

# Untethered On-The-Fly Radio Assembly With Wires-On-Demand

Jorge Surís, Matthew Shelburne, Cameron Patterson,  
Peter Athanas, John Bowen, Timothy Dunham, Justin Rice  
Virginia Tech

Configurable Computing Lab

Bradley Department of Electrical and Computer Engineering  
Blacksburg, Virginia 24061

Email: {jasuris,mshelbur,cdp,athanas,jobowen2,tdunham,ricej}@vt.edu

**Abstract**—In systems typified by software defined radio, existing flows for run-time FPGA reconfiguration limit resource efficiency when constructing datapaths. We present the Wires-on-Demand framework that allocates a *sandbox* region in which modules from a library are flexibly placed and interconnected rapidly and autonomously in an embedded platform without vendor tools.

## I. INTRODUCTION

With the introduction of Software Defined Radios (SDR) [1] platforms capable of supporting a wide variety of demodulation schemes became possible. Flexibility is achieved by transferring functionality from fixed hardware to software routines. At run-time, the behavior of the system can be altered by changing the software routine that is called. Replacing application specific hardware, which is inherently parallel, with the General Purpose Processors (GPP) required to execute the software routines results in an increase in system requirements. For example, an FIR filter with 51 taps requires 101 arithmetic operations (51 multiplies and 50 additions) to process one sample. The high number of operations leads to required GPP operating frequencies greater than 100 times the sampling frequency, assuming that the GPP is only doing the filtering. Further signal processing would require even higher operating frequencies or the use of multiple GPPs. With the availability of 3.2GHz processors, such high operating frequencies might not seem to be a problem, but embedded systems rarely contain such powerful processors.

To reduce the burden on GPPs many SDR platform designers have chosen to add Field Programmable Gate Arrays (FPGA) to their systems. The reconfigurability offered by FPGAs, along with their parallel nature makes them a suitable co-processor for SDR platforms. High-end GPPs can be replaced with a low-end GPP, such as an ARM, and an FPGA. Instead of software routines, hardware modules can be used for high data-rate operations such as filtering, down-converting, timing recovery and synchronization amongst others. The GPP is then freed-up to handle more complex lower data-rate operations such as MAC layer protocols, encryption and system administration.

FPGAs however do not provide the level of agility provided by software routines because the FPGA's configuration

bitstreams are developed at design-time and new combinations of existing functionality are not easily created at run-time. For example, if the expected waveforms are QPSK at frequency  $f_1$  and BPSK at  $f_2$ , then two configuration bitstreams are created. One with a filter centered around  $f_1$  and a QPSK demodulator and one with a filter centered around  $f_2$  and a BPSK demodulator. If the need arose for a BPSK waveform at  $f_1$ , an all-software-based radio can create the new waveform by using the filtering routine of the first waveform and the demodulation routine of the second waveform. Accomplishing the same task on an FPGA is not as trivial, because the module combinations in each configuration bitstream are defined at design-time and mainstream tools do not allow the modification of the configuration at run-time.

In [2], Xilinx introduced a partial reconfiguration tool flow that allows the reconfiguration of part of the FPGA. Their flow allows the designer to separate the static and dynamic portions of the design by pre-allocating slots where the dynamic modules will reside. Slots are areas in the reconfigurable fabric reserved for the exclusive use of dynamic modules. All communication to and from a dynamic module must be routed through the provided bus macros that serve as the bridge between regions. By providing the ability to reconfigure each slot separately, the agility of the implemented design is increased. The problems evident in the previous example can be addressed by having two dynamic slots in the base design. One slot can be used for the bandpass filter and another can be used for the demodulator. This gives the FPGA software-like flexibility, allowing any filter/demodulator combination.

