Obstacles to Greater Reliability of RHIC

Al Marusic

2017 RHIC Retreat

State of FEC Software and Hardware

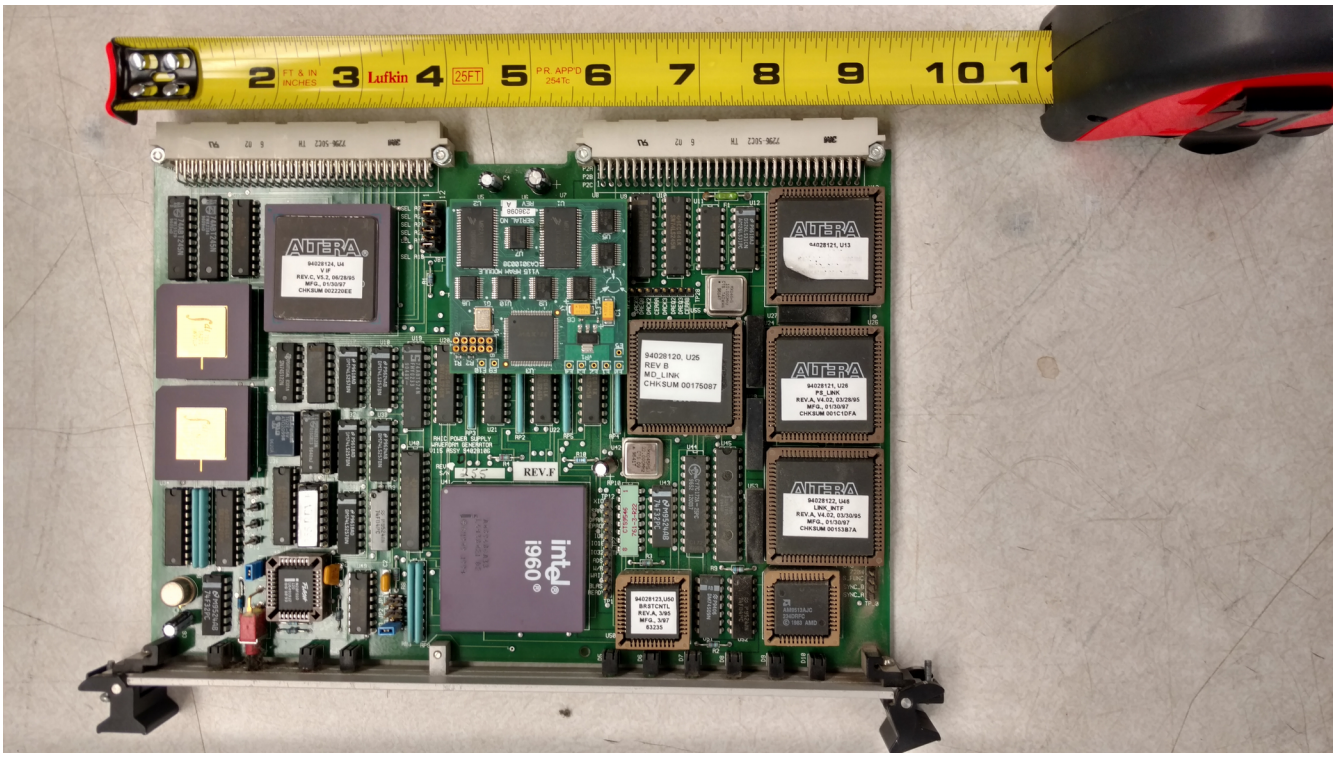
In CAD we use a number of different boards and FECs for control of the hardware. Most of those FECs and boards are VME based and most of them were designed and built ~20 years ago.

These are the types of FECs in use:

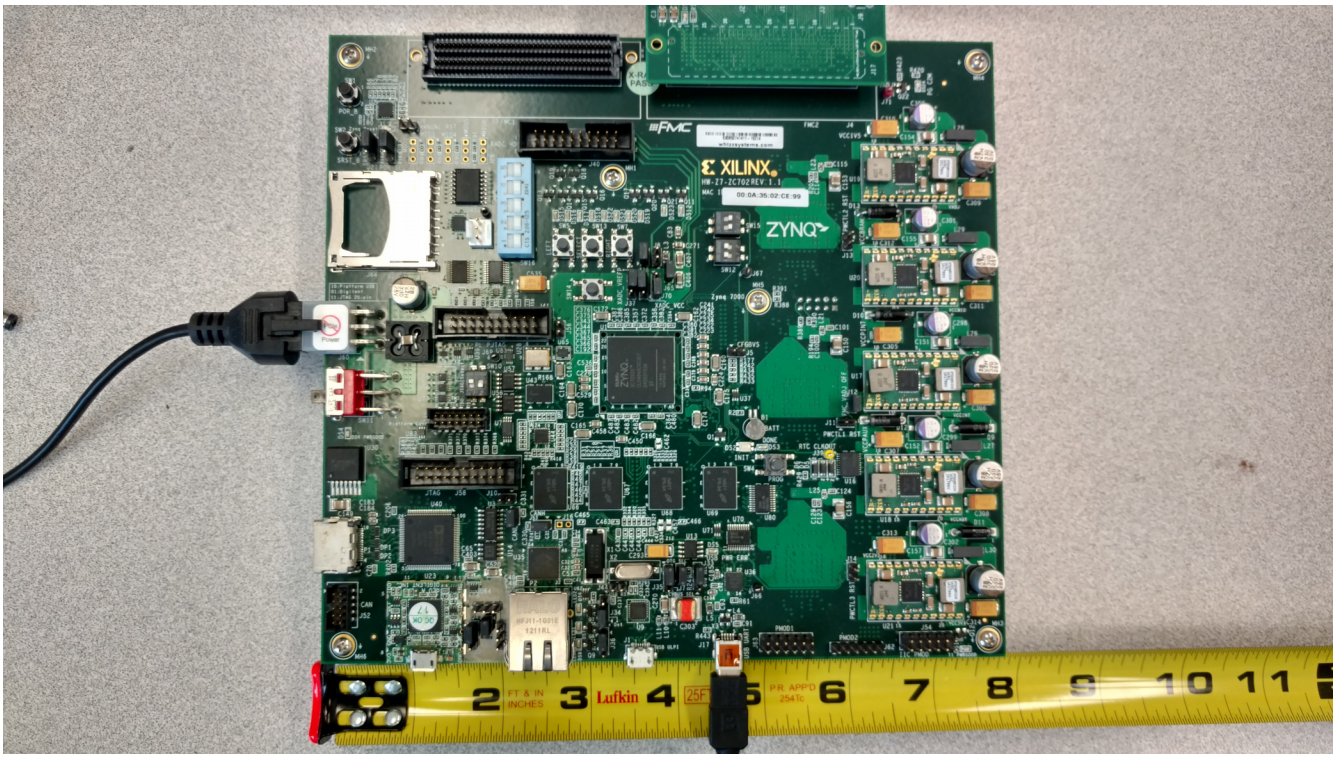
Type:	CPU frequency:	OS:	number of units:
MVME162	20 MHz	VxWorks 5.5.1	~35
MVME2100	200 MHz	VxWorks 5.5.1	~90
POWER3E	200 MHz	VxWorks 5.5.1	~60
MVME3100	667 MHz	VxWorks 5.5.1	~120
Xilinx Virtex-5	300 MHz	VxWorks 6.5	~80
Xilinx Zynq 70?0	677 MHz and above	Linux	?

