

The PicoRadio Test Bed

Fred Burghardt, Susan Mellers, Jan Rabaey

December 30, 2002

Berkeley Wireless Research Center
U.C. Berkeley
<http://bwrc.eecs.berkeley.edu>
flb@eecs.berkeley.edu

ABSTRACT

In this paper, we describe the architecture and use of the PicoRadio Test Bed, a general-purpose emulation platform for exploring new wireless communication technologies. It primarily supports the PicoRadio program at the Berkeley Wireless Research Center, a research facility of the University of California at Berkeley. The Test Bed provides a programmable hardware platform with basic radio connectivity, processor, and configurable logic along with a rich set of hardware abstractions and a complete programming environment for researchers wishing to explore PicoRadio network technologies in a real-world environment.

1. INTRODUCTION

The PicoRadio group at the Berkeley Wireless Research Center is conducting research into small, very low power¹ *system-on-chip* (SOC) devices and the wireless network environment in which they will operate. This work encompasses a wide range of disciplines, including basic RF design, configurable logic, communication vs. computation tradeoffs, protocol design at several layers, and applications. A PicoRadio network, composed of potentially many thousands of PicoNodes, is optimized for minimal power consumption and is self-configuring; bandwidths are generally low, on the order of what might be expected from an environmental sensor such as temperature, humidity, or light intensity.

There are many potential application areas for PicoRadio networks. Of particular interest to PicoRadio researchers is the smart building. Most modern large buildings do not provide an accurate picture of internal environmental conditions. For example, a single thermostat on one wall of a large room controls temperature accurately only for the few feet immediately adjacent. This arrangement is also susceptible to design or construction oversights e.g. a thermostat located next to a heater vent or air return. Also, an enormous amount of energy is wasted heating unoccupied space; if a large office is occupied by a small number of people, it is wasteful to heat the entire office.

A smart building requires fine-grain control over the environment within its walls. In order to achieve this level of control, a great deal of information on the current conditions is needed. This information must include temperature, humidity, and light

measurements from points in close proximity, sometimes a meter or two away. PicoRadio research targets this scenario.

A very large number of unobtrusive sensors would provide the data needed. These sensors must be self-powered; changing batteries is not an option due to the number of nodes and inaccessibility of some of them. Because the nodes are self-powered, they must consume very little power to maximize lifetime (a sensor embedded in the wall would have to last the lifetime of the building)². They must be very inexpensive for such a dense network to be affordable. They also must be able to transfer their information to a place where it can be utilized. These are all challenging problems that push technology in semiconductors, power delivery, and networking. [5][6][7][8]

Members of the PicoRadio group have developed unique and powerful network and media access control (MAC) protocols that enable a set of PicoNodes to communicate over radio links. The network implemented by these protocols is ad-hoc, meaning that it self-configures according to the environment and the presence of nodes in the near vicinity. At the network level, there is no central point of failure or single critical resource. The driving principle behind the protocols is to place minimal energy demands on the nodes.

In order to enable real-world investigation into system-level aspects of a PicoRadio network before the SOC devices are available (and also to help determine how a PicoNode should be designed), a prototype environment has been built. This environment is referred to as the PicoRadio Test Bed. The Test Bed is a collection of hardware nodes, software and programmable logic algorithms that run on the nodes, and design entry and compilation flows required to implement the algorithms. Each node is composed of two major parts: a set of custom circuit boards with commercial off-the-shelf (COTS) components and a software infrastructure that allow PicoRadio designers to make use of the hardware.

The Test Bed provides a service to PicoRadio researchers. It is a general-purpose embedded systems test platform intended for limited deployment in support of a wide spectrum of research activities related to the PicoRadio project.

This paper discusses PicoRadio networks in general, motivation and design goals for the Test Bed, hardware architecture, design

¹ Average power consumption less than 100uW.

² A cooperative research effort in Mechanical Engineering is looking into 'energy-harvesting', or gathering energy from environmental sources such as ambient vibration.

entry points, libraries, and design flows. It also describes some past and current applications for the Test Bed.

2. DESIGN GOALS

The intent of the Test Bed is to provide a platform for real-world emulation of PicoRadio technologies. To make it usable to researchers, it requires:

- Characteristics and capabilities similar to a PicoRadio SOC node e.g. configurable logic, custom logic emulation, programmable processor/controller, knobs for power control, short range radio.
- Easy to use design entry methods
- Simple interfaces to hardware resources. A user should not have to know about arcana such as processor register bit assignments, FPGA memory mapping, interface timing, etc.
- Full software compile, execute and debug environment
- Radio independent, to allow experimentation with various RF front-ends.
- Able to interoperate with custom application hardware such as various sensors and actuators.
- Power consumption low enough to allow continuous battery operation for at least 24 hours.
- Size suitable for unobtrusive placement but large enough to accommodate all desired features and some unanticipated needs.

3. HARDWARE ARCHITECTURE



Figure 1: A fully assembled Test Bed node with sensor board, and a case with the core boards and radio board removed

The PicoRadio Test Bed physical node (figure 1) is composed of a digital board, a power board, a radio board, a sensor board, and a case³.

The digital and power boards are collectively referred to as the *core*. This term indicates that all nodes contain at least these two boards and that these two boards are not subject to modification or replacement by users. A node with only core boards is usable for situations where real sensor data or radio connectivity is not needed.

³ The case was designed by a student from the Dept. of Mechanical Engineering, as part of an ongoing cooperative relationship between BWRC and ME. [3]

Complexity is highest on the digital board. The basic hardware architecture was developed about five years ago for a previous project and leveraged into PicoRadio. The first digital board for the Test Bed was built in 1999. Board size is dictated by the digital board layout, which is dense on both sides. All other boards use the same basic form factor regardless of density, at least with respect to the placement of the large 120 pin board-to-board connectors visible in the photos.

Boards are “stacked” one on top of the other through these connectors. In figure 1, the sensor board in the leftmost image is plugged into a digital board; its connectors pass through the case lid.

