Embedded Systems Research at DemoFest'07

Oscar Almeida, Alessandro Forin, Philip Garcia, Johannes Helander, Nishith Khantal, Hong Lu, Karl Meier, Sibin Mohan, Henrik Nielsen, Richard Neil Pittman, Risto Serg, Bharat Sukhwani, Margus Veanes, Ben Zorn

Microsoft Research

Sarah Berry, Chris Boyce, David Chaszar, Brandon Culrich, Mikhail Kisin, Gabe Knezek, Warren Linam-Church, Steve Liu, Michael Stewart, Doug Toney

Texas A&M University

July 2007

Technical Report
MSR-TR-2007-94

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Embedded Systems Research at DemoFest'07

Oscar Almeida, Alessandro Forin, Philip Garcia, Johannes Helander, Nishith Khantal, Hong Lu, Karl Meier, Sibin Mohan, Henrik Nielsen, Richard Neil Pittman, Risto Serg, Bharat Sukhwani, Margus Veanes, Ben Zorn

Microsoft Research

Sarah Berry, Chris Boyce, David Chaszar, Brandon Culrich, Mikhail Kisin, Gabe Knezek, Warren Linam-Church, Steve Liu, Michael Stewart, Doug Toney

Texas A&M University

Abstract

The Embedded Systems group at Microsoft Research has been engaged in joint academic collaborations spanning both teaching and research activities. The results of these collaborations are jointly reviewed, and highlighted to the academic community at the Faculty Summit at Microsoft headquarters in Redmond, WA, during the DemoFest event. This is also a good opportunity to review some of the other research projects that the group is engaged in, with special consideration to the research performed as part of the summer internships. This report presents the demonstrations that took place during the 2007 DemoFest. Two undergraduate student projects from the Real Time Distributed System group at Texas A&M illustrate the results of the teaching activities. The first was a six-motor robotic assembly capable of solving the Rubik Cube puzzle interactively, on an actual cube. The second was a 3D spherical display system realized with a rotating ring of LEDs controlled by twelve microcontroller boards, each board controlling a row of 18 multi-color LEDs. The research projects presented included the introduction of the first working prototype of the MSR eMIPS dynamically self-extensible microprocessor and three current intern projects that leverage the eMIPS architecture. Additional joint projects included a prototype of a data management system based on multiple trust evidence sources, illustrating a new approach to online and embedded security developed in collaboration with the Software Design and Implementation group. A walking stick for outpatient monitoring was the result of the collaboration between the Robotics and the Embedded Systems groups. Finally, scalability in Embedded Systems was the focus of two closely related projects. The first project realized the world's smallest Web2.0 server, using 2,636 bytes of code and 92 bytes of data on an 8-bit AVR microprocessor. This result leveraged the analysis and modeling tools realized in the second project, a collaboration with the Foundations of Software Engineering group that led to a hybrid static-dynamic tool for worst-case execution time estimation and off-line scheduling generation.

1 Introduction

The Microsoft Faculty Summit is a workshop that has grown considerably over the years. This year, the attendance totaled nearly 350 highly qualified participants from universities across the globe. One of the most favorably received events at the Summit is the DemoFest event, when a large number of research projects and related activities are presented in a fair-like environment over a very short period of time (just about three hours!). This year the floor space, and consequently the number of booths, was somewhat reduced due to the widespread renovation work being carried out at the Microsoft Conference Center where the Summit takes place. The total number of booths was still high at 35, down only slightly from the 39 of the previous year.

The Embedded Systems Group has been present at the Summit since its inception, demonstrating both the results of its own research and the results of joint collaborations with academic partners and other researchers. This year we highlighted the ongoing collaboration with the Real Time Distributed Systems Group at Texas A&M University, with Microsoft Robotics, and with MSR researchers from the Software Design and Implementation Group and from the Foundations of Software Engineering Group.

The goal of this document is to attempt to communicate the vitality and excitement of the DemoFest event for those who could not attend it. While the atmosphere of free-flow communications and discussions is clearly impossible to reproduce, we can at least recapture and recount the artifacts and some of the practical demonstrations that took place in our small section of the event. Each of the document’s sections is dedicated to one of the demonstrations that occurred at the event; it describes the demo in simple terms and reproduces and explains its outputs and screenshots. Each section has a corresponding poster that was displayed at the booth and which is reproduced in Appendix A. Short movies and memorable moments are recorded in Appendix B.

It is likewise impossible to describe the atmosphere of the days that immediately preceded the event, when the demos where finalized sometime very late into the night and very many and very busy people crammed into a couple of offices, interacting, arguing, feeding, competing, laughing, sleeping and generally helping each other reach a common goal of complete and amazing success. Our heartfelt thanks go especially to Sara Dangel and Claudia Boyle for their help with the ensuing bedlam, and to all our significant others for their patience and tolerance.

Educational Experiences in Embedded Systems: A Goal Driven, Self-Propelling Process

Steve Liu
Texas A&M University

Abstract

Over the past five years, in partnership with MSR researchers, we have developed an experimental teaching strategy to explore more effective ways of teaching embedded computing systems to undergraduate students. Since its humble start, valuable experiences and tangible results continue to emerge. In this DemoFest we presented and shared the project's results. We found that disciplined creativity and explorative processes can steer students into a self-learning pathway that would be difficult to achieve by traditional lectures, tests and lab exercises alone.

In this strategy, the first half of the semester is driven by traditional lectures, exams and lab exercises aimed at imparting a baseline knowledge base. The second half of the semester is driven by open projects that require the students to deliver tangible prototypes, using the basic tools and materials learned in the first half of the semester. Specific learning objectives and methods are developed for different stages of the semester in order to achieve the stated technical goals of the class: "Exploration of students' creativity through group creation of entry-level, industrial grade toy projects using basic parts and tools."

1 Introduction

A recent National Academy report, "Rising Above the Gathering Storm: Energizing and Employing America for a Brighter Economic Future (2007) ," cites science and technology as critical assets to the security and prosperity of the United States. "Death by distance," a phrase used by the authors to describe the painful prospect of outsourcing professional jobs overseas, is a subtle yet real threat to the U.S. leadership in technology. While this issue needs to be addressed by the American society as a whole, higher education institutions face the immediate challenge of educating junior engineers who can serve the industry needs in the furiously competitive global market. This is particularly true for the information technology (IT) industry, which does not require heavy machinery investment to establish an operational base. The ongoing upheaval in the U.S. automotive industry is a clear hint of what will happen tomorrow if we do not act today on the pressing issues facing the American IT industry. Undergraduate education is responsible for producing the entry level engineers who perform the basic work. If domestic students cannot meet or surpass the productivity standards in the global market, the only alternative for a corporation to survive from global competition is to migrate the technical jobs to those places that can provide qualified workers at competitive wages. The last wave of outsourcing should be a clear reminder of this possibility. Unless we escalate both the number and professional preparedness of our engineering graduates, it will be only a matter of time before the U.S. loses its competitiveness in the global IT market.

Public universities are chartered with educating the larger student body with a broad background, within reasonable financial burdens for the students. Recruiting and retaining the undergraduate engineering students without compromising the curriculum's quality is a standing priority issue for many schools. Our experience is related to this issue, but some of the experiences provide greater insights on the effects of the new teaching approach. Our goal is to escalate the students' ability to transform technical knowledge into working systems at a minimal cost. Some self-imposed questions and our field experiences in addressing these questions in a large public university are highlighted in this report. Despite the progress made in the past several years (2002-present), our discoveries, experiences and approaches were limited to the Texas A&M campus. They may not necessarily be applicable to other institutions, and they should be interpreted with caution.

In this experiment, we took a clean slate approach in revising the course; literally everything related to the class—technical materials, labs, assignments, textbooks, technology, even the lab physical layout itself—are new. Our goal is to advance the technical competence of the students within existing credit-hour and prerequisite constraints. To achieve this goal, we based our experiment on the adjustment of the teaching-learning strategies, of the class objectives and of the technical materials. The entire process was supported by industry partners from Microsoft, Xilinx and ARM.

The Changes

Before our experiment started, the class was structured like most other classes in the curriculum but with the addition of a lab session. A textbook based on a popular microcontroller family was the only technical reference. The lab was based on a few exercises using the microcontroller covered in the book. Homework assignments, tests, and labs were the main tools for student learning and evaluation. The lecture materials were well structured and followed the book, and the lab was based on a few fixed exercises based on a separate lab manual. For students, this class setup represented a gradual transition from a learning process dominated by the textbook-exam scheme into an environment with some lightweight hands-on lab activity, and before they entered into their final senior capstone design class. Unfortunately, previous experience in teaching the senior capstone design course had shown that most students did not even know how to start a project, or were unable to transform their knowledge into a working system and therefore most students failed to bring the project to a successful completion.

We began by researching incremental changes to the existing technical materials and procedures in order to achieve our goals. This was eventually abandoned because of the following overriding concerns: (1) the textbook was very rich in content, but it was a major burden for both students and faculty. If a book is used as the main technical thrust there is little room left for other teaching options. (2) There was a significant gap between the lab materials and the textbook. (3) The existing process served the learning purpose well, but only for the heavily structured material in the book. Students would garner little experience in the discovery, usage and cross-reference of the broad and diverse technical data required to build practical working systems.

After an extensive search for textbooks, we found none that served our purposes effectively. We concluded that the only option left was to restructure the class from scratch. With the endorsement of both industrial collaborators and colleagues whose interests directly related to and were affected by the class, we decided to restructure it using a total hands-on approach to teaching and learning. We replaced the textbook with real microelectronic chip datasheets and schematics, supplemented by lectures on basic system design. Instead of developing monolithic but trivial applications on the microcontroller board, we choose to run a small open source, componentized, real time operating system (*Microsoft Invisible Computing, aka MMLite*). We consider the addition of the real time operating system to the microcontroller an important decision because, for instance, most modern handheld devices require some operating system functions. Students would learn to create embedded software that can coexist with and leverage other pre-existing software modules. We added an FPGA board to the hardware contents because of its widespread use in modern microelectronics systems. We chose not to use testing as the primary tool to assess the students' performance in class; instead, we gave more weight to the lab sessions and, most importantly, to the final open project. This reflected our view that basic knowledge can be checked by written tests, but system design skills and creativity can only be tested by labs and projects.

Faculty and Student Roles

A class is a human-centric system where faculty, students and teaching materials are the inputs, and the new knowledge and experience acquired by the students are the outputs. The traditional wisdom is to deliver the material in the most structured way possible in order to maximize the amount of knowledge that the students will gain from the class. The structure of the body of knowledge, the communication between faculty and students, the assignments, the tests, and the labs form the crux of most classes. For many fields in science, mathematics, engineering and technology (SMET), interaction mechanisms that are student-oriented and multidimensional can serve the learning objectives effectively. But whether the same can be said about education in information technology needs to be examined carefully.

In our approach, the semester is informally divided into two parts. In the first half the instructor and teaching assistants deliver the basic system concepts, lab procedures and examples to bring the students to the minimal technical level sufficient to perform basic development functions. The second part of the semester is instead driven by the students to bring their open project to completion. The faculty plays many roles in this second half of the semester:

- **Consultant** for project ideas and strategies;
- **Mediator** for dispute arbitration;
- **Sponsor** to finance the projects' purchases;

- **Judge** for the final evaluation.

Students begin the semester as passive learners, but soon they must explore their design options and the available parts needed to complete their project. The second half of the semester essentially becomes a self-propelled, self-learning process.

Hands-on and Fun

Fun and curiosity are the key factors that stimulate the students' willingness to tackle advanced ideas and transform them into working systems. The value of building an artifact is lost on the students when it cannot be shown to friends, or if they do not find it interesting. This is a particularly important issue for embedded computing systems, because of the tight integration of hardware, software, algorithms and even mechanical design into any complete, working system. Granted, learning to construct computing systems is important in its own right, and the integration of the computing devices with their environment, e.g., power, sizes, mechanical structures, etc., is also a major challenge. But rather than attempt to instill the proper design rules in advance, experience has taught us that leaving all options open to the students leads to a more effective self-learning experience. Our approach is to use a challenge-response process while mentoring the students in the pursuit of their technical goals. Students are challenged at every stage about the scope and progress of their projects, but there is no definite correlation between their adoption of the instructors' inputs and the final project outcomes. In fact, some of the best projects were realized in total defiance of the instructors' suggestions. Nonetheless, when the challenge-response process is used in conjunction with the instructor's sponsorship incentives, it clearly stimulates the students to think out of the box and to take on more interesting projects.

Students can use any information and any available design reference, provided that they can properly justify the value-added to their own project. They are allowed to regroup, and to scale up or scale down as their projects progresses with time. They can even redirect the project goals if appropriate. Freedom however comes with responsibility; students are held responsible for the way they use the information, for how they manage the intra-team relationships, and for the ultimate outcome of their project. Projects are graded based on the value that is added during the process. This value added could be quantified in terms of creativity, engineering process, or even the (failed) attempt to take on exceedingly difficult problems. Safe or "canned" projects receive the lowest and possibly failing grades.

From the teacher's perspective, the objective of hands-on and fun is not the artifact itself, but rather the stimulation of the students' creativity, knowledge usage and teamwork. The result of the final project is only a portion of the final grade; even a failed project can receive a good grade if the team can demonstrate high quality in their project development process. The level of challenges, creativity, and teamwork vs. accountability are all important decision factors. Students are encouraged to follow and to expand upon the successful projects from previous semesters. Admittedly, some projects did resemble each other but every semester has produced at least one very interesting project that is worth preserving and that will be cited as example in the next semester. Outstanding projects are submitted to the DemoFest fair to share our experiences with participating faculty. The history of exceptional projects is maintained in a website, and together with the exposure at DemoFest it creates further motivation for the new students.

Literature vs. Textbooks

A critical decision made at the beginning of this project was to eliminate the textbook in favor of real microelectronic datasheets, source codes and some basic system design lectures. We did not make this choice lightly. For generations the textbook has been the proven tool to pass knowledge from one generation to the next. It will continue to play this critical role in our teaching system, but it is not the only tool that can be used for a design-oriented, embedded computing class. Rarely do two embedded computing systems look alike; for decades in this field, very few convergent theories, architectures, or computing models have been agreed upon. New technologies constantly emerge from industry and many of them have very short life spans. Although datasheets are not written in the clear format and style of textbooks, most are written in fairly simple language and with complete technical information. We have classified the datasheets in two types: architectural and functional pages, and reference pages. Reference pages are not covered in the exams but functional pages are. While it may appear less convenient to students, there is a compelling reason to expose them to the original technical documents as they learn real system design.

Some websites are easy to navigate, while others are very difficult. Similarly, some datasheets are much easier to read than others; it may take weeks for a full-time engineer to go through some particularly voluminous and detailed set of datasheets. Unless they have some prior internship experience in the industry, most students assume that the textbook is their only source of knowledge. Luckily, technical information for undergraduate-level students is abundantly available from many different sources. Very few students, however, are able to navigate through the many sources of information and their required tool chains and then transform them into working systems. Helping students make the breakthrough and move from the class to industry is very difficult if we maintain the teaching and learning processes of the past.

The other major concern with a textbook is the timely delivery of contemporary and updated knowledge to the students, because it can take months or even years before a good textbook can be written about a new subject. Requiring students to use industry datasheets in the class can be quite intimidating, but we might not be able to find a textbook that balances the well structured abstraction against the detailed and complete information. Therefore, we have no alternative but to make selective use of some well written datasheets to better serve the purpose of bringing the students closer to the real world of system design and development.   "Time to market" is an industry expression that captures the tension to deliver a product to the consumer in the shortest time possible, to avoid rapid obsolescence and loss of consumer interest. Similarly, we might define as "time to classroom" our own struggle in the education field to bring to the student materials that are current and therefore improve on the market value of the knowledge that the students acquire.

Industry Collaboration

A primary objective of this experiment is to find the teaching methods that most elevate the students' professional preparedness. Experience tells us that there is a strong correlation between the degree of interaction between industry and university and these educational outcomes. This interaction must lead to face-to-face dialogues that can be translated into classroom-business actions. Ever since the inception of our collaboration, our MSR partners have played a critical role in the progress that we have been able to make. The first natural question was how to better match students' skills and industry expectations. It turns out that this issue does not have a standard answer, and, rightfully, should not. Industrial needs change with time, and the delivering of contemporary technological knowledge to the classroom is only one small part in the effort to make a quantum leap in educational outcomes. Material engagement in the classroom activities and genuine interests from industry are critical to keep the class lively and interesting for both faculty and students. Our joint projects in the DemoFest have proven highly effective in keeping the collaboration energy focused and vibrant.

In summary, we believe we have identified a sustainable, productive educational model that can enable other faculty with similar interests to reenergize their classes. It is not easy to get this far, particularly considering the existing fiscal and curricula restrictions and the students' resistance to change, but the outcomes speak for themselves. Students are able to complete far more advanced projects in their capstone design courses, and they demonstrate greater confidence in their job interviews. And the project results get stronger and stronger with the years.   This year is another good year, and we look forward to an even better future.

Rubik's Cube Solver

Chris Boyce, David Chaszar, Michael Stewart, Doug Toney, Steve Liu
Texas A&M University

Abstract

The Rubik's cube solver is a multilayer embedded computing-motion control system that is capable of moving a Rubik's cube from an arbitrary initial state to the solved state. The high level cube solver is based on the Kociemba's algorithm and was implemented by Michael Reid (source: http://www.cubeman.org/). An EB63 microcontroller evaluation board running an application built with Microsoft Invisible Computing receives the motion commands from the high- level solver running on a PC, and moves the cube accordingly by rotating one of the faces at a time. We built a hardwood frame, mounted six stepper motors on the frame and directly coupled them to the cube to make the moves. Each axel links one stepper motor to the center element of one of the cube's faces. The whole project was completed within a few weeks from the assignment and demonstrates the students' abilities to transform their knowledge into a working system. Except for the RubikSolver software, all other software and hardware circuits and their controlling functions were realized by the students with minimal assistance from the instructor.

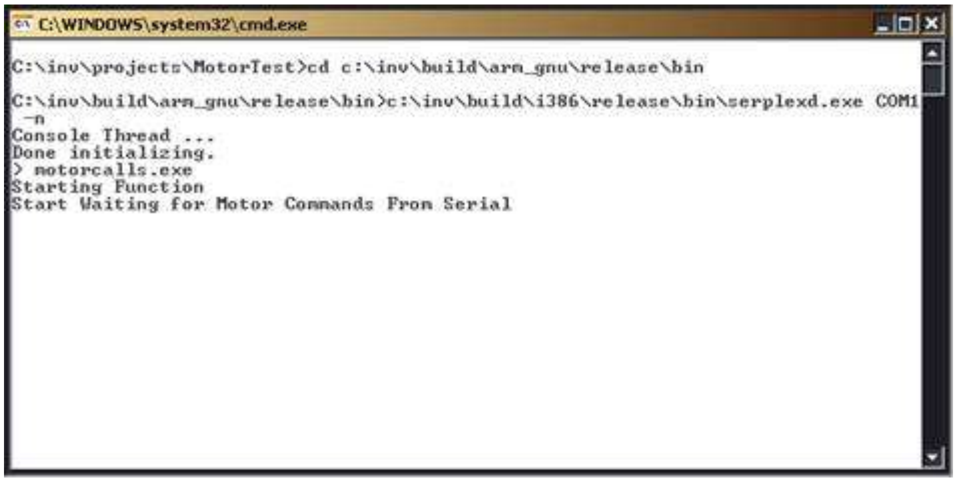
1 Introduction

RubikSolver is the GUI/solving portion of the Rubik's cube solver. It is solved separately on the PC because the algorithm used [1] (Kociemba's algorithm) is fairly space and CPU intensive, more than the EB63 can perform within a reasonable time frame. It uses the COM2 port on the PC in order to send the motor commands to the EB63, but it is also possible to just solve the puzzle without sending the solution to the motors.

Kociemba's algorithm is implemented as a non-optimal solver first formalized by Herbert Kociemba. The solving algorithm we used is modified from source code provided by Michael Reid. The code was originally acquired from http://www.cubeman.org/, where Reid's Near-Optimal Cube Executable and Source is provided freely to the public. We have made many changes to the original version. We have ported the source code to C++, and moved all of the algorithm code into its own class (*KociembaAlgorithm*) under its own namespace (*Kociemba*). Additionally, several of the structs have been changed into classes and their members made private (*Cube*, *Options*), as well as using C++ friend declarations to ensure smooth transition (*Options*). *Cube* has been moved into its own namespace (*Rubik*) along with several constants that might be useful for future algorithm implementations. *KociembaAlgorithm* derives from a base *Algorithm* class which also helps make it easier to implement other algorithms in the future. The end result is that we have reduced pollution of the global namespace with algorithm specific constants and classes. Additionally, many of the #define'd constants have been made either static const or made into enums to help enforce type safety. Finally, the original printf and scanf routines have been replaced with much safer C++ streams, which makes input and output to the algorithm much more flexible.

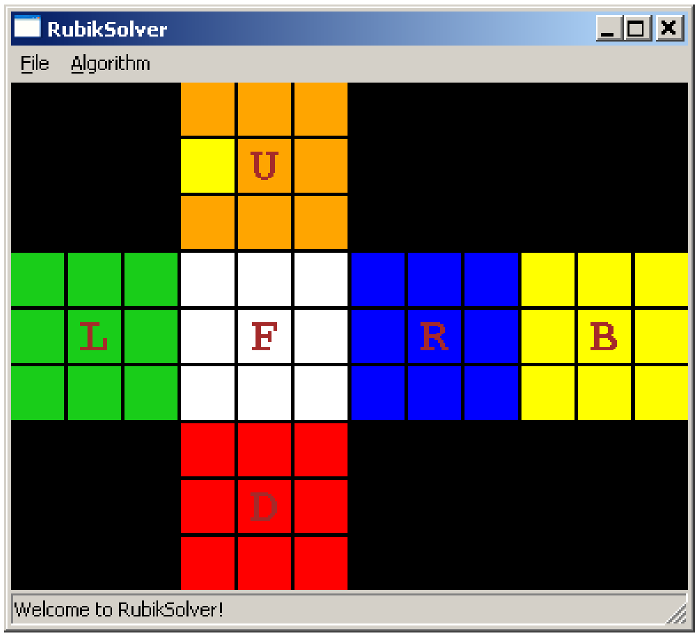
2 Demonstration

We start the demonstration by running RubikGUI.exe on the PC side and motorcalls.exe on the EB63 side. When motorcalls is initialized and ready for operation the console screen output from the board is as shown in the following screen. This indicates that the controller is waiting for the string of commands, which are expected on the second serial line connector of the EB63 board.

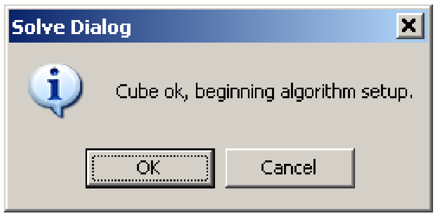


```
C:\inv\projects\MotorTest>cd c:\inv\build\arm_gnu\release\bin

C:\inv\build\arm_gnu\release\bin>c:\inv\build\i386\release\bin\serplexd.exe COM1
 -n
Console Thread ...
Done initializing.
> motorcalls.exe
Starting Function
Start Waiting for Motor Commands From Serial
```

Once the GUI screen appears, there are several options. We can load a preset cube puzzle, create and save a new one, or enter the current state of the actual cube. The color of a square is selected by rotating through the six available color options, a left-click cycles in one direction and right-click in the other. Cubes must be entered from left to right, top to bottom facing the side. The following picture shows the GUI offered to the user. Faces are marked individually to help locating the corresponding ones on the actual cube.



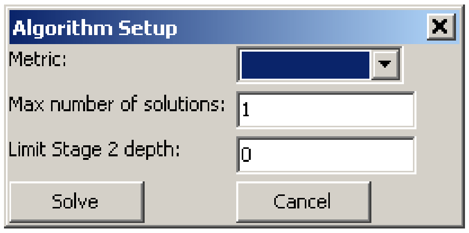
Once the cube puzzle has been entered in the GUI, we can select **Algorithm->Solve** or **Algorithm->Unsolve** and the program will pop up a window saying whether or not the current entered cube state is valid.



Solve Dialog

Cube ok, beginning algorithm setup.

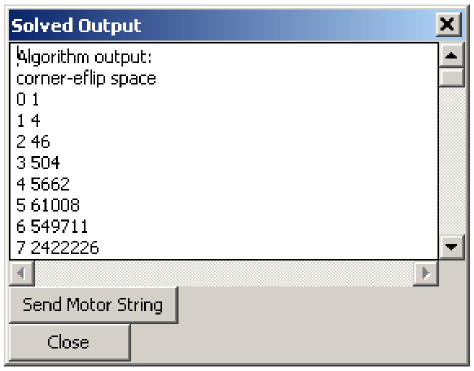
After confirming that the cube state is correct, the GUI creates a new window titled '**Algorithm Setup**'.



```
Metric:
Max number of solutions: 1
Limit Stage 2 depth: 0
```

Here the user is presented with three options: the metric to solve for, the maximum number of solutions, and the depth limit of the Stage 2 solver. The metric determines what the solver tries to minimize, either face turns or quarter turns (face turn is faster). Since there is potentially more than one possible solution to a puzzle, the user can select the maximum number of solutions to collect before exiting (suggested 1, 0 for no limit). The depth parameter controls the search space expansion strategy; we recommend keeping it at 0 (indicating no limit).

Once the options are set we click **Solve**, and a solution window appears with the moves required:

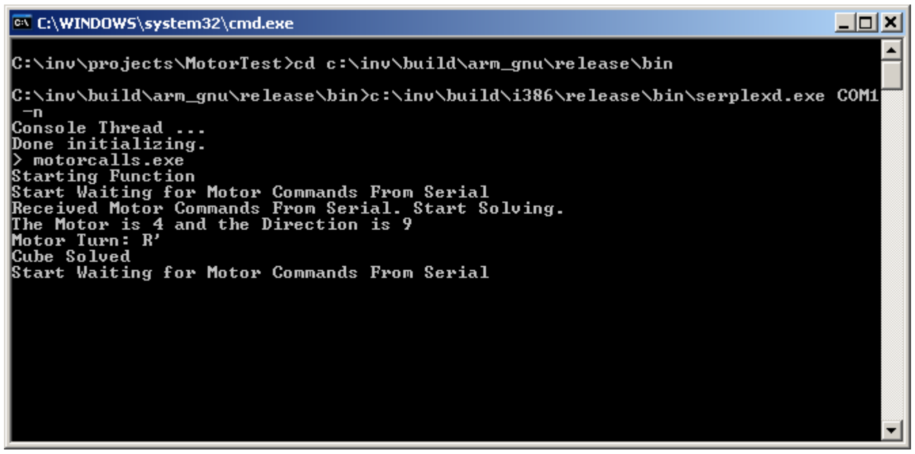


```
Algorithm output:
corner-eflip space
0 1
1 4
2 46
3 504
4 5662
5 61008
6 549711
7 2422226
```

To actually move the real cube we click **Send Motor String**, and the string of commands is sent to the EB63 over the serial line COM2. The final result of the motion commands is as follows:

- If we fist clicked **Algorithm**->**Solve**, the motor string will solve a scrambled cube to a clean cube.
- If we fist clicked **Algorithm**->**Unsolve**, the motor string will solve a clean cube to a scrambled cube.