Although a more flexible environment, the Partial Reconfiguration (PR) flow does have its limitations. Slots for dynamic module placement are allocated at design time, making the number of modules that can be instantiated at the same time a compile-time constant. For example, if a de-scrambler is needed for the new waveform from the previous example, it would not be possible to add it since there are only two slots in the system. Another limitation is that the communication scheme is static, requiring all modules sharing a slot to have the same ports. Finally, because of its complexity and lack of support (it is not a mainstream tool) the PR flow has a high learning curve.

In [3], a run-time reconfiguration framework, called Wires-on-Demand (WoD), was presented. WoD expands on Xilinx's PR flow to address some of its limitations. In this paper present the implementation of WoD on an embedded SDR platform to build a radio receiver.

## II. ON-THE-FLY ASSEMBLY

Recognizing the performance benefits of FPGAs, many vendor and academic Cognitive Radio (CR) endeavors have included them into their SDR development platforms [4]–[6]; however due to the lack of a light-weight dynamic reconfiguration framework, CR platforms have used FPGAs in a somewhat static manner using a few pre-built bitstreams. In the USRP platform presented in [4], the FPGA is primarily used for up/down converting the signal. Most of the signal processing is either done prior to the FPGA in a daughter-board or in the GPP.

For FPGAs to become an integral component of SDR platforms, they must be capable of changing their behavior at the same granularity level of software. This implies that for every functional software block to be replaced an equivalent hardware block must exist. For example, if a software receiver is composed of three functions (filtering, demodulation and descrambler), it is not sufficient to replace it with a monolithic hardware block that accomplishes all three. Using one large block would not allow the re-use of one of its components, such as the filter in other implementations.

Because the length of a demodulation chain can change over time, the number of functional blocks used must also be a run-time variable. Although a variable length chain is easily accomplished in software, accomplishing it in hardware is much more difficult. Using Xilinx's PR flow, several slots for dynamic placement of blocks could be allocated to account for an expanding chain length. Unfortunately in the PR flow, the number of slots is a design time constant and cannot be changed at run-time to account for changing needs.

To obtain the required flexibility required for SDRs, a design flow capable of creating new configuration bitstreams at run-time is needed. If the SDR platform is connected to a host PC, regular vendor tools can be used to generate the new configurations as needed, but this is an unlikely scenario as many SDR systems are embedded platforms. A light-weight framework capable of modifying bitstreams at run-time is therefore needed to supply a flexibility similar to that of software.

## III. WIRES-ON-DEMAND

Wires-on-Demand presents a reconfiguration framework that allows on-the-fly assembly of IP blocks to create a new configuration. This allows systems to adapt to unforeseen circumstances. Because no vendor tools are used by the framework at run-time, it can be run on any embedded platform with support for C++. At run-time, a server accepts requests to modify the active system and returns a partial bitstream with the changes. Because modules are placed and connected at run-time, the length of the receiver chain is only limited by

the size of the dynamic region allocated at design-time and the size of the modules.

### A. WoD: Design-Time

To implement the WoD framework, two tasks must be completed at design-time. First, the FPGA must be partitioned into two areas a dynamic region, called a *sandbox*, and a static region. This is done using Xilinx's partial reconfiguration flow, where the sandbox is a large reconfigurable region. Bus-macros are used to connect the sandbox with the static logic. Because modules can only access resources lying inside the sandbox, it must encompass any resources such as DSPs and BRAMs that may be required by the modules.

Second, IP blocks used by the server are prepared by an automated tool using Xilinx's partial reconfiguration flow. The design-time flow for preparing modules is shown in Figure 1. The tool automatically inserts bus macros to tie ports to known locations and creates the files required to generate the module. The PR flow is then automatically invoked to create a partial bitstream containing the module. A description file for the module is created which specifies the locations of ports and special columns as well as device specific information. Figure 2 shows a module which has been processed by the design-time tools. Notice the row of bus macros at the top and bottom of the module marked by the blue squares. Also notice that all module routing and logic is contained within a specific area.