3.1 The Digital Board

The digital board (figure 2) contains a Strong ARM 1100 embedded microprocessor with 4Mb of DRAM and 4Mb of flash memory, and a Xilinx XC4020XLA Field Programmable Gate Array (FPGA) with dedicated external SRAM and flash. The ARM is used to emulate functionality that may be mapped into a general-purpose processor or microcontroller. It contains a CPU core and a variety of peripheral controllers for services such as standard I/O control and timers. The FPGA is used to emulate tasks that would be assigned to configurable or custom logic on a PicoNode SOC. Radios are connected directly to the FPGA so that time-sensitive radio register programming and control can take place in an FPGA block.



Figure 2: Digital board

Just about every useful signal from the processor and FPGA is exported from the digital board via the board-to-board connectors, including:

- ARM address and data busses
- Memory bus control (CS, CAS/RAS, OW, WE)
- All ARM serial ports (UART, SPI, Ti, Microwire, IrDA, SDLC)
- FPGA general-purpose pins
- ARM GPIO pins
- Dynamic Voltage Scaling (DVS) control
- Power supply fault pins
- Radio digital interface signals
- LCD interface

- USB interface
- PCMCIA interface
- Audio and telecom codecs
- Touch screen interface
- Battery

Researchers wishing to make use of these capabilities build a board conforming to the connector specification and add whatever components are necessary. Our sensor boards are examples of this.

3.2 The Power Board

The power board (figure 3) provides 5.0v, 3.3v, and 1.5v power. The 3.3v supply is connected to the processor I/O pads and the FPGA. The 1.5v supply is for processor core power; this supply can be dynamically adjusted between 0.925v and 2.0v. The 5.0v supply is an auxiliary. The 1.5v supply adjustments are coupled with the processor clock for power control. A node is booted at 60MHz@0.925v. A 'C' function interface allows the user to adjust power levels in integral steps from 1 to 20 if needed, where 1 is the boot level and 20 is 200MHz@1.5v (nominal for the ARM). Power consumption can be adjusted by up to an order of magnitude; lower is better, obviously, but there are applications that require a faster CPU clock such as audio tone generation. The power board exists to service the digital board only; additional boards must include their own power supplies⁴.



Figure 3: Power Board

3.3 Radio Boards

We currently have boards that support two radio options: Proxim RangeLAN and Bluetooth (figure 4). The Bluetooth radio is the primary RF front end for the Test Bed, because it models the short range expected from PicoRadio SOC nodes. It is based on an Ericsson RF front-end module that provides a serial bit stream interface. We do not use the Bluetooth protocols; only the analog portion of the Ericsson Bluetooth implementation is utilized⁵. All

⁴ The 5.0v auxiliary supply is a carry-over from an earlier radio board, which did not have a supply. It is exported and can be used by other boards, but with no guarantee that capacity will be sufficient.

⁵ There is actually an A/D and a little bit of digital to create the bit streams and manage programming.

PicoRadio baseband and network protocols are custom designed by PicoRadio researchers. The Bluetooth radio is capable of transmitting and receiving on 128 channels; it is normally used with frequency-hopping. It transmits in the ISM band (2.4GHz) at 1Mbps.



Figure 4: Bluetooth Radio Board

3.4 Sensor Boards

We currently have two sensor board designs. Board #1 contains sensors for temperature, humidity, light, and sound. Board #2 has sensors for temperature, acceleration, and magnetic fields (figure 5). Sensor board #2 also has provisions for GPS.

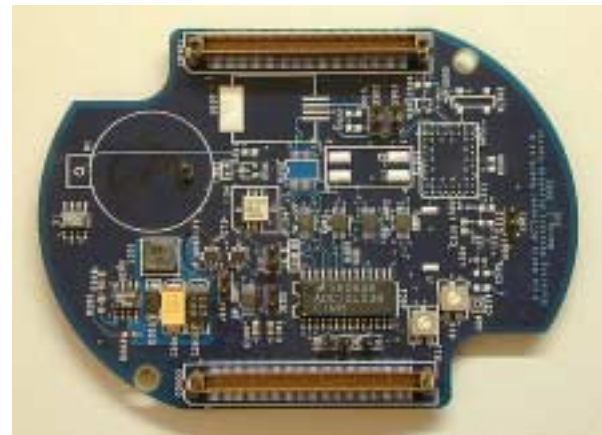


Figure 5: Sensor Board #2

All of the circuit boards built for the Test Bed were designed in-house. Fabrication and assembly were contracted, and test was done by BWRC staff and students.

3.5 Power Consumption

The Test Bed node was designed to run for at least 24 hours on a battery charge⁶. This varies substantially depending on the application and use of power control facilities, from just a few hours to over 60. Table 1 shows the results of power measurements we took for a node with no sensor board. The dashed box indicate the typical operating environment for a node running PicoRadio protocols. The solid box indicates the point of

⁶ Two 1200mAH StarTac lithium ion batteries in series.

absolute maximum battery life. The differences between P_1 and P_2 highlight the advantages of DVS.

ARM	Xilinx	Radio	P_1 (mw)	P_2 (mw)	Battery Lifetime (hours)
Run	Run	Tx	677	302	29
		Rx	691	331	26
		Idle	583	209	41
		Off	569	202	42
	Sleep	Tx	612	238	36
		Rx	634	299	32
		Idle	518	144	60
		Off	511	137	63
Idle	Run	Tx	583	288	30
		Rx	605	317	27
		Idle	497	194	44
		Off	490	182	46
	Sleep	Tx	526	223	39
		Rx	554	252	34
		Idle	439	137	63
		Off	432	130	67

P_1 : ARM Clk = 200MHz, $V_{ee} = 1.5v$
 P_2 : ARM Clk = 60MHz, $V_{ee} = 0.925v$

Table 1: PicoNode power consumption

4. USERS VIEW

A user of the Test Bed is typically a graduate student at BWRC who is working with the PicoRadio project and has an algorithm or application to evaluate. As stated in the goals, a primary objective of the design was to allow users to implement their processor or FPGA designs without detailed knowledge of the hardware or software architecture. The users view of a Test Bed node is comprised of four elements:

1. A workspace
2. Entry points
3. A set of libraries
4. A design flow

There is a separate set of these elements for the processor and FPGA.

