Launching GUPPI: the Green Bank Ultimate Pulsar Processing Instrument

Ron DuPlain[*a], Scott Ransom[a], Paul Demorest[a], Patrick Brandt[b], John Ford[b], Amy L. Shelton[b]
[a]National Radio Astronomy Observatory, 520 Edgemont Rd, Charlottesville, VA, USA 22903-2475;
[b]National Radio Astronomy Observatory, P.O. Box 2, Green Bank, WV, USA 24944-0002

ABSTRACT

The National Radio Astronomy Observatory (NRAO) is launching the Green Bank Ultimate Pulsar Processing Instrument (GUPPI), a prototype flexible digital signal processor designed for pulsar observations with the Robert C. Byrd Green Bank Telescope (GBT). GUPPI uses field programmable gate array (FPGA) hardware and design tools developed by the Center for Astronomy Signal Processing and Electronics Research (CASPER) at the University of California, Berkeley. The NRAO has been concurrently developing GUPPI software and hardware using minimal software resources. The software handles instrument monitor and control, data acquisition, and hardware interfacing. GUPPI is currently an expert-only spectrometer, but supports future integration with the full GBT production system. The NRAO was able to take advantage of the unique flexibility of the CASPER FPGA hardware platform, develop hardware and software in parallel, and build a suite of software tools for monitoring, controlling, and acquiring data with a new instrument over a short timeline of just a few months. The NRAO interacts regularly with CASPER and its users, and GUPPI stands as an example of what reconfigurable computing and open-source development can do for radio astronomy. GUPPI is modular for portability, and the NRAO provides the results of development as an open-source resource.

Keywords: pulsar, FPGA, agile development, rapid application development, instrument control, data acquisition, hardware interface, Berkeley Emulation Engine (BEE)

1. INTRODUCTION

The Green Bank Ultimate Pulsar Processing Instrument (GUPPI) is a next-generation flexible digital signal processor (radio telescope backend) designed for pulsar studies using the Green Bank 43m (140ft) telescope and the Robert C. Byrd Green Bank Telescope (GBT). The National Radio Astronomy Observatory (NRAO) has been developing GUPPI using field programmable gate array (FPGA) hardware and design tools developed by the Center for Astronomy Signal Processing and Electronics Research (CASPER) at the University of California, Berkeley.

1.1 Design approach

The GBT Spectrometer, which took approximately 10 years to design, build, and deploy, is now more than 15 years old. Owing to the complex nature of its design, any significant enhancements to the GBT Spectrometer's pulsar search and precision timing capabilities would prove to be prohibitive in terms of both hard costs and staffing needs. Alternatively, NRAO has been exploring the development of radio telescope backends using reconfigurable off-the-shelf hardware platforms and software tools that allow rapid design, verification, and deployment of signal processing systems.

A reconfigurable computing platform enables iterative design, and as such, GUPPI's development consists of three distinct phases. As of spring 2008, GUPPI is at the end of Phase I.[1]

- Phase I: provide capability for standard pulsar observations using an incoherent filter bank and Full-Stokes.
- Phase II: provide wide-bandwidth (200-800 MHz) coherent-dedispersion observation modes.
- Phase III: fully integrate into the GBT Monitor & Control (M&C) system to become a common-user backend.

[*] rduplain@nrao.edu; phone +1 434-244-6845; www.nrao.edu

1.2 Science drivers and target capabilities

The design goals for GUPPI are to create a pulsar backend with high dynamic range (at least 8-bit sampling), large bandwidth (800-1000MHz), full polarization capabilities, better radio frequency interference (RFI) immunity, and high time and frequency resolution (at least 4096 channels dumped approximately 20000 times per second for GUPPI Phase I).[2, †] Current generation pulsar instruments are all missing one or more of these components, each of which are crucial for certain types of pulsar observations.

Detailed studies of the currently poorly understood pulsar emission process typically involve studying the brightest pulsars with spin periods between 0.1 to 10 seconds. These observations require full polarization data with high dynamic range and significant resistance to RFI in order to measure high-quality and high-signal-to-noise individual pulses.