### B. WoD: Run-Time

At run-time a program called the dynamic module server is used to make changes to the current bitstream. Figure 3 shows the flow of control through server when a change request is issued. When a change to the system is requested the server determines if modules need to be added or removed from the datapath. If a module is to be added, the module library is searched for a version of the requested module that has been created for the client device as a single server can host multiple clients. Modules are stored in the library as partial bitstreams along with an description file detailing the module requirements and properties. The new module is passed to the placer, which determines the correct placement and defines routing channels to connect the modules. When placing modules any use of special resources, such as DSP blocks, is taken into consideration. An FIR filter that uses a DSP block, for example, cannot be placed in an area that does not contain one. The placer will try to place the new module without affecting surrounding modules moving them only if the module being placed does not fit. The placer uses the module port location and type information contained within the module's XML description to determine how the new module should be connected to the adjacent modules. Routing between modules is accomplished through rectangular regions between placed modules, essentially forming routing channels. A list of new routing channels is created, each containing a list of new signals to be routed. A dynamic router then generates the bitstream changes required to implement the connections

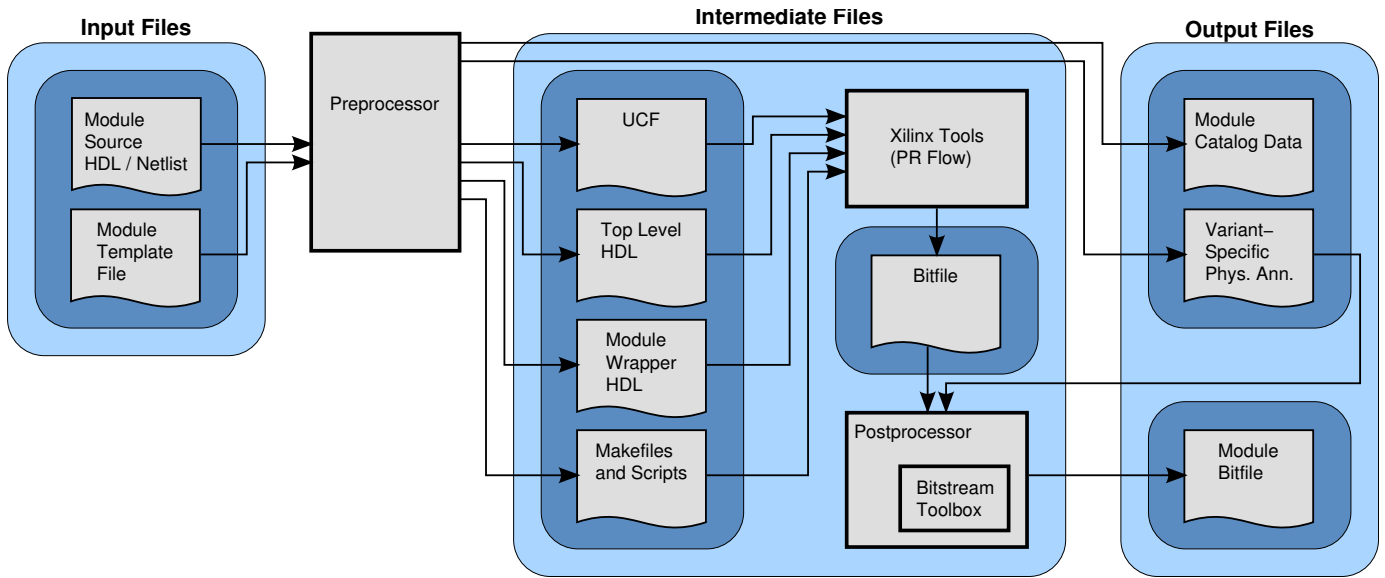


Fig. 1. Design-time module creation

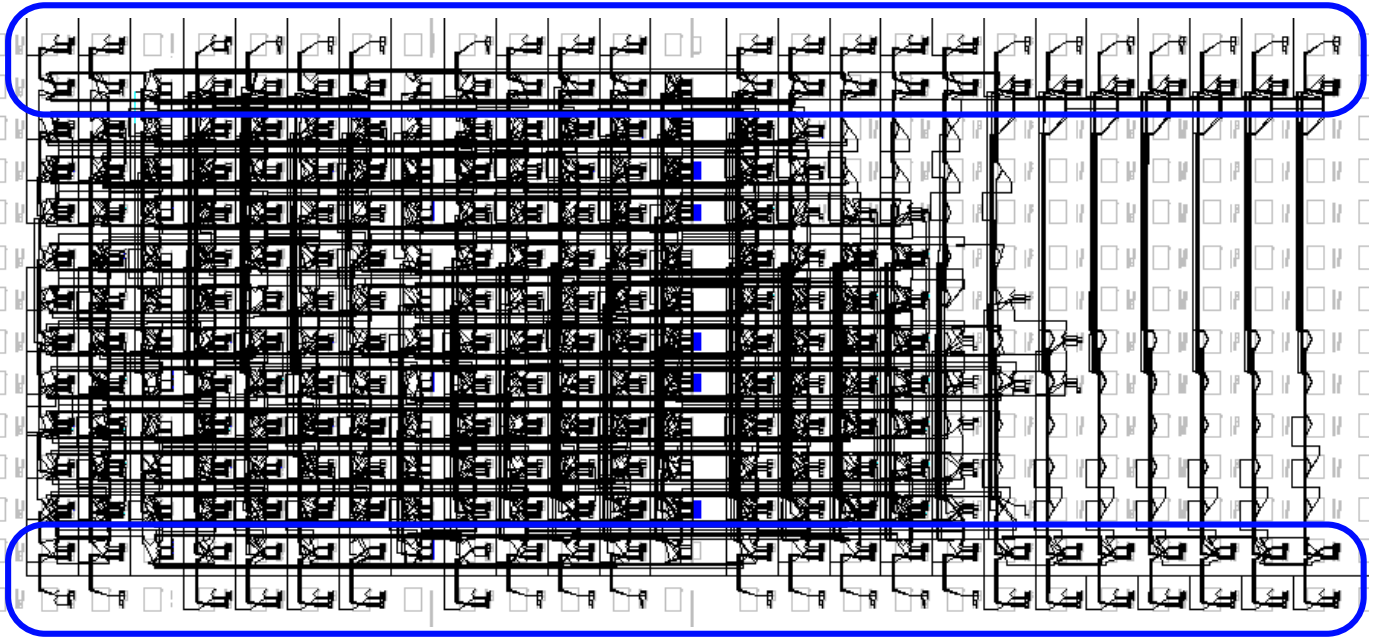


Fig. 2. Pre-processed module with bus macros

identified by the placer. A simple channel routing algorithm is used to avoid large data structures and to obtain deterministic execution times. Memory requirements are further reduced by utilizing a custom wiring database that has been compiled into the router. Once all changes (new/deleted modules, module relocations and new routing channels) have been identified a partial bitstream is generated and provided to the client application.

#### IV. SAMPLE APPLICATION

A simple envelope detection AM receiver was chosen as the sample application. Figure 4 shows the radio topology.

The light gray area designates the dynamic region which approximately covers 50% of the FPGA. Note that all processing is done by dynamically placed modules. The static logic in the sample application consists of: a bus interface to communicate with the host processor, the Internal Configuration Access Port (ICAP) to reprogram the FPGA, and an ADC interface block that obtains the input data. The SDR platform used has an ARM processor with 256MB of memory running Linux on which the server is run. The ARM is connected to a Virtex 4 XC4VLX60 FPGA operating at 96 MHz through an EMIF bus. Generated bitstreams are provided to the FPGA ICAP via the EMIF bus interface.