Together, the entry points and libraries are referred to as 'the kernel'. The kernel is not an operating system in the traditional sense. Rather, it is a mechanism that provides more-or-less transparent access to capabilities of the Test Bed that are important as a means to facilitate research but not part of the research itself. In other words, a researcher doesn't have to waste time figuring out how to program the processors interrupt

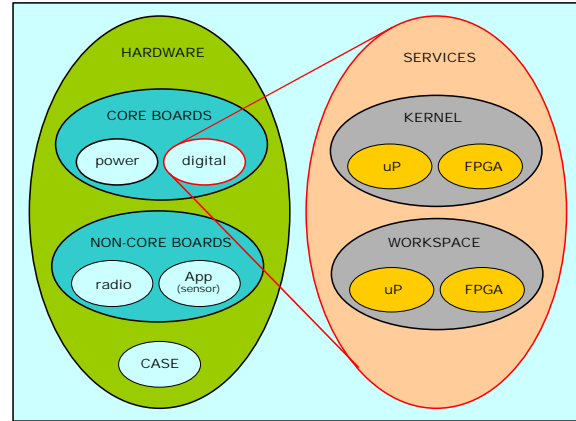


Figure 6: System view. Services are mapped onto the hardware of the digital board

controller registers in order to install an interrupt, or build a processor bus arbitration circuit for the FPGA.

A new project is created by copying a template into a design directory of the users choosing. The project template contains a minimal set of source files and FPGA blocks, a compile and edit environment for the processor, a design and build environment for the FPGA, a library for the processor, a library for the FPGA, and a basic API. The workspace is the starting point for the application programmer.

The programming model for the Test Bed is to build applications that are invoked by the kernel via the entry points, and to access the capabilities of the underlying hardware through the libraries. The libraries are represented by system calls for the processor and pre-constructed objects presented as schematic symbols for the FPGA. No assumptions about applications are built into the system except that they are suitable for a PicoRadio SOC.

The processor uses a single-threaded interrupt driven model of computation. PicoRadio protocols and applications do not use advanced features of a traditional operating system such as multithreading, virtual memory, and processes because they are targeted to a system (the PicoRadio SOC) with minimal resources and a very limited power budget. The kernel is likewise minimal, consisting of initialization code that includes loading a FPGA configuration, a kernel main loop that idles the CPU when possible, and a large set of function calls for accessing processor resources.

4.1 The Workspace

The workspace is the first thing a user sees when implementing a design. It is the primary design entry environment.

4.1.1 Processor Workspace

A user views the processor through a 'C' code editor and design manager from ARM Ltd (figure 7). The processor workspace interacts with the processor itself via a PC-to-node serial port until the code is sufficiently mature to be embedded into flash memory. The default processor workspace includes several 'C' source files that provide an entry point to the kernel and some basic services.

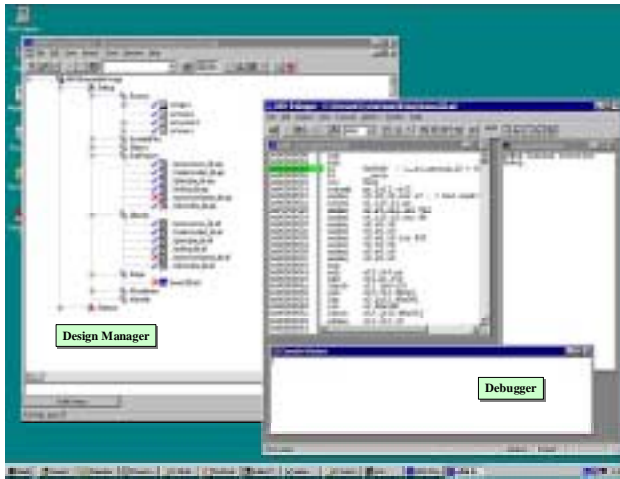


Figure 7: ARM Project Manager

4.1.2 FPGA Workspace

The FPGA workspace is represented by a mostly blank schematic displayed in the Viewdraw editor from Workview Office (figure 8) or the ISE editor from Xilinx. The only object in the schematic is an interface block that acts as I/O between the FPGA and the processor. Users instantiate library blocks onto the schematic page as symbols or create new blocks that contain either lower-level schematics, VHDL, Verilog, or EDIF models. The blocks are interconnected by drawing busses (an arbitrary number of grouped wires) and nets (single wires) to pins on a schematic symbol. It is possible to design entirely in a text-based language and instantiate a single symbol that references the design. The only interconnect requirement in that case is to connect the single symbol with the interface block.

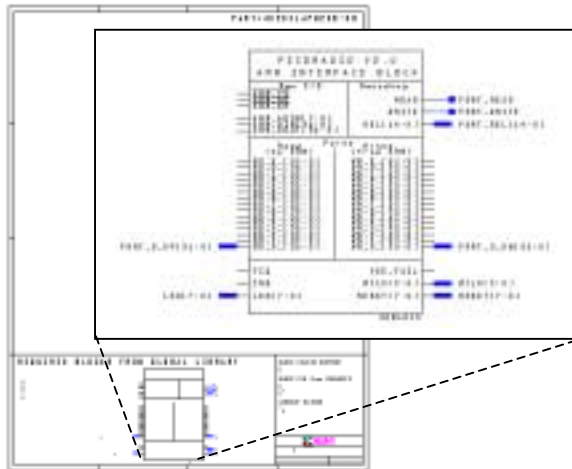


Figure 8: Empty FPGA workspace with processor interface symbol shown in inset. Busses and nets are attached to the interface.

Several methods are available for describing behavior:

- Viewdraw schematics
- Xilinx ISE schematics
- Handel-C
- StateCAD
- VHDL

As noted above, schematics represent the highest level of an FPGA design – the place where sub blocks are “glued” together. Selection of Viewdraw or ISE as the schematic editor is fixed by the initial choice of workspace.

Handel-C is a hardware description language based on ANSI ‘C’, produced by Celoxica Ltd as part of their DK1 design environment. It is a “super-subset” of standard ‘C’, in that some capabilities are disallowed (e.g. dynamic memory allocation) while keywords indicating specific hardware behaviors are added. The latter set includes *par*, which indicates that a basic block should be executed in parallel. Since the target is hardware, parallel in this context actually means ‘at the same time’. The Handel-C compiler can generate VHDL or EDIF.