Similarly, ultra-high-precision millisecond pulsar (MSP) timing for tests of General Relativity and the direct detection of nanohertz (nHz) gravitational waves requires each of the design goals mentioned above. MSPs are weak radio sources, yet their pulsar arrival times require measurements to tens or hundreds of nanoseconds – a factor of tens of thousands of times shorter than their typical spin periods. GUPPI aims to provide the processing power to support these observations for a common user backend on the GBT.

2. CASPER PLATFORM

As stated on its homepage, "The goal of CASPER is to streamline and reduce the current radio astronomy instrumentation design flow through the development of an open-source, platform-independent design approach."[3, ‡] CASPER provides several hardware board designs which use Xilinx FPGAs, and while CASPER in general does not provide turnkey instrumentation, it has been hosting regular workshops, engaging in much collaboration, and supporting a wide open-source community.[4]

2.1 Hardware summary

The CASPER platform primarily consists of three different types of hardware boards:

- iADC – an analog-to-digital converter that provides capture of the input signal.
- IBOB – the Internet Break-Out Board which provides packetization of the input signal.
- BEE2 – the Berkeley Emulation Engine 2 which provides flexible, high-performance signal processing.

CASPER-based designs are upgradeable. Various designs are underway for a new generation of boards, including the IBOB2 (known as ROACH) which holds promise for even more flexibility.[5]

2.2 Development toolflow

CASPER integrates various tools into a streamlined design toolflow called MSSGE, which stands for Matlab/Simulink/SystemGenerator/EDK. This toolflow links Matlab/Simulink with the Xilinx System Generator for DSP, Xilinx ISE, and Xilinx EDK. Together, these tools allow the design, simulation, and implementation of FPGAs through visual programming blocksets and various functions in Matlab/Simulink. CASPER adds several libraries to the toolflow for common radio astronomy functions, integrating custom hardware descriptor code with various Xilinx and Simulink blocks. The final result is a set of parameterized blocks which allow users to design radio astronomy instruments while largely avoiding low-level details of key modules and functions (e.g. analog-to-digital converter and Fast Fourier Transform). The MSSGE toolflow allows for full simulation and verification within Matlab/Simulink, so users can make all major design decisions before loading onto FPGAs. This provides a productive user experience when applied to CASPER's standardized boards with Xilinx FPGAs (i.e. iADC, IBOB, and BEE2) for which it provides the necessary tools for configuration and interfacing.[6, §]

† The GUPPI project originates in notes on a Next-Generation "Dream" Pulsar Machine.
‡ CASPER has an extensive list of collaborators, available at http://casper.berkeley.edu/collaborators.php.
§ This subsection is an adaptation (used with permission) from the University of Cincinnati CASPER user guide.