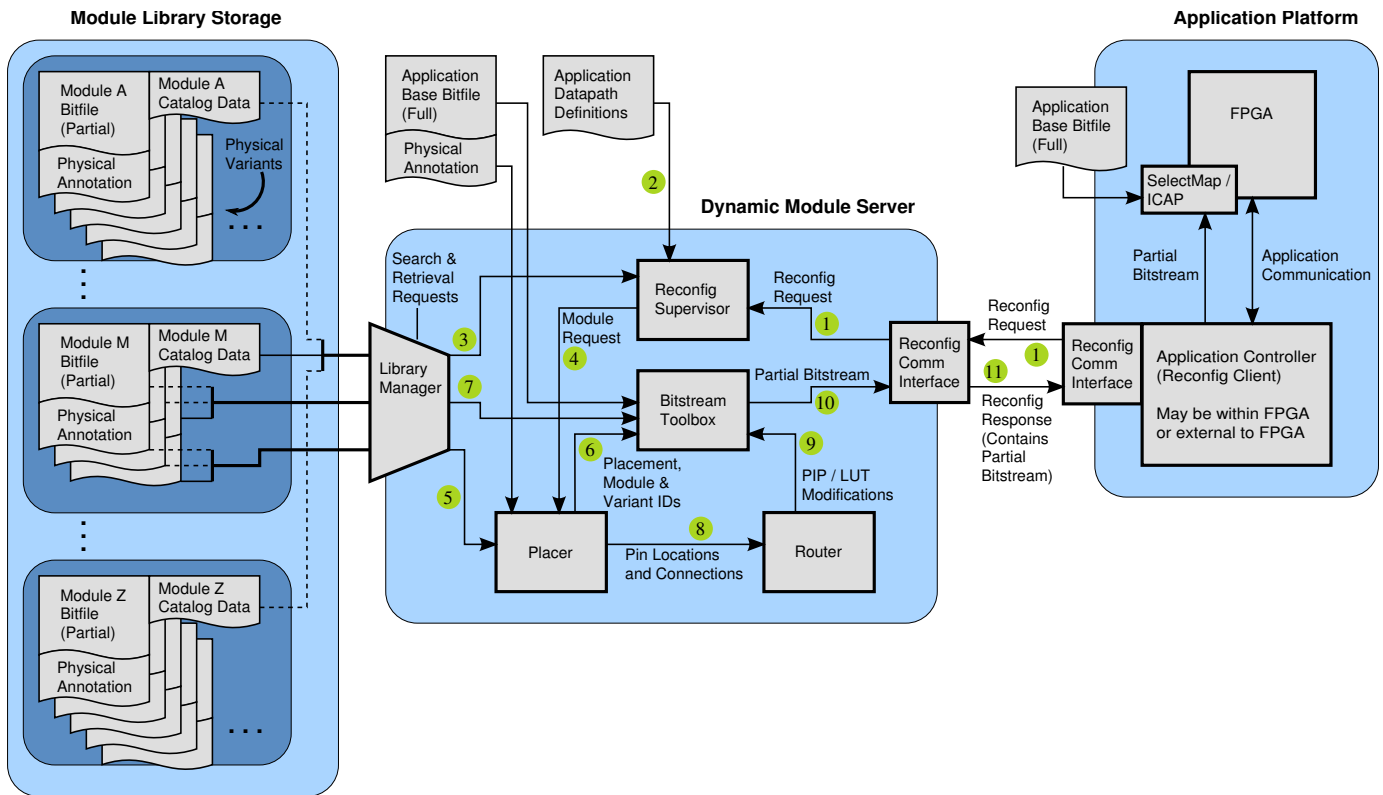


Fig. 3. On-the-Fly system construction

The demonstration radio is structured to demodulate only one station; however, the architecture is dynamically updated, giving it the ability to select different radio stations. For the fixed-tuner AM radio, nine dynamic modules were generated: six FIR filters, an NCO, a multiplier (mixer) and an absolute value core. Because very little filtering is done in the front-end analog stage, frequency selection must be done through the use of a high-pass or low-pass filter as the second module. This allows mixing a station down to the intermediate frequency without interference from other stations. If the desired radio station is outside the range of the current in use filter, it is removed by the server and replaced with a new filter module. Note that such a system could be implemented within the regular PR flow by using one slot for this second module, but that would result in a very limited system. Only an envelope detector would be possible. Because the number of modules in the system is not restricted, the WoD framework allows the server to change the implementation to better quality AM receiver such as a quadrature detector or something totally different such as a BPSK receiver.