StateCAD is a graphical extended finite state machine (EFSM) designer from the Xilinx ISE tool set. It’s an alternative to Handel-C; we leave it up to the application programmer to determine which model of computation is appropriate for the circumstance. Handel-C has a significant edge in descriptive power while StateCAD is good for simple control algorithms. Most Test Bed application work uses Handel-C, but many library blocks have StateCAD control elements.

The workspace will also accept hand-written VHDL.

Behavior is ultimately represented by a symbol in a schematic. The design flow generates EDIF for each HDL block, and the EDIF is referenced by symbol name in the final build.

4.2 The Entry Points

In the Test Bed, the entry points are mechanisms for hooking user designs into the Test Bed system itself. As with the other elements of the users view, there is a set of entry points for the processor and a set for the FPGA.

4.2.1 Processor Entry Points

There are two aspects to the processor entry point: 1) a set of empty ‘C’ functions: an initialization function, a main function, and a global fast interrupt service routine (ISR) and 2) standard interrupt service routines. See figure 9.

The initialization function is run once at startup, after the kernel has been initialized (the FPGA is programmed during kernel initialization). The main function is called from a loop in the kernel at some periodicity not guaranteed by the kernel. User code is not allowed to block in main, since the mechanism for sharing the processor is to call user code entry points sequentially in the main function. Although the libraries provide access to 32 separate interrupt sources, the processor provides for a single ‘fast’ interrupt vector that is global (FIQ). The template ISR `app_FIQISR()` is provided for this interrupt.

The `app_Init()` and `app_Main()` entry points may be hierarchical: a complex application may have calls to multiple init or main functions within `app_Init()` or `app_Main()`.

After initialization, `app_Main()` represents a poll. Generally, housekeeping chores such as managing databases are done in this function.

```

/*****
// application.c
*****/
void app_Init(InitOpts *opts)
{
}

void app_Main(InitOpts *opts)
{
}

void interrupt_InstallIRQHandler(unsigned source, void
(*handler), unsigned data);

void interrupt_Enable(unsigned source);

void interrupt_Disable(unsigned source);

```

Figure 9: Processor entry points

The PicoNode protocols and most other applications for the Test Bed are highly event-driven. In an ideal world, the node is sleeping until some event occurs that requires servicing. Wake up in this architecture is generally caused by an interrupt of some kind. Therefore, the other primary entry point to the processor is via the standard interrupt mechanism, or IRQ. A programmer writes an ISR for a specific hardware interrupt source, for instance the FPGA, and registers the interrupt through a library call. This ISR is invoked when the interrupt condition specified by the programmer is met, usually a positive edge on a processor I/O pin. The design within the FPGA can generate several interrupt sources 'or'-ed into the single FPGA IRQ line. In this case, the ISR has the responsibility for detecting which interrupt has been generated.

IRQs are used for packet arrivals, timers, and other events asynchronous to the main thread.

4.2.2 FPGA Entry Point

The FPGA entry point is represented by the ARMIF block in the default FPGA workspace and pre-defined blocks in the FPGA libraries. The ARMIF block has several features:

- An abstraction of processor memory-mapped I/O that looks like a set of 15 32-bit write ports into the FPGA and an independent set of 15 32-bit read ports out of the FPGA.
- Port activity and handshake signals (`select[]`, `read`, `write`).
- FIQ and IRQ connections to the processor.
- Three 20MHz master clocks.
- Eight programmable reset lines.
- DVS power control interface

- A raw interface to the processor bus for applications that can't use the port abstraction (haven't found any yet).

Ports are accessed from the processor via the library function calls `xilinx_WritePort(n,m)` and `m = xilinx_ReadPort(n)` where `n` is a 32-bit integer port number and `m` is a 32-bit integer value. When writing, `m` appears on the port `n` bus as a big endian binary value. Write ports are latched so the value is held until the next write on the same port. When reading, the reverse occurs.

The FPGA initiates events in the processor via the FIQ and IRQ interrupt lines. Generally, an interrupt results in a series of `xilinx_ReadPort()` and `xilinx_WritePort()` calls as the processor queries state and performs the action indicated by the interrupt source.

4.3 The Libraries

Access to underlying hardware capabilities and software packages are included in the system libraries. There is a library each for the processor and the FPGA (figure 10).

Both libraries are extensively documented, with functional prototypes and usage instructions for processor APIs and data sheets for FPGA blocks.

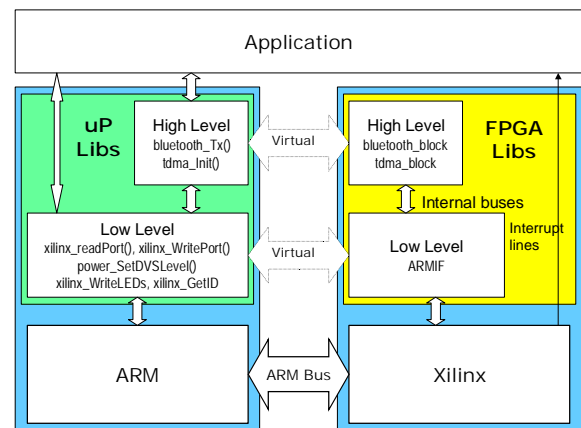


Figure 10: Diagram of service abstractions offered by the kernel.

4.3.1 Processor Libraries

The processor libraries provide easy access to underlying processor resources. Users interface with the libraries through function calls. These function calls represent varying levels of abstraction. For instance, say a designer wants to use the Bluetooth radio in an application. He or she would instantiate the pre-built Bluetooth controller block into the FPGA, connect nets and busses according to the data sheet for the block, and use the Bluetooth processor API to control the block. The Bluetooth API includes function calls for setting channels, controlling the transmitter, controlling the receiver, controlling the radio power supply etc. The Bluetooth API represents a relatively high level of abstraction for the processor library. The API employs low-level library calls such as `xilinx_WritePort()` to make the Bluetooth control block in the FPGA do the right thing.