Now the move instructions go to the EB63. The EB63 will echo in the console (visible on the DOS window below) each command it receives:



```
C:\inv\projects\MotorTest>cd c:\inv\build\arm_gnu\release\bin

C:\inv\build\arm_gnu\release\bin>c:\inv\build\i386\release\bin\serplexd.exe COM1
 -n
Console Thread ...
Done initializing.
> motorcalls.exe
Starting Function
Start Waiting for Motor Commands From Serial
Received Motor Commands From Serial. Start Solving.
The Motor is 4 and the Direction is 9
Motor Turn: R'
Cube Solved
Start Waiting for Motor Commands From Serial
```

As shown above, in this case the program received a single command to move the **Right Motor** Counterclockwise 90 degrees. The encoding for "Motor Turn" is:

R = 90 degrees Clockwise
R' = 90 degrees Counterclockwise
R2 = 180 degrees Clockwise

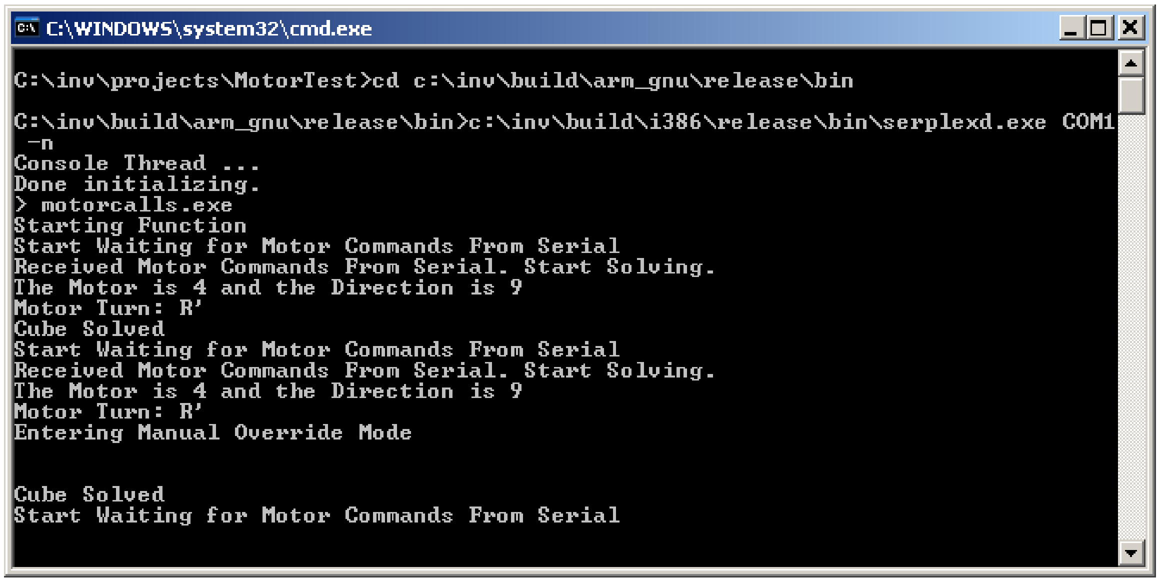
And similarly for the other motors, marked as in the GUI with the letters U, L, F, R, B, and D. The program will indicate **Cube Solved** when it has reached the end of the solve string.

3 Lockup Resolution

It is possible for the cube to either lock up due to excessive friction, or to perform a partial rotation due to insufficient friction. In case the cube malfunctions and is locked in an invalid configuration it is possible to manually operate the motors and to continue solving. The **Manual Override** feature allows the user to move the sides of the cube manually. Three buttons on the board are used to implement this feature:

PIOB3	*Enter or exit Manual Override mode*
PIOB4	*Rotate Clockwise*
PIOB5	*Rotate Counterclockwise*

The rotations apply to the currently selected motor. Once the adjustments are complete, pressing *PIOB3* again will exit **Manual Override** and the solver will continue with the next move.



```
C:\inv\projects\MotorTest>cd c:\inv\build\arm_gnu\release\bin

C:\inv\build\arm_gnu\release\bin>c:\inv\build\i386\release\bin\serplexd.exe COM1
 -n
Console Thread ...
Done initializing.
> motorcalls.exe
Starting Function
Start Waiting for Motor Commands From Serial
Received Motor Commands From Serial. Start Solving.
The Motor is 4 and the Direction is 9
Motor Turn: R'
Cube Solved
Start Waiting for Motor Commands From Serial
Received Motor Commands From Serial. Start Solving.
The Motor is 4 and the Direction is 9
Motor Turn: R'
Entering Manual Override Mode


Cube Solved
Start Waiting for Motor Commands From Serial
```

[1] Herbert Kociemba. Cube Explorer. `http://kociemba.org/cube.htm`, 2006.

Orb Spherical Display

Sarah Berry, Brandon Culrich, Mikhail Kisin, Gabe Knezek, Warren Linam-Church, Steve Liu
Texas A&M University

Abstract

The Aggie Orb is a spherical display system capable of displaying images on a rotational ring, which rotates at a speed of 1100 RPM (corresponding to approximately 20fps) to create the spherical visual effects. This project represents two generations of development efforts by undergraduate students in the computer science department of the Texas A&M University. The first generation Orb I was a light-weight prototype built using hand-made circuit boards, connectors, and pedestal. The success of Orb I led to the creation of Orb II, which was built on a heavy duty metal frame and pedestal, a Plexiglas safety enclosure, twelve custom-built LED controller modules connected in a daisy chain, and a high performance slip ring to implement the spherical display concept. Due to the heavy weight (over 200 pounds) and large size, two students loaded the artifact on a van and drove for three days from College Station, TX to Redmond, WA to present the project results at the Summit.

1 Introduction

The Orb II project is an extension of the Global Spinomatic 3000 project (Orb I) created by Sarah Berry, Mikhail Kisin, and Gabe Knezek in the CPSC 462 Microcomputer Systems course. Given the time allotted (about 1 month) and the budget constraints, the Orb I was an excellent proof of concept with regard to spherical rendering. At completion of the semester the students were invited to extend the project with higher grade material, more advanced manufacturing processes, and more advanced computing and communication architectures. Some students in the Orb I project chose not to continue the endeavor, and two new students joined the continuation project. Orb I was implemented in a class setting within a planned schedule, whereas Orb II was implemented in a small group, under the direct supervision of the instructor and under a directed study class code.

2 Cyber-physical system principles

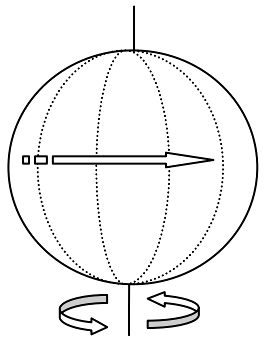
The problem space of the spherical display idea is the synchronization of the temporal and spatial activities in the cyber and physical dimensions. It went through three generations of testing in an integrated teaching and research environment before it was formalized into the lab exercises of the class. The first generation product (shown at side) is an air banner controlled by a PIC processor and capable of visualizing characters input from the PC. This project was mentioned at DemoFest'04 in the Embedded Systems education panel.

The second generation prototype is a laser pointer writer, designed to plot graphs on the wall using the combined motions of a laser pointer and a mirror controlled by stepper motors. The controlling software runs on the EB63 evaluation board. The Microsoft Invisible Computing runtime provides the resource management services and real time management functions for the display system. The pictures show the prototype as well as its projected output on a dark background. The laser display system was first reported at DemoFest'05 in the Embedded Systems education panel.

The idea of using laser pointer and mirror to demonstrate the temporal and spatial synchronization appeared to be a cost-effective approach for the undergraduate microprocessor class. A new version of the basic design based on servo motors was created and further expanded into a dual platform system.

A gyro chip was used to realize an additional an input command device that allows the interactive control of the drawings. This prototype was demonstrated at DemoFest'06. At this point, the concept was mature enough to be formalized into a series of lab exercises for students to learn the basic principles of temporal and spatial synchronization through prototyping. It proved a highly effective platform for students to understand the issues in the cyber (temporal) and physical (motion) synchronization. The Orb system therefore is only the latest result in a long thread of evolution.

3 System concept and the Orb architecture



The Orb architecture is based on a single-axis mechanical structure. All the electronic parts are integrated into a spinning ring; the visual effects are generated by multi-color LEDs moving along a spherical trajectory at a rate high enough to be undetectable to the human eye (approx 15-20 fps). A stationary magnet is placed near the spinning ring, and its coupling with a Hall Effect sensor placed on the ring itself indicates the current angle of the ring (latitude) to the controlling software. Using the latitude information, software controls the timing and color of the LEDs to create the pseudo-stationary patterns. Effectively, while in motion the device creates a sphere of "pixels" usable to display color images. The number of pixels in the vertical dimension (longitude) is statically defined by the number of LEDs placed on the ring. Notice that it is possible to alternate LEDs from the two sides of the ring to obtain a finer resolution in this dimension. Software could conceivably reduce the resolution by darkening alternating rows of LEDs. The number of pixels in the horizontal dimension (latitude) is instead dynamically defined only by software, but in reality it is limited by the maximum switching rate of the LEDs (at the pole).

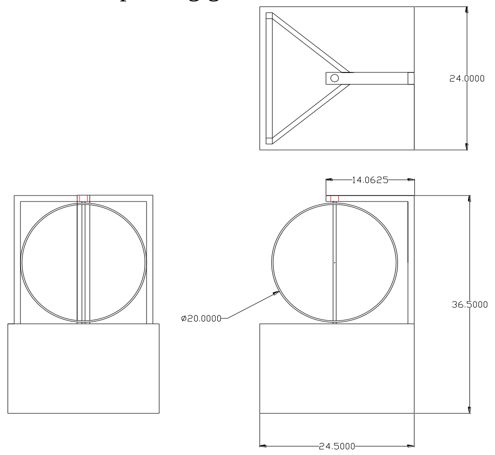
The Orb I used a centralized computing architecture to control a small number of high intensity LEDs directly placed into a small aluminum ring, and all the control circuits were hand made, physically bounded on the ring with hot glue.

The Orb II has a steel beam based mechanical frame, which was tested by running it for an hour of constant high-speed rotation to ensure that the mechanical strength is adequate. A total of 12 LED-controlling modules based on high intensity LEDs, LED drivers and AVR microcontrollers were designed and fabricated for the spherical illumination system. Each of the 12 LED modules is equipped with 18 tri-color LEDs. The modules are interconnected via a daisy chain bus to transfer data and power to all the modules on the ring, and to the external computing system outside the ring. The image resolution is 100x162 pixels. The off-ring computing system is interfaced to the on-ring LED modules through a six-lead slip ring. Each pair of matching leads on the slip ring provides the continuous connectivity while the slip ring is spinning.

4 Mechanical design and safety considerations

After the mechanical structure passed the stress test, all cables and their connectors, the microcontroller boards for the LED displays, the communications and other supporting functions were realized and tested off-line. The daisy chain bus successfully passed the functional test the very first time when the LED modules were powered up and interconnected. The most challenging issues are related to securing the modules to the ring so that parts do not get loose and fly out. After multiple tests, we found that the high strength fishing lines and Velcro tapes can indeed secure the boards, cables and their connectors under an estimated 11G of spinning force. For extra precaution, we added a transparent enclosure in Plexiglas that protects the spectators from catastrophic rupture events.

The quality of the visual effect improves with a higher spinning rate, but this also leads to stronger structural vibrations. Great care must be taken to balance the weight distributions to avoid potentially severe property damage or even injuries. The spinning generates considerable lateral forces; see below the G force calculations for the ring.



$$d = 20'' = 0.508m$$
$$r = 0.254m$$
$$v = 2\pi r(rps) = 2\pi(0.254)\left(\frac{1075rpm}{60\sec/_m}\right) = 28.594\,{}^{m}\!/_{s}$$
$$F_c = \frac{mv^2}{r}, F = ma$$
$$a_c = \frac{v^2}{r} = \frac{28.594\,{}^{m}\!/_{s}}{0.254m} = 112.574\,{}^{m}\!/_{s^2}$$
$$a_g = 9.8\,{}^{m}\!/_{s^2}$$
$$G - forces = \frac{a_c}{a_g} = \frac{112.574}{9.8} = 11.487$$

5 The journey

The plan was to demonstrate the Orb II and two other projects at the DemoFest'07 event, but the deadline approached and the project infact was far from complete. Because of its heavy weight (estimated over 200 *lb*s) and its large size, transport of the Orb II is not trivial. We inquired multiple carriers of their time and other delivery restrictions. It was concluded that the lead time, packaging and shipping could become a major hurdle to the plan. Eventually Gabe Knezek and Warren Linam-Church decided to simply rent a minivan and embark in the three days road trip from College Station, TX to deliver the prototype and development computers to Redmond, WA. Their GPS location was relied to the instructor and other collaborators, who followed Gabe and Warren until their safe arrival.

6 The demo

The prototype was tested for low bandwidth communications through the slip ring but the powerful spinning motor caused significant noise on the DC power and communication links. The image mapping software, whose original function was to drive the Orb display patterns in real time, was instead used to transform any image into the display format of the spherical surface. The most popular pattern used in the demo is the planar earth, as shown in the short movie in Appendix B. It took a few seconds for the motor to reach its table speed, and also for the software system to calibrate to its average readings. A short movie illustrating the Orb in action is shown in Appendix B. The spherical display shows the Earth, using a bitmap taken from the Windows time-zone applet and downloaded in advance to the microcontrollers.

7 Operations

Currently, the LED modules are powered by eight rechargeable batteries, which should be fully charged before a demo session. The batteries are located at the bottom of the ring and need to be securely placed in their casing, with all the flat cables tugged in over them. The protective cage and the front cover must be tightened before the motor power is turned on. The prototype is not designed for long time of continuous operation. If the noise increases (loosening of screws on the cover), odd smell emerges (overheated motor) the power should be shut down immediately.

eMIPS Prototype Introduction

Richard Neil Pittman, Alessandro Forin
Microsoft Research

Abstract

The eMIPS architecture is able to realize the performance benefits of application-specific hardware optimizations in a general-purpose, multi-user system environment using a dynamically extensible processor architecture. It allows multiple secure Extensions to load dynamically and to plug into the stages of a pipelined data path, thereby extending the core instruction set of the microprocessor. Extensions can also be used to realize on-chip peripherals and if area permits even multiple cores. The first eMIPS prototype is implemented on a Xilinx ML401 development board for the Virtex-4 XC4LX25 FPGA. A working prototype was introduced for the first time to the academic community at the Microsoft Faculty Summit in July 2007. Practical demonstrations included executing the hardware validation tests, a simple program demonstrating the new functionality provided by the architecture, and an HTTP server running on top of an embedded RTOS.

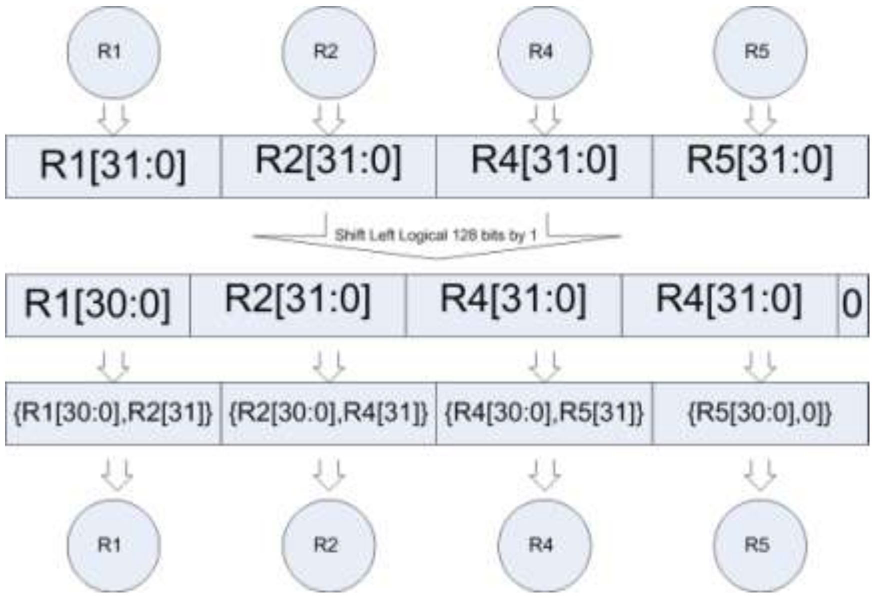
1 Introduction

The eMIPS dynamically extensible processor implements the MIPS RISC architecture, augmented with hardware Extensions that are dynamically loaded and can interact directly with the RISC pipelined data path. The eMIPS processor prototype is implemented on a Xilinx Virtex 4 LX25 device (XC4LX25) integrated in a Xilinx ML401 development board. The partial reconfiguration feature available on Xilinx Virtex FPGAs makes it possible for the eMIPS processor to change dynamically the level of performance or functionality to adapt to the changing needs of its applications. In the first part of the demonstration, we used the hardware validation tests and an automated test driver to run a number simple programs. For each program, the processor is reset, the program is downloaded to RAM over a serial line connection by the on-chip bootloader, control is transferred to the program and the pass/fail state of the test is detected by capturing its output to the serial line. These tests verify the proper functioning of the processor itself and of the on-chip peripherals; they are used by the eMIPS developers to verify any changes they make before they attempt to load a full OS.

In the second part of the demonstration, we booted a small embedded operating system and ran a test using the first practical Extension that was developed. This Extension is used to accelerate (a portion of) a software-implemented 64-bit division using 32-bit arithmetic. The intermediate step in the division is shown below in MIPS assembly. It involves shifting 128-bit quantities across four 32-bit registers, comparing two 128-bit values, 128-bit subtraction and 32-bit OR to raise a quotient bit if necessary.

```
sll   r1,r1,1
srl   r3,r2,31
or    r1,r1,r3
sll   r2,r2,1
srl   r3,r4,31
or    r2,r2,r3
sll   r4,r4,1
srl   r3,r5,31
or    r4,r4,r3
sltu  r3,r1,r6
beq   r0,r3,40
sll   r5,r5,1
```

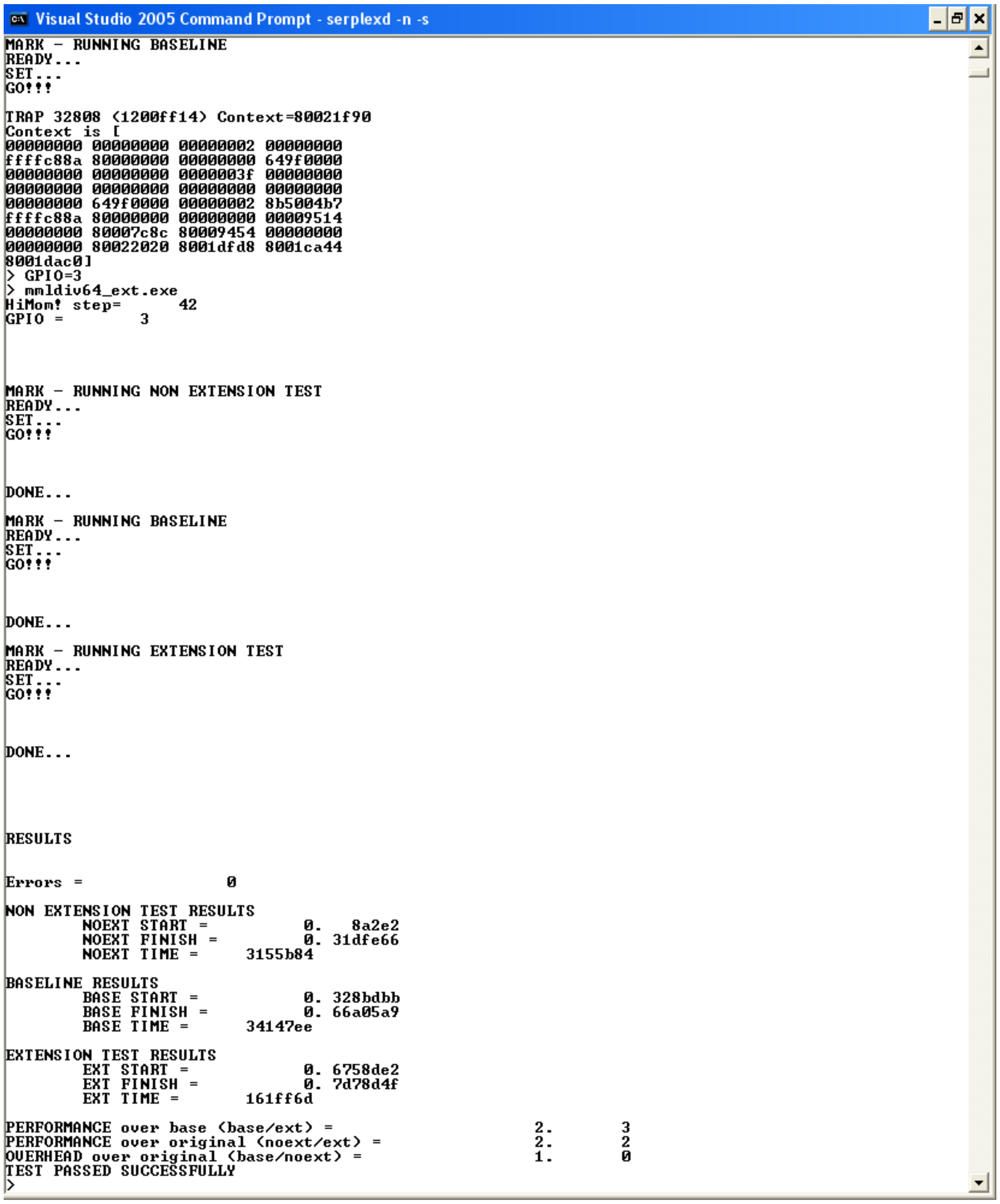
The Extension targets this 128-bit shift-by-one in the division process. Shifting a bus by a single bit is a trivial task for hardware but the software is forced to work with 32-bit registers at a time. This requires shifting one register by one, getting the most significant bit of the next register by shifting it, OR the shifted register with the most significant bit of next register and repeat. The Extension is able to read four registers from the register file, input them into a shift register, perform the shift by one and write the registers back. This is illustrated in the figure below.



The Extension also performs a conditional jump, as indicated by the last three MIPS instruction in the block above. The Extension compares the value of R1 with another register R6 (not in the figure) and based on that comparison the Extension either updates the PC before relinquishing control or it allows the flow to fall through to the current next instruction.

We demonstrated the division Extension using a test program called "mmldiv64_ext.exe" and using the Microsoft Invisible Computing Embedded Operating System. The picture below shows the output captured from a PC. In the first attempt to execute the program the Extension is not loaded, which results in the program attempting to execute an Extended instruction which is not recognized by the processor. This results in a Reserved Instruction exception that is captured by the OS handler, which then dumps the content of the processor registers and terminates the program.

In the next step, we use the XILINX iMPACT tool to load the Extension over the JTAG cable, and the test program this time runs successfully to completion. The same set of 64-bit divisions are executed (and the results checked) in three different ways by the test. The first time they are executed by the original version of the 64-bit division and timed using the on-chip 10MHz free-running counter. The second time we execute the modified version of the 64-bit division, which differs from the previous one only in the insertion of the Extended instruction at the start of the suindicated basic block. For timing purposes, the first time around we nullify this instruction, e.g. we treat it as a NOP instruction. The results are the same as before, but the test takes slightly longer to execute due to the fetching of the extra instruction. In the third and final pass we do recognize the Extended instruction and execute the basic block directly in the Extension. The test then compares the timings for the three runs and prints the speedups that were obtained.