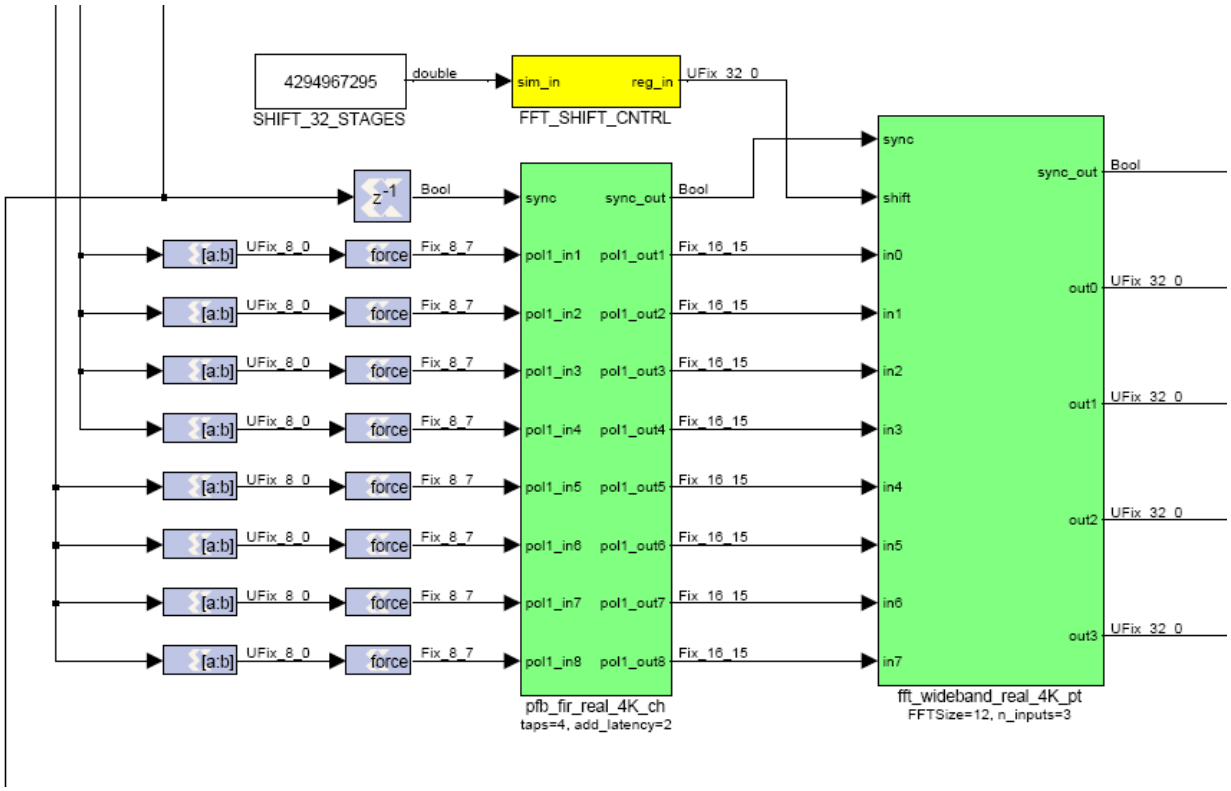

Figure 1. Above is an excerpt from a design using MSSGE. Users are able to use high-level block diagrams to design powerful signal processors and digital design systems. The block labeled "FFT_SHIFT_CNTRL" is an example of a software register, which is then accessible by that name through simple software-to-hardware interfaces.

2.3 Software interfaces

Developers can specify parameters with hardware registers and memory locations on an IBOB or BEE2 using a variable name, which is then accessible by the given name through a simple shell. The IBOB communicates via a "Tiny Shell" (TinySH, minimal command-line shell) interface. The BEE2 communicates via a Debian GNU/Linux operating system, known as the Berkeley Os for ReProgrammable Hardware (BORPH), running on a PowerPC platform (resident in a Xilinx Virtex-II Pro FPGA). BORPH provides access to hardware components (via the GNU/Linux proc file system), as well as a facility for quickly loading new FPGA configurations on the available BEE2 FPGAs.[7]

Further, the IBOB has a C compiler and the BEE2 can run anything that is possible with a 200 MHz processor-driven GNU/Linux system with a minimal amount of memory, which allows for further flexibility in low-level interfacing. These interfaces can communicate over-the-wire via on-board 100 Mbps Ethernet (100 MbE). These features provide for a relatively straightforward scheme in accessing hardware parameters, and further, for simple "snapshots" of values at various points in the data flow. That is, parameters and snapshot values are stored in registers (called "software registers") and memory locations (called "BRAMs"), which in turn are accessed through these on-board software interfaces.[8]

3. GUPPI HARDWARE OVERVIEW

GUPPI includes iADCs attached to two IBOBS, which connect directly to a single BEE2. The hardware components are accessible as follows:

- The iADC configures at the time of hardware design compilation. In addition to the compile-time configuration, several adjustable parameters may be available through an IBOB interface.
- The IBOB communicates via the TinySH, over a serial or 100 MbE (more reliable than serial) link.

- The BEE2 communicates via the BORPH GNU/Linux system, which is accessible through secure shell over a 100 MbE link.

A GNU/Linux host (named “beef”) connects to the BEE2 through a 10 GbE switch for data transfer, and connects to the IBOBs and BEE2 through link local 100 MbE connections for control links. This host machine has 8 CPU cores with 16 GB RAM and several terabytes of spinning disks, which can record up to 300 MB /s, sustained. This host is responsible for running all of GUPPI’s software, recording observation data to disk, and providing a physical control interface to GUPPI’s boards.