The processor library includes:

APIs for ARM subsystem control

- ARM GPIO pins
- Interrupts
- Timers
- Power
- Real-time clock
- Reset
- Serial ports

FPGA control and data transport

- Basic I/O (xilinx_ReadPort(), ...)
- Proxim radio block control
- Bluetooth radio block control
- Pre-built datapath block control
- Pre-built TDMA MAC block control

General support

- Data structure packages (FIFO, linked list, queue, heap)
- Flash programming
- Debug
- Virtual channel abstraction

A user can choose the level of control needed for the application. Some examples:

To initialize an IRQ interrupt, use

```
interrupt_InstallIRQHandler(source, handler, data);
```

where source is one of the 32 IRQ interrupts, handler is the user-provided ISR to call when the interrupt occurs, and data is an arbitrary value passed as an argument to the ISR. To enable the interrupt, use

```
interrupt_Enable(source);
```

Similarly, to initialize a timer, use

```
ostimer_Start(timernum, timevalue, handler, data);
```

To stop the timer, use

```
ostimer_Stop(timernum);
```

To restart the timer (generally in the ISR if the timer is to be periodic) use

```
ostimer_Restart(timernum, value);
```

The ostimers generate interrupts on expiration, so ostimer functions call into the interrupt API in addition to setting up timer controller registers in the processor. The 32 interrupt sources referred to earlier come from ARM subsystems like timers, clocks, serial ports, and so on.

On the other hand, if a user wants fine grain control over the interrupt controller, a large set of macros is available that directly query or modify the controller registers, such as

```
interrupt_mask(source)
interrupt_is_pending(source)
interrupt_level_is_IRQ(source)
```

The ARM subsystem interfaces are comprehensive and exhaustive. There are over three hundred functions and macros in the ARM subsystem library alone.

4.3.2 FPGA Libraries

For the FPGA, a set of blocks are available that provide functions such as:

- Basic processor I/O (ARMIF)
- Commonly used functions, such as byte and word counters, signal shaping FSMs⁷, and odd-sized registers and multiplexers.
- Mappings for all I/O pins
- FIFOs based on internal Xilinx RAM
- Radio interfaces
- A parameterizable TDMA MAC
- Full-duplex Rx and Tx datapaths with IRQ support

The last four are complex and powerful subsystems. They can save a designer a substantial amount of effort when the design calls for the creation of standard but time-consuming sub-blocks. For instance, the time division multiple access (TDMA) block includes all timing, synchronization, transmit/receive logic, and slot management needed to create a network, all in a bolt-on module that works with both Test Bed radios. Instantiate the module into the FPGA workspace, hook it up, and go. In addition, frame and slot length are parameterizable from a processor API.

There are also several services provided that overlap both spaces. For instance, a ‘channels’ abstraction provides a byte-oriented FIFO interface at the application level for arbitrary radio channels (figure 11). How radio channels map to the channel abstraction is up to the data link layer designer. With the TDMA block, channels correspond to time slots.

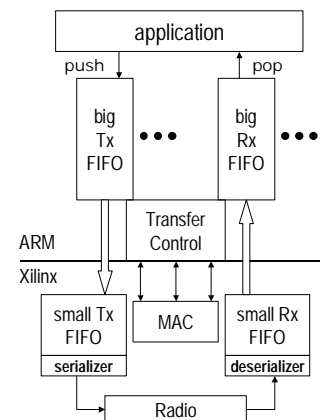


Figure 11: Diagram of channel service. Implemented by a collection of processor library calls and FPGA blocks.

⁷ Good for interfacing clock domains e.g. radio clock, FPGA master clock, processor bus cycle times, etc.

4.4 The Design Flow

The design flow for the Test Bed also adheres to the principle of abstracting detail from the user. Like the workspace, entry points and libraries, there is a distinct design flow each for the processor and the FPGA (figure 12).

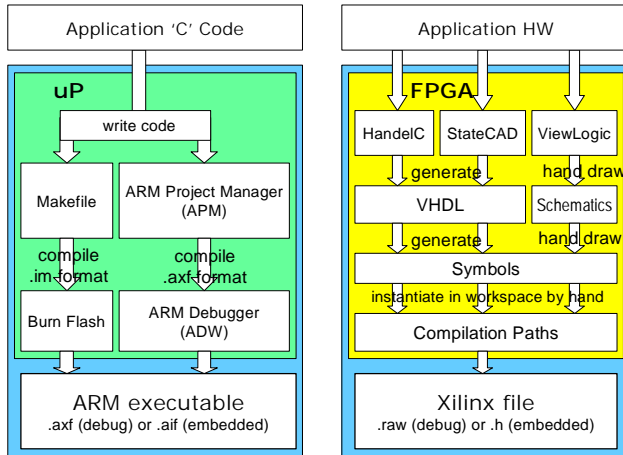


Figure 12: Processor and FPGA design flow.

4.4.1 Processor Design Flow

From the programmers viewpoint, building an executable follows the typical ‘C’ compile/link procedure. The compiler, assembler, and linker are part of the ARM Development Suite mentioned in 4.1.1.

There are three types of executables for a Test Bed application: the debug, standalone, and rom variants. Each is respectively closer to an image that will be burned into flash. Debug allows file I/O (printf, scanf) and loads the FPGA configuration file from a network partition. With standalone, a programmer can optionally turn off I/O and the FPGA file is stored in the executable as a character array. Both debug and standalone contain complete ‘C’ libraries. Rom uses a limited ‘C’ library and does not contain debug symbols. The FPGA file for rom is also stored internally.

Variants are determined by compile flags stored in the project templates, so as far as image building is concerned the differences are transparent to the programmer. A programmer must be aware of two issues: 1) debug file I/O must be protected by compile flags or the rom variant build will fail in the linking phase (no support in the rom clib), and 2) rom builds do not support dynamic memory allocation because the PicoRadio SOC won’t.

Most program development is done with the debug variant. Standalone is used primarily for testing the internally stored FPGA program. The build environment for these two variants is the ARM Project Manager (APM). They are loaded into the processor via an RS-232 serial connection between a laptop and the processor using a graphical debugger, and run-time control (breakpoints, etc.) is maintained by the laptop. The ARM debugger supports semi-hosting: a GUI runs on the PC side while a boot/debug agent is resident in the target flash.