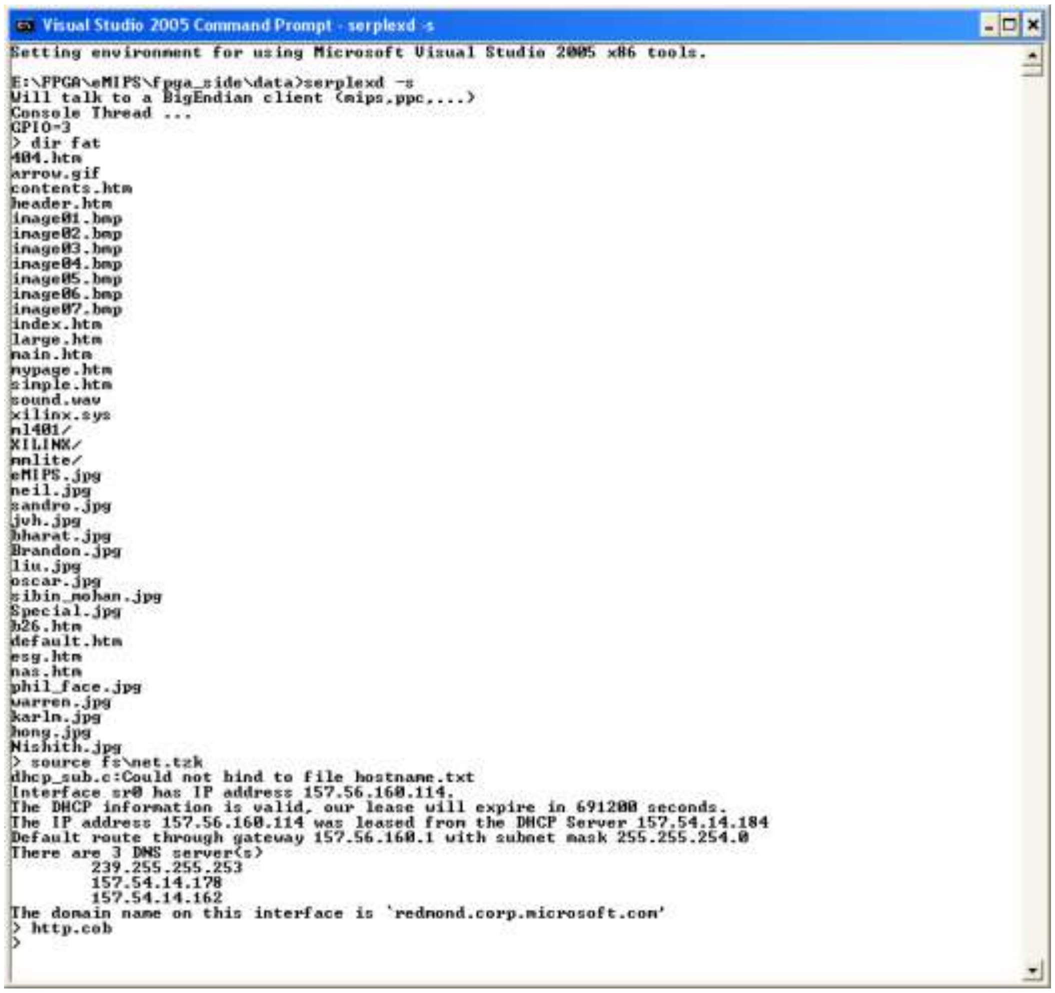


Visual Studio 2005 Command Prompt - serplexd -n -s

```
MARK - RUNNING BASELINE
READY...
SET...
GO!!!

TRAP 32808 (1200ff14) Context=80021f90
Context is [
00000000 00000000 00000002 00000000
ffffc88a 80000000 00000000 649f0000
00000000 00000000 0000003f 00000000
00000000 00000000 00000000 00000000
00000000 649f0000 00000002 8b5004b7
ffffc88a 80000000 00000000 00009514
00000000 80007c8c 80009454 00000000
00000000 80022020 8001dfd8 8001ca44
8001dac0]
> GPIO=3
> mmldiv64_ext.exe
HiMom! step=       42
GPIO =         3



MARK - RUNNING NON EXTENSION TEST
READY...
SET...
GO!!!


DONE...

MARK - RUNNING BASELINE
READY...
SET...
GO!!!


DONE...

MARK - RUNNING EXTENSION TEST
READY...
SET...
GO!!!


DONE...




RESULTS


Errors =                0

NON EXTENSION TEST RESULTS
        NOEXT START =           0.   8a2e2
        NOEXT FINISH =          0. 31dfe66
        NOEXT TIME =      3155b84

BASELINE RESULTS
        BASE START =            0. 328bdbb
        BASE FINISH =           0. 66a05a9
        BASE TIME =       34147ee

EXTENSION TEST RESULTS
        EXT START =             0. 6758de2
        EXT FINISH =            0. 7d78d4f
        EXT TIME =        161ff6d

PERFORMANCE over base (base/ext) =                      2.       3
PERFORMANCE over original (noext/ext) =                 2.       2
OVERHEAD over original (base/noext) =                   1.       0
TEST PASSED SUCCESSFULLY
>
```

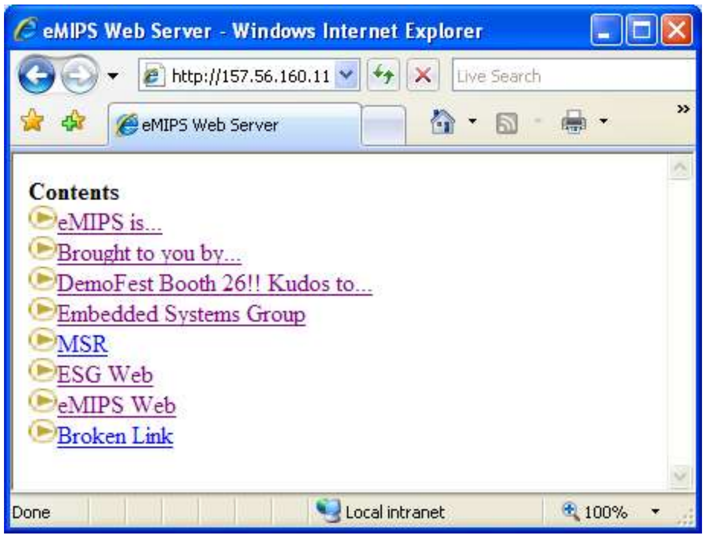
The system has now become stable enough to execute a more complex test, involving the same embedded operating system as above but using more complex software and a more diverse set of peripherals. As shown in the figure below, we can access the FAT16 file system on the ML401 Compact Flash card and list its content by talking to a simple command interpreter. We can then execute a script to start the network protocol stack, which communicates to the corporate Ethernet using the same serial line and relaying packets using the VirtualPC NIC driver. As shown in the figure, the system uses the DHCP protocol to acquire an IP address for itself, and the IP addresses of the DNS servers.

We then start a Web server service as the last command in the picture.



```
Setting environment for using Microsoft Visual Studio 2005 x86 tools.

E:\FPGA\eMIPS\fpga_side\data>serplexd -s
Will talk to a BigEndian client (mips,ppc,...)
Console Thread ...
GPIO=3
> dir fat
404.htm
arrow.gif
contents.htm
header.htm
image01.bmp
image02.bmp
image03.bmp
image04.bmp
image05.bmp
image06.bmp
image07.bmp
index.htm
large.htm
main.htm
mypage.htm
simple.htm
sound.wav
xilinx.sys
ml401/
XILINX/
mmlite/
eMIPS.jpg
neil.jpg
sandro.jpg
jvh.jpg
bharat.jpg
Brandon.jpg
liu.jpg
oscar.jpg
sibin_mohan.jpg
Special.jpg
b26.htm
default.htm
esg.htm
nas.htm
phil_face.jpg
warren.jpg
karln.jpg
hong.jpg
Nishith.jpg
> source fs\net.tzk
dhcp_sub.c:Could not bind to file hostname.txt
Interface sr0 has IP address 157.56.160.114.
The DHCP information is valid, our lease will expire in 691200 seconds.
The IP address 157.56.160.114 was leased from the DHCP Server 157.54.14.184
Default route through gateway 157.56.160.1 with subnet mask 255.255.254.0
There are 3 DNS server(s)
        239.255.255.253
        157.54.14.178
        157.54.14.162
The domain name on this interface is 'redmond.corp.microsoft.com'
> http.cob
>
```

Moving to a separate PC, we can start a web browser and point to the IP address of the ML401 board, resulting in the following output.



Contents
- eMIPS is...
- Brought to you by...
- DemoFest Booth 26!! Kudos to...
- Embedded Systems Group
- MSR
- ESG Web
- eMIPS Web
- Broken Link

Extensible Peripherals for eMIPS

Bharat Sukhwani
Boston University

Richard Neil Pittman, Alessandro Forin
Microsoft Research

Abstract

The area limitations of the XC4LX25 device on which eMIPS is implemented have spurred innovative solutions in the I/O peripherals support. Rather than permanently devoting area resources to the peripherals in the core section of the microcomputer (the TISA), eMIPS peripherals can be allocated to the extension area and loaded and unloaded dynamically, as required by the software application. At the summit we demonstrated the overall scheme in action by loading and unloading a timer under software control. The demonstration was performed using the Giano simulator. We also showed the progress already made towards implementing this functionality on the XC4LX25 by means of a simple extension that makes use of the additional privileged interface signals that are required for I/O peripherals to access interrupts and physical memory, to be configured into the system memory map, all the while maintaining the security and trust levels of the core TISA module.

1 Introduction

Dynamic loading and unloading of on-chip peripherals is performed by the system software using the extension slots in the eMIPS processor. System software initiates the loading of the extension peripheral by sending a signal to the Xilinx System Ace FPGA control chip, which in turn loads the corresponding bit file for the extension into the *"soft fabric*" of the reconfigurable hardware (Appendix A:Poster 5). The extension then sends an interrupt signal to the on-chip Bus Master (on the *"hard fabric"*) to indicate that it needs to be configured. The Bus Master acts as a bridge between the extension peripheral and the memory controller and is used to configure the extension peripheral. All the resources requested by the peripheral (including the interrupt request, if any) are passed to the memory controller via the Bus Master. When the peripheral is configured it communicates directly with the memory controller, bypassing the Bus Master. Once the system software is finished using the peripheral, it can unload it, freeing up both the portion of the memory address space used by the peripheral as well as the reconfigurable extension slot which can then be reused by another extension.

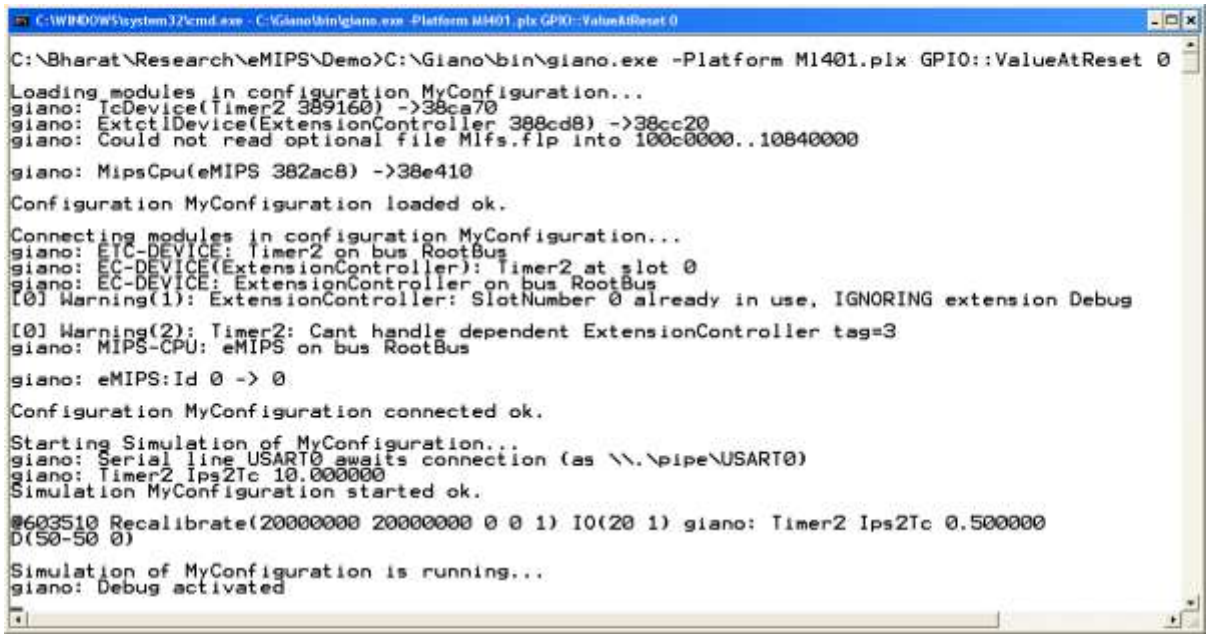
2 Demonstrations

2.1 Load and Unload Simulation

Our first demonstration presented the overall plan for extensible on-chip peripherals. We presented a test performing the processes of loading an extension peripheral and then unloading it. The software program "pload" was compiled for eMIPS and then executed on the Giano simulator, configured for the ML401 board.

The Giano simulation of the eMIPS processor is started by executing the following command

```
> giano.exe -Platform Ml401.plx GPIO::ValueAtReset 0
```



```
C:\WINDOWS\system32\cmd.exe - C:\Giano\bin\giano.exe -Platform M1401.plx GPIO::ValueAtReset 0

C:\Bharat\Research\eMIPS\Demo>C:\Giano\bin\giano.exe -Platform M1401.plx GPIO::ValueAtReset 0

Loading modules in configuration MyConfiguration...
giano: IcDevice(Timer2 389160) ->38ca70
giano: ExtctlDevice(ExtensionController 388cd8) ->38cc20
giano: Could not read optional file Mlfs.flp into 100c0000..10840000

giano: MipsCpu(eMIPS 382ac8) ->38e410

Configuration MyConfiguration loaded ok.

Connecting modules in configuration MyConfiguration...
giano: ETC-DEVICE: Timer2 on bus RootBus
giano: EC-DEVICE(ExtensionController): Timer2 at slot 0
giano: EC-DEVICE: ExtensionController on bus RootBus
[0] Warning(1): ExtensionController: SlotNumber 0 already in use, IGNORING extension Debug

[0] Warning(2): Timer2: Cant handle dependent ExtensionController tag=3
giano: MIPS-CPU: eMIPS on bus RootBus

giano: eMIPS:Id 0 -> 0

Configuration MyConfiguration connected ok.

Starting Simulation of MyConfiguration...
giano: Serial line USART0 awaits connection (as \\.\pipe\USART0)
giano: Timer2 Ips2Tc 10.000000
Simulation MyConfiguration started ok.

@603510 Recalibrate(20000000 20000000 0 0 1) IO(20 1) giano: Timer2 Ips2Tc 0.500000
D(50-50 0)

Simulation of MyConfiguration is running...
giano: Debug activated
```

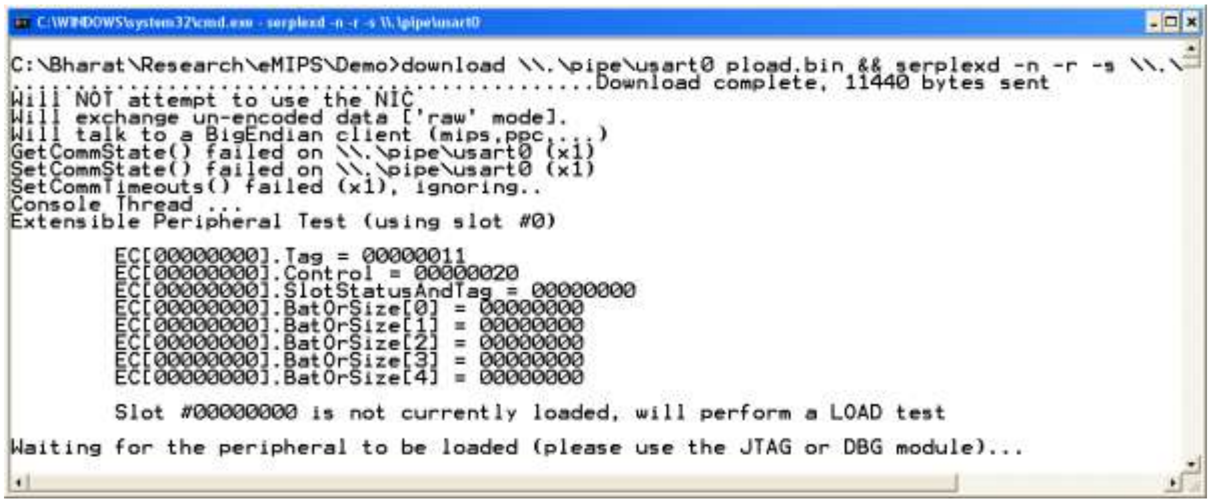
Figure

Here, `giano.exe` is the executable for the Giano simulator, and the file. `Ml401.plx` represents the eMIPS system and contains block level representations of the different modules (baseline processor, memory module, memory bus, extensions, peripherals etc.); each block in turn corresponding to a .dll file that implements the functionality of that block. The `ValueAtReset` attribute sets the values of the GPIO pins on the board to zero, which indicates to the boot loader that it should download executable code from the serial line, rather than jumping directly to the code in the on-board FLASH. Figure 1 shows the output on the command window after running the above command.

Once Giano is started the eMIPS boot loader waits for the software program to be downloaded, which is done using the following command:

```
> download \\.\pipe\usart0 pload.bin && serplexd -n -r -s \\.\pipe\usart0
```

The corresponding command window is shown in Figure 2

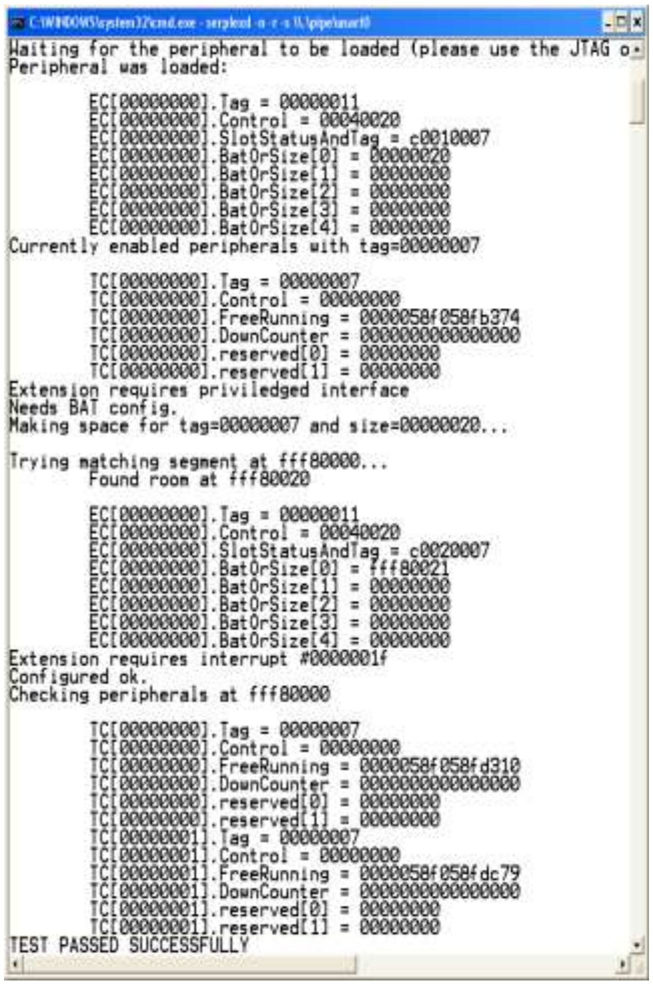

C:\WINDOWS\system32\cmd.exe - serplexd -n -r -s \\.\pipe\usart0

```
C:\Bharat\Research\eMIPS\Demo>download \\.\pipe\usart0 pload.bin && serplexd -n -r -s \\.\
.............................................Download complete, 11440 bytes sent
Will NOT attempt to use the NIC
Will exchange un-encoded data ['raw' mode].
Will talk to a BigEndian client (mips,ppc,...)
GetCommState() failed on \\.\pipe\usart0 (x1)
SetCommState() failed on \\.\pipe\usart0 (x1)
SetCommTimeouts() failed (x1), ignoring..
Console Thread ...
Extensible Peripheral Test (using slot #0)

        EC[00000000].Tag = 00000011
        EC[00000000].Control = 00000020
        EC[00000000].SlotStatusAndTag = 00000000
        EC[00000000].BatOrSize[0] = 00000000
        EC[00000000].BatOrSize[1] = 00000000
        EC[00000000].BatOrSize[2] = 00000000
        EC[00000000].BatOrSize[3] = 00000000
        EC[00000000].BatOrSize[4] = 00000000

        Slot #00000000 is not currently loaded, will perform a LOAD test

Waiting for the peripheral to be loaded (please use the JTAG or DBG module)...
```

Figure

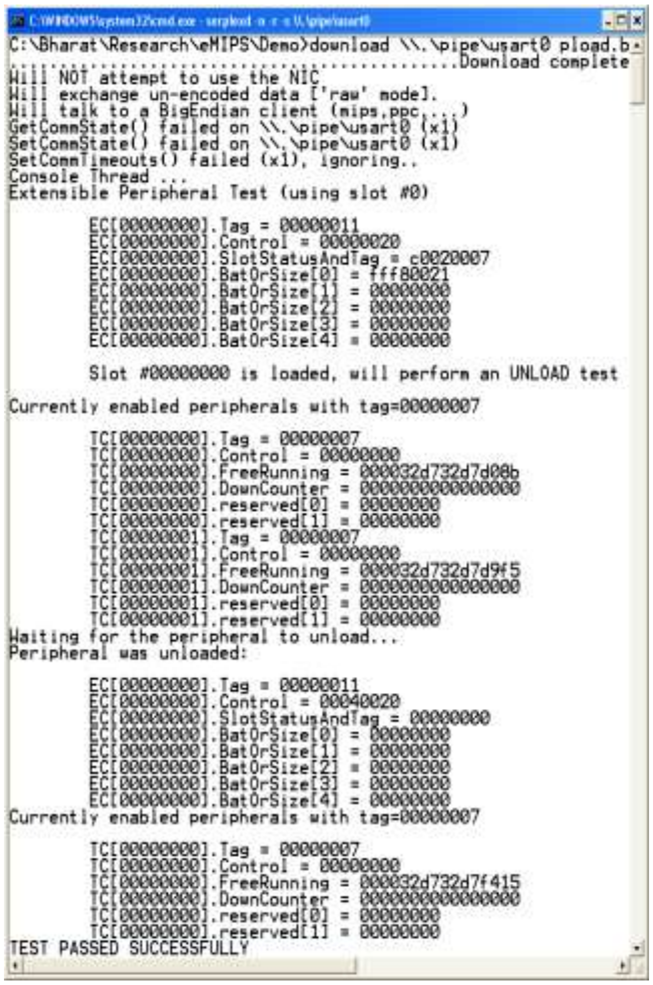
`Pload.bin` is the software test program used to perform the loading/unloading of peripherals using extension slot number 0. If the program finds that the peripheral is not loaded, it performs the peripheral load test otherwise it unloads the peripheral. The screenshot of Figure 2 shows that the program initially finds that there is no peripheral loaded in extension slot 0, and the program therefore proceeds with the load test.

When an extension peripheral is loaded, the Extension Controller (EC) receives an indication (from the extension) as to the presence of an extension in slot 0. The EC in turn raises an interrupt signal to indicate to the system software that the peripheral needs to be configured. In the current demo, we pass an input “I1 0” to the debug module inside the Giano simulator (previous command window, Figure 1) to simulate the hardware indication, so that the software program can then continue with the configuration of the peripheral. Once EC receives this input, the PLOAD program starts the actual


C:\WINDOWS\system32\cmd.exe - serplexd -n -r -s \\.\pipe\usart0

```
Waiting for the peripheral to be loaded (please use the JTAG o
Peripheral was loaded:

        EC[00000000].Tag = 00000011
        EC[00000000].Control = 00040020
        EC[00000000].SlotStatusAndTag = c0010007
        EC[00000000].BatOrSize[0] = 00000020
        EC[00000000].BatOrSize[1] = 00000000
        EC[00000000].BatOrSize[2] = 00000000
        EC[00000000].BatOrSize[3] = 00000000
        EC[00000000].BatOrSize[4] = 00000000
Currently enabled peripherals with tag=00000007

        TC[00000000].Tag = 00000007
        TC[00000000].Control = 00000000
        TC[00000000].FreeRunning = 0000058f058fb374
        TC[00000000].DownCounter = 0000000000000000
        TC[00000000].reserved[0] = 00000000
        TC[00000000].reserved[1] = 00000000
Extension requires priviledged interface
Needs BAT config.
Making space for tag=00000007 and size=00000020...

Trying matching segment at fff80000...
        Found room at fff80020

        EC[00000000].Tag = 00000011
        EC[00000000].Control = 00040020
        EC[00000000].SlotStatusAndTag = c0020007
        EC[00000000].BatOrSize[0] = fff80021
        EC[00000000].BatOrSize[1] = 00000000
        EC[00000000].BatOrSize[2] = 00000000
        EC[00000000].BatOrSize[3] = 00000000
        EC[00000000].BatOrSize[4] = 00000000
Extension requires interrupt #0000001f
Configured ok.
Checking peripherals at fff80000

        TC[00000000].Tag = 00000007
        TC[00000000].Control = 00000000
        TC[00000000].FreeRunning = 0000058f058fd310
        TC[00000000].DownCounter = 0000000000000000
        TC[00000000].reserved[0] = 00000000
        TC[00000000].reserved[1] = 00000000
        TC[00000001].Tag = 00000007
        TC[00000001].Control = 00000000
        TC[00000001].FreeRunning = 0000058f058fdc79
        TC[00000001].DownCounter = 0000000000000000
        TC[00000001].reserved[0] = 00000000
        TC[00000001].reserved[1] = 00000000
TEST PASSED SUCCESSFULLY
```

Figure


C:\WINDOWS\system32\cmd.exe - serplexd -n -r -s \\.\pipe\usart0

```
C:\Bharat\Research\eMIPS\Demo>download \\.\pipe\usart0 pload.b
..............................................Download complete
Will NOT attempt to use the NIC
Will exchange un-encoded data ['raw' mode].
Will talk to a BigEndian client (mips,ppc,...)
GetCommState() failed on \\.\pipe\usart0 (x1)
SetCommState() failed on \\.\pipe\usart0 (x1)
SetCommTimeouts() failed (x1), ignoring..
Console Thread ...
Extensible Peripheral Test (using slot #0)

        EC[00000000].Tag = 00000011
        EC[00000000].Control = 00000020
        EC[00000000].SlotStatusAndTag = c0020007
        EC[00000000].BatOrSize[0] = fff80021
        EC[00000000].BatOrSize[1] = 00000000
        EC[00000000].BatOrSize[2] = 00000000
        EC[00000000].BatOrSize[3] = 00000000
        EC[00000000].BatOrSize[4] = 00000000

        Slot #00000000 is loaded, will perform an UNLOAD test

Currently enabled peripherals with tag=00000007

        TC[00000000].Tag = 00000007
        TC[00000000].Control = 00000000
        TC[00000000].FreeRunning = 000032d732d7d08b
        TC[00000000].DownCounter = 0000000000000000
        TC[00000000].reserved[0] = 00000000
        TC[00000000].reserved[1] = 00000000
        TC[00000001].Tag = 00000007
        TC[00000001].Control = 00000000
        TC[00000001].FreeRunning = 000032d732d7d9f5
        TC[00000001].DownCounter = 0000000000000000
        TC[00000001].reserved[0] = 00000000
        TC[00000001].reserved[1] = 00000000
Waiting for the peripheral to unload...
Peripheral was unloaded:

        EC[00000000].Tag = 00000011
        EC[00000000].Control = 00040020
        EC[00000000].SlotStatusAndTag = 00000000
        EC[00000000].BatOrSize[0] = 00000000
        EC[00000000].BatOrSize[1] = 00000000
        EC[00000000].BatOrSize[2] = 00000000
        EC[00000000].BatOrSize[3] = 00000000
        EC[00000000].BatOrSize[4] = 00000000
Currently enabled peripherals with tag=00000007

        TC[00000000].Tag = 00000007
        TC[00000000].Control = 00000000
        TC[00000000].FreeRunning = 000032d732d7f415
        TC[00000000].DownCounter = 0000000000000000
        TC[00000000].reserved[0] = 00000000
        TC[00000000].reserved[1] = 00000000
TEST PASSED SUCCESSFULLY
```

Figure

configuration process.

Figure 3 shows the output from PLOAD during the extension load test. The program initially prints the contents of the registers of the EC and TC peripherals. EC represents the extension controller and is used to control the configuration of the extension peripheral. TC represents the Timer Controller, which is the type of extension peripheral being configured in this demonstration. As shown in the figure, once the peripheral is loaded PLOAD looks at the EC state to identify the resources required. The SlotStatusAndTag register shows that the peripheral has a tag 7 (which classifies it at a Timer) and requests privileged access as well as interrupts. The BatOrSize [0] register shows that the peripheral requires 32 bytes of address space. The memory controller first checks if a peripheral with the same tag already exist, and if so, it tries to allocate space for the new peripheral at a memory address contiguous to the existing peripheral. In the present test, a timer peripheral already exists (TC [0] in Figure 3) and hence PLOAD maps the new timer to a location next to the current one. To do this, software updates the BatOrSize [0] register of the EC to indicate that the starting address of the extension peripheral is address 0xfff80020, with the valid bit (least significant bit) set to 1. PLOAD then moves the peripheral to the “Run” state and configuration is complete. The test finally indicates that it passed successfully and the memory map indicates now two timer controllers TC [0] and TC [1].

Figure 4 shows the output on the command window when the test is run again. In this case, the peripheral is already loaded in slot 0 and the test will therefore try to unload it. PLOAD writes to the SlotStatusAndTag register a new value for the state machine that controls the peripheral (see [Appendix A Poster 5]). As indicated, writing a value of Absent will inactivate the peripheral and notify the EC of the event. PLOAD notices this event ("Peripheral was unloaded" in Figure 4) and dumps the content of EC. The SlotStatusAndTag register shows the extension slot 0 is now free, because the extension peripheral was successfully unloaded. This leaves only a single timer controller TC[0] in the memory map.