Table 1: FEC types

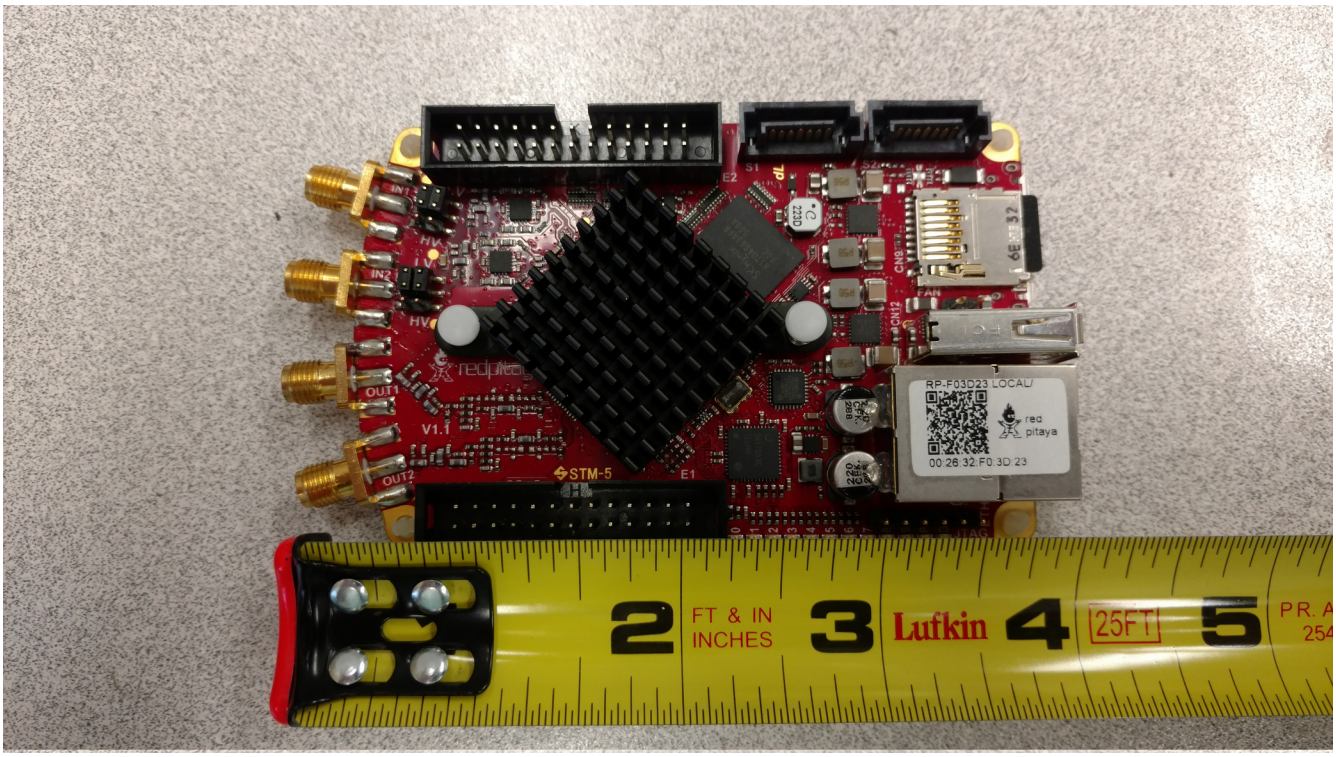
Only FECs with the type in bold are not obsolete. MVME162, MVME2100, POWER3E and MVME3100 FECs are VMEbus based. Xilinx Virtex-5 systems are used mostly by RF. The newest FECs in use are built around Xilinx Zynq 7000 microchips which combine FPGA with ARM CPU with two 32-bit cores.



Picture 1: WFG board



Picture 2: Zynq ZC702 Evaluation Board



Picture 3: Red Pitaya

These FECs and their associate boards differ in various ways, and the main difference is in their capabilities which stem from the difference in their age: most of our VME boards are 20 years old or use 20-year-old technologies. But not only our old boards are lacking in capabilities, also the tools used to develop those boards (which still have to be used to modify those boards) are lacking in capabilities. Most of our VME boards have defects which we live with, and which we rediscover from time to time. To fix those boards or change their functionality in any way, requires many orders of magnitude more effort than changing / designing new boards would require.

The table below lists some pros and cons of different types of FECs:

FECs which use VMEbus:	
CONs:	Max transfer rate achieved with our boards is ~7 MB/s, so there are FECs which can not read data from all the boards in the same chassis.
	VME chassis are rather large and not energy efficient.
PRO:	VME chassis provide enclosure and common power supply for many VME modules.
FECs which are running VxWorks operating system:	
CONs:	Compared to Linux, VxWorks lacks many features, such as memory protection, so it is very hard to develop code with no defects, i.e. it is hard to find bugs. Also VxWorks lacks libraries Linux systems have.
	It is very hard to find info about VxWorks.
	The code for FECs running VxWorks 5.5.1 has to be developed on Solaris / SPARC computers (because we lost Linux compiler for MV3100s).
	VxWorks 5.5.1 is no longer supported. The compiler used for compiling code for MVME162 / MVME2100 / POWER3E FECs is GCC 2.96 and for MVME3100 is GCC 3.3 (for comparison GCC version on Linux machines is 4.4.7, while the latest version of GCC is 7.1). That means that C and C++ language features which were added in the last 20 years can not be used.
PRO:	Most of the bugs in our and VxWorks 5.5.1 code were found and fixed (but some remain). However, we did not put as much effort into fixing bugs in FECs running VxWorks 6.5 and therefore those systems are less stable. We also did not apply all existing patches to VxWorks.
FECs which are Zynq based:	
CON:	Boards have to be provided with enclosure and power supply.
PRO:	Much easier to develop for, great tools, tools updated regularly, much better documentation, much greater performance.

Table 2: Pros and cons of FECs of various kinds

To increase reliability and ability of our hardware, and to increase productivity of our engineers, we have to start using new technologies and new tools. That implies that VxWorks should not be used by new types of FECs, and that we should try to avoid using VME based FECs and boards in new projects. To enable migration to modern hardware and software, Zynq based version of existing generic VME boards should be developed, for example, delay module / beam sync link / digital input combo board in smaller form factor should be developed.