The rom variant uses a batch make rather than the APM GUI. The programmer simply types ‘make’ in a shell (usually tcsh in the Cygwin environment). The file produced contain a fully

executable image that includes the boot monitor and the program executable.

A node boots from the monitor in flash and immediately copies the flash contents to DRAM, then transfers execution to DRAM. The flash is no longer used at run-time except as non-volatile storage for an application. To burn a new application into flash, the debug variant of a special purpose program is loaded into the application area in DRAM using the graphical debugger. The program then transfers the rom variant of the application into DRAM from a network file and programs the flash. The new application will run the next time the node is booted.

4.4.2 The FPGA Design Flow

The FPGA design flow includes compilation and synthesis paths for all methods discussed in 4.1.2. Once the schematic is entered and the HDL models are ready, the Xilinx configuration file is generated through a batch process using Cygwin make. The Makefile invokes tools specific for the design entry format e.g. the Handel-C compiler, FPGA Express for VHDL, or an EDIF generator for Viewdraw schematics. All of the top level compilation tools target EDIF; from there the Makefile invokes Xilinx ISE tools to generate an EXO format file. The EXO file is refined further into an image suitable for downloading to flash with tools written by BWRC staff and students.

5. APPLICATIONS

The term “application” in the context of the PicoRadio Test Bed means anything not part of the kernel. An application is typically some component of the PicoRadio system that a researcher wants to examine, such as a protocol, a radio baseband algorithm (e.g. error-correction coding), or system for collecting and logging environmental data such as temperature, using the communication facilities provided by protocols and baseband algorithms. Applications are designed, implemented, and debugged by researchers using the Test Bed workspace, entry points, libraries, and design flow described earlier.

Following are several example applications that have been implemented on the Test Bed or are currently in testing.

5.1 Local Positioning

Among a number of unique aspects of the PicoRadio network is the notion of location-based addressing. Rather than a fixed, global identifier such as an IP address each node uses its current location in physical space, represented by an XYZ coordinate triple relative to an arbitrary origin. Requests for sensor data, for instance, are directed to a cuboid in space bounded by a pair of XYZ coordinates, and nodes within that cuboid respond with the appropriate values. In sensor networks we are more interested in an area in space than a particular node. For instance, a user will most likely want the temperature in the north-west corner of a room rather than the reading from a specific sensor. In a dense PicoRadio network, multiple samples may be generated from a single request to that area. Differentiation is done on where in space the sample was taken, not which node returned it.

In addition, location must be dynamically determined by a node itself because network density prevents manually configured static locations. Also, node locations aren’t necessarily fixed, although it’s much easier to deal with nodes that don’t move. The most obvious solution to this problem is something like GPS, but most

PicoRadio deployments will be indoors and nodes may be embedded in walls or furniture. GPS will not operate sufficiently well (or at all) in this environment.

PicoRadio researchers are investigating methods of node locationing using information gathered from the network itself. Possible candidates are triangulation from distance measurements based on received signal strength indication (RSSI) and distance measurements based on hop counts. In the former, due to the nature of a radio environment RSSI range errors are typically very large (around 50%), so a great deal of redundancy is required to obtain accuracy. Hop count based algorithms suffer from the same difficulty, but do not require any support below the network level; this makes them independent of radio technology. The PicoRadio network provides the density needed in both cases.

In both methods, a small subset of nodes in a network have fixed, predetermined locations. These nodes, called *anchors*, transmit beacons that are forwarded by all nodes that hear them. Nodes that are not anchors determine their position by triangulating on anchors or nodes that have a good idea of their location from previous computations. One challenge of these algorithms is to ensure convergence to a stable state.

Two experiments were performed with an algorithm based on least-squares triangulation using RSSI. In the first experiment, one “target” node attempted to locate itself based on information from four anchors. The anchors were pre-programmed with fixed XYZ coordinates. Results from this experiment were interesting but less than spectacular. The target node could reliably detect movement, but its idea of absolute position was generally poor. In the second experiment, the number of anchor nodes was increased to eight. The results were significantly better than the first experiment, confirming that redundancy is required for this algorithm using RSSI as a distance metric

The hop-based algorithm (Hop-Terrain) is currently implemented in the Test Bed. This algorithm is expected to be ported to the single-chip PicoNode.

[1][9][10]

5.2 PicoRadio Protocols

At the network level PicoRadio SOC nodes will communicate with each other in an ad-hoc manner with no central point of contention or failure. This behavior is determined by the network and media access control (MAC) protocols.

PicoRadio uses request-response semantics for messages. A requesting node sends a broadcast request directed toward a cuboid, and the request is forwarded in that direction by other nodes along the way. As the request propagates outward through forwarding, each node collects data on the most immediate source of that request (a neighbor). Since request propagation depends on directed broadcast forwarding, a node may see multiple requests from the same source arriving on different paths (from different neighbors). Included in this data is an energy metric and the location of the node that originally requested information. The node records the energy, neighbor ID, and the source address; when responses are returned the network protocol knows which return paths are available, and chooses a path based on energy and a probability algorithm that ensures no path is overused. This routing works only because all nodes know their relative position in the space.

Nodes that are in direct radio contact with each other are said to be in the same *neighborhood*. In a dense network, neighborhoods intersect, and if conditions are dynamic (typical for wireless) membership in a neighborhood can change over time. Essentially, membership in a neighborhood implies that there is “acceptable” communication between all nodes in the neighborhood. What is acceptable is determined by the deployment and the type of higher-level message passing (e.g. sensor data). Each node in a neighborhood is dynamically assigned an integer ID which must be unique in any neighborhood in which the node resides.

The network protocol is responsible for global end-to-end routing, that is, from requestor to sensor node and vice-versa. The MAC protocol is responsible for transmitting packets between neighbors. A network path can be many ‘hops’ long while a MAC path is always between two directly connected nodes.

Because the MAC protocol is responsible for low-level tasks such as gaining access to the wireless media and bitwise transmission and recovery of packets, timing resolutions are relatively fine and real-time behavior is mandatory. On the other hand, the network protocol generally has wide latitude with respect to timing i.e. environmental sensor data is relatively low duty cycle, and latency is not a big concern. For these reasons, most of the MAC protocol is implemented in the Test Bed FPGA while all of the network protocol is implemented in the processor.