2.2 Accessing Physical Memory from an Extension

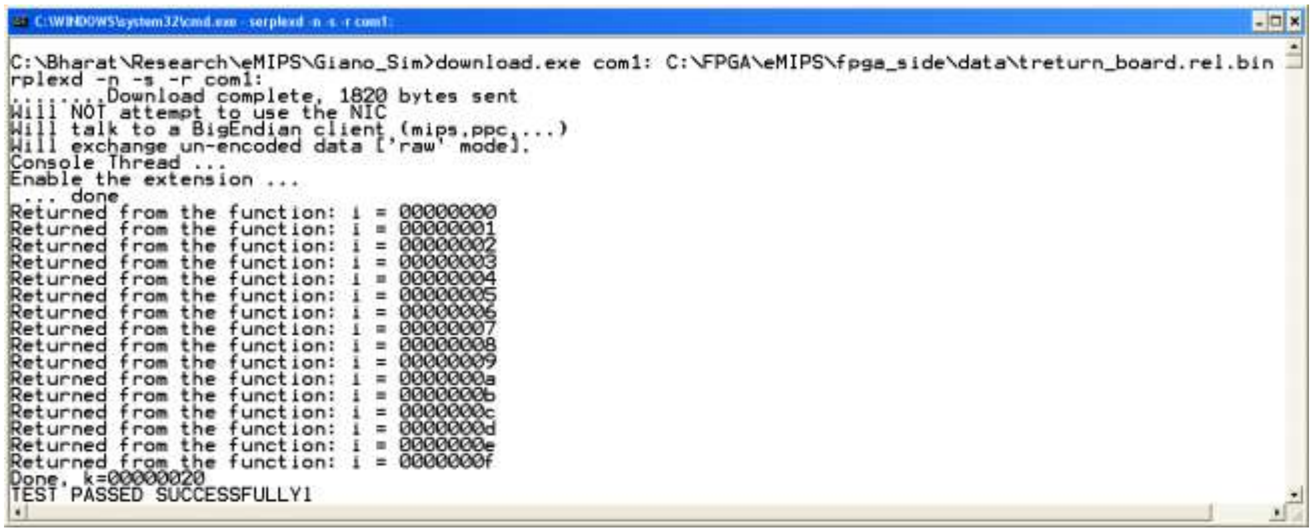
Our second demonstration presented our progress towards implementing the extensible on-chip peripherals on the FPGA. Some peripherals need privileged physical memory access to the system memory bus. The purpose of this demo is to show the capability of memory access from the extension. This feature has been successfully implemented, as shown by the current demo.

We augmented the eMIPS processor ISA with an extended instruction that replaces the basic block executed for returning from a function call. The basic block consists of the following instructions

```
lw      ra      (immediate - 8)(sp)
jr      ra
addiu   sp      sp      immediate
```

The first instruction loads the return address at an offset from the current value of the stack pointer register. The second instruction jumps to the return address. The third instruction is the so-called "delay slot", on the MIPS architecture it is also executed as part of the previous jump instruction. In this case the instruction pops the stack frame.
The extended instruction used to replace that basic block is

```
op7c   ra     sp     immediate
```



```
C:\Bharat\Research\eMIPS\Giano_Sim>download.exe com1: C:\FPGA\eMIPS\fpga_side\data\treturn_board.rel.bin
rplexd -n -s -r com1:
........Download complete, 1820 bytes sent
Will NOT attempt to use the NIC
Will talk to a BigEndian client (mips,ppc,...)
Will exchange un-encoded data ['raw' mode].
Console Thread ...
Enable the extension ...
 ... done
Returned from the function: i = 00000000
Returned from the function: i = 00000001
Returned from the function: i = 00000002
Returned from the function: i = 00000003
Returned from the function: i = 00000004
Returned from the function: i = 00000005
Returned from the function: i = 00000006
Returned from the function: i = 00000007
Returned from the function: i = 00000008
Returned from the function: i = 00000009
Returned from the function: i = 0000000a
Returned from the function: i = 0000000b
Returned from the function: i = 0000000c
Returned from the function: i = 0000000d
Returned from the function: i = 0000000e
Returned from the function: i = 0000000f
Done. k=00000020
TEST PASSED SUCCESSFULLY!
```

Figure

The extended instruction encodes the three parameters that define the original basic block, namely the size of the stack frame, the register that acts as stack pointer and the one that acts as the return address.
Our demo runs a simple program that performs a function call in a loop. After every return from the function, the calling function prints a string indicating that the control has returned. We patched the binary of this program (using the automatic binary patching tool part of the **bbtools**) to insert the extended instruction before the above mentioned basic

block. We then run this binary on the eMIPS processor implemented on the Xilinx FPGA on the ML401 development board.
Figure 5 shows the result of running the demo program on the board. `Treturn.rel.bin` is the patched binary which is downloaded to the FPGA board using the *com1:* serial line. As shown, the program calls the function 16 times and each time prints a string after returning from the call. After the loop has been executed, the program displays the success message.

Project P2V: An Architecture for Zero-Overhead Online Verification of Software Programs

Hong Lu
Texas A&M University

Alessandro Forin
Microsoft Research

Abstract

The PSL-to-Verilog (P2V) compiler translates a set of assertions about a block-structured software program into a hardware design to be executed concurrently with the execution of the software program. The assertions validate the correctness of the program without altering its temporal behavior in any way, a result that has never been previously achieved by any online model-checking system. The assertions are expressed in the simple subset of the IEEE Standard Property Specification Language PSL and the target execution system is the eMIPS processor, a dynamically self-extensible processor. At the summit we demonstrated a working prototype of the P2V compile, implemented in Python. The compiler generates code for the implementation of the eMIPS processor running on the Xilinx ML401 development board. The input to the compiler is either a PSL file or it is typed by the user on the Python interpreter's interactive window. The output is a set of Verilog modules that implement the Verification Unit.

1 Introduction

In project P2V, we concern ourselves with zero-overhead monitors for software programs, monitors whose execution does not interfere with the target programs. Zero-overhead monitoring involves both passive observation and online verification. Passive observation refers to the non-intrusive collection of relevant information from an executing program, and online verification refers to the detection of requirement violations on-the-fly at the earliest possible moment using the collected information.

Most existing monitoring techniques are based on code instrumentation. Such techniques cause the unavoidable probe effect which changes the timing behavior of the target program. P2V takes a different approach, where the monitoring task is performed by a separate piece of hardware which is generated automatically from the PSL specification. As a result, the executable of the software program does not need to be modified, and furthermore, its temporal behavior does not change in any way.

Software correctness specifications are expressed in sPSL, a subset of PSL adapted for C requirement specifications. The dataflow of P2V is shown in Figure . The top side of the diagram depicts the normal compilation flow for C, using the standard compiler and tools and resulting in the executable image file. The bottom part of the diagram shows the symmetric flow for the sPSL specifications, where the P2V compiler translates sPSL assertions into Verilog code. Then the manufacturer's FPGA tools take this Verilog source file and create the binary file that is actually used for partial reconfiguration of the FPGA at run time, when the program is started. When compiling a sPSL specification, P2V uses the symbol table generated by the C compiler to keep track of C objects.

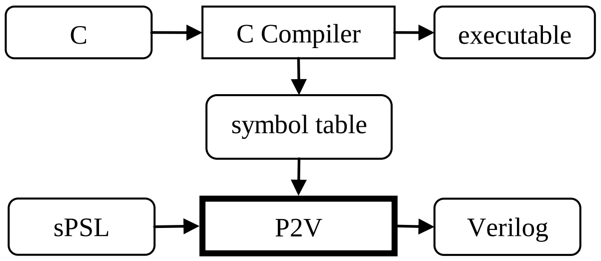


Figure : P2V data flow

The execution platform for the compiled C code and its Verilog monitor is eMIPS, a dynamically extensible processor implemented on FPGAs. eMIPS allows multiple extensions of a MIPS processor to load dynamically and to plug into the stages of a pipelined CPU data path. The zero-overhead monitor unit, abbreviated as MU from this point on, is deployed as one of the extensions. MU has two major components, an observing unit OU and a verification unit VU, as shown in Figure . OU is closely integrated with the eMIPS core data path, and can passively access all relevant signals and registers, including the program counter, the stack pointer, the current instruction register, memory write addresses and values, and the general purpose registers. In Figure 2, this interaction is depicted by the input signals PC, INSTR and MEMVAL among others. VU verifies sPSL assertions using the observations collected by OU. It takes a list of atomic propositions a_1, a_2, …, a_n as inputs (generated by OU), and outputs two signals VIOLATED and SATISFIED.

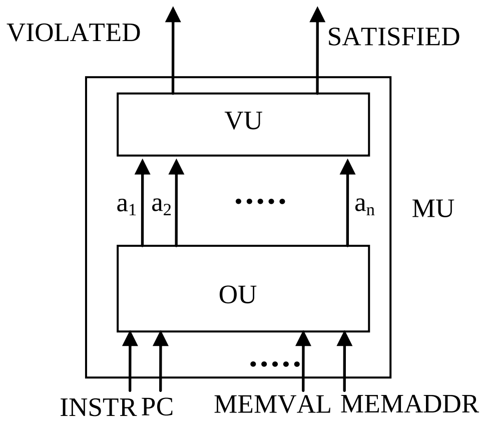


Figure : MU architecture

Besides being completely transparent, the other distinctive feature of P2V is its flexibility. As an extension to eMIPS core, the logic of MU can be synthesized on a per-program basis. Furthermore, MU is loaded and executed at runtime together with the target program. More than one program can be executing on the same microprocessor under system software control. This flexibility is achieved via the dynamic partial reconfiguration capabilities of modern FPGAs, something that is simply not possible for ASIC platforms.

2 Project Demo

In our demo, we used a simple traffic light control program as an example (the C code is omitted due to space limit). The requirement that if a green light should be followed by a yellow light can be specified in sPSL as: `always (g imply (not r) until y)`. The following demo session (from a Python interpreter) shows the Verilog code of the verification unit generated for this sPSL property.

```
>>> demo()
```

```
Input a PSL property: always (g imply (not r) until y)
Output Verilog code:
module VU(CLK,g,r,y,SCOPE,SATISFIED,VIOLATED);
    input CLK;
    input g,r,y;
    input SCOPE;

    output SATISFIED,VIOLATED;
    reg SATISFIED,VIOLATED;

    reg[20:0] STATE;
    wire[2:0] OBSERVATION={g,r,y};

    always @(posedge SCOPE)
    begin
        STATE = 0;
        SATISFIED = 0;
        VIOLATED = 0;
    end

    always @(negedge SCOPE)
        case (STATE)
            0: STATE=1;
            1: STATE=1;
            2: STATE=2;
            3: STATE=1;
        endcase

    always @(posedge CLK)
        if (SCOPE)
            case({STATE,OBSERVATION})
                {21'd0,3'b111}: STATE=0;
                {21'd0,3'b011}: STATE=0;
                {21'd0,3'b101}: STATE=0;
                {21'd0,3'b001}: STATE=0;
                {21'd0,3'b110}: STATE=2;
                {21'd0,3'b010}: STATE=0;
                {21'd0,3'b100}: STATE=3;
                {21'd0,3'b000}: STATE=0;
                {21'd2,3'b111}: STATE=2;
                {21'd2,3'b011}: STATE=2;
                {21'd2,3'b101}: STATE=2;
                {21'd2,3'b001}: STATE=2;
                {21'd2,3'b110}: STATE=2;
                {21'd2,3'b010}: STATE=2;
                {21'd2,3'b100}: STATE=2;
                {21'd2,3'b000}: STATE=2;
                {21'd3,3'b111}: STATE=0;
                {21'd3,3'b011}: STATE=0;
                {21'd3,3'b101}: STATE=0;
                {21'd3,3'b001}: STATE=0;
                {21'd3,3'b110}: STATE=2;
                {21'd3,3'b010}: STATE=2;
                {21'd3,3'b100}: STATE=3;
                {21'd3,3'b000}: STATE=3;
            endcase

    always @(posedge CLK)
        if (STATE == 1) SATISFIED = 1;

    always @(posedge CLK)
        if (STATE == 2) VIOLATED = 1;
endmodule
```

In the generated Verilog code, module VU takes signal CLK, g, r, y, SCOPE as input and outputs signal SATISFIED and VIOLATED. The positive and negative edge of SCOPE indicate the start and end of monitoring, and signals g, r, y indicate the status of the green, red, and yellow lights. The main body of VU implements a verification finite state machine, where state 0, 1, and 2 are the initial, satisfied and violated state respectively.

Besides outputting the Verilog code for VU, P2V also generates the following components of OU, including the monitor activator, stack pointer observer, and scope detector.

```
module monitor_activator(CLK,RESET,ENEXT_EXT,RI,PI);
    input CLK,RESET,ENEXT_EXT;
    output RI, PI;
    reg RI, PI;

    initial
    begin
        RI = 0;
        PI = 0;
    end

    // request monitoring
    always@(posedge RESET)
    begin
        RI = 1;
        PI = 0;
    end

    always@(posedge ENEXT_EXT)
     begin
        RI = 0;
        PI = 1;
        @(posedge CLK)
        RI = 1;
        PI = 0;
    end
endmodule

module SP_observer(CLK,WRREG1_RG,WRDATA1_RG,WRREG2_RG,WRDATA2_RG,SP);
    input CLK;
    input [4:0] WRREG1_RG, WRREG2_RG;
    input [31:0] WRDATA1_RG, WRDATA2_RG;
    output [31:0] SP;
    reg [31:0] SP;

    initial   SP = 0;

    always @ (WRREG1_RG)
        if (WRREG1_RG == 29) //$sp = 29
            @ (posedge CLK) SP = WRDATA1_RG;

    always @ (WRREG2_RG)
        if (WRREG2_RG == 29) //$sp = 29
            @ (posedge CLK) SP = WRDATA2_RG;
endmodule

module scope_detector(PC,SP,ENEXT_EXT,SCOPE);
    input [31:0] PC, SP;
    input ENEXT_EXT;
    reg [31:0] FP;
    output SCOPE;
    reg SCOPE;

    initial
    begin
        SCOPE = 0;
        FP = 0;
    end

    //when PC = func, it's a scope entry
```

```
   always @ PC
       if (PC == 32'h800003e4)
          if (!SCOPE)
          begin
             FP = SP;
             SCOPE = 1;
          end

   //when SP = FP, it's a scope exit
   always @ SP
      if (SCOPE & (FP == SP))
         SCOPE = 0;

   always @ (posedge ENEXT_EXT)
      SCOPE = 0;
endmodule
```

Project M2V

Karl Meier
University of Washington

Alessandro Forin
Microsoft Research

Abstract

The MIPS-to-Verilog (M2V) compiler translates blocks of MIPS machine code into a hardware design represented in Verilog. The design constitutes an Extension for the eMIPS processor, a dynamically extensible processor realized on the Virtex-4 XC4LX25 FPGA. The Extension interacts closely with the basic pipeline of the microprocessor and recognizes special Extended instructions, instructions that are not part of the basic MIPS ISA. Each instruction is semantically equivalent to one or more blocks of MIPS code. The purpose of the M2V compiler is to automate the process of creating Extensions for the specific purpose of accelerating the execution of software programs.

At the summit we demonstrated the current state of the compiler, which is implemented as a C++ program. The compiler accepts as input basic blocks in the form generated by the eMIPS bbtools*, a set of programs for the analysis and instrumentation of MIPS ELF images. The parser recognizes the full MIPS ISA and creates a graph representing the dependencies between instructions and between registers. The semantic analysis is currently restricted to the R set of instructions. The scheduling algorithm is complete but the timing analysis is not yet sufficient to produce correct outputs.*

1 Introduction

The eMIPS platform consists of a standard MIPS pipeline and an extension unit (EU). The EU contains programmable logic that is used for extensions to the MIPS instruction set. These extensions are used to accelerate the execution of an application. The machine code for the extended instruction is inserted before the accelerated basic block in the MIPS binary. When the extended instruction completes, program execution will proceed at the address following the basic block or at the address of a branch target.

The objective for the M2V compiler is to automatically create the logic for eMIPS extensions using a .bbw file as the hardware specification. The .bbw file is a basic block that is extracted from a MIPS binary image using the bbtools. A basic block is a candidate for extension when it is identified as a block that takes considerable execution time when an application is running. The Giano simulator and the bbtools are used to identify the blocks that should be accelerated.

The M2V compiler generates synthesizable Verilog which is synthesized using the standard Xilinx place and route tool-chain to create a bit file that can be loaded onto the eMIPS platform. In an embedded platform, the extension can be loaded at power-up. In a more general purpose system, the extension can be dynamically loaded when a binary image is loaded. Dynamic loading of the extension requires partial reconfiguration of the programmable logic.

2 M2V Compiler Implementation

The M2V compiler is a multi-pass compiler. The first pass parses the .bbw file, semantically analyzes the MIPS instructions, and builds and connects nodes in a dependency graph. There are two types of nodes in the graph: register nodes and instruction nodes. The semantic analysis provides the function and cost for each instruction and also the register dependencies. The register nodes represent a register access which could go to the register file or to a temporary storage location in the EU. The register table tracks whether a register has already been read from the register file, where the last update to the register is stored, and whether the register needs to be written back to the register file. When multiple instructions read the same unchanged register value, the register table provides information so only a single register node is created. Register nodes may or may not result in an actual clocked hardware register. The final schedule for the extension determines when pipeline stages are added and whether a register node will result in a hardware register.

A major challenge for the M2V compiler is to constrain the EU such that it does not interfere with instructions flowing through the eMIPS pipeline before and after the extended instruction executes. eMIPS uses a standard five stage RISC pipeline, with IF, ID, EX, MA, and WB stages. This pipeline is tightly integrated with the EU. An extended instruction will take multiple cycles to execute since it is semantically equivalent to all of the MIPS instructions in a basic block. During ID, the extension will snoop the register reads that are visible to the primary eMIPS pipeline. If the instruction is an extended instruction, the EU will claim the instruction and stall the instructions behind it while it executes. Instructions before the extended instruction complete normally and must have access to the same resources that they would normally use. Figure 1 shows how the instructions proceed through the eMIPS pipeline, where instruction *m* is the only extended instruction in the example.

Cycle Number	0	1	2	3	4	5	...	n+2	n+3	n+4	n+5	n+6	n+7	n+8
Instruction m-2	IF	ID	EX	MA	WB									
Instruction m-1		IF	ID	EX	MA	WB								
Extended Instruction m			IF	ID	EX1	EX2	...	Exn-2	EXn-1	EXn	MA	WB		
Instruction m+1				IF										
Instruction j									IF	ID	EX	MA	WB	
Instruction j+1										IF	ID	EX	MA	WB

Figure 1: Scheduling of instructions through the eMIPS Pipeline with the extended instruction (Instruction m)

During cycle 3, the EU will decode and claim the extended instruction, snoop the reads from the register file, store the register reads, and prepare to stall the trailing instructions in cycle 4. The instruction fetch in cycle 3 does not perform useful work since this instruction is the first instruction of the accelerated basic block. During cycle 4, instruction m-2 has control of the register write-back logic, instruction m-1 has control of the memory access logic, and the extended instruction begins stage EX1. In EX1, reads to the register file are controlled by the EU since future instructions are stalled. In EX2, the EU can read from the register file and access memory through the main memory unit. The EU is in steady-state from EX3 until EXn-2 and it can control all ports on the register file and access to the memory logic. In stage EXn-1, instruction j must be fetched and so any branch conditions and branch addresses must be resolved by this stage. In cycle n+4, the EU must relinquish control of the register read ports to instruction j which is in the ID stage. In cycle n+5, the EU performs its last memory access and can also write to the register file. In cycle n+6, the EU performs its last write to the register file and the extended instruction is complete.

The second pass of the M2V compiler creates the schedule for register and memory accesses by doing a constrained depth-first traversal of the dependency graph created in the first pass of compilation. The traversal begins at the register nodes and continues until a dependency can not be met. When the node can not be completed, it is placed on a queue to be traversed in the next cycle. The nodes with unmet dependencies at the end of a cycle mark where pipeline stages will be inserted. Constraints on the schedule are the register and memory resources available in a given cycle and the delay through a sequential stream of operations. The register-to-register delay is estimated from the complexity of the instruction and the fan-out of the register nodes. The semantic analyzer provides the complexity of the instruction and the dependency graph yields the fan-out from each node. When the delay exceeds the cycle-time threshold, a pipeline stage is added.

A third pass through the dependency graph generates the EU using synthesizable Verilog as the target.

3 Results

The M2V compiler has been developed using an example basic block from video games that implements part of a 64-bit division. The disassembly of the block is given below:

```
sll   r1,r1,1
srl   r3,r2,31
or    r1,r1,r3
sll   r2,r2,1
srl   r3,r4,31
or    r2,r2,r3
sll   r4,r4,1
srl   r3,r5,31
or    r4,r4,r3
sltu  r3,r1,r6
beq   r0,r3,40
sll   r5,r5,1
```

The intermediate output from the M2V compiler is listed below in Figure 2. The hand-generated pictorial representation of the dependency graph for this basic block is given in Figure 3.

```
> c:\fpga\BasicBlocks\m2v.exe -v small.bbw

Generating IL and Dependency Graph...
............

Depth-first walk through the dependency graph - infinite resources
NODE 1: Register
   [0x0]    Read r1 (canonical) from the RF
NODE 2: Instruction
   [0x0]    0x10840     sll   r1, r1, 1
NODE 3: Register
   [0x0]    Write r1 (canonical)
      used by: [0x8]

NODE 4: Register
   [0x4]    Read r2 (canonical) from the RF
      used by: [0xc]
NODE 5: Instruction
   [0x4]    0x21fc2     srl   r3, r2, 31
NODE 6: Register
   [0x4]    Write r3 (canonical)
      used by: [0x8]
NODE 7: Instruction
   [0x8]    0x230825    or    r1, r1, r3
NODE 8: Register
   [0x8]    Write r1 (canonical) to the RF
      used by: [0x24]
NODE 9: Instruction
   [0xc]    0x21040     sll   r2, r2, 1
NODE 10: Register
   [0xc]    Write r2 (canonical)
      used by: [0x14]

NODE 11: Register
   [0x10]   Read r4 (canonical) from the RF
      used by: [0x18]
NODE 12: Instruction
```

```
   [0x10]   0x41fc2     srl   r3, r4, 31
NODE 13: Register
   [0x10]   Write r3 (canonical)
      used by: [0x14]
NODE 14: Instruction
   [0x14]   0x431025    or    r2, r2, r3
NODE 15: Register
   [0x14]   Write r2 (canonical) to the RF
NODE 16: Instruction
   [0x18]   0x42040     sll   r4, r4, 1
NODE 17: Register
   [0x18]   Write r4 (canonical)
      used by: [0x20]

NODE 18: Register
   [0x1c]   Read r5 (canonical) from the RF
      used by: [0x2c]
NODE 19: Instruction
   [0x1c]   0x51fc2     srl   r3, r5, 31
NODE 20: Register
   [0x1c]   Write r3 (canonical)
      used by: [0x20]
NODE 21: Instruction
   [0x20]   0x832025    or    r4, r4, r3
NODE 22: Register
   [0x20]   Write r4 (canonical) to the RF
NODE 28: Instruction
   [0x2c]   0x52840     sll   r5, r5, 1
NODE 29: Register
   [0x2c]   Write r5 (canonical) to the RF

NODE 23: Register
   [0x24]   Read r6 (canonical) from the RF
NODE 24: Instruction
   [0x24]   0x26182b    sltu  r3, r1, r6
NODE 25: Register
   [0x24]   Write r3 (canonical) to the RF
      used by: [0x28]

NODE 26: Register
   [0x28]   Read r0 (canonical) from the RF
NODE 27: Instruction
   [0x28]   0x10030005  beq   r0, r3
```

Figure 2: Intermediate output from the M2V compiler for the input in Figure 1.

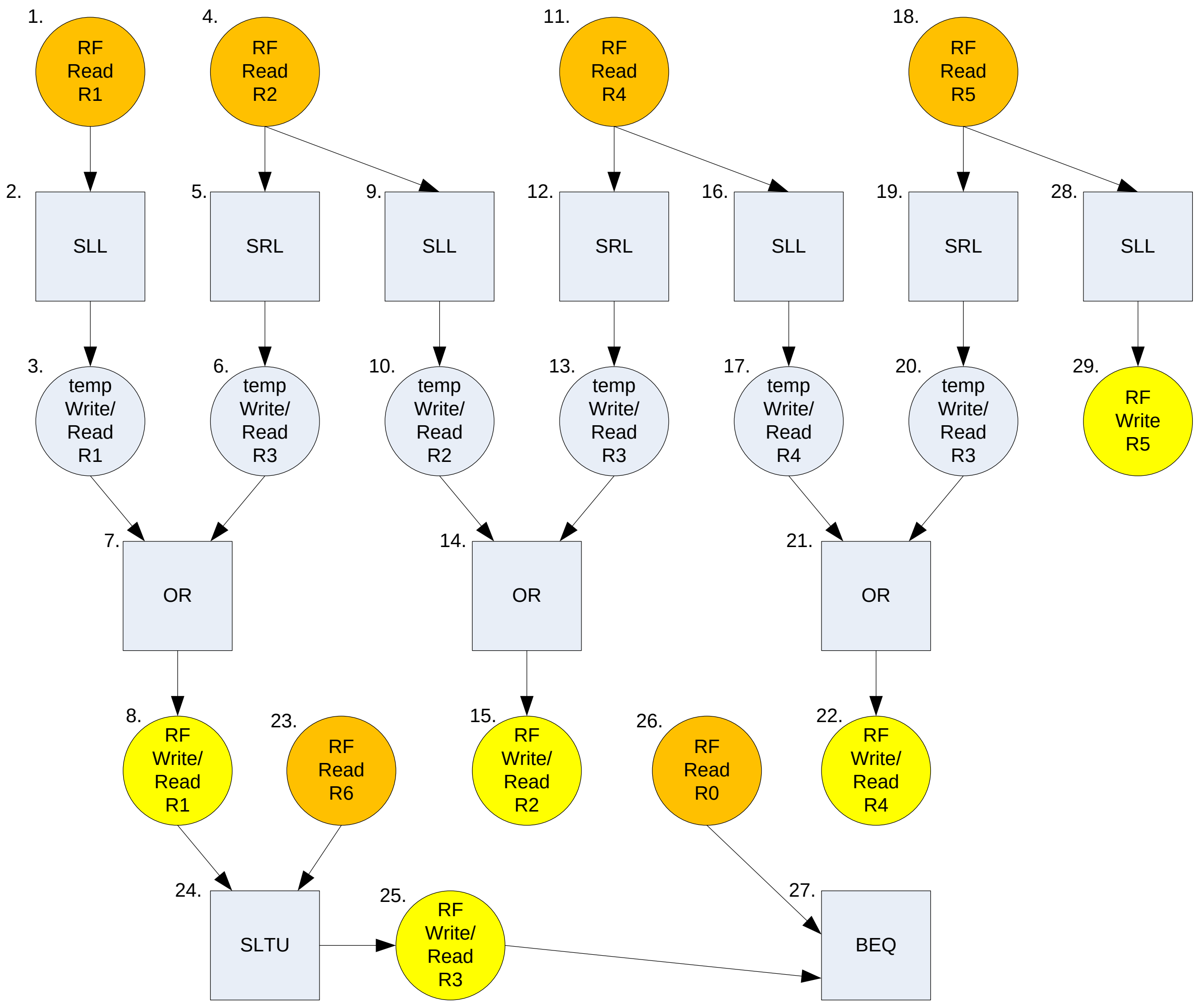


Figure 3: Dependency graph for the basic block in Figure 1.