4. GUPPI CONTROLLER

GUPPI has a controller to provide a basic human and software interface for monitor and control of the instrument. The controller must also interact with a data acquisition system.

4.1 Requirements

The GUPPI Controller shall:

- Provide read and write access to all on-board parameters.
- Provide instrument start and stop commands.
- Run independently of the hardware boards, with the exception of the on-board interfaces of TinySH and BORPH.
- integrate with the GUPPI Data Acquisition (GDAQ) system by:
    - Providing all parameters and external information needed by GDAQ.
    - Reading status, diagnostic, error and logging messages from GDAQ.
    - Providing timely start and stop commands.
- Be compatible for future integration with the GBT M&C system.

In turn, the GUPPI Controller shall be provided with at least:

- from the GUPPI Data Acquisition module:
    - Simple status codes with an established convention.
    - A "heartbeat" in the form of a timestamp.
- from the GUPPI Data Acquisition module developers:
    - Required parameters and external information.
    - Conventions for all shared parameters and information, defined before development.

4.2 Plumbing overview

The GUPPI Controller has a top-down architecture, where the top level has abstracted parameters, which pass between the controller's server and an "external" client, which is either a user at a command prompt, or a simple API client. The bottom level has low-level board interactions. The IBOB interface simply makes sense of the TinySH prompt and simplifies its interaction. The BEE2 interface connects the Demux module to a custom TCP server running on the BEE2, which accepts connections and interacts with parameters through the BORPH file system.

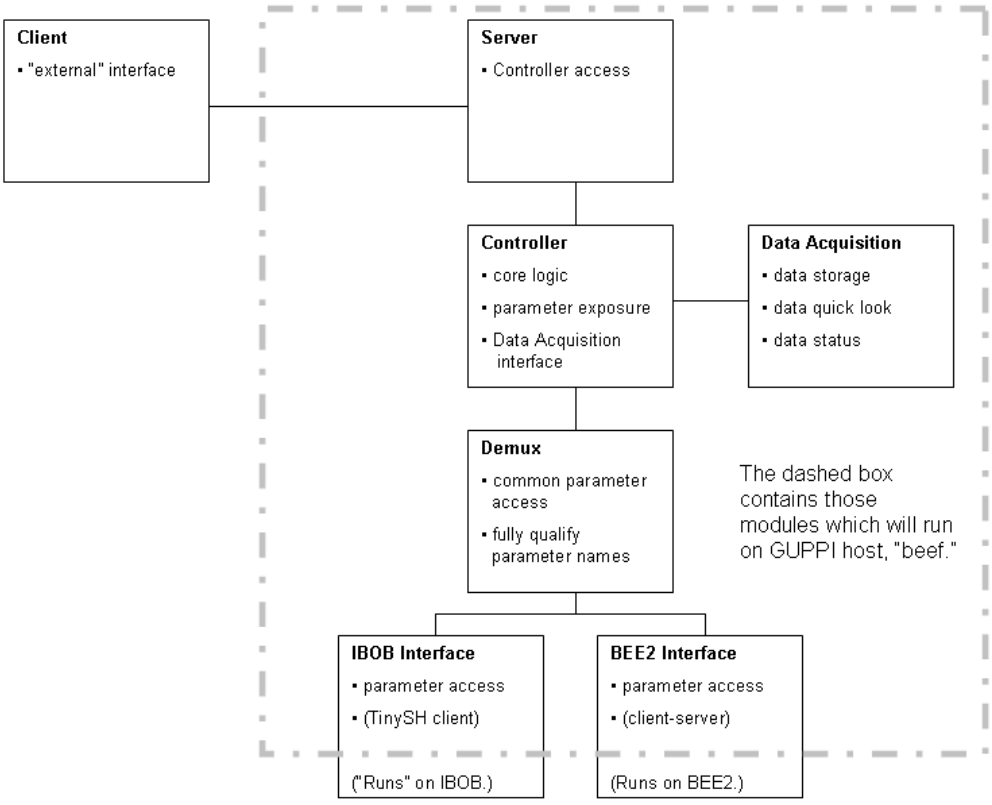

Figure 2. Above is a plumbing diagram, showing key controller modules, their key responsibilities, and information flow between them.