We will also have to live for a long time with these 2 quite different systems, and therefore we need to have a way to transfer data between them. That would allow piecemeal replacement of existing systems. For example, that is the only way BBQ could be improved.

We should also exploit the opportunity provided by the need to redesign existing VME boards due to obsolescence of their parts, to design better boards and not just equivalent ones. Redesign of PS controls, for example, would allow us to remove FECs from alcoves.

Picture 4: 100 kHz PSI

State of Software on Consoles / Servers

There are numerous problems with our Linux software. We can divide those problems into two categories: the problems with the design of the programs, and the problems with the tools they are made with, and the environment in which they are made.

We have many programs and servers which are designed to do certain things, but they do not ensure or inform user that those things can not be done, in particular:

- We have many programs which are supposed to do one particular thing, but they fail to check if all prerequisites for doing the work are satisfied and consequently using these programs requires “expert” knowledge. For example, due to such deficiency in RhicChromaticity we lost 6 hours of APEX time.
- Our managers are supposed to manage particular device or large group of devices. But some managers do not take care of all aspects of the hardware they are managing. For example, magman / RhicInjection do not stop injection into RHIC if one PS in ATR which is supposed to be on, is not on, or if it is not at setpoint, i.e. operators depend on alarms to find out that something is wrong with ATR line.
- We have managers which do not implement functionality which one would assume they should be responsible for. For example, it is very hard to change frequency of IPM data taking in RHIC. Such functionality should be one of the main purposes of IPM manager. Instead various tape sequences were written to provide that functionality (but of course not in a user friendly way).

In short, we should try to ensure that our programs / managers are designed in such a way that they allow user to specify what he/she wants to be done and then they do whatever is necessary to achieve that, or inform user that because of the state of the system that can not be achieved. Tape and alarms should not be substitute for properly designed programs / managers.

We also do not use many tools which are normally used in software development, and we use tools which are not widely used or even used by us only. We pay for these distinctions in many ways:

- We do not have a way to record the problems with programs, for example their bugs or issues mentioned above. Normally bug / issue trackers are used for such purposes, but we do not have one, and therefore we can not keep track of issues and make plans to fix them before next run.
- We use ClearCase as version control system, whereas the most widely used version control system is Git. Even Microsoft switched from their proprietary version control system to Git. Not using Git limits how we developed software in a way which the following bullet point demonstrates.
- We use Red Hat Linux distribution which is generally known for having not up to date software, but we do not even use the latest version of it (we currently use RH 6.7, while the latest version is RH 7.4, our computers run Linux kernel 2.6.32, RH 7.4 comes with version 3.10, while the latest kernel version is 4.12), and therefore the software we use is very old. Then, we go to extraordinary lengths to install software required by new hardware into our old distribution, instead of just using new version of RH (or even different distribution such as Ubuntu) and developing software on it (we can not do that because we use ClearCase, but if we were using