Figure 13 shows the advantages of packet forwarding. Percent success is the ratio of times a packet reached a particular receiver. The deep fade in the dashed line indicates a null in the radio signal. The fade appears to go away (solid line) when a third node is added that is in a more advantageous location and can forward the packet to the receiver.

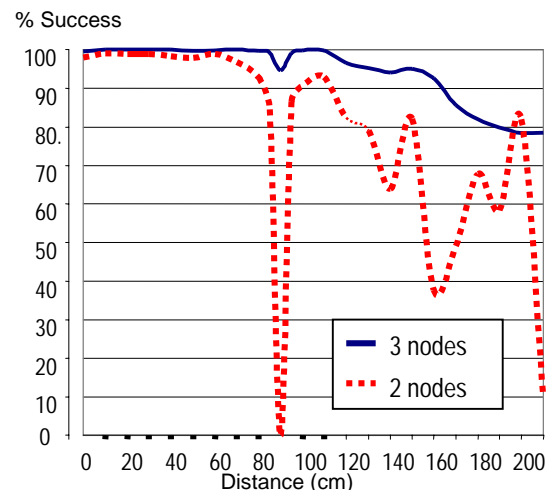


Figure 13: Effect of forwarding.

The protocols have been under evaluation on the Test Bed since early 2002. Initially, the most rudimentary aspects such as broadcast forwarding were implemented, followed by increasingly sophisticated mechanisms such as energy-based routing. The Test Bed has enabled us to understand how the PicoRadio network behaves in a real deployment. Important lessons have been learned in this process. Currently, the protocols are fully realized on the Test Bed, and the design is being ported to the PicoNode 3 architecture.

[11][12][13]



Figure 14: Reto taking measurements on PicoRadio network performance

5.3 Sensor Measurements using TDMA

In general, a network provides transport for some content of interest. For instance, the Internet TCP/IP protocols allow two computers to trade web pages with only the knowledge of a counterparts IP address. The location of the computer and the number of hops from source to destination is irrelevant. Similarly, in PicoRadio, requests for sensor values and sensor readings are transferred between two nodes that can be many hops away. The entity that requests the information (person or controller) and the sensor node that generates it do not care about network routing as long as the data get to the right place.

One of the great advantages of the Test Bed is the ability to test features of PicoRadio before underlying dependencies are available. Decoupling of request and response generation and the way in which they are transferred from requestor to responder allows us to use network/MAC protocols other than the one described above in testing of the higher layers. While the PicoRadio network and MAC protocols were in early development, we deployed a sensor network based on a TDMA MAC protocol, with no network protocol. A TDMA network uses a star topology with a basestation at the center and remote nodes at the periphery. Time is divided into chunks, and each remote node gets a guaranteed chunk of the radio bandwidth that repeats at periodic intervals. The basestation provides global timing synchronization for the remotes, and multi-hop is not practical. This network is decidedly un-PicoRadio. It is not scalable, it is not low-power, and it uses a highly critical centralized resource. However, it does provide node-to-node connectivity for requestors and responders. In addition, because the Test Bed is a general-purpose platform, TDMA is relatively easy to implement, and when the PicoRadio protocols were ready it was a fairly simple matter of swapping out one protocol for another.

Figure 15 shows measurements taken over a nine hour period for temperature, humidity, and light intensity. Each line represents a data source. Note the periodic nature of temperature and humidity readings due to HVAC⁸ cycling. The large spike in the bottom graph come from someone opening the window blinds for a hour or so.

⁸ Heating, Ventilating, and Air Conditioning

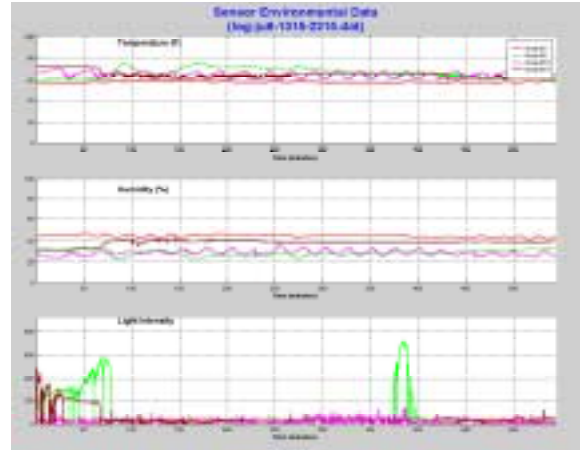


Figure 15: Sensor measurements from experiments at BWRC

5.4 Acoustic Anemometer

Sensor board #1 was originally designed with a specific experiment in mind. Air flow through a space can be detected by variations in an audio signal passing through the space. The sensor board contains a speaker for producing tones and a microphone for detecting the tones. Signal processing on the received tones can be done in the FPGA and processor to measure the rate of flow of air along the axis connecting the nodes.

The Test Bed was used for a series of experiments in acoustic anemometry. Results were published in a masters thesis.[2]

5.5 Removing Redundancy Sensor Values

For PicoRadio networks, sensor values have inherent redundancy because, in a dense sensor network, the readings of nearby sensors will be highly correlated. It would be beneficial if some readings could be compressed, combined, or disregarded.

Say X and Y are correlated sensors. Wyner and Ziv stated in 1978 that even without knowing the exact value of X, we only need to know the correlation between X and Y to determine Y. We can express Y as a scaled and noisy version of X. In the past three years codes have been found by Pradhan and Ramchandran that utilize this theory.

Simulations of a correlation tracking algorithm derived from the codes were run on light, temperature, and humidity data collected by Test Bed nodes [4]. Figure 16 shows time on the horizontal axis and correlation noise on the vertical axis. The lower values represent the actual correlation noise between two sensors. The upper line represents the correlation noise that could have been tolerated by the codebook in use at the time. As long as the correlation limit is above the actual values there is no decoding error; however, if the correlation limit line is too far above the data, it means that the encoder is too conservative i.e. it is sending more bits than absolutely necessary.