Currently, the IBOB client is unused since no IBOB interface is required for the GUPPI Phase I design.

4.3 Expert user interface

The client-server model upstream from the controller allows users and other software tools to connect to the GUPPI Controller from anywhere on the network. The server extends Python's built-in SimpleXMLRPCServer module, which takes care of all underlying connection details; the client need only use Python's xmlrpclib to connect. Combining all of the components in Figure 2 provides a high-level control path from a Python interpreter at the client to send and receive parameter data to all of GUPPI's boards and their parameters; this interpreter is a single portal to all parameters with fully qualified names (i.e. (board)/(component)/(parameter)) thanks to the dumux module.

A simple command prompt wraps the client module. Python's native interpreter can easily extend into a custom interpreter thanks to the built-in sys and readline (which integrates the system tool GNU readline) modules. The readline

module allows for custom tab completion, which is excellent for agile projects. Tab completion allows for a simple, but effective user dialog at the command prompt, and in a way, provides documentation through the tab key. Further, the command prompt module provides simple top-level functions (such as high-level instrument functions and special monitoring methods), and the ability to script commands (thanks to Python's natural scripting ability). Even Python import statements work easily; this allows for easy integration of plotting tools such as Matlab-inspired matplotlib for Python. A simple extension could use standard input/output (i.e. UNIX-like stdin, stdout, and stderr) to adapt this command prompt into a graphical interface.

In its simplest form, the core expert user interface is four operations.

- `value = get([key])` – returns the `value` of parameter with name `key`; if no `key` is given, returns a list of known parameters.
- `success = set(key, value)` – sets the value of the parameter with name `key` to `value` returns true if successful (false otherwise).
- `success = load([profile])` – loads the specified `profile` (FPGA configuration) and returns true if successful (false otherwise); if no `profile` is given, returns a list of known profiles.
- `success = unload(profile)` – unloads the specified `profile` (partial matches accepted) and returns true if successful (false otherwise)

These four functions provide a basis for all GUPPI interaction.

4.4 Portability

As long as a design has known parameters, this controller can easily provide a simple command prompt, even for impromptu test designs. The generic parameter-passing scheme of the GUPPI Controller applies to nearly any project. As such, other CASPER-based instruments at NRAO benefit from GUPPI's code base. The availability of GUPPI's code will certainly increase as development continues; in general, NRAO source code is publicly licensed (GPL). IBOB users should note that since the GUPPI Phase I design does not require direct communication with the IBOBs, the controller's IBOB client would remain untested until later phases.

The CASPER community has been developing a UDP packet protocol and related tools for controlling boards over the network. The GUPPI Controller is an alternative to those tools. GUPPI Controller allows control of several boards from a single custom command line interface and uses TCP for a more reliable connection.

GUPPI manages its components, including the controller server and the BEE2 server, using a package called Supervisor,[**] which is free software built on Python and installs easily using "easy_install."[††] Supervisor runs processes persistently, restarts crashed processes, provides remote access over HTTP and UNIX sockets, and manages process logging. It is not NRAO-specific, and it will easily port to other systems and applications.

5. GUPPI DATA ACQUISITION (SUB)SYSTEM

The GUPPI data acquisition software subsystem is responsible for receiving the data stream output by the hardware over 10 GbE, and writing this to disk. It currently fulfills this role, and future development will allow for additional real-time software signal processing before the data is stored. The system's design aims to be modular, flexible, and independent of the detailed implementations of the GUPPI Controller software and the hardware data source. The data passes between independent, asynchronous processing threads via a series of one or more shared memory ring buffers. This design allows for optimum use of multi-core computing resources, and permits external monitor or debugging applications to inspect the data easily at any point. The core data acquisition programs consist of C source code, and a Python interface is included for control and simple data inspection.