4 Status of M2V Development

The M2V compiler will eventually automatically generate logic for eMIPS extended instructions. The final compiler will have three phases: one for graph creation, one for cycle selection, and one for emitting Verilog. At the time of this writing, the infrastructure for graph creation is complete but the semantic analysis for the full MIPS instruction set is not. The R instructions are implemented as well as some of the I instructions. The R instructions take 2 registers as operands and the I instructions take 1 register and an immediate as an operand. Pass 2 of the compiler, cycle selection, is in progress. Pass 3 has been architected but not yet implemented.

Connecting Microsoft Robotics Studio and Embedded Web Services

Oscar Almeida, Johannes Helander, Nishith Khantal
Microsoft Research

Henrik Nielsen
Microsoft Robotics

Abstract

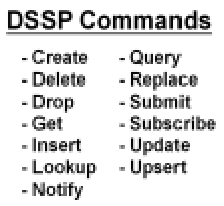
Microsoft Robotics Studio is a Windows-based environment that allows developers to easily create robotics applications across a wide variety of hardware. The environment is driven by the web services concept using the Decentralized Software Services Protocol (DSSP). The commands supported by DSSP provide an extension to the HTTP protocol, with provision for structured data retrieval, manipulation, and event notification. Our project essentially consists of implementing DSSP on the Microsoft Invisible Computing platform. The purpose is two-fold: (1) to show compatibility between Robotics Studio and an embedded system, and (2) to further develop and increase the interoperability of the Invisible web services platform. The DSSP commands were seamlessly inserted into the existing Invisible platform, meaning that the web services created by DSSP are now accessible to any other protocol such as WS-Management, adding to the system's interoperability. Ultimately, our goal is to provide a means of communication between Robotics Studio and a sensor via the Invisible platform. This demo was performed in an outpatient monitoring medical context. The angular velocity of a walking stick prototype, intended for use by the elderly, is measured by an embedded gyro sensor. Such measurements can be used to detect when the user may have fallen and generate an alarm to the medical personnel. The scope of this demo is to poll the gyro with a microcontroller and send the data over the network to Robotics Studio using DSSP.

1 The Walking Stick

The microcontroller used in the demo is the Atmel EB63 Arm7 board, which runs at 25MHz. This board is loaded with the Microsoft Invisible Computing platform. Tasks include polling the gyro and managing the web services used to communicate with Robotics Studio. Because the board has no Ethernet port, the Ethernet packets generated by the network protocol are sent via a serial line to the PC and are then forwarded using a Virtual NIC driver.

The gyro used is the Gyration MG1101 MicroGyro. It contains two axes of measurement and operates over a 2-wire serial bus. In this project, we placed the gyro inside and at the base of the walking stick prototype, shown at right. The gyro measures the angular velocity of the walking stick.

The Robotics Studio DSSP commands used by the walking stick are:

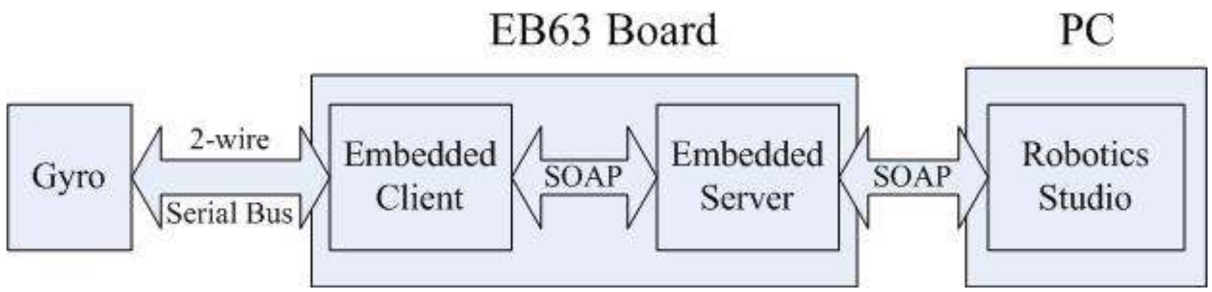


DSSP Commands

- Create	- Query
- Delete	- Replace
- Drop	- Submit
- Get	- Subscribe
- Insert	- Update
- Lookup	- Upsert
- Notify	

2 Demonstration

Three parties are involved in the communications, as shown in the picture. "Embedded Client" is a local application running on the board, which is the software that actually controls the gyro. It communicates locally with the "Embedded Server" via the loopback network interface. "Robotics Studio" runs on a PC and communicates with "Embedded Server" using the Ethernet link and the IP address 172.31.43.192.



At the start, Embedded Client asks Embedded Server to create a web service for itself. This service is called 'TEST' and will serve to hold the gyro data. The SOAP messages exchanged are as follows.

Embedded Client: Create Request

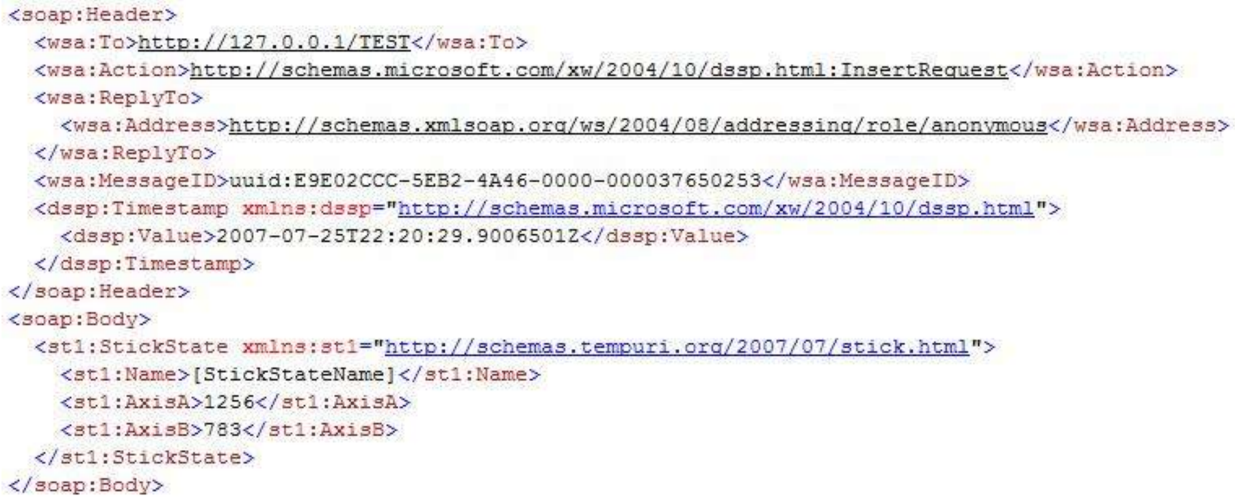
```
<soap:Header>
  <wsa:To>http://127.0.0.1/TEST</wsa:To>
  <wsa:Action>http://schemas.microsoft.com/xw/2004/10/dssp.html:CreateRequest</wsa:Action>
  <wsa:ReplyTo>
    <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address>
  </wsa:ReplyTo>
  <wsa:MessageID>uuid:E9E02CCC-5EB2-4A46-0000-000037650253</wsa:MessageID>
  <dssp:Timestamp xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html">
    <dssp:Value>2007-07-25T22:20:29.7901361Z</dssp:Value>
  </dssp:Timestamp>
</soap:Header>
```

Embedded Server: Create Response

```
<soap:Header>
  <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:To>
  <wsa:Action>http://schemas.microsoft.com/xw/2004/10/dssp.html:CreateResponse</wsa:Action>
  <wsa:RelatesTo>uuid:E9E02CCC-5EB2-4A46-0000-000037650253</wsa:RelatesTo>
  <wsa:MessageID>uuid:F9A17CD5-44A1-3A29-0113-526972419783</wsa:MessageID>
  <dssp:Timestamp xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html">
    <dssp:Value>2007-07-25T22:20:29.8686460Z</dssp:Value>
  </dssp:Timestamp>
</soap:Header>
```

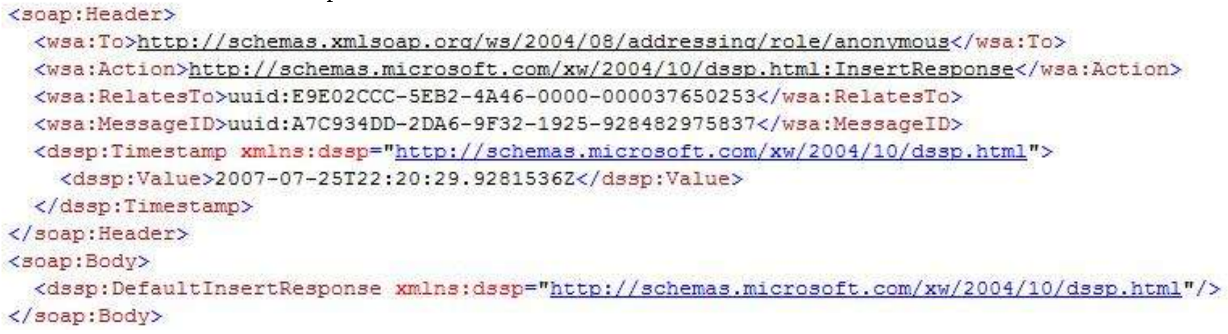
Embedded Client then inserts the initial state of the gyro into the 'TEST' web service on Embedded Server. The messages are as follows.

Embedded Client: Insert Request



```
<soap:Header>
  <wsa:To>http://127.0.0.1/TEST</wsa:To>
  <wsa:Action>http://schemas.microsoft.com/xw/2004/10/dssp.html:InsertRequest</wsa:Action>
  <wsa:ReplyTo>
    <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address>
  </wsa:ReplyTo>
  <wsa:MessageID>uuid:E9E02CCC-5EB2-4A46-0000-000037650253</wsa:MessageID>
  <dssp:Timestamp xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html">
    <dssp:Value>2007-07-25T22:20:29.9006501Z</dssp:Value>
  </dssp:Timestamp>
</soap:Header>
<soap:Body>
  <st1:StickState xmlns:st1="http://schemas.tempuri.org/2007/07/stick.html">
    <st1:Name>[StickStateName]</st1:Name>
    <st1:AxisA>1256</st1:AxisA>
    <st1:AxisB>783</st1:AxisB>
  </st1:StickState>
</soap:Body>
```

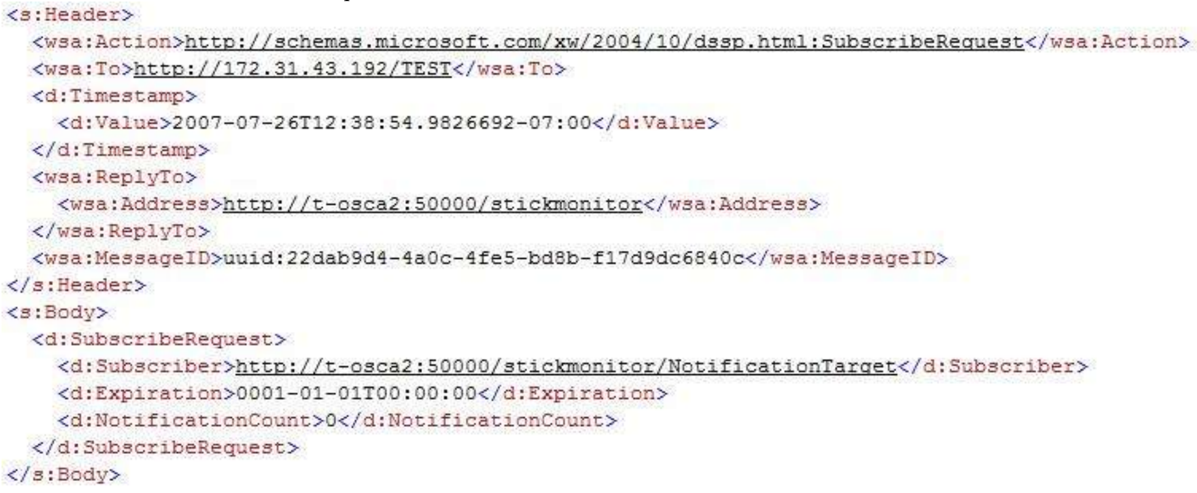
Embedded Server: Insert Response



```
<soap:Header>
  <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:To>
  <wsa:Action>http://schemas.microsoft.com/xw/2004/10/dssp.html:InsertResponse</wsa:Action>
  <wsa:RelatesTo>uuid:E9E02CCC-5EB2-4A46-0000-000037650253</wsa:RelatesTo>
  <wsa:MessageID>uuid:A7C934DD-2DA6-9F32-1925-928482975837</wsa:MessageID>
  <dssp:Timestamp xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html">
    <dssp:Value>2007-07-25T22:20:29.9281536Z</dssp:Value>
  </dssp:Timestamp>
</soap:Header>
<soap:Body>
  <dssp:DefaultInsertResponse xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html"/>
</soap:Body>
```

At this point, Robotics Studio subscribes to the service 'TEST', to be notified of any changes in the state of the gyro. The message exchange is:

Robotics Studio: Subscribe Request

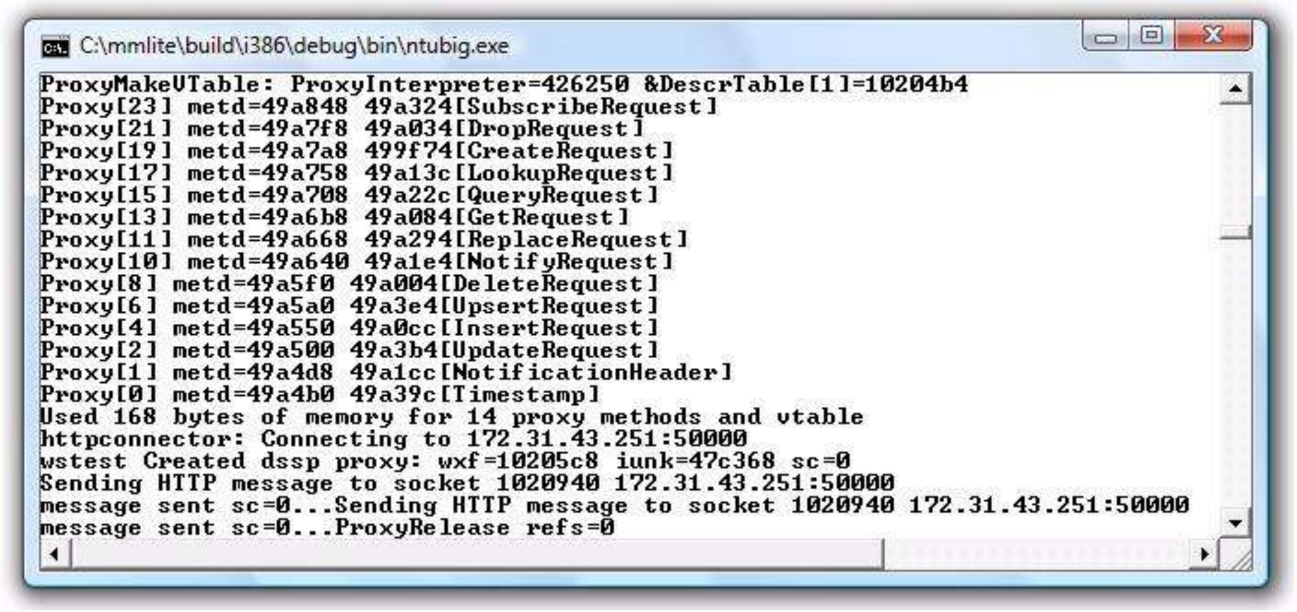


```
<s:Header>
  <wsa:Action>http://schemas.microsoft.com/xw/2004/10/dssp.html:SubscribeRequest</wsa:Action>
  <wsa:To>http://172.31.43.192/TEST</wsa:To>
  <d:Timestamp>
    <d:Value>2007-07-26T12:38:54.9826692-07:00</d:Value>
  </d:Timestamp>
  <wsa:ReplyTo>
    <wsa:Address>http://t-osca2:50000/stickmonitor</wsa:Address>
  </wsa:ReplyTo>
  <wsa:MessageID>uuid:22dab9d4-4a0c-4fe5-bd8b-f17d9dc6840c</wsa:MessageID>
</s:Header>
<s:Body>
  <d:SubscribeRequest>
    <d:Subscriber>http://t-osca2:50000/stickmonitor/NotificationTarget</d:Subscriber>
    <d:Expiration>0001-01-01T00:00:00</d:Expiration>
    <d:NotificationCount>0</d:NotificationCount>
  </d:SubscribeRequest>
</s:Body>
```

Embedded Server: Subscribe Response

```
<s:Header>
  <wsa:To>http://t-osca2:50000/stickmonitor</wsa:To>
  <wsa:Action>http://schemas.microsoft.com/xw/2004/10/dssp.html/SubscribeResponse</wsa:Action>
  <wsa:RelatesTo>uuid:22dab9d4-4a0c-4fe5-bd8b-f17d9dc6840c</wsa:RelatesTo>
  <wsa:MessageID>uuid:17ebb7a4-2c9f-1aa4-ca7d-a29e7ab2573d</wsa:MessageID>
  <dssp:Timestamp xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html">
    <dssp:Value>2007-07-26T19:38:55.2120720Z</dssp:Value>
  </dssp:Timestamp>
</s:Header>
<s:Body>
  <d:SubscribeResponse xmlns:d="http://schemas.microsoft.com/xw/2004/10/dssp.html">
    <d:SubscriptionManager>http://172.31.43.192/SUBSCRIPTIONS/TEST</d:SubscriptionManager>
    <d:Subscriber>http://t-osca2:50000/stickmonitor/NotificationTarget</d:Subscriber>
  </d:SubscribeResponse>
</s:Body>
```

At this point the system is initialized and enters the operational state. The EB63 board polls the gyro at a set rate, and accordingly Embedded Client updates the state of the gyro in Embedded Service. This update triggers the creation of a proxy between Embedded Server and Robotics Studio. Once the connection is established, the updated gyro measurements are sent to Robotics Studio. The Robotics Studio web service collecting this data (gyro measurements) can be viewed in a web browser in real time. Below is the printed debug information displayed when Embedded Server creates the proxy to Robotics Studio and sends it a message.



C:\mmlite\build\i386\debug\bin\ntubig.exe

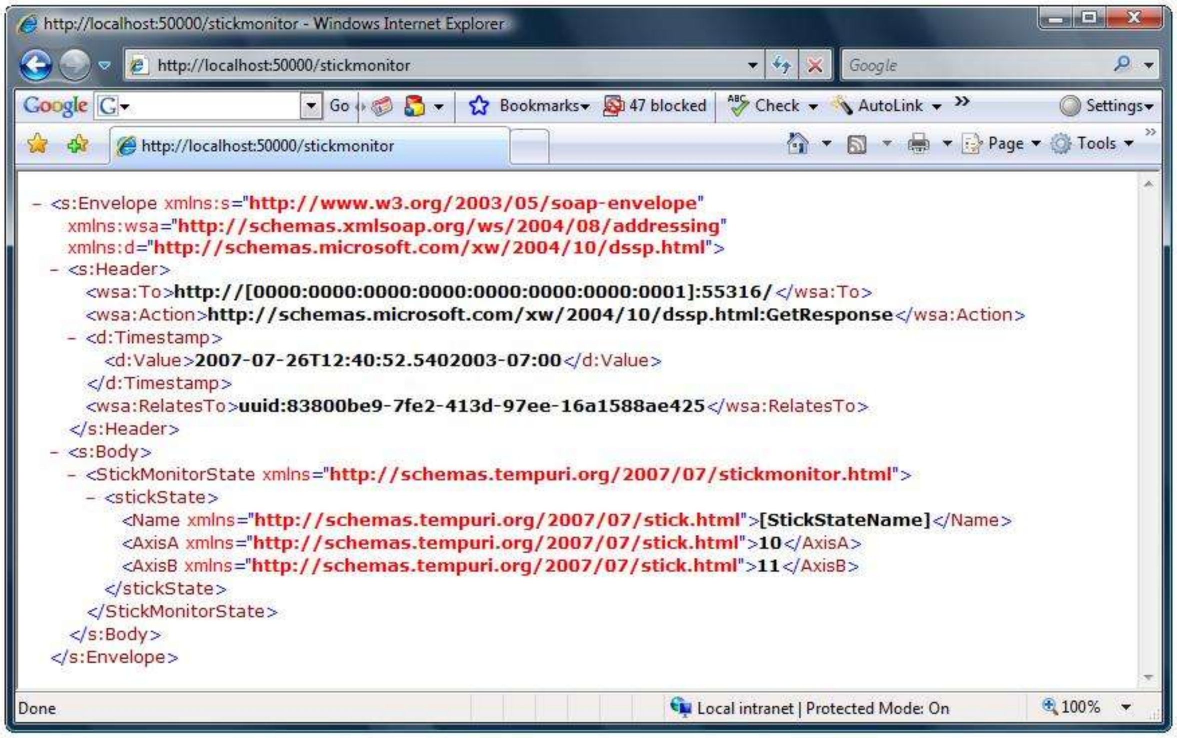
```
ProxyMakeVTable: ProxyInterpreter=426250 &DescrTable[1]=10204b4
Proxy[23] metd=49a848 49a324[SubscribeRequest]
Proxy[21] metd=49a7f8 49a034[DropRequest]
Proxy[19] metd=49a7a8 499f74[CreateRequest]
Proxy[17] metd=49a758 49a13c[LookupRequest]
Proxy[15] metd=49a708 49a22c[QueryRequest]
Proxy[13] metd=49a6b8 49a084[GetRequest]
Proxy[11] metd=49a668 49a294[ReplaceRequest]
Proxy[10] metd=49a640 49a1e4[NotifyRequest]
Proxy[8] metd=49a5f0 49a004[DeleteRequest]
Proxy[6] metd=49a5a0 49a3e4[UpsertRequest]
Proxy[4] metd=49a550 49a0cc[InsertRequest]
Proxy[2] metd=49a500 49a3b4[UpdateRequest]
Proxy[1] metd=49a4d8 49a1cc[NotificationHeader]
Proxy[0] metd=49a4b0 49a39c[Timestamp]
Used 168 bytes of memory for 14 proxy methods and vtable
httpconnector: Connecting to 172.31.43.251:50000
wstest Created dssp proxy: wxf=10205c8 iunk=47c368 sc=0
Sending HTTP message to socket 1020940 172.31.43.251:50000
message sent sc=0...Sending HTTP message to socket 1020940 172.31.43.251:50000
message sent sc=0...ProxyRelease refs=0
```

The notifications are one-way SOAP messages, meaning that they will not wait for a response from the target. A notification can be configured in Robotics Studio in the form of any of the state changing commands (Replace, Insert, Update, and Upsert). For this demo, the notification is the equivalent of a Replace call, meaning that the entire state kept in Robotics Studio will be replaced by the entire body of the incoming message.

Embedded Server: Notification Response

```
<soap:Header>
  <wsa:To>http://172.31.43.251:50000/stickmonitor/NotificationTarget</wsa:To>
  <wsa:Action>http://schemas.microsoft.com/xw/2004/10/dssp.html:NotifyRequest</wsa:Action>
  <wsa:MessageID>uuid:E9E02CCC-5EB2-4A46-0000-003728965848</wsa:MessageID>
  <dssp:Timestamp xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html">
    <dssp:Value>2007-07-26T19:40:49.3270627Z</dssp:Value>
  </dssp:Timestamp>
  <dssp:NotificationHeader xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html">
    <dssp:ForwardedAction>
      http://schemas.microsoft.com/xw/2004/10/dssp.html:ReplaceRequest
    </dssp:ForwardedAction>
  </dssp:NotificationHeader>
</soap:Header>
<soap:Body>
  <st1:StickState xmlns:st1="http://schemas.tempuri.org/2007/07/stick.html">
    <st1:Name>[StickStateName]</st1:Name>
    <st1:AxisA>10</st1:AxisA>
    <st1:AxisB>11</st1:AxisB>
  </st1:StickState>
</soap:Body>
```

Shown below is the user interface of this demo. While Robotics Studio is running, a user may simply open a web browser to the URL of the Robotics Studio service. If the service state is desired, the web browser will perform a Get function call on the service and display the response. Here the measurements on each axis of the gyro are shown in the body of the message.


http://localhost:50000/stickmonitor - Windows Internet Explorer
http://localhost:50000/stickmonitor
Google
Go
Bookmarks
47 blocked
Check
AutoLink
Settings
Page
Tools
Done
Local intranet | Protected Mode: On
100%

```
- <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:d="http://schemas.microsoft.com/xw/2004/10/dssp.html">
  - <s:Header>
      <wsa:To>http://[0000:0000:0000:0000:0000:0000:0000:0001]:55316/</wsa:To>
      <wsa:Action>http://schemas.microsoft.com/xw/2004/10/dssp.html:GetResponse</wsa:Action>
    - <d:Timestamp>
        <d:Value>2007-07-26T12:40:52.5402003-07:00</d:Value>
      </d:Timestamp>
      <wsa:RelatesTo>uuid:83800be9-7fe2-413d-97ee-16a1588ae425</wsa:RelatesTo>
    </s:Header>
  - <s:Body>
    - <StickMonitorState xmlns="http://schemas.tempuri.org/2007/07/stickmonitor.html">
      - <stickState>
          <Name xmlns="http://schemas.tempuri.org/2007/07/stick.html">[StickStateName]</Name>
          <AxisA xmlns="http://schemas.tempuri.org/2007/07/stick.html">10</AxisA>
          <AxisB xmlns="http://schemas.tempuri.org/2007/07/stick.html">11</AxisB>
        </stickState>
      </StickMonitorState>
    </s:Body>
  </s:Envelope>
```

Data Centric Embedded Services

Johannes Helander, Nishith Khantal, Ben Zorn, Oscar Almeida
Microsoft Research

Abstract

Data Centric Embedded Services are based on the core idea of an object manager, which provides both flexibility and code sharing. The Object Manager is flexible because it can handle a wide variety of objects ranging from hardware sensors to trust policies. At the same time, it has a lot of code sharing with common modules for data representation, communication, security etc. We have implemented a trust management service based on the object manager model, and used it in one of the other demos (Ref: Connecting Microsoft Robotics Studio and Embedded Web Services) to communicate with Microsoft Robotics Studio.