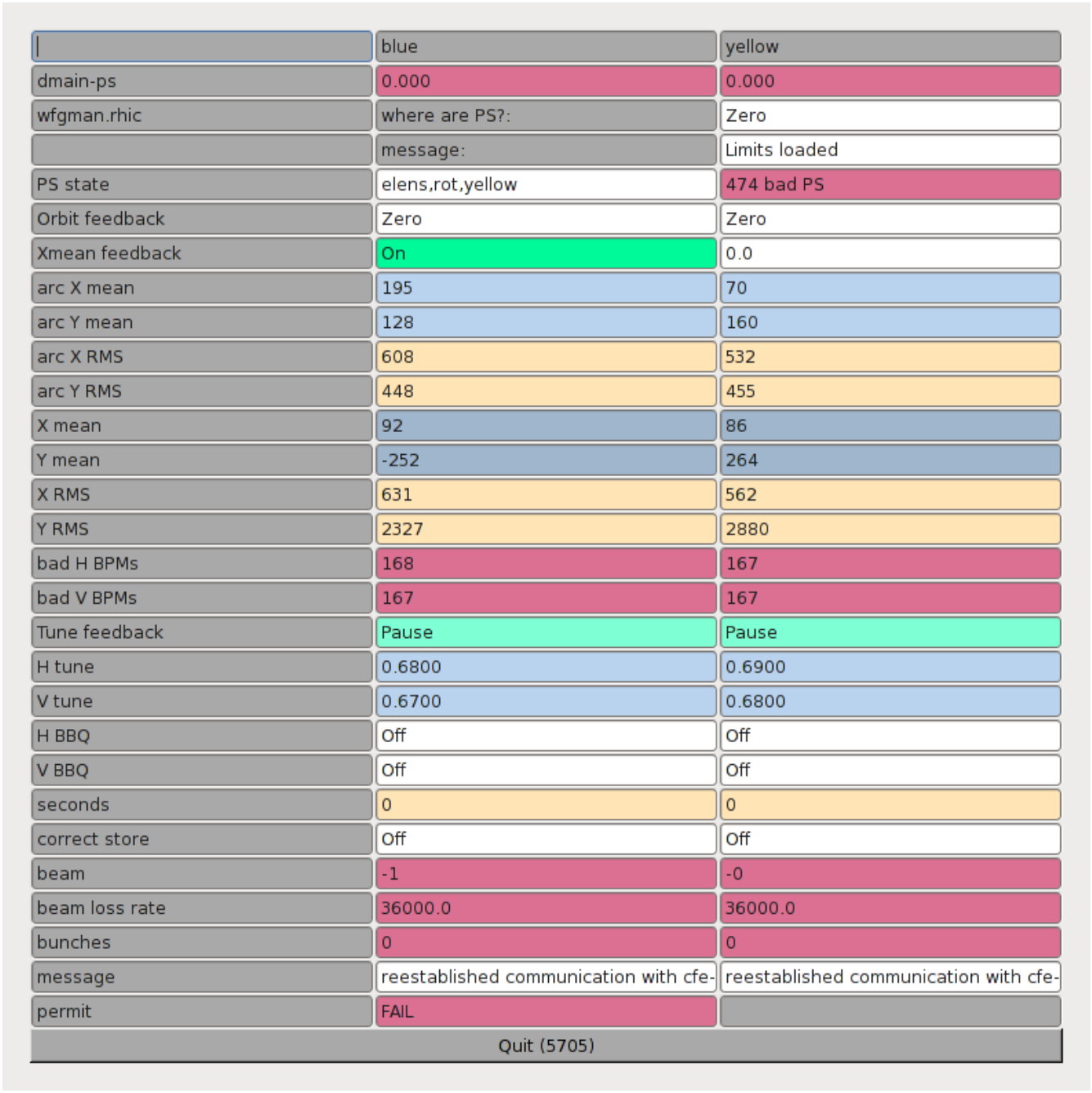
Git we could...). We should abandon the requirement that all computers should run the same Linux version.

- We do not use widget toolkits generally used in Linux for creating graphical user interface, i.e. GTK+ or Qt. Instead we have our own. That makes development of our GUI programs much harder and their quality much lower.
- We use and continue to develop programs which use CDEV networking code developed by JLAB and abandoned by JLAB in 2002 (and still not properly incorporated with the rest of our code). We then put a lot of effort to make software using CDEV behave as if it is using ADOs, for example storage server is CDEV server, but it could have been made much easier as ADO manager.

There are other issues related to our hardware and software:

- existing documentation is mostly obsolete,
- many programs are effectively abandoned, and issues with them never get fixed.
- there are no attempts to clean up anything, old files and directories are not deleted, even the most minimal maintenance of the system is not performed, e.g. out of 26 directories in PATH shell variable, 6 do not exist and have not existed for 20 years,
- security updates are performed only once per year,
- some programs due to many reasons ended up with very poor performance, for example LogView.

Many of our programs could be much better. smartpet.py, which is my version of pet, shows how pet could be made much more useful.



	blue	yellow
dmain-ps	0.000	0.000
wfgman.rhic	where are PS?:	Zero
	message:	Limits loaded
PS state	elens,rot,yellow	474 bad PS
Orbit feedback	Zero	Zero
Xmean feedback	On	0.0
arc X mean	195	70
arc Y mean	128	160
arc X RMS	608	532
arc Y RMS	448	455
X mean	92	86
Y mean	-252	264
X RMS	631	562
Y RMS	2327	2880
bad H BPMs	168	167
bad V BPMs	167	167
Tune feedback	Pause	Pause
H tune	0.6800	0.6900
V tune	0.6700	0.6800
H BBQ	Off	Off
V BBQ	Off	Off
seconds	0	0
correct store	Off	Off
beam	-1	-0
beam loss rate	36000.0	36000.0
bunches	0	0
message	reestablished communication with cfe-	reestablished communication with cfe-
permit	FAIL	

Quit (5705)

Picture 5: smartpet.py during shutdown

Conclusions

- We should develop Zynq version of generic VME modules. We should give up on development of VMEbus boards.
- We should fix design issues with many of our programs.
- We should rethink the need for many idiosyncrasies of our software environment.