[**] The Supervisor homepage is http://supervisord.org/.
[††] The Python program "easy_install" makes installing Python packages easy, even on the BEE2. See http://peak.telecommunity.com/DevCenter/EasyInstall for more details.

This system must also gather all relevant monitor and control parameters, which it records in the output files. These files are the final data products that astronomers using GUPPI will take away for further offline analysis. GUPPI stores its output data in the PSRFITS format,[9] an emerging FITS-based international standard for pulsar data storage.

5.1 Monitor and control architecture

The data acquisition system passively receives monitor and control parameters through a single shared memory status buffer. Access to the status buffer is synchronized using POSIX semaphores to denote whether it is in a locked or unlocked state. The status buffer stores parameters using pseudo-FITS syntax of 80 character blocks containing ASCII keyword/value pairs, which makes the raw buffer contents human-readable as well as easily parsed by existing FITS libraries. In GUPPI Phase III, the status buffer will also contain information such as telescope position, source names, and frequency, available through the GUPPI Controller software (which in turn gathers the information from observatory M&C systems). For early testing or for use of the data acquisition software as a standalone system, a simple Python interface allows for manual and script-generated parameters.

5.2 Data architecture

The data flow in the data acquisition system is built upon an architecture of independent processes (or threads) passing information through a set of shared memory data ring buffers. This concept is similar to that used in previous software-based pulsar machines such as the Astronomy Signal Processors.[10] Each data buffer logically consists of a series of blocks, each of which can individually be marked as "full" or "free" through a semaphore array. The buffer then essentially functions as a single-reader, single-writer FIFO where a writing process waits for a free block before filling it with data, and a reading process waits for full blocks then processes the data and frees the block. Multi-step data processing can chain several buffers together, with processes reading from one and writing to another. The structure of each block within a buffer is very general, consisting of a header that uses the same pseudo-FITS syntax described above, and a data area (typically 32 MB per block). This allows reuse of same memory allocation and locking/unlocking code for buffers holding a variety of types of data. A simple Python module for inspecting and monitoring the data buffer contents provides a simple user and script interface to these buffers.

In the most basic mode of operation, the software only uses two processing threads and a single data buffer. A network processing thread reads incoming UDP packets sent from the GUPPI FPGA hardware, and fills these packets into the data buffer. Each packet contains a single spectrum, typically integrated over 50 μs, and a packet sequence number that functions as the data timestamp. The network thread strips off the sequence number and places the spectrum data in the correct time order in the data buffer, filling dropped packets as zeroes in the data. The thread also reads header information at this time from the status buffer and copies it into the data block header. When a block is full, the disk thread picks it up and parses the data block header parameters, and records them and the actual data to disk in PSRFITS format. The data ordering within the packets is as close as possible to PSRFITS specifications; this minimizes any requirement for software reordering.

The system runs at high data rates, with the hard drive write speed being the limiting factor. GUPPI's current data recording RAID system can write up to 300 MB/s continuously, though 200 MB/s is the typical target rate. The 10 GbE networking code and hardware can receive data at up to about 800 MB/s with minimal (less than 10^{-4}) packet loss. Currently in development is a real-time "folding" thread. This would fit in the chain between the network and disk threads, and will average the incoming spectra modulo the period of the pulsar under observation. This style of observing is ideal for monitoring known pulsars, and can reduce the final output data volume (and disk write speed) by a factor of several hundred or more.

5.3 Portability

The data acquisition system runs portably and independently of the other GUPPI components. In May 2008, the data acquisition code was able to run almost completely unmodified to record data from an alternate FPGA spectrometer developed independently of NRAO efforts.[11] In this case, the data only required a simple reordering to conform to the PSRFITS standard, demonstrating the data acquisition system's modularity for reuse. GUPPI Phase II involves exploring different designs for a high-bandwidth coherent dedispersion backend, one of which is software based. For that application, this same data acquisition framework could run on a cluster of machines, receive channelized baseband data from the FPGA hardware, and coherently dedipserse and fold it.