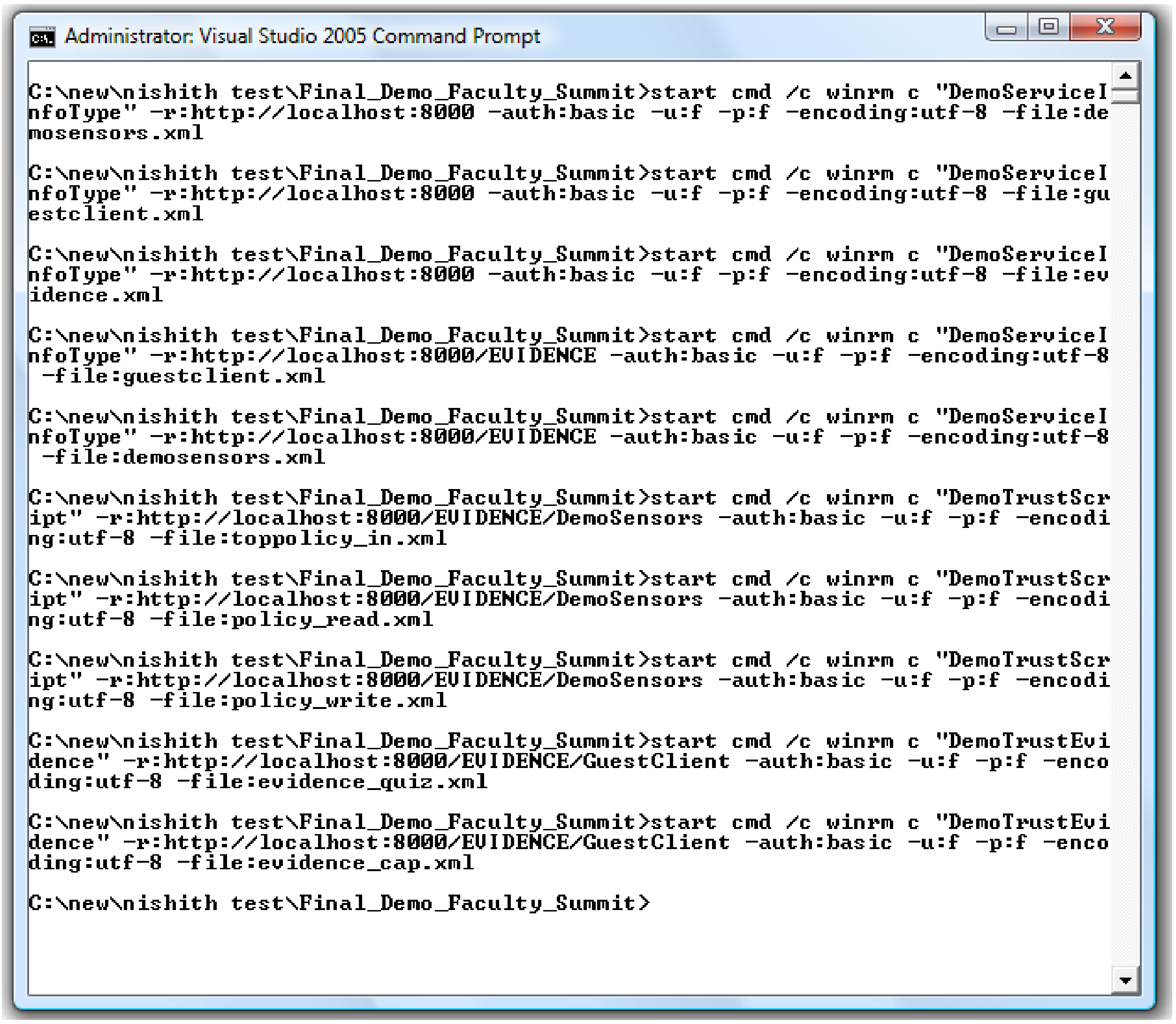
Trust among parties is modeled in the form of policies and evidences. The policies and evidences are stored and managed by the trust management service, which is referred to as Trust Manager in the poster in Appendix A. The service evaluates policies, based on the evidences and makes online access check decisions. For example, we can have a simple policy stating that to access a particular resource a client needs "to know the password" and "must be human". Hence we have a policy that is defined by the resource and the evidences that is to be provided by the client. The communication pattern for an access check consists of the client communicating with the Trust Manager and requesting access to the resource, followed by the Trust Manager evaluating the policies based on the evidences it holds on behalf of the client. This interaction is illustrated in the figure at the bottom left of the poster in Appendix A. The output is either "access granted" (if access is given) or "the URL to the evidence provider service" (if access is not given). The client can provide the necessary evidence at the evidence provider service, which will update the value in the Trust Manager, thus allowing access to the client.

Demo

Bootstrapping

Step 1: Starting the trust management service at a central location
A trust management service is started. The service in this demo is run on an i386 processor but it could run on an embedded platform. In another demo (Ref: Connecting Microsoft Robotics Studio and Embedded Web Services) the same code is run on an ARM processor.

Step 2: Configuring the trust management service (by the administrator)
In order to bootstrap the system, we assume that an administrator creates the object hierarchy represented at the bottom right of our poster in appendix A. In this demo the objects are created locally, but the Trust Management Service allows the creation of objects from a remote machine as well.

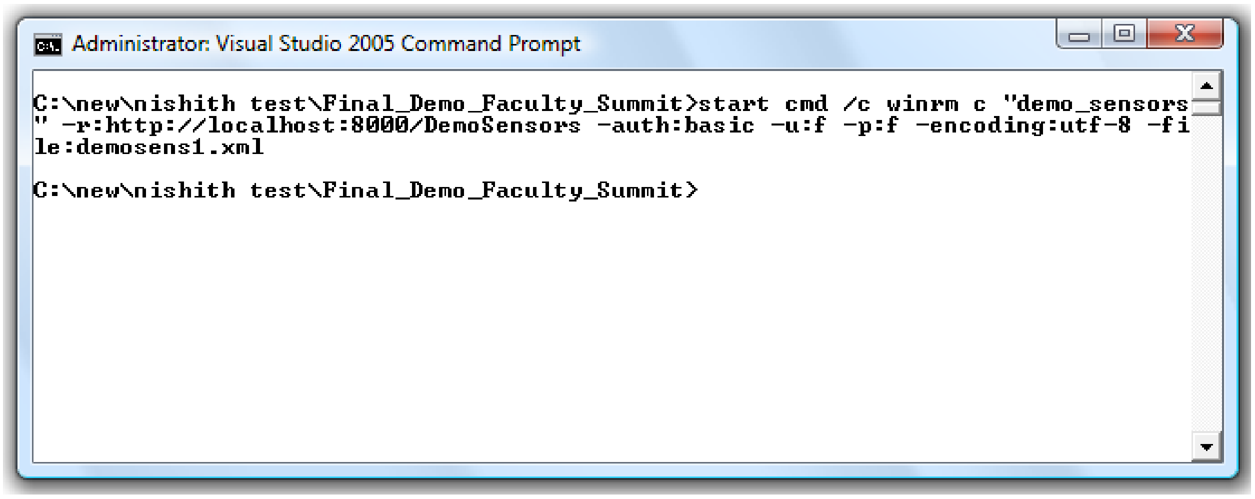


```
C:\new\nishith test\Final_Demo_Faculty_Summit>start cmd /c winrm c "DemoServiceI
nfoType" -r:http://localhost:8000 -auth:basic -u:f -p:f -encoding:utf-8 -file:de
mosensors.xml

C:\new\nishith test\Final_Demo_Faculty_Summit>start cmd /c winrm c "DemoServiceI
nfoType" -r:http://localhost:8000 -auth:basic -u:f -p:f -encoding:utf-8 -file:gu
estclient.xml

C:\new\nishith test\Final_Demo_Faculty_Summit>start cmd /c winrm c "DemoServiceI
nfoType" -r:http://localhost:8000 -auth:basic -u:f -p:f -encoding:utf-8 -file:ev
idence.xml

C:\new\nishith test\Final_Demo_Faculty_Summit>start cmd /c winrm c "DemoServiceI
nfoType" -r:http://localhost:8000/EVIDENCE -auth:basic -u:f -p:f -encoding:utf-8
 -file:guestclient.xml

C:\new\nishith test\Final_Demo_Faculty_Summit>start cmd /c winrm c "DemoServiceI
nfoType" -r:http://localhost:8000/EVIDENCE -auth:basic -u:f -p:f -encoding:utf-8
 -file:demosensors.xml

C:\new\nishith test\Final_Demo_Faculty_Summit>start cmd /c winrm c "DemoTrustScr
ipt" -r:http://localhost:8000/EVIDENCE/DemoSensors -auth:basic -u:f -p:f -encodi
ng:utf-8 -file:toppolicy_in.xml

C:\new\nishith test\Final_Demo_Faculty_Summit>start cmd /c winrm c "DemoTrustScr
ipt" -r:http://localhost:8000/EVIDENCE/DemoSensors -auth:basic -u:f -p:f -encodi
ng:utf-8 -file:policy_read.xml

C:\new\nishith test\Final_Demo_Faculty_Summit>start cmd /c winrm c "DemoTrustScr
ipt" -r:http://localhost:8000/EVIDENCE/DemoSensors -auth:basic -u:f -p:f -encodi
ng:utf-8 -file:policy_write.xml

C:\new\nishith test\Final_Demo_Faculty_Summit>start cmd /c winrm c "DemoTrustEvi
dence" -r:http://localhost:8000/EVIDENCE/GuestClient -auth:basic -u:f -p:f -enco
ding:utf-8 -file:evidence_quiz.xml

C:\new\nishith test\Final_Demo_Faculty_Summit>start cmd /c winrm c "DemoTrustEvi
dence" -r:http://localhost:8000/EVIDENCE/GuestClient -auth:basic -u:f -p:f -enco
ding:utf-8 -file:evidence_cap.xml

C:\new\nishith test\Final_Demo_Faculty_Summit>
```

- Each command in the above screen shot creates one object in the object hierarchy (refer to the poster).
- The first command creates the directory 'DemoSensors' per the specifications defined in the file 'demosensors.xml', on the Trust Manager running as a service at the address localhost:8000.
- Subsequent commands define other directories, policies and evidences according to their corresponding configuration files (.xml files). For instance, the location of the trust management service and the path in the hierarchy is specified by the URL http://localhost:8000/EVIDENCE/GuestClient
- At this point, the Trust Manager is fully responsible for the resource. The presence of a 'DemoSensors' object indicates that 'DemoSensors' is registered with the Trust Manager; any resource creation/editing/read in this directory is governed by its policies and interactions with the clients.

Access Checks

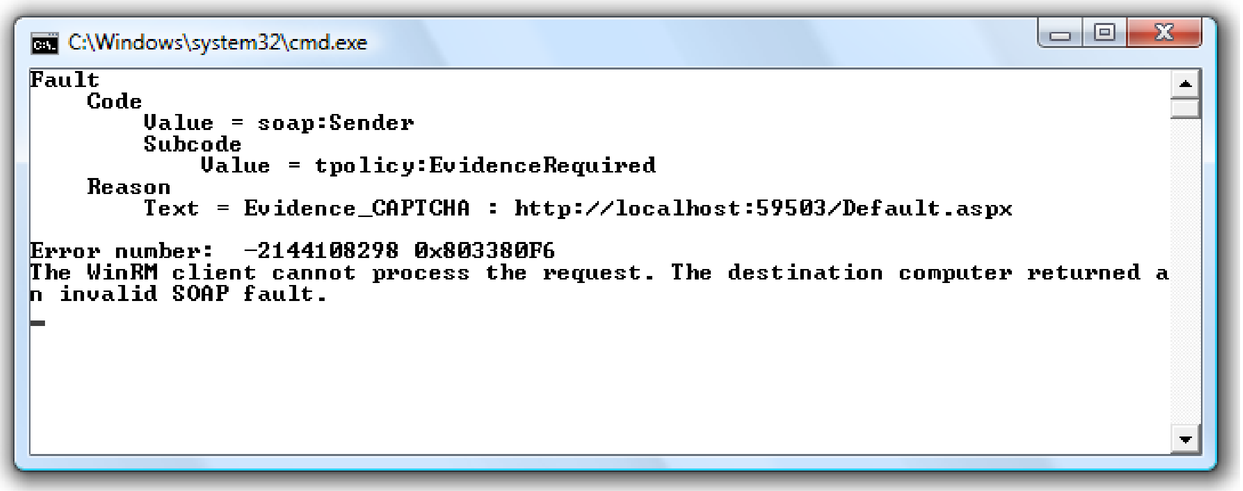
Step 1: Client tries to access the resource.


Administrator: Visual Studio 2005 Command Prompt

```
C:\new\nishith test\Final_Demo_Faculty_Summit>start cmd /c winrm c "demo_sensors
" -r:http://localhost:8000/DemoSensors -auth:basic -u:f -p:f -encoding:utf-8 -fi
le:demosens1.xml

C:\new\nishith test\Final_Demo_Faculty_Summit>
```

- In the above screen shot, the client tries to access the Trust Management Service. The URL in the above command defines the location of the Trust Management Service the client wants to use, in this case the DemoSensors directory as indicated by the URL http://localhost:8000/DemoSensors.
- The Trust Manager receives the request and applies the policies related to “DemoSensors”. It finds a Top Policy (Ref: Object Hierarchy in the poster), which redirects to a more specific policy that is based on the access mode. In this case the mode is ‘create’ and therefore we must apply Policy_write.
- Policy_write in turn requires Evidence_CAP, a CAPTCHA[1] evidence, and Evidence_quiz, an evidence based on the knowledge of the subject. At this point the client has not passed the CAPTCHA test and the value of Evidence_CAP is false (Ref: Object Hierarchy in the bottom-right of the poster).
- Hence the access check fails, as indicated in the figure below.

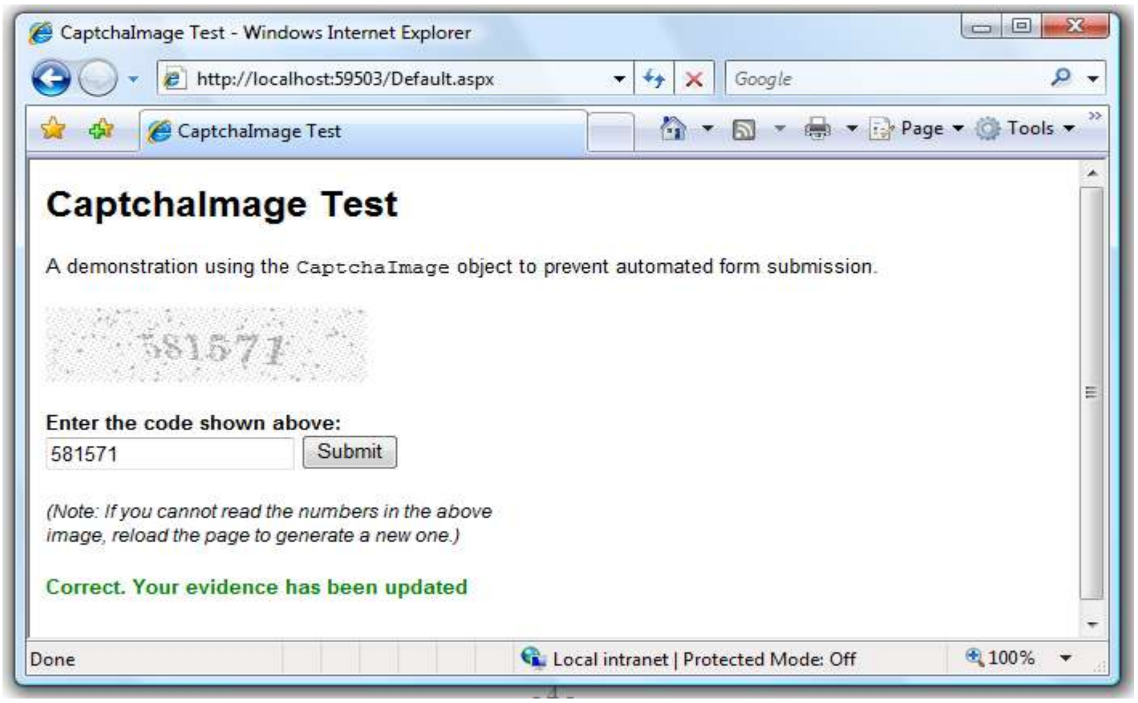
Step 2: Trust Manager sends a reply with the URL of evidence provider (Access Check Failed).


C:\Windows\system32\cmd.exe

```
Fault
    Code
        Value = soap:Sender
        Subcode
            Value = tpolicy:EvidenceRequired
    Reason
        Text  = Evidence_CAPTCHA : http://localhost:59503/Default.aspx

Error number:  -2144108298 0x803380F6
The WinRM client cannot process the request. The destination computer returned a
n invalid SOAP fault.
_
```

- The reply from the Trust Manager to the client indicates that “Access check” failed because evidence is required and the client needs to go to http://localhost:59503/Default.aspx to provide it.

[1] Completely Automated Public Turing test to tell Computers and Humans Apart


CaptchaImage Test - Windows Internet Explorer
http://localhost:59503/Default.aspx
Google
CaptchaImage Test
Page
Tools

CaptchaImage Test

A demonstration using the `CaptchaImage` object to prevent automated form submission.

Enter the code shown above:
581571
Submit

(Note: If you cannot read the numbers in the above image, reload the page to generate a new one.)

Correct. Your evidence has been updated

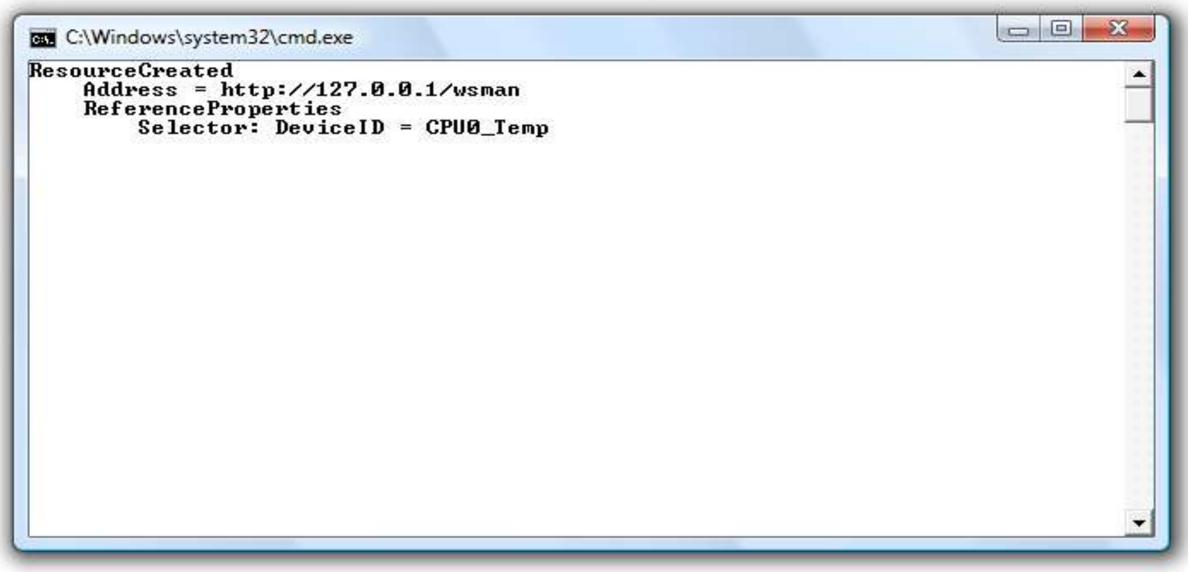
Done
Local intranet | Protected Mode: Off
100%

Step 3: Client goes to the evidence provider

- The client now goes to the evidence provider and provides the evidence, as shown in the picture above for the CaptchaImage Test.
- Correct evidence is provided by the client, and hence the client's evidence is updated in the Trust Manager.

Step 4: Client again tries to access the resource and now Access is provided

- The client again tries to access the resource, using exactly the same command as described in Step 1.
- Similar evaluation policies as described in Step 1 are applied. This time the client has both evidences, and hence access is granted and the required object is created.



```
ResourceCreated
    Address = http://127.0.0.1/wsman
    ReferenceProperties
        Selector: DeviceID = CPU0_Temp
```

World's Smallest Web 2.0 Server

Risto Serg, Johannes Helander
Microsoft Research

Abstract

The Service oriented systems architecture is gaining popularity among system designers because it can help reduce the development and maintenance costs. Communication between systems can be arranged using either some custom, application specific protocol or some standard protocol such as XML Web Services and reap the advantages that come from standardization and interoperability. One concern is that implementing XML Web Services on small embedded systems causes unnecessary overheads. This might well be true if we were to implement the entire SOAP protocol support. Fortunately, in most cases this is not necessary. Many small embedded systems provide a very limited and predefined set of services and therefore it is possible to implement only a subset of the protocol -- only what is explicitly needed for the application. Using some specific optimization techniques we showed that a subset of the XML Web Services protocol can be implemented on a tiny platform – an 8-bit AVR microcontroller with minimal overhead. Our sample application provides two sample services – addition and subtraction. It was implemented using a total of 92 bytes of RAM and 2,636 bytes of code. The XML messages that the system receives as service requests and send out as replies are all several hundreds of bytes long. The whole system doesn't need any hardware or software interrupts for operating, making it possible to target even simpler hardware platforms.

1 Demo

The following SOAP message is sent to the AVR controller over a serial line, requesting it to perform a simple addition.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:tns="http://soapinterop.org/">
  <soap:Header>
    <wsa:To>http://127.0.0.1/COB/calc.cob</wsa:To>
    <wsa:Action>http://soapinterop.org//Add</wsa:Action>
  </soap:Header>
  <soap:Body>
    <tns:Add xmlns:tns="http://soapinterop.org/">
      <tns:A>35</tns:A>
      <tns:B>7</tns:B>
    </tns:Add>
  </soap:Body>
</soap:Envelope>
```

The reply message from the AVR is as follows:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:tns="http://soapinterop.org/">
  <soap:Header>
```

```
      <wsa:Action>http://soapinterop.org//AddResponse</wsa:Action>
    </soap:Header>
    <soap:Body>
      <tns:AddResponse xmlns:tns="http://soapinterop.org/">
        42
      </tns:AddResponse>
    </soap:Body>
  </soap:Envelope>
```

Similarly, the message that is understood by the AVR for performing a subtraction is as follows:

```
  <?xml version="1.0"?>
  <soap:Envelope xmlns:tns="http://soapinterop.org/">
    <soap:Header>
      <wsa:To>http://127.0.0.1/COB/calc.cob</wsa:To>
      <wsa:Action>http://soapinterop.org//Subtract</wsa:Action>
    </soap:Header>
    <soap:Body>
      <tns:Subtract xmlns:tns="http://soapinterop.org/">
        <tns:A>49</tns:A>
        <tns:B>7</tns:B>
      </tns:Subtract>
    </soap:Body>
     </soap:Envelope>
```

The sizes for code segments are reported in Figure 1. Note that they are inclusive of some extra code to drive and print on the LCD display.

2 Techniques employed

2.1 Minimal real-time scheduling and offline analysis

One of the techniques used in the design of the system is to separate the functional and the temporal behaviors of the application, allowing us to analyze both aspects separately and independently of each other. This separation simplifies the design task and has the additional benefit of facilitating reuse of the code when it is later on transitioned to different hardware platforms. Sophisticated off-line analysis tools evaluate all the feasible execution schedules and help us pick the best fixed schedule that can satisfy the requirements for a hard real-time system and still provide the dynamic behavior required by the system. The models currently developed for this application include triggering, counting, timing and composite model.

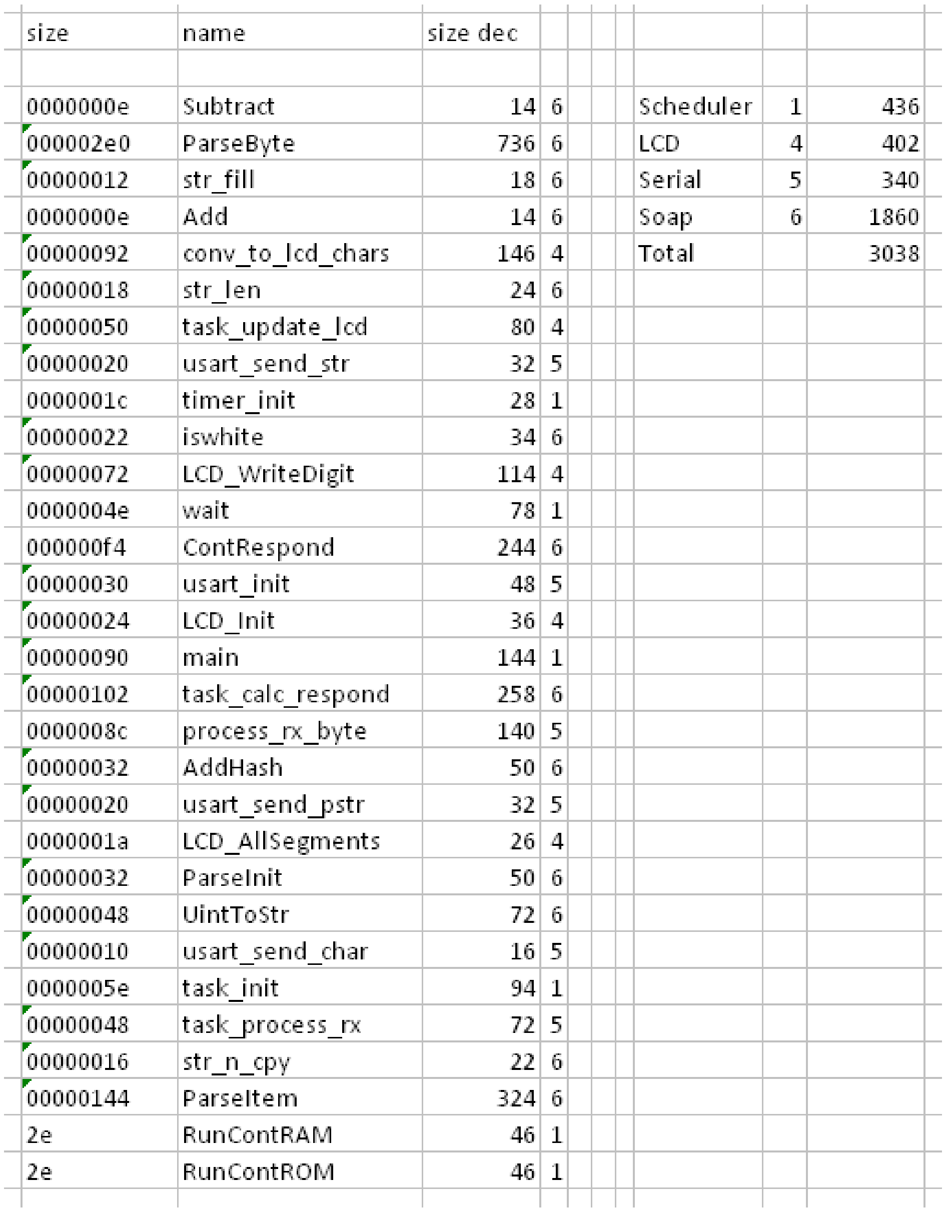
2.2 Implementation details

Scheduler

The schedule is a fixed data structure that contains one node per schedulable entity. Nodes are kept in an array, which requires less memory compared to a more traditional linked list approach. Each member of the schedule array is a structure containing:

1. a pointer to a future, which is a continuation expressed in a separate structure
2. flags
3. the earliest and latest start times

The scheduler must keep track of the current scheduling node; if the schedule array contains up to 256 nodes it is enough to use a single byte for this index. This byte is maintained in one of the general purpose processor registers, and therefore the RAM consumption of the scheduler is 0 (zero) bytes. Reserving a register for a special use does not create much impact because the AVR processor has a full complement of 32 general purpose registers available. The schedule is fixed and known at compile time and therefore we can put it into ROM.



size	name	size dec				
0000000e	Subtract	14	6	Scheduler	1	436
000002e0	ParseByte	736	6	LCD	4	402
00000012	str_fill	18	6	Serial	5	340
0000000e	Add	14	6	Soap	6	1860
00000092	conv_to_lcd_chars	146	4	Total		3038
00000018	str_len	24	6			
00000050	task_update_lcd	80	4			
00000020	usart_send_str	32	5			
0000001c	timer_init	28	1			
00000022	iswhite	34	6			
00000072	LCD_WriteDigit	114	4			
0000004e	wait	78	1			
000000f4	ContRespond	244	6			
00000030	usart_init	48	5			
00000024	LCD_Init	36	4			
00000090	main	144	1			
00000102	task_calc_respond	258	6			
0000008c	process_rx_byte	140	5			
00000032	AddHash	50	6			
00000020	usart_send_pstr	32	5			
0000001a	LCD_AllSegments	26	4			
00000032	ParseInit	50	6			
00000048	UintToStr	72	6			
00000010	usart_send_char	16	5			
0000005e	task_init	94	1			
00000048	task_process_rx	72	5			
00000016	str_n_cpy	22	6			
00000144	ParseItem	324	6			
2e	RunContRAM	46	1			
2e	RunContROM	46	1			

Figure 1: Code sizes for the Web server

Once a node is done executing the scheduler takes control. It checks for a possible delay before executing the next node by examining the start time field, and if necessary it performs the necessary wait operation. The scheduler then checks the validity of the next scheduling node. If it is valid then the next node is executed otherwise the node is skipped.