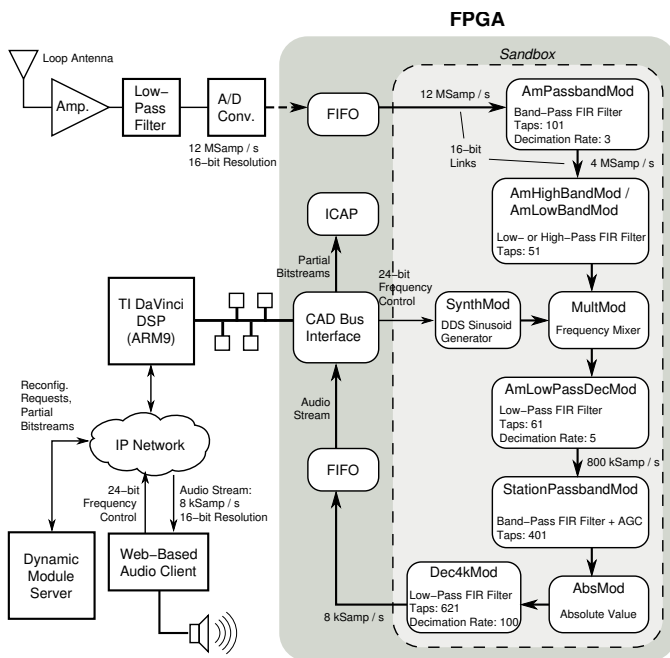


Fig. 4. Module placement

For rapid development module add and remove requests are made through a web-based interface hosted on the ARM web-server. Figure 5 shows a graphical representation of the FPGA with the generated AM radio. Blue areas represent the dynamically placed modules and pink areas represent dynamic routing channels.

