Extended Abstract: A Toolchain for Rapid Prototyping of Underwater Communication Systems

Gregory Parsons, Shaolin Peng, Alexander G. Dean

gsparsons@nc.rr.com, speng2@ncsu.edu, alex_dean@ncsu.edu

Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27695-7256

1. INTRODUCTION

This paper introduces the RaPTEX toolchain and its use for the rapid prototyping and evaluation of an underwater ultrasonic communication system. The main goal of our RaPTEX toolchain is to bridge this gap between the need for embedded communication systems on low-cost low-energy hardware and the difficulty in their design. To this end, we have developed new methods and implemented them in a tool enabling rapid prototyping of communication protocols for embedded systems. The tool offers a collection of commonly used communication protocol components. A graphical user interface allows users to quickly build a system by selecting, configuring and linking components. From the user’s design, the toolchain will automatically generate code, compile, link and analyze it, providing performance and resource requirement predictions to the user. The modular interface of the tool simplifies the creation of new, custom protocols, which is useful if an uncommon protocol is needed, or if the tool is used by networking researchers testing their yet unpublished communication protocols.

A critical feature of the toolchain is the ability to estimate the performance of the selected communication implementation, providing instant feedback to the user. This enables informed choices and a more effective exploration of the offered trade-off space. Furthermore, the tool can realistically simulate an embedded system using a cycle-accurate functional simulator for each processor of the system. This enables exact measurements of relevant performance parameters such as energy consumption and resource requirements such as memory usage, while facilitating the debugging of these notoriously difficult-to-debug distributed embedded systems. However, we do not discuss the simulator further in this current work.

[1] shows a platform of software and hardware for prototyping sensor network applications, however, it requires a Bluetooth-based sensor node hardware which is not commonly used in underwater communication systems. [2] presents a tool called Que, to assist sensor network deployments. It supplies help from the prototyping stage to early production stage by addressing the issues of scale, longevity, data and integration. However, neither of the above tools provides information about the estimation of the timing, energy and memory cost of the prototyped systems.

We target the underwater acoustic channel, which presents formidable challenges but is often the only feasible medium [3]. Choosing the right protocol, or a combination thereof, can be nontrivial, often necessitating extensive experimentation. To this end, we have implemented an ultrasonic communication system for biotelemetry in extremely shallow waters [4]. It offers a collection of simple communication protocol components. In this paper we use this system as the experimental platform to demonstrate and evaluate our toolchain.

2. RAPTEX SOFTWARE TOOLCHAIN

RaPTEX offers the user the power of iterative design: for given hardware choices and protocol configurations the performance of the system will be estimated automatically. Hence, the user will be able to adjust the hardware parameters and protocol choices until the desired performance is achieved. Alternatively, if the system is over-designed, the user will be able to choose less expensive hardware in terms of money, size or power that will still satisfy the targeted performance.

The RaPTEX software toolchain consists of a closed loop of four stages (as shown in Figure 1), enabling the user to get quick feedback on on-going system customization, utilizing iterative design techniques. The toolchain targets the 8-bit Atmel AVR architecture. The first stage in the closed loop process is a library of configurable software building blocks representing various potential system capabilities. Once the target application has been decomposed into its component software blocks, each needs to be abstracted for easy handling by the toolchain into what are called XML Description Libraries (XMDL). An XMDL is written in eXtensible Markup Language (XML) for both ease of use and resiliency. The format consists of five sections, each describing an aspect of the software module in question, such as inputs, outputs, user configurability, energy modelling, and any linking behavior with other modules.

The second stage in the process consists of the stand-alone Java application serving as the interface between the designer and the toolchain. The target application is shown in the center of the interface in the form of its component software blocks with each block’s possible connection points. All available software blocks are listed on the left side of the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WUWNet '09 November 3, 2009, Berkeley, CA, USA.
Copyright 2009 ACM 978-1-60558-821-6...$10.00.
interface and can be added to the application through a simple click and drag. Once added, application blocks can be linked together also through click and drag to form a completed application. Individual software blocks can also be highlighted and then configured on the right side of the interface. The user, after constructing the target application, will click 'Run' and the GUI will generate a design descriptor file which can be interpreted by the code generator.

Once a complete system is configured, the third stage in the process interprets the user’s inputs to transform the chosen application modules into finalized C-level source code. This finished code is then compiled using AVR-GCC into both assembly and machine code that will be used for static timing analysis.

Finally, the fourth stage performs automated code analysis using the static timing analysis capabilities of the Thrint assembly code optimization tool [5]. This tool reads in the AVR assembly code generated by the AVR-GCC compiler and creates a Control Dependence Graph (CDG) representation of the program. The results of this analysis includes best and worst case execution times and memory usage information, which is then returned and interpreted by the GUI to complete the closed toolchain loop. At this point, the GUI will use the timing and energy models described in the original XMDL to determine a host of design information about the current iteration of the design. This information can then be used by the user to determine in the configured application will satisfy design criteria.

3. DEMONSTRATION RESULTS

We use an underwater ultrasonic communication system consisting of a customizable ultrasonic transmitter and receiver to demonstrate the toolchain’s analysis and successful prediction of timing, energy and memory characteristics.

To verify the accuracy of the timing results generated by the toolchain, we run the machine code created by the toolchain on the target hardware platform and record the actual execution time using a cycle counter we call the External Timing Harness (ETH). For the two platforms, the ETH measures the duration of execution by toggling output pins to signify when the processor was busy executing or sleeping. Each platform and configuration was automated to run for a period and the results captured by the ETH were then sent to a desktop PC for analysis. Then we compared the results with the best case execution (BCE) and worst case execution (WCE) bounds calculated analytically by the toolchain. We similarly collected all the results from different configurations for the transmitter. From these results, we could see clearly that all executions of the target hardware were in a short range between the BCE and WCE bounds predicted by the toolchain.

To verify the energy estimation of the toolchain, the transmitter hardware was configured to transmit continuously and the energy consumption of the processor was tracked. We created and used an external power supply with energy metering we call the Joule-o-tron to determine the total energy consumption per packet. The Joule-o-tron accurately integrates power use to determine energy by monitoring switch-mode power supply switching activity. This procedure was performed across the spectrum of available encoding schemes. The energy model utilizes the average of the BCE and WCE bounds as calculated by the code analysis stage of the toolchain. As we can see from such results, the energy estimation of the toolchain was within 10% of the measured values in all instances.

4. REFERENCES