Futures

The term “future” was originally used in the MultiLisp programming language to express potential concurrency and lazy evaluation of functional expressions. In our system a future is an execution node structure that contains:

1. a pointer to a function code
2. the stack frame size
3. a variable number of parameters

If a future needs to preserve state across evaluations then at least one of the parameters is a pointer. Evaluations of the future consist of loading the parameters into the processor registers and on the current stack frame (from the future node structure) and then invoking the future’s function according to the C calling convention. The future node can usually be allocated in ROM, unless it needs to preserve state e.g. over-write one of the parameters.

Drivers

The AVR AtMega 169 processor includes a built-in serial line interface and a passive LCD screen controller. Both peripherals are managed in software by objects with different interface methods. Since each method is a function we can turn every method into a future and execute it as part of the schedule.
The serial line is used as the communication channel to carry XML messages. The send and receive routines poll the corresponding hardware registers to achieve the necessary synchronization.
The LCD screen is capable of displaying 6 alphanumeric characters. Its driver includes the necessary conversion tables between segment data registers and ASCII codes, plus an intermediate buffer for atomic LCD updates. Currently the LCD screen is used to just display arguments and result of the implemented WEB Service calls, and it can be easily omitted to further reduce the memory footprint. Figure 1 indicates the corresponding costs.

XML parser

The parser is called from the serial line driver one byte at a time. It uses a nested state machine to preserve its state across invocations. The future that is performing the data receive process is described in a fixed schedule and is executed frequently enough to get the serial data without the need of an intermediate buffer. If the message start marker is received then the state of the future changes and all subsequent received bytes are passed to the XML parser. The corresponding method is called until the message end marker is received. The XML parser parses incoming message one byte at a time with the aid of internal nested state machine. If it detects a match against a predefined string then it fills an already scheduled future node with valid data, which activates execution of the service. Effectively, this converts the output of the XML parser into a future. As described in the scheduler section, an execution node is only run if it contains valid data.

WEB Service

Many WEB Services require that a response message is sent back to the requestor. When this is the case, the service response is generated by a future that is scheduled to run after the execution of the Web Service future. The Response method checks if the future that computes the results has ran and if so it sends back the response message.

Scalable Embedded Systems

Johannes Helander, Risto Serg, Sibin Mohan, Margus Veanes, Philip Garcia
Microsoft Research

Abstract

Embedded applications in the physical world including robotics, industry, and civic infrastructure often require multiple devices to achieve a single goal or to perform a given task. These multiple devices must be able to interoperate and adapt in an often chaotic and distributed environment. Many of these systems have timing constraints imposed upon them and are thus referred to as distributed real-time systems. Achieving predictable behavior in distributed real-time systems is difficult, as complex timing information of the distributed real-time tasks must be known. Unfortunately, in contemporary software design, timing information and temporal behavior is hidden and hard coded inside the implementations of individual software components. In addition to making analysis of such systems difficult, this makes it challenging to adapt software to platforms with differing capabilities.

Working with distributed embedded systems requires knowledge of parallel program design. Parallel programming is a difficult task that has not been adequately addressed by current tools and programming languages. Designing such applications often requires the developer to explicitly order the execution of threads, and ensure the correctness of (near) infinite overlays of the threads in the system.

In this work we propose techniques to build robust distributed service-oriented embedded systems. To achieve this goal, we first deconstruct a program into two separate aspects -- the functional implementation and a temporal pattern. Each aspect is expressed separately in an appropriate programming language. We process the temporal aspect of the program separately with various tools and techniques, including a combination of static and dynamic program analyses and model-based testing.

To handle the concurrency that is explicitly given in the temporal patterns, we have designed a series of stream buffers that abstract away the details of concurrent programming. By using these abstractions, a programmer can develop parallel, streaming applications as if they were straightforward serial programs.

1 Introduction

When examining real-time systems, tools either perform static or dynamic timing analysis to determine the worst-case temporal aspects of the program. But such analysis is limited in the scope of applications that it can analyze. Analysis based on dynamic traces, on the other hand, is also incomplete since it cannot guarantee that complete code coverage was achieved.

In this work, we present techniques that use ideas based on objects, aspects, patterns and services combined with parallelism concepts. We also utilize the idea of developing different facets of an application (which we call "aspects") separately. Two of these aspects are – *futures* and *partitures* which we explain in Section 2. To aid this process, we analyze (a) the process of extracting temporal behavior from distributed applications and (b) implementing a new stream buffer structure which can transparently handle concurrency between the various components within a program.

Extracting temporal behavior from distributed programs is achieved by using a "*temporal analyzer*". The temporal analyzer uses a combination of static analyses of the program coupled with dynamic traces and domain knowledge to extract temporal information from the program (“*partitures*”). The temporal analyzer is explained in more detail in Section 3.

To help facilitate concurrent application execution, and simplify the description of dataflow in programs we have developed a concurrent stream buffer, and runtime environment. The details of this system are described in Section 4.

2 Futures and Partitures

Our tools have been designed to ease the development of distributed embedded applications. To achieve this goal, we use the programming paradigm of *futures* and *partitures*. These constructs allow the programmer to express the intent of sequential dependencies and temporal parameters more easily.

2.1 Futures

Futures were originally proposed in 1976, and are the principal synchronization construct used in MultiLisp. Futures are used to express concurrency and potential parallelism and sequential behavior between program components. By allowing the programmer to freely and lightly express parallelism in the program it is possible for the runtime system to adjust the level of parallelism so that it can run optimally on the target system’s available resources. The future allows a common programming paradigm for cyber-physical programs across, virtually, the entire spectrum of programmable devices. Futures are encapsulated method calls that are evaluated (executed) at some point in time between when they are created and when the result is used. A future does not require a stack until it is ready to execute, thus, to save memory, stack allocation can be deferred until the future’s actual execution. This means that stacks can be efficiently recycled, thus significantly decreasing the cost of creating a future and the RAM footprint of an application. Futures can be executed whenever the scheduler decides that they must execute. In small embedded devices, these futures can be executed in a serial fashion to allow for low memory utilization. In larger systems, many futures can be launched simultaneously to take advantage of available resources. As futures are cheap to create (and can be efficiently executed on both small and large systems), it is expected that programmers will not hesitate to use them to express the concurrency inherent in their applications.

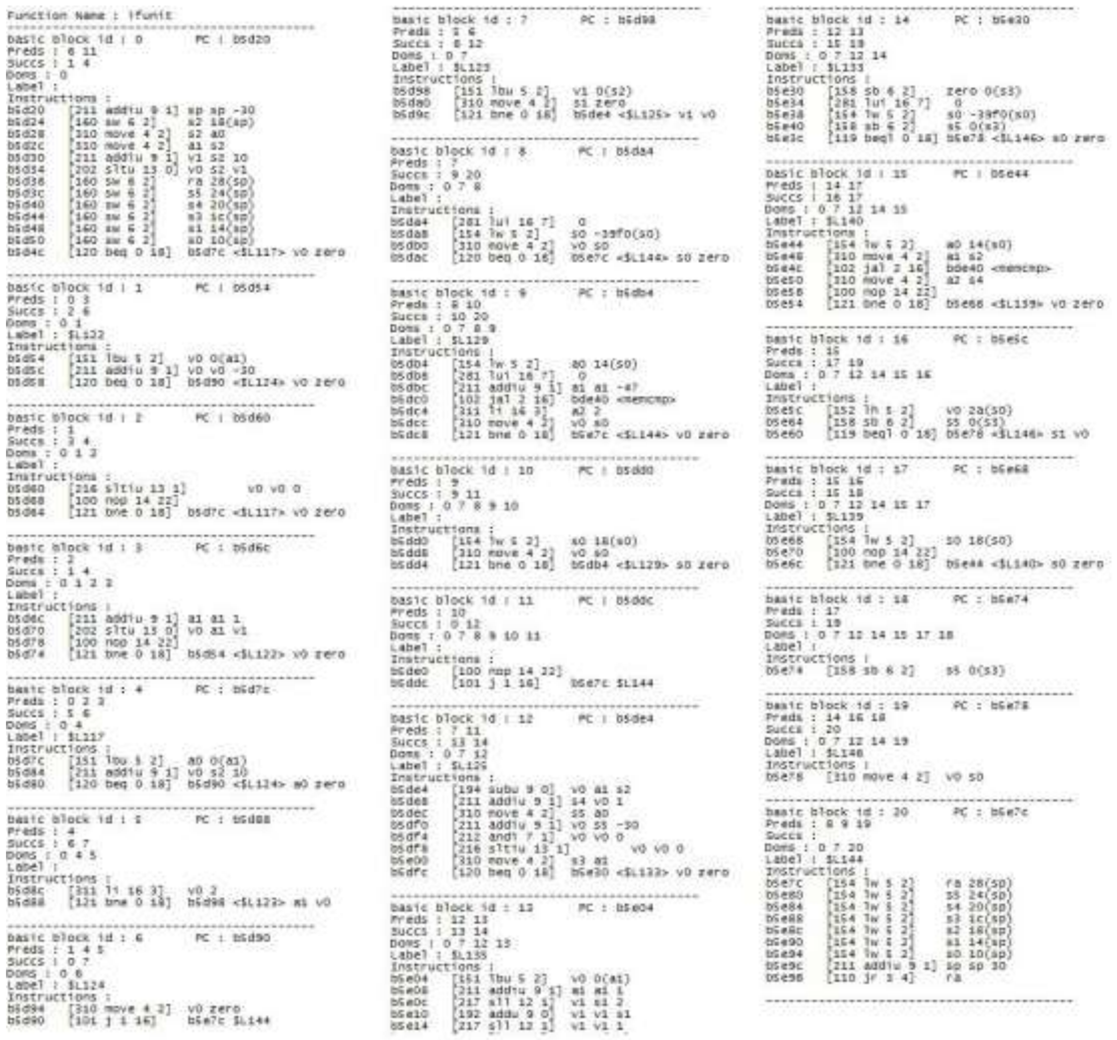
2.2 Partitures

The partiture is analogous to the score of a symphony orchestra and is a separate program that describes the temporal aspect of an application, leaving implementation details to the regular programming language, such as C or Java/C#. The partiture is a collection of scores, where each score describes the expected sequence of events and message pattern of a single task as a collection of bars. Each bar prescribes the estimated time and duration when something should happen, as well as where messages are sent and what other bars are triggered. The partiture does not prescribe what the content of messages is, what functions are called, or what the implementation does. Instead, the partiture is an estimate of the program’s temporal behavior. The partiture may be authored by hand, generated from an execution log, generated by analysis tools, or as a combination of the above. If the partiture is based on solid worst-case analysis, it is a hard real-time partiture.

In our work, we are developing tools which will help us automate the process of automatically extracting partitures from distributed programs.

3 Analysis Tools: The Temporal Analyzer

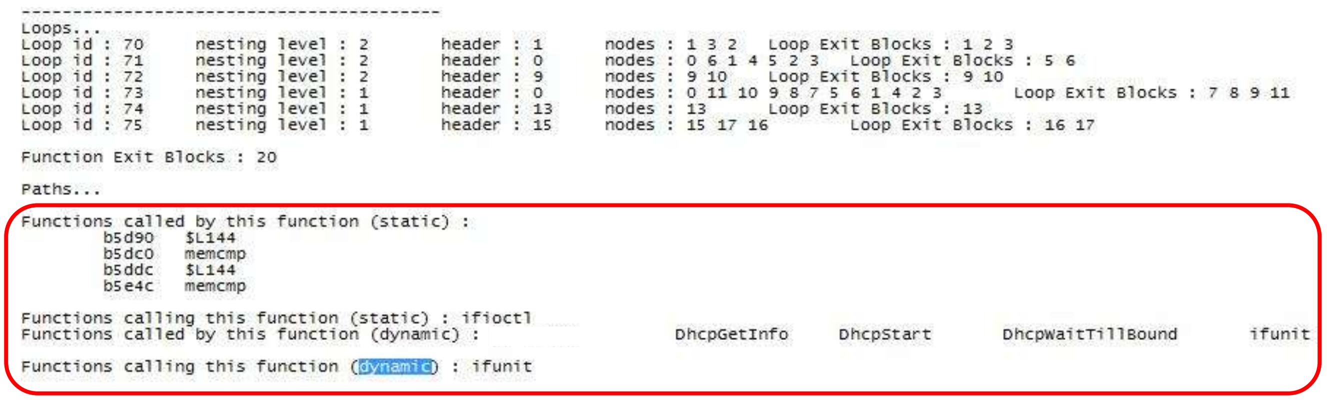
The *temporal analyzer* is a graph-based program analysis tool. It draws upon information from various sources, starting with static analysis of the control flow graph of the program. To validate our tool, we use it to examine the network stack in the Microsoft Invisible Computing framework which is compiled down to a single object file. The disassembly of this file is broken down into its constituent basic blocks to form the static control flow graph of the program. Some function calls cannot be resolved at compile-time, as they are made through the v-table or other function pointers. Hence the control flow graph is missing some edges between functions. To fill in this gap, we use function call traces from the execution of the network stack on the Giano simulation framework. These traces provide us with some of the missing edges from the control flow graph.



```
Function Name : lfunit
-----------------------------------------
basic block id : 0        PC : b5d20
Preds : 6 11
Succs : 1 4
Doms : 0
Label :
Instructions :
b5d20    [211 addiu 9 1] sp sp -30
b5d24    [160 sw 6 2]    s2 18(sp)
b5d28    [310 move 4 2]  s2 a0
b5d2c    [310 move 4 2]  a1 s2
b5d30    [211 addiu 9 1] v1 s2 10
b5d34    [202 sltu 13 0] v0 s2 v1
b5d38    [160 sw 6 2]    ra 28(sp)
b5d3c    [160 sw 6 2]    s5 24(sp)
b5d40    [160 sw 6 2]    s4 20(sp)
b5d44    [160 sw 6 2]    s3 1c(sp)
b5d48    [160 sw 6 2]    s1 14(sp)
b5d50    [160 sw 6 2]    s0 10(sp)
b5d4c    [120 beq 0 18]  b5d7c <$L117> v0 zero

-----------------------------------------
basic block id : 1        PC : b5d54
Preds : 0 3
Succs : 2 6
Doms : 0 1
Label : $L122
Instructions :
b5d54    [151 lbu 5 2]   v0 0(a1)
b5d5c    [211 addiu 9 1] v0 v0 -30
b5d58    [120 beq 0 18]  b5d90 <$L124> v0 zero

-----------------------------------------
basic block id : 2        PC : b5d60
Preds : 1
Succs : 3 4
Doms : 0 1 2
Label :
Instructions :
b5d60    [216 sltiu 13 1]        v0 v0 0
b5d68    [100 nop 14 22]
b5d64    [121 bne 0 18]  b5d7c <$L117> v0 zero

-----------------------------------------
basic block id : 3        PC : b5d6c
Preds : 2
Succs : 1 4
Doms : 0 1 2 3
Label :
Instructions :
b5d6c    [211 addiu 9 1] a1 a1 1
b5d70    [202 sltu 13 0] v0 a1 v1
b5d78    [100 nop 14 22]
b5d74    [121 bne 0 18]  b5d54 <$L122> v0 zero

-----------------------------------------
basic block id : 4        PC : b5d7c
Preds : 0 2 3
Succs : 5 6
Doms : 0 4
Label : $L117
Instructions :
b5d7c    [151 lbu 5 2]   a0 0(a1)
b5d84    [211 addiu 9 1] v0 s2 10
b5d80    [120 beq 0 18]  b5d90 <$L124> a0 zero

-----------------------------------------
basic block id : 5        PC : b5d88
Preds : 4
Succs : 6 7
Doms : 0 4 5
Label :
Instructions :
b5d8c    [311 li 16 3]   v0 2
b5d88    [121 bne 0 18]  b5d98 <$L123> a1 v0

-----------------------------------------
basic block id : 6        PC : b5d90
Preds : 1 4 5
Succs : 0 7
Doms : 0 6
Label : $L124
Instructions :
b5d94    [310 move 4 2]  v0 zero
b5d90    [101 j 1 16]    b5e7c $L144

-----------------------------------------
basic block id : 7        PC : b5d98
Preds : 5 6
Succs : 8 12
Doms : 0 7
Label : $L123
Instructions :
b5d98    [151 lbu 5 2]   v1 0(s2)
b5da0    [310 move 4 2]  s1 zero
b5d9c    [121 bne 0 18]  b5de4 <$L125> v1 v0

-----------------------------------------
basic block id : 8        PC : b5da4
Preds : 7
Succs : 9 20
Doms : 0 7 8
Label :
Instructions :
b5da4    [281 lui 16 7]   0
b5da8    [154 lw 5 2]    s0 -39f0(s0)
b5db0    [310 move 4 2]  v0 s0
b5dac    [120 beq 0 18]  b5e7c <$L144> s0 zero

-----------------------------------------
basic block id : 9        PC : b5db4
Preds : 8 10
Succs : 10 20
Doms : 0 7 8 9
Label : $L129
Instructions :
b5db4    [154 lw 5 2]    a0 14(s0)
b5db8    [281 lui 16 7]   0
b5dbc    [211 addiu 9 1] a1 a1 -4?
b5dc0    [102 jal 2 16]  bde40 <memcmp>
b5dc4    [311 li 16 3]   a2 2
b5dcc    [310 move 4 2]  v0 s0
b5dc8    [121 bne 0 18]  b5e7c <$L144> v0 zero

-----------------------------------------
basic block id : 10       PC : b5dd0
Preds : 9
Succs : 9 11
Doms : 0 7 8 9 10
Label :
Instructions :
b5dd0    [154 lw 5 2]    s0 18(s0)
b5dd8    [310 move 4 2]  v0 s0
b5dd4    [121 bne 0 18]  b5db4 <$L129> s0 zero

-----------------------------------------
basic block id : 11       PC : b5ddc
Preds : 10
Succs : 0 12
Doms : 0 7 8 9 10 11
Label :
Instructions :
b5de0    [100 nop 14 22]
b5ddc    [101 j 1 16]    b5e7c $L144

-----------------------------------------
basic block id : 12       PC : b5de4
Preds : 7 11
Succs : 13 14
Doms : 0 7 12
Label : $L125
Instructions :
b5de4    [194 subu 9 0]  v0 a1 s2
b5de8    [211 addiu 9 1] s4 v0 1
b5dec    [310 move 4 2]  s5 a0
b5df0    [211 addiu 9 1] v0 s5 -30
b5df4    [212 andi 7 1]  v0 v0 0
b5df8    [216 sltiu 13 1]        v0 v0 0
b5e00    [310 move 4 2]  s3 a1
b5dfc    [120 beq 0 18]  b5e30 <$L133> v0 zero

-----------------------------------------
basic block id : 13       PC : b5e04
Preds : 12 13
Succs : 13 14
Doms : 0 7 12 13
Label : $L135
Instructions :
b5e04    [151 lbu 5 2]   v0 0(a1)
b5e08    [211 addiu 9 1] a1 a1 1
b5e0c    [217 sll 12 1]  v1 s1 2
b5e10    [192 addu 9 0]  v1 v1 s1
b5e14    [217 sll 12 1]  v1 v1 1

-----------------------------------------
basic block id : 14       PC : b5e30
Preds : 12 13
Succs : 15 19
Doms : 0 7 12 14
Label : $L133
Instructions :
b5e30    [158 sb 6 2]    zero 0(s3)
b5e34    [281 lui 16 7]   0
b5e38    [154 lw 5 2]    s0 -39f0(s0)
b5e40    [158 sb 6 2]    s5 0(s3)
b5e3c    [119 beql 0 18] b5e78 <$L146> s0 zero

-----------------------------------------
basic block id : 15       PC : b5e44
Preds : 14 17
Succs : 16 17
Doms : 0 7 12 14 15
Label : $L140
Instructions :
b5e44    [154 lw 5 2]    a0 14(s0)
b5e48    [310 move 4 2]  a1 s2
b5e4c    [102 jal 2 16]  bde40 <memcmp>
b5e50    [310 move 4 2]  a2 s4
b5e58    [100 nop 14 22]
b5e54    [121 bne 0 18]  b5e68 <$L139> v0 zero

-----------------------------------------
basic block id : 16       PC : b5e5c
Preds : 15
Succs : 17 19
Doms : 0 7 12 14 15 16
Label :
Instructions :
b5e5c    [152 lh 5 2]    v0 2a(s0)
b5e64    [158 sb 6 2]    s5 0(s3)
b5e60    [119 beql 0 18] b5e78 <$L146> s1 v0

-----------------------------------------
basic block id : 17       PC : b5e68
Preds : 15 16
Succs : 15 18
Doms : 0 7 12 14 15 17
Label : $L139
Instructions :
b5e68    [154 lw 5 2]    s0 18(s0)
b5e70    [100 nop 14 22]
b5e6c    [121 bne 0 18]  b5e44 <$L140> s0 zero

-----------------------------------------
basic block id : 18       PC : b5e74
Preds : 17
Succs : 19
Doms : 0 7 12 14 15 17 18
Label :
Instructions :
b5e74    [158 sb 6 2]    s5 0(s3)

-----------------------------------------
basic block id : 19       PC : b5e78
Preds : 14 16 18
Succs : 20
Doms : 0 7 12 14 19
Label : $L146
Instructions :
b5e78    [310 move 4 2]  v0 s0

-----------------------------------------
basic block id : 20       PC : b5e7c
Preds : 8 9 19
Succs :
Doms : 0 7 20
Label : $L144
Instructions :
b5e7c    [154 lw 5 2]    ra 28(sp)
b5e80    [154 lw 5 2]    s5 24(sp)
b5e84    [154 lw 5 2]    s4 20(sp)
b5e88    [154 lw 5 2]    s3 1c(sp)
b5e8c    [154 lw 5 2]    s2 18(sp)
b5e90    [154 lw 5 2]    s1 14(sp)
b5e94    [154 lw 5 2]    s0 10(sp)
b5e9c    [211 addiu 9 1] sp sp 30
b5e98    [110 jr 3 4]    ra

-----------------------------------------
```

Figure 1

Figure 1 shows the basic block structure of one function. It also shows a list of all loops in the function.



```
-----------------------------------------
Loops...
Loop id : 70     nesting level : 2      header : 1      nodes : 1 3 2   Loop Exit Blocks : 1 2 3
Loop id : 71     nesting level : 2      header : 0      nodes : 0 6 1 4 5 2 3   Loop Exit Blocks : 5 6
Loop id : 72     nesting level : 2      header : 9      nodes : 9 10    Loop Exit Blocks : 9 10
Loop id : 73     nesting level : 1      header : 0      nodes : 0 11 10 9 8 7 5 6 1 4 2 3       Loop Exit Blocks : 7 8 9 11
Loop id : 74     nesting level : 1      header : 13     nodes : 13      Loop Exit Blocks : 13
Loop id : 75     nesting level : 1      header : 15     nodes : 15 17 16        Loop Exit Blocks : 16 17

Function Exit Blocks : 20

Paths...

Functions called by this function (static) :
        b5d90   $L144
        b5dc0   memcmp
        b5ddc   $L144
        b5e4c   memcmp

Functions calling this function (static) : ifioctl
Functions called by this function (dynamic) :               DhcpGetInfo     DhcpStart       DhcpWaitTillBound       ifunit

Functions calling this function (dynamic) : ifunit
```

Figure 2

Figure 2 shows the function calls relative to this function – static calls made by this function, dynamic calls made by this function and all the functions calling this function. It also shows the list of all loops in the current function.

Once we have information on at least some of these dynamic function calls, we can reason about other functions that can be called from the same call site – basically an estimate of all the functions that the v-table entry could lead to. This is gleaned from domain knowledge and type information. The type information of the called function can be used to determine the list of all other functions which have the same type. Hence, these are the only functions that could potentially be called from that particular call site. This provides us with more information about the control flow graph of the program being analyzed.

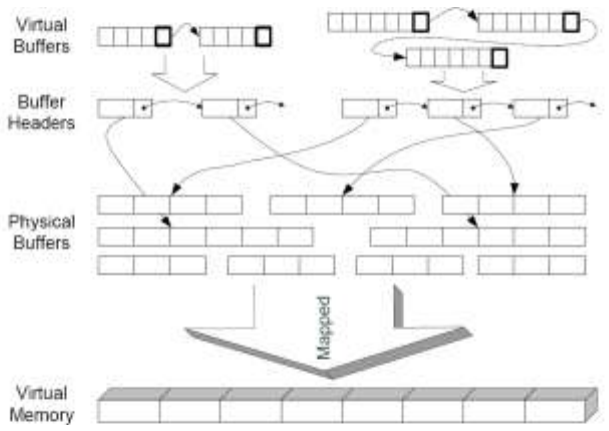
We can also make use of a second set of traces – those of the synchronization constructs used in the execution of the network stack and the threads that make these calls. We use this information to determine cycles and dependencies between threads in a multi-threaded application.