6. FIRST LIGHT

GUPPI saw first light with the GBT on 17 Apr 2008, observing the pulsar B1824-24 (also known as J1824-2452A or M28A). At the time, GUPPI was in development during Phase I; regardless, it exhibited two times the time resolution of the GBT's previous pulsar instruments, which revealed features not previously observed with the GBT.

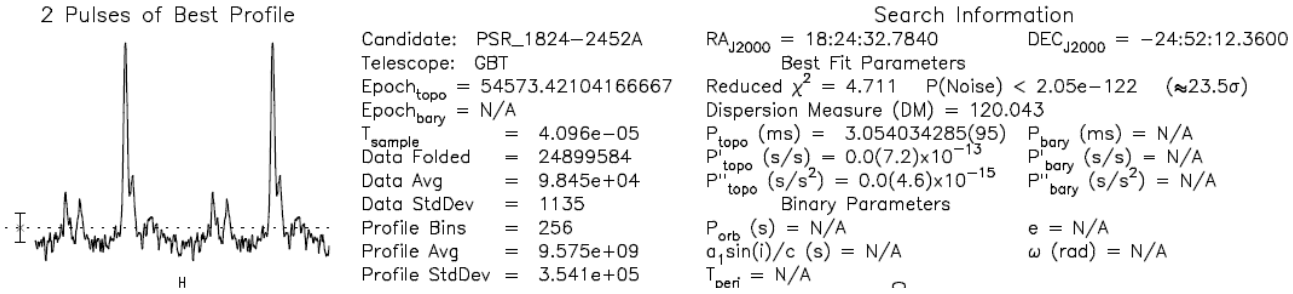

Figure 3. Above is an informational plot of GUPPI's first light. Other GBT pulsar instruments do not resolve the "notch" in the observed pulses, but GUPPI's higher time resolution reveals additional detail.

7. RECONFIGURABLE COMPUTING

Through its reconfigurable computing platform, GUPPI has significant potential for sharing modules and resources with other signal processing instruments. All of GUPPI's implementation details are essentially software, whether they exist as FPGA configurations or CPU instructions. Hence, various components and whole modules share easily with similar projects.[12]

7.1 CICADA: the bigger picture

GUPPI is part of a larger program at NRAO, Configurable Instrument Collaboration for Agile Data Acquisition (CICADA), which aims to deploy instruments for the GBT and 43m (140ft) telescopes in Green Bank while supporting CASPER-based efforts within the university community. In the months leading to GUPPI's first light, dedicated staffing for CICADA and GUPPI combined was roughly 1 FTE hardware development, roughly 1 FTE software development, and approximately 1.5 FTEs scientific support. Though most CICADA development in the spring of 2008 supported GUPPI directly, CICADA in general benefited greatly. Since the hardware itself does not change (e.g. an IBOB is still an IBOB even if its FPGA configuration changes), design tools, knowledge bases, and implementations can easily apply to any of CICADA's projects.

7.2 Cloning GUPPI

GUPPI is not a solo instrument. GUPPI's goal is to be just as modular and new-user-friendly as CASPER. As GUPPI matures, its hardware and software code bases will readily adapt where others seek to put together a clone. In order to do so, an institution must first adopt, deploy, and become familiar with the CASPER platform, and be willing to support in-house development, at least at a minimum, to ensure that GUPPI is adapted to the specific needs of that institution.

8. CONCLUSIONS

Development for GUPPI was able to make substantial strides over the course of a few months in early 2008. CASPER's reconfigurable computing platform and high-level programming tools allowed for rapid, iterative development of hardware implementations, while in software, parameter access by name allowed for simple design patterns to provide an effective control interface.

GUPPI's expert user command prompt proved to be a simple and elegant solution for low-level parameter monitor and control. By extending Python's native interpreter, the GUPPI command interpreter became easily extensible to quickly accommodate new requirements and feature requests, through the ability to import standard and third-party Python modules and packages, implementation of user-friendly tab completion, and use of high-level scripting constructs (such as function declaration).