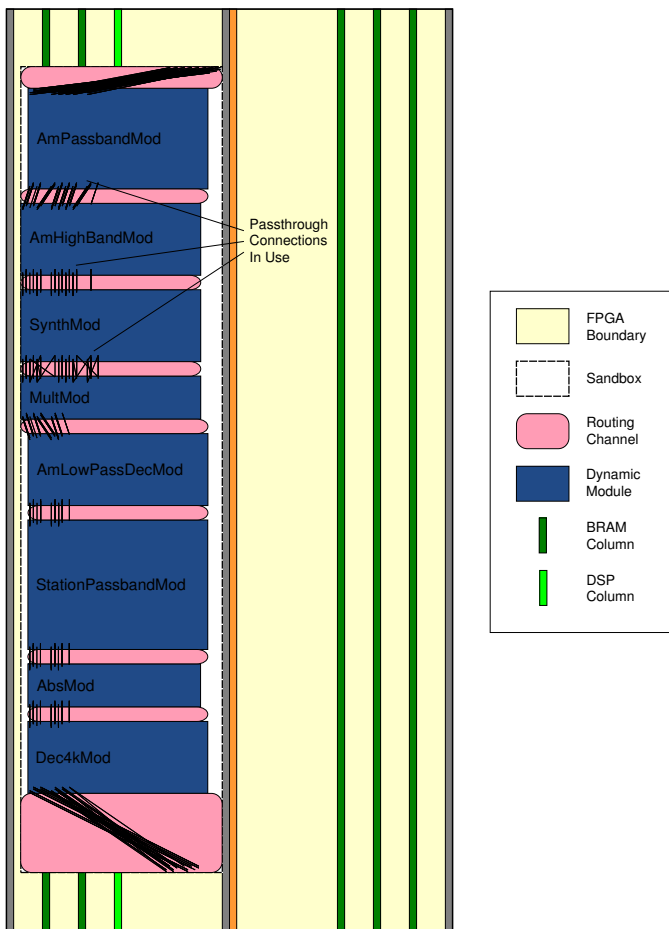


Fig. 5. Module placement

## V. CONCLUSIONS

The high resource requirements of digital signal processing have led to the inclusion of FPGAs in many embedded software defined radio platforms. The complexity of designing for FPGAs and the lack of a flexible reconfiguration framework however have limited the use of their reconfigurable nature leaving dynamic functionality to be executed in software.

In this paper we present the application of the Wires-on-Demand reconfiguration framework to the SDRs. WoD constructs the configuration bitstream of the FPGA at run-time providing the user with fine grain control over its behavior. The increased flexibility provided by WoD allows SDRs and Cognitive Radio applications to migrate dynamic functionality to the FPGA. Reduced functionality executing on the GPP and the modest memory and computational requirements of WoD allows the use of smaller more efficient GPPs on SDR platforms.

Using a simple AM radio implementation we demonstrated the functionality of the WoD framework and how it can be applied to more complex SDR applications. Hardware modules could be added and removed from the system. New routing channels were constructed to connect the dynamically placed modules, and a valid configuration bitstream was produced

at run-time without the use of vendor tools. The framework was run on an embedded platform with limited memory demonstrating its modest system requirements.

Configurable radios have traditionally been the domain of DSPs because of the flexibility of software, but the level of flexibility and reduced design complexity introduced by WoD allows FPGAs to compete with DSP platforms in the Software-Defined Radio realm.

## VI. ACKNOWLEDGEMENTS

This work is supported by the United States Air Force under Contract Number FA8651-06-C-0126. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force.

## REFERENCES

- [1] J. Mitola, "The software radio architecture," *Communications Magazine, IEEE*, vol. 33, no. 5, pp. 26–38, 1995.
- [2] Xilinx Inc., *XAPP290: Two flows for partial reconfiguration: Module based or difference based*, September 2004.
- [3] P. Athanas, J. Bowen, T. Dunham, C. Patterson, J. Rice, M. Shelburne, J. Surís, M. Bucciero, and J. Graf, "Wires on Demand: Run-time communication synthesis for reconfigurable computing," in *FPL 2007: Int. Conf. on Field Programmable Logic and Applications*, pp. 513–516.
- [4] M. Ettus, *Universal Software Radio Peripheral*, Ettus Research, Mountain View, CA.
- [5] L. Belanger, "Advanced SDR platform eases multiprotocol radio development," *RF Design Magazine*, pp. 36,38,40,42, Jan. 2006.
- [6] W. Zhu, B. Daneshrad, J. Bhatia, J. Chen, H.-S. Kim, K. Mohammed, O. Nasr, S. Sasi, A. Shah, and M. Tsai, "A real time MIMO OFDM testbed for cognitive radio & networking research," in *WiNTECH '06: Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization*. New York, NY, USA: ACM Press, 2006, pp. 115–116.
- [7] R. Lysecky, F. Vahid, and S. X.-D. Tan, "Dynamic FPGA routing for just-in-time FPGA compilation," in *DAC '04: Proceedings of the 41st Annual Conference on Design Automation*. New York, NY, USA: ACM Press, 2004, pp. 954–959.
- [8] S. A. Guccione and D. Levi, "XBI: A Java-based interface to FPGA hardware," in *Configurable Computing: Technology and Applications, Proc. SPIE 3526*.
- [9] A. Megacz, "A library and platform for FPGA bitstream manipulation," 23-25 April 2007, pp. 45–54.
- [10] M. Hübner, C. Schuck, and J. Becker, "Elementary block based 2-dimensional dynamic and partial reconfiguration for Virtex-II FPGAs," in *IPDPS. IEEE*, 2006.