The simulations consisted of a 100,000 samples from each sensor, and we used the Chebyshev bound to calculate how many bits need to be sent each time in order to guarantee that $P_e < 1/1000$. The Chebyshev bound is not very tight, so there were no decoding errors and sometimes more bits were sent than was necessary.

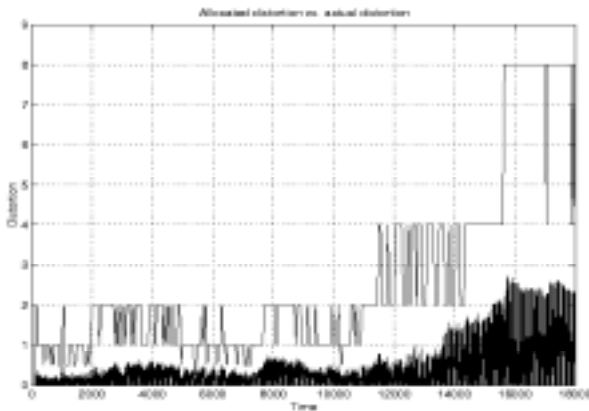


Figure 16: Correlation noise vs. time

6. CONCLUSIONS

The PicoRadio Test Bed has been and will continue to be a central resource for researchers with the PicoRadio project. It is currently fully deployed and highly utilized on a daily basis by students at the BWRC. Experience gained using the Test Bed is being directly applied to the PicoRadio SOC development; a first generation chip is due out in two months.

In order to replicate network density in the Test Bed, fifty nodes have been built and most are currently in operation for various experiments throughout the BWRC. Of the fifty, only two are currently unusable due to hardware problems. Four are in use at the Technical University of Berlin.

The Test Bed represents a major effort by a staff, undergraduate students, graduate students, and faculty at BWRC over a three year period. As a general-purpose emulation platform, it is expected to be useful for exploring wireless networking beyond PicoRadio. It has been in use in varying forms for over two years, and has continued to mature over that time.

All design goals discussed in this paper have been met or exceeded.

7. ACKNOWLEDGMENTS

This work was funded under the DARPA PAC/C program, grant #F29601-99-1-0169.

The following people were instrumental in making the Test Bed a useful system, or putting it to good use:

Ed Arens, faculty, Center for the Built Environment, ARCH
 DeLynn Bettencourt, instrumentation, summer intern, CSU Fresno
 Tiffany Crawford, network modeling, summer intern, Howard Univ
 Mark Feng, user interfaces and testing, undergrad, EECS
Dietrich Ho, sensor board #1, undergrad, EECS
Jason Hu, sensor board #2, undergrad, EECS
 Tufan Karalar, locationing & anemometer, graduate student, EECS
 Gary Kelson, executive director, BWRC
 Mike Montero, case design, graduate student, ME

Dan Odell, case design, graduate student, ME

Dan Petkovsek, system profiling, summer intern, Univ of Maryland
 Dragan Petrovic, data analysis, graduate student, EECS
 Muhua Pang, radio interface, summer intern, Univ of Rochester
Johnathan Reason, protocol implementation, post-doc, EECS
 Brian Richards, technical staff, BWRC
 Chris Savarese, locationing, graduate student, EECS
 Rahul Shah, network protocols, graduate student, EECS
 Mike Sheets, packet formats, graduate student, EECS
Hugo Shi, protocol implementation, undergrad, EECS
 Reto Stutz, test & measurement, visiting scholar, EPFL
Howard Tsai, system development, undergrad, EECS
 Ruth Wang, protocol implementation, undergrad, EECS
 Paul Wright, faculty, ME

Bold – special recognition for key support and massive effort.

8. REFERENCES

- [1] Beutel, J. "Geolocation in a Pico Radio Environment". Masters thesis, University of California at Berkeley. 2000.
- [2] Karalar, T. "An Acoustic Digital Anemometer," Masters Thesis, University of California at Berkeley. 2002.
- [3] Odell, D., Wright, P. "Concurrent Product Design: A Case Study on the Pico Radio Test Bed", Masters Thesis, University of California at Berkeley.
- [4] Petrovic, D. "Removing Redundancy from Wireless Sensor Networks," Presentation, BWRC Summer Retreat 2002
- [5] Rabaey, J., "Wireless Beyond the Third Generation — Facing the Energy Challenge," ISLPED 01.
- [6] Rabaey, J., Arens, E., Federspiel, C., Gadgil, A., Messerschmitt, D., Nazaroff, W., Pister, K., Oren, S., Varaiya, P. "Smart Energy Distribution and Consumption: Information Technology as an Enabling Force," Center for Information Technology Research in the Interest of Society (CITRIS). White paper.
- [7] Rabaey, J., Ammer, J., Karalar, T., Li, S., Otis, B., Sheets, M., Tuan, T. "PicoRadios for Wireless Sensor Networks: The Next Challenge in Ultra-Low-Power Design," Proceedings of the International Solid-State Circuits Conference, San Francisco, CA, February 3-7, 2002.
- [8] Rabaey, J., Ammer, J., da Silva, J., Patel, D. "PicoRadio: Ad-hoc Wireless Networking of Ubiquitous Low-Energy Sensor/Monitor Nodes," WVLSI April 2000.
- [9] Savarese, C. "Robust Positioning Algorithms for Distributed Ad Hoc Wireless Sensor Networks," Masters thesis, University of California at Berkeley. 2002.
- [10] Savarese, S., Rabaey, J., Beutel, J. "Locationing in Distributed Ad-Hoc Wireless Sensor Networks," ICASSP 2001.
- [11] Shah, R., Rabaey, J. "Energy Aware Routing for Low Energy Ad Hoc Sensor Networks", IEEE Wireless Communications and Networking Conference (WCNC), March 17-21, 2002, Orlando, FL.
- [12] Zhong, L., Rabaey, J., Guo, C., Shah, R., "Data Link Layer Design for Wireless Sensor Networks," Proceedings of IEEE MILCOM 2001, Washington D.C., October 28-31, 2001.
- [13] Zhong, L., Shah, R., Guo, C., Rabaey, J. "An Ultra-Low Power and Distributed Access Protocol for Broadband Wireless Sensor Networks," IEEE Broadband Wireless Summit, Las Vegas, N.V., May 2000.