we get a LOT more functionality and STILL have space for additional code and/or data.

**Longer term advantages of porting a controller application to DEBROS:**

The longer term advantages of porting the older code to DEBROS can be summarized in one word: Standardization.

The vendor’s Dynamic-C approach is a completely proprietary one. The tools and methodology are very unconventional and don’t allow the user to control how memory is used. The only support for multi-tasking is a proprietary form of cooperative multitasking, which results in source code that is very large and complex.

In comparison, the compiler used to write DEBROS is ANSI-compatible (nearly 100% ANSI-***compliant*** as far as I can tell).

The tools produce conventional re-locatable libraries and object files, and allow the user ***complete*** control over how memory is used. Armed with these tools and an excellent book on the Linux kernel, I was able to create DEBROS: A fully pre-emptive, priority-based, soft Real-Time OS based on Linux/Unix.

The system calls in DEBROS are the same as the ones in Linux and Unix. They take the same parameters and produce the same effects. The result is that ***the majority of the code can be used with little or no changes on DEBROS, Linux, and Unix*** (as Vasu proved by doing development and testing of the early CA server for DEBROS on a Linux machine).

The one significant difference across platforms is, of course, the hardware-specific portions. But this is fairly well encapsulated in the DEBROS model, so duplicating the DEBROS hardware abstraction layer for a given platform covers the majority of the effort needed to move any of the existing DEBROS applications to newer, more powerful hardware (and it only has to be done once for a given platform).

**Summary:**

There are costs to the continued use of controllers running Dynamic-C code:

* The code base is effectively frozen. In at least one case, adding or even changing a single line of code causes it to exceed the available memory. So even if we know how to fix a problem, we can’t.
* The lack of any useful diagnostic capabilities means when problems do occur (and they do still occur), we have no means of diagnosing them.
* They generate excessive amounts of ARP traffic which causes problems for other devices on the network.

For each controller that is switched to running a DEBROS version of the code, we immediately gain the following:

* A reduction in the excessive ARP traffic
* Numerous diagnostic tools
* Much simpler maintenance and monitoring
* The ability to make changes or additions simply and to apply them quickly with little or no disruption
* Code that can moved to another hardware platform without rewriting most it

The DEBROS model is a conventional, industry standard one that is compatible with the one OS that is available on nearly every new hardware platform and most existing ones. This means it will be a long time before the code becomes obsolete or unusable, and greatly reduces the learning curve for anyone not already familiar with it.

**A breakdown of the existing Rabbit-based controllers:**

DEBROS (priority-based, preemptive OS): Dynamic-C (single-threaded, complex, unwieldy state machines):

Type aprox # Type aprox #

------------ ------- ------------ -------

BCM 45 \* BBServo 32

DVR 8 DCServo 3

DigPhaseCtlr 12 FastRFOff 1 (a trivially simple application to port)

ECROven 2 \* MWaveAmp 4

LLRFTurnOn 11 RFPhase 9

ModularPS 18 RFServo 57

PS8Rack 38 RevSwitch 34 (DEBROS version in progress)

TCPS 22 \*

\* Indicates controllers now running newer DEBROS version of older Dynamic-C application