GUPPI's implementation took advantage of the CASPER platform's clean, well-understood interfaces to hardware parameters. As a result, the CICADA/GUPPI team developed hardware and software components for GUPPI concurrently. By keeping to a generic parameter-passing scheme, maintenance of the expert user interface primarily involved maintaining a listing of hardware parameters. This allowed NRAO to apply a small set of resources very effectively and productively when deploying GUPPI.

Early testing and first light results of GUPPI are promising. As the project moves into the next phases, the command prompt interface will extend to provide even more functionality and the new pulsar processor will fully integrate into the GBT, while the data acquisition system will provide for more real-time processing on GUPPI's GNU/Linux host.

ACKNOWLEDGEMENTS

GUPPI would not be possible without the wide collaboration of the CICADA participants.

Thanks to the University of Cincinnati (UC) team and its supporters for foundational work. The UC team consists of Mike Borowczak, Ron DuPlain, Michael Teuschler, and Andy Severyn, with advisor Dr. Philip A. Wilsey and generous support and guidance from Dr. Ranga Vemuri.

Thanks to the West Virginia University (WVU) team and its supporters for continued collaboration. The WVU team includes Duncan Lorimer, Maura McLaughlin, Mitch Mickaliger, Patrick Brandt, and Brandon Rumberg.

The GUPPI project team includes Randy McCullough, Jason Ray, and Glen Langston, with thanks to other engineers and scientists at NRAO Green Bank, who provide ongoing hardware research and development. Thanks to Rich Lacasse, Rick Fisher, and other engineers and scientists at NRAO Charlottesville for ongoing advisement and support. Thanks to Walter Brisken at NRAO for guidance and contributions.

Special thanks to the CASPER group and its community for developing an excellent program for astronomical signal processing, for supporting extensive open-source collaboration, and for active engagement in its collaboration with NRAO.

Special thanks to Xilinx, Inc, for its support and generous donations to the CICADA projects.

The National Radio Astronomy Observatory is a facility of the National Science Foundation, operated under cooperative agreement by Associated Universities, Inc.

REFERENCES

[1] NRAO, "GUPPI on the Wiki," https://wikio.nrao.edu/bin/view/CICADA/GUPPIOnTheWiki (2008).

[2] Ransom, S. "Next-Generation 'Dream' Pulsar Machine Notes," http://www.gb.nrao.edu/~jlockman/new_pulsar_instrument.txt (2006).

[3] CASPER, "CASPER: Center for Astronomy Signal Processing and Electronics Research," http://casper.berkeley.edu (2008).

[4] Werthimer, D., Backer, D., and Wright, M., "PetaOp Signal Processing for Radio Astronomy," http://www.aui.edu/future_committee/getfile.php?filename=%2F3_Presentations_Charlottesville_Meeting%2FWerthimer.pdf (2007).

[5] CASPER, "CASPER Hardware," http://casper.berkeley.edu/hardware.php (2008).

[6] DuPlain, R., Borowczak, M., Severyn, A., and Teuschler, M., "FPGA Spectrometer: Virtual Modeling and Simulation using Matlab/Simulink," http://www.cv.nrao.edu/~rduplain/fpga-dev/Cincinnati_CASPER_UserGuide.pdf (2007).

[7] So, H. K.-H., and Brodersen, R., "A Unified Hardware/Software Runtime Environment for FPGA-Based Reconfigurable Computers using BORPH," *ACM Transactions on Embedded Computing Systems (TECS)*, Volume 7, Issue 2, Feb, 2008, New York, NY, USA.

[8] Parsons, A., et al., "PetaOp/Second FPGA Signal Processing for SETI and Radio Astronomy," Asilomar Conference on Signals, Systems, and Computers, November 2006.

[9] Hotan, A. W., van Straten, W., and Manchester, R. N., PASA, 21, 302 (2004).

[10] Demorest, P. B., Ph.D. Thesis, University of California, Berkeley (2007).

[11] McMahon, P., "Parspec," http://casper.berkeley.edu/wiki/index.php?title=Parspec (2008).

[12] Hauck, S., and DeHon, A., *Reconfigurable computing: the theory and practice of FPGA-based computation.* Systems on silicon. San Francisco, CA, Morgan Kaufmann (2007).