4 Stream Buffers for Component-based Programming

To help simplify the task of generating multithreaded programs, we can first separate the application flow from the control and data flow that exists within a single routine. To accomplish this goal, we break down an application using a form of the pipe-and-filter design pattern. This splits the application program into a series of filters or ``components" and a graph that expresses the flow of data between these components. We propose doing this by using a standardized buffer structure combined with the pipe-and-filter design paradigm to create applications. Applications for our system are expressed using an XML representation of the data flow between the components of the application. This approach allows for hierarchical formations, and furthermore we can use patterns to simplify the application's description.

For the first implementation of our system we are developing a ``data pump" that pushes data through the various components in the program's dataflow graph. The data pump is responsible for initializing the buffers that are used by the application, as well as the transfer of data between the various routines.

The buffers and their accessors are designed to be both efficient and easy to use. This task is difficult as not all applications can specify the size of data that is held within their buffers. The various constraints placed on the buffer means that no single buffer structure is sufficient for all situations. Because of these constraints, we have created a layered approach to the buffers. Figure 3 shows a mapping of how the buffers are layered on top of the system's virtual memory layer.


Virtual Buffers
Buffer Headers
Physical Buffers
Mapped
Virtual Memory
(Figure 3)

Appendix A: Posters

The following posters decorated the walls of our booth at DemoFest.

Education Experiences in Embedded Systems
A Goal Driven, Self Propelling Process

Steve Liu, Texas A&M University Computer Science Department
In collaboration with
Alessandro Forin and Johannes Helander, Microsoft Research

Microsoft Faculty Summit Demofest
July 17, 2007

Motivations and needs

- Training students to meet industrial requirements within financial & credit hour budgets
- Recruiting and retention of engineering students is a national concern
- Growing gap between analytical knowledge, textbooks, industrial tools and real systems
- Standardized materials and exams alone produce unimaginative students
- Creativity, risk taking spirit and entrepreneurial skills are critically lacking
- Need sustainable models for research & education integration
- New teaching platforms to deal with increasingly complex technology landscape

Our approach: A goal driven, self propelling process

- Goal: drive students to produce tangible artifacts and stimulate self learning abilities
- Holistic process: basic learning (lectures), develop skills (lab exercises), self exploration (project proposals), resource management (parts, time and manpower), execution (prototypes)
- Simplicity: Rigorously focus on a small set of core material
- Self guided: After the basic training, students lead their own projects
- Multi-role: Instructor is also mediator, customer, sponsor, partner
- Realism: live codes and working prototypes, data sheets, schematics, industrial tools
- Fun & risk taking: encourage creativity, high risk project concepts, building from scratch, discourage shrink wrap kits
- Accumulative: good projects recognized, modeled after, and expanded in following years

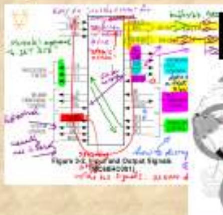
Software: Microsoft Invisible Computing, XML Web Services Giano Simulator

Device drivers, Scheduler & software timer
Loader, Debugger, Memory and other resource map
Interrupt

Hardware: Atmel (ARM7) Microcontroller, Xilinx FPGA

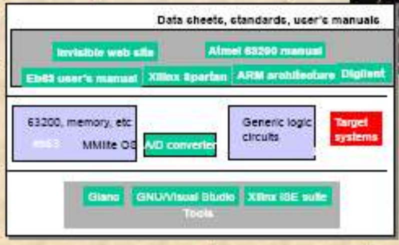
Atmel eb63 board, Xilinx Spartan 3
SoC architecture, flash, on-board buses,
inter-board interfaces

Tablet lecturing



FPGA & Microcontrollers

Technical Scope



Development platforms

Labs

Emphasis on fundamental skills:
Fully supervised sessions

Lab 1: Instrument practice
Lab 2: hello world (FPGA, microcontroller)
Lab 3: FPGA ⇒ eb63
Lab 4: eb63 ⇐ FPGA
Lab 5: Interrupts

Lab 6: projects

Light-weight networking

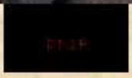
SOAP information center

Embedded System Principles

Microprocessor, I/O devices, and timers/counters
(ARM, Atmel 63200)
Memory, Atmel memory map
Tier 1 (motherboard) bus
Tier 2 (peripheral) bus: USB, JTAG, PCI
Tier 3 (sensor) bus: Maxim (DS) 1-wire
Analog sensors
Interconnect and signals
Datapaths, Address paths. Power, Clocks
clock distribution, signaling (level, edge triggered, combinations)
Architecture of the operating system

Real-time

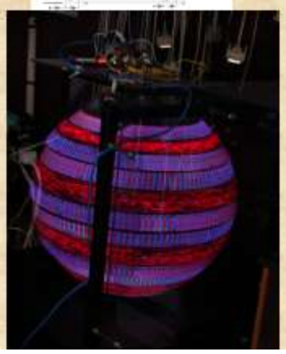
Air banner



Sign language glove

Mini-elevator

mechanics-electronics-software-art



Rubik's Cube Solver

Chris Boyce, David Chaszar, Michael Stewart, Doug Toney

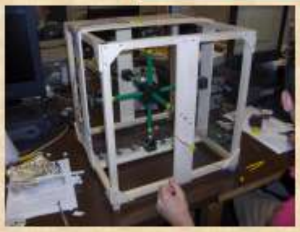
Advisor: Steve Liu
Computer Science Department
Texas A&M University

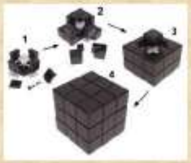
Microsoft Research Faculty Summit
July 15-17, 2007

Motivations and Needs:
- Solving the Rubik's cube is a combinatorial problem that requires significant computing cycles and memory
- Intelligent machines may be required to solve combinatorial problems within a module
- A simple exercise to examine the integration of high level algorithms, digital control, and mechanical designs
- A cute toy to teach both algorithms and embedded computing

Our approach, a layered system design:
- High level layer: a Rubik's cube algorithm running on a PC to generate a sequence of moves
- Motion control layer: Six motors execute each move based on signals from a microcontroller
- Mechanical construct: direct coupling of shafts from motors to the cube
- An open loop system

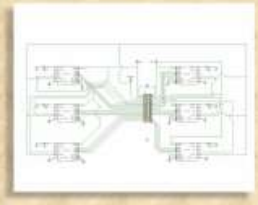
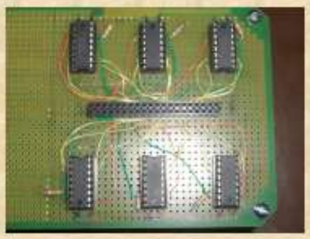
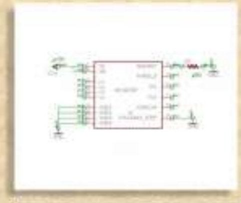
Mechanical Designs





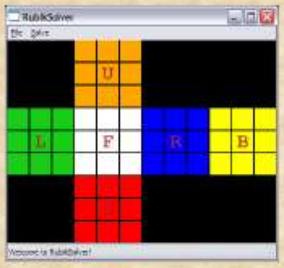
Cube pieces preparation
- DIY kit (www.Rubiks.com)
- Remove the springs of inner cube axles
- Inner edges are sanded down
- Center piece modification for tube coupling

Hardware and Circuits

Electronics:
- Six Bipolar Stepper Motors
- EB63 for the clock signals
- Stepper driver chips -- one for each motor.

User Software



RubikSolver:
- Koolemba's algorithm
- GUI
- Initial configuration
- Un-jamming of cube moves

Orb 3D Display

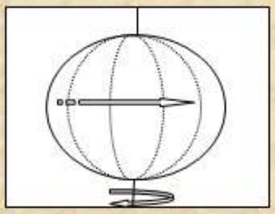
Orb I: Gabe Knezek, Sarah Berry, Mikhail Kisin
Orb II: Gabe Knezek, Brandon Culrich, Warren Linam-Church

Advisor: Steve Liu
Computer Science Department
Texas A&M University
Microsoft Research Faculty Summit
July 15-17, 2007

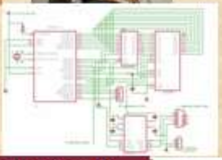
Motivations and Needs
- Most displays are 2-D flat planes that cannot render depth
- By rotating the display devices along a 3-D trajectory and synchronizing temporal and spatial timings we can create 3-D stereo vision effects

Our approach:
- A spinning ring with LEDs placed around the ring to light up pixels around a sphere
- Sensing of the spinning ring to synchronize temporal and spatial coordinates
- A daisy chain network implemented on a display bit bus
- Multiple microcontrollers use the daisy chain to coordinate with each other

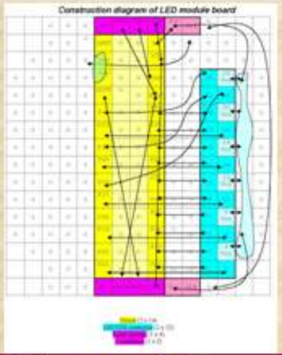
Conceptualization



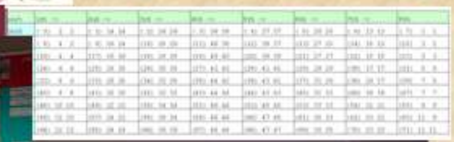
Critical Components: Slipring, location sensor, high intensity LEDs



LED controller board

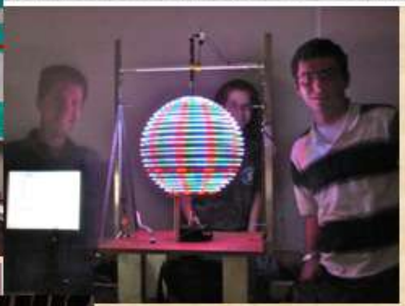


Memory map, LED map

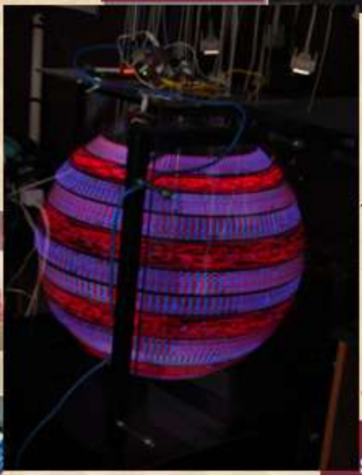


[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Orb II platform stress test



Orb I



Orb II

Mechanical Forces: 11G!

$$d = 20'' = 0.508m$$

$$r = 0.254m$$

$$v = 2\pi r(rps) = 2\pi(0.254)\left(\frac{1075rpm}{60\,{}^{sec}/_{m}}\right) = 28.594\,{}^{m}/_{s}$$

$$F_c = \frac{mv^2}{r}, F = ma$$

$$a_c = \frac{v^2}{r} = \frac{28.594\,{}^{m}/_{s}}{0.254m} = 112.574\,{}^{m}/_{s^2}$$

$$a_g = 9.8\,{}^{m}/_{s^2}$$

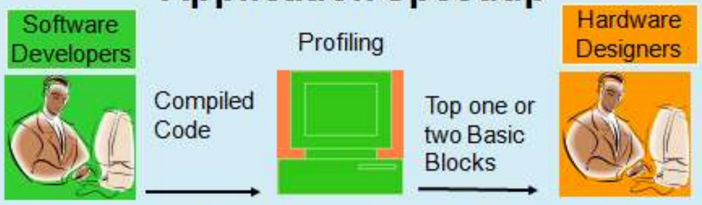
$$G - forces = \frac{a_c}{a_g} = \frac{112.574}{9.8} = 11.487$$

Warning:
Slipring may contain mercury
Orb platform must be tested for safety

eMIPS: the Dynamically Extensible Processor

Neil Pittman, Karl Meier, Alessandro Forin

Application Speedup



Original Binaries Modified to utilize Hardware Acceleration

Basic Blocks Implemented as Hardware Extensions

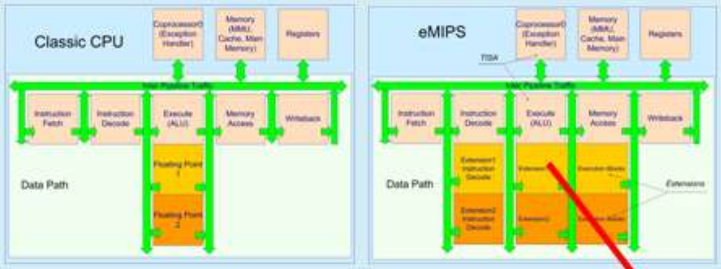
Result: Speedup Execution using a CPU Optimized to the Application

Binaries with Hardware Acceleration

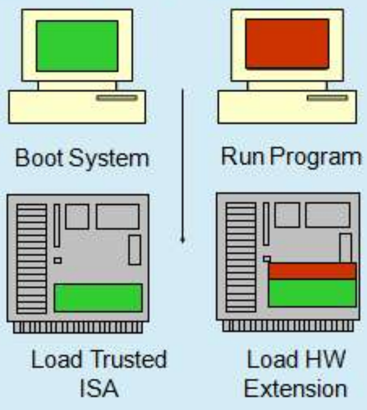
Extended instructions are inserted into the Binaries. If the HW Extension is loaded the instruction executes and skips the basic block. Otherwise eMIPS interprets the instruction as a NOP and executes the block.

```
Other code…
Op78      sp,ra,10      New
                        Instr.
Lw        ra,10(sp)     Basic
Jr        ra            Block
Addiu     sp,sp,18
Other code…
```

Architectural Comparison

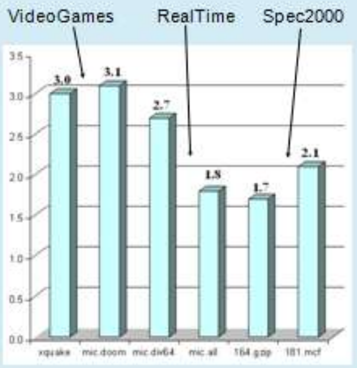


eMIPS Workstation



Time and Area Evaluation

Software Benchmarks



Area Consumption

eMIPS on the Virtex4 XC4LX25 FPGADevice

MODULE	SLICES	%Total	LUTs	%Total	CLOCKS	%Total
AvailableOnChip	10752		21504		32	
DataPath	3915	36.41	6812	31.68	2	6.25
Peripherals	2558	23.79	4714	21.92	4	12.50
BaseTotal	6473	60.20	11526	53.60	6	18.75
Extension	983	9.14	1608	7.48	2	6.25
Total	7456	69.35	13134	61.08	8	25.00

Implementation

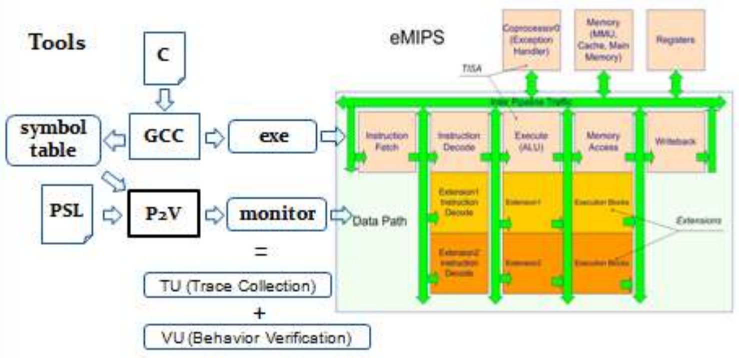
FPGA: Xilinx ML401 Virtex4 Eval. Board

eMIPS is synthesized for the FPGA. Add keyboard, mouse and disk for a complete workstation!

eMIPS Applications

Hong Lu, Bharat Sukhwani, Alessandro Forin

Zero-Overhead Online Verification of Software Programs



Concepts

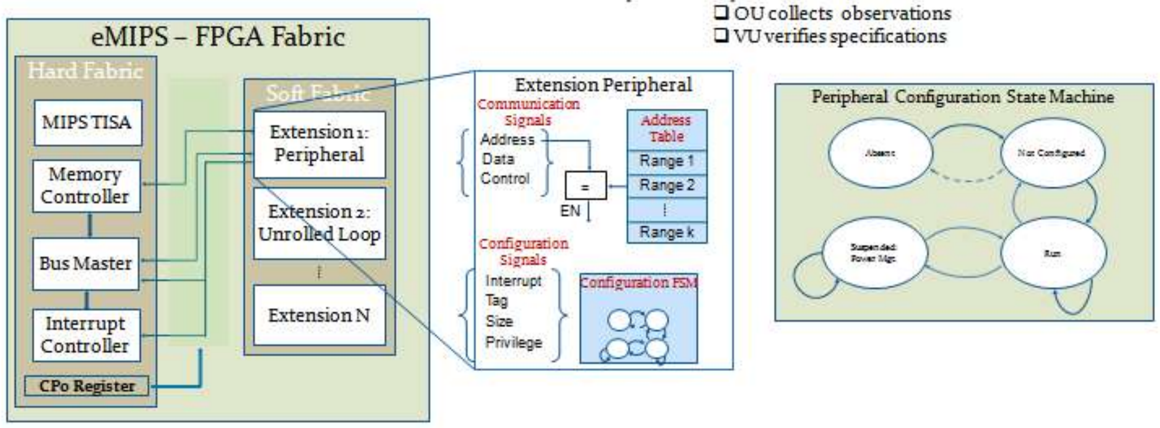
- ❑ PSL: property specification language
  - ❑ IEEE standard
  - ❑ system behavior specification
- ❑ P2V: PSL to Verilog Compiler

Use Cases

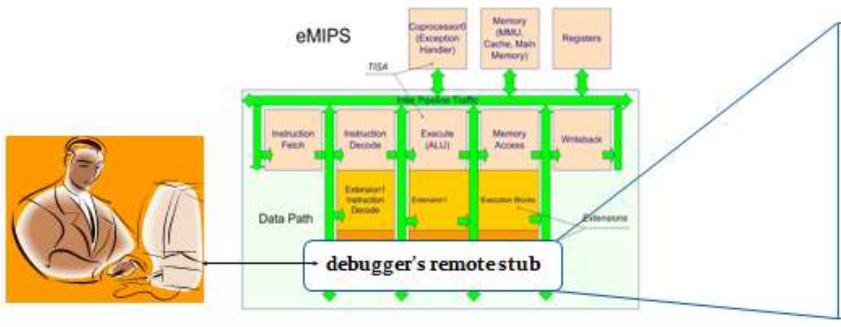
- ❑ Debugging and Testing
- ❑ System Intrusion Detection
- ❑ Real-Time Behavior Verification

Extensible On-Chip Peripherals

- ❑ OU collects observations
- ❑ VU verifies specifications



Safe, In-Process, JTAG-style Software Debugger



Debug Extension

- Take over the "break" instruction

Commands (from serial line):

-R/W registers & PC

-R/W memory

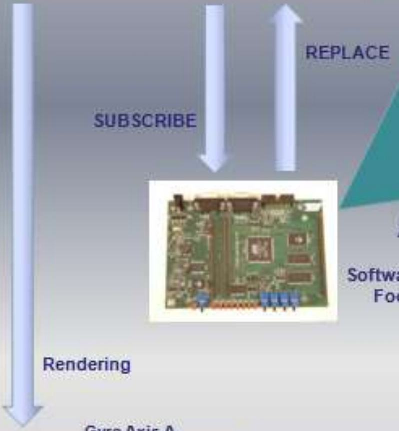
-Stall/Step/Continue

Connecting Microsoft Robotics Studio and Embedded Web Services

Oscar Almeida, Johannes Helander, Nishith Khantal, Henrik Nielsen
Microsoft Research & Microsoft Robotics

Robotics Studio

- Define & create services
- Support desired DSSP commands
- Communicate with other services
- Visualization (Simulation or Real)



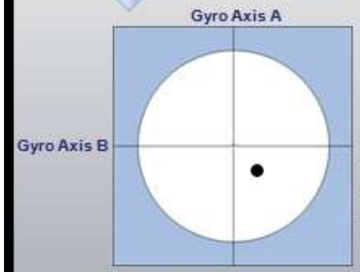
Embedded Service Directory Structure

- Service: Root
  - Service: Demo
    - State: Walking Stick
    - Service: Subscription Manager
      - State: List of Subscribers
  - Service: Trust Scripts

Atmel EB63 ARM Board
Hardware: 25MHz Arm7
Software: Microsoft Invisible Computing
Footprint: 32KB RAM 256KB ROM

Service Partitioning

Robotics Studio:	Subscribe to walking stick state
Embedded Client:	Poll gyro
Embedded Client:	Change state on Embedded server
Embedded Server:	Notify Robotics Studio of state change
Robotics Studio:	Change visual output



● Cane user's stability
□ Normal Operation
▨ Fall Detected!

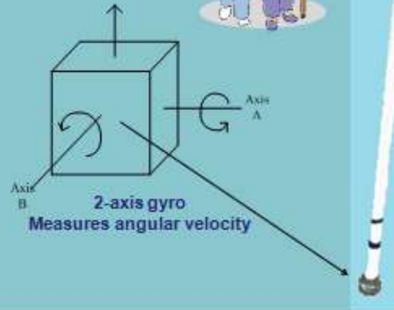
Stick Monitor Output

DSSP Commands

-Create	-Query
-Delete	-Replace
-Drop	-Submit
-Get	-Subscribe
-Insert	-Update
-Lookup	-Upsert
-Notify	

Medical Scenario

- Fall detection system built into cane.
- If a fall occurs, send alert via DSSP.
- Aid will arrive to the victim sooner.



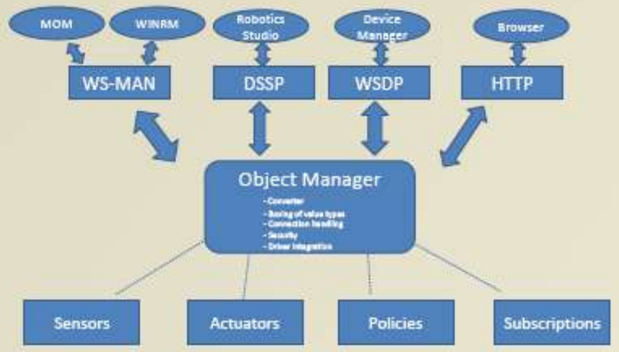
http://www.microsoft.com/robotics http://research.microsoft.com/invisible http://research.microsoft.com/research/EmbeddedSystems/EWS

Microsoft Research Faculty Summit

Data Centric Embedded Services

Johannes Helander, Nishith Khantal, Ben Zorn, Oscar Almeida
Microsoft Research

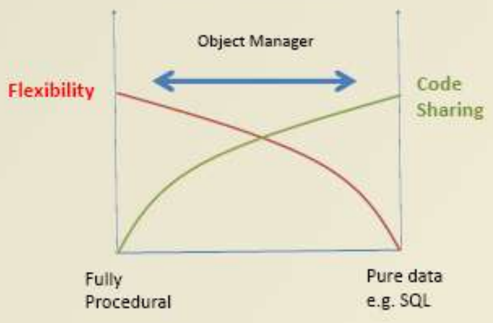
The Object Manager Provides Common Functionality



- Many kinds of objects, representing hardware, data, or both
- Accessible through multiple protocols
- Hybrid data model allows both flexibility and sharing
- Extends sophisticated applications to embedded systems
- Makes it easy to turn everything into a service

Hybrid Object Model

part *data*, part *procedural*



Combines the strengths of both models
- Data representations, communication, security, object initializations handled by common code
- Black box behavior when desired

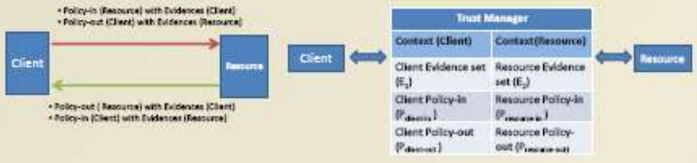
Trust Evaluation Based on Evidence

Combining evidences to build Trust

- Evidence: Passwords, HIP, Proximity, Puzzle, Quiz, Biometric, Peer Rating, etc.
- Incomplete and imprecise (fuzzy) evidence that changes over time
- Flexible policy description and evaluation
- Decentralized decision making

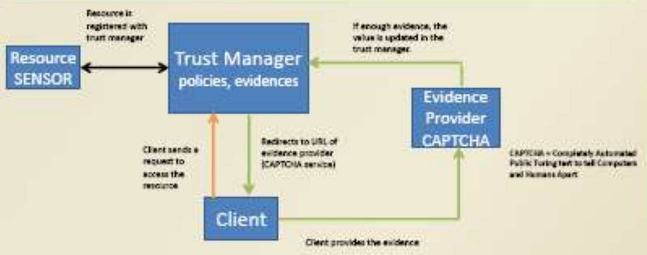
Access checks for peer-to-peer communication

- Resource specific policies (based on evidence known about client)
- Client specific policies (based on evidence known about resource)
- Trust evaluation for both incoming and outgoing messages.


• Policy-in (Resource) with Evidences (Client)
• Policy-out (Client) with Evidences (Resource)
Client
Resource
• Policy-out (Resource) with Evidences (Client)
• Policy-in (Client) with Evidences (Resource)
Client
Resource

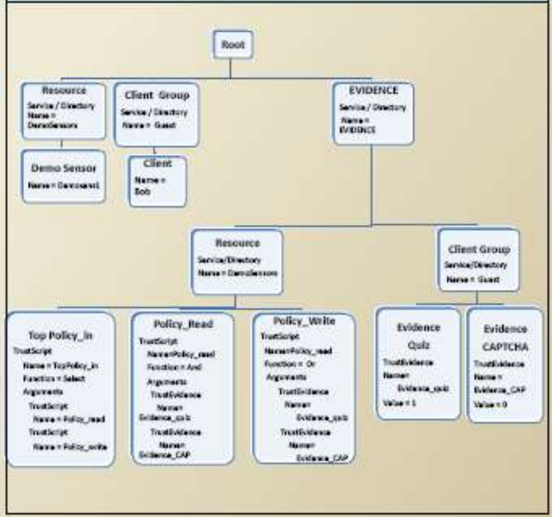
Trust Manager	
Context (Client)	Context(Resource)
Client Evidence set (E_c)	Resource Evidence set (E_r)
Client Policy-in ($P_{client-in}$)	Resource Policy-in ($P_{resource-in}$)
Client Policy-out ($P_{client-out}$)	Resource Policy-out ($P_{resource-out}$)

Demo: Communication Pattern


Resource is registered with trust manager
Resource SENSOR
Trust Manager policies, evidences
If enough evidence, the value is updated in the trust manager.
Evidence Provider CAPTCHA
Client sends a request to access the resource
Redirects to URL of evidence provider (CAPTCHA service)
CAPTCHA = Completely Automated Public Turing test to tell Computers and Humans Apart
Client
Client provides the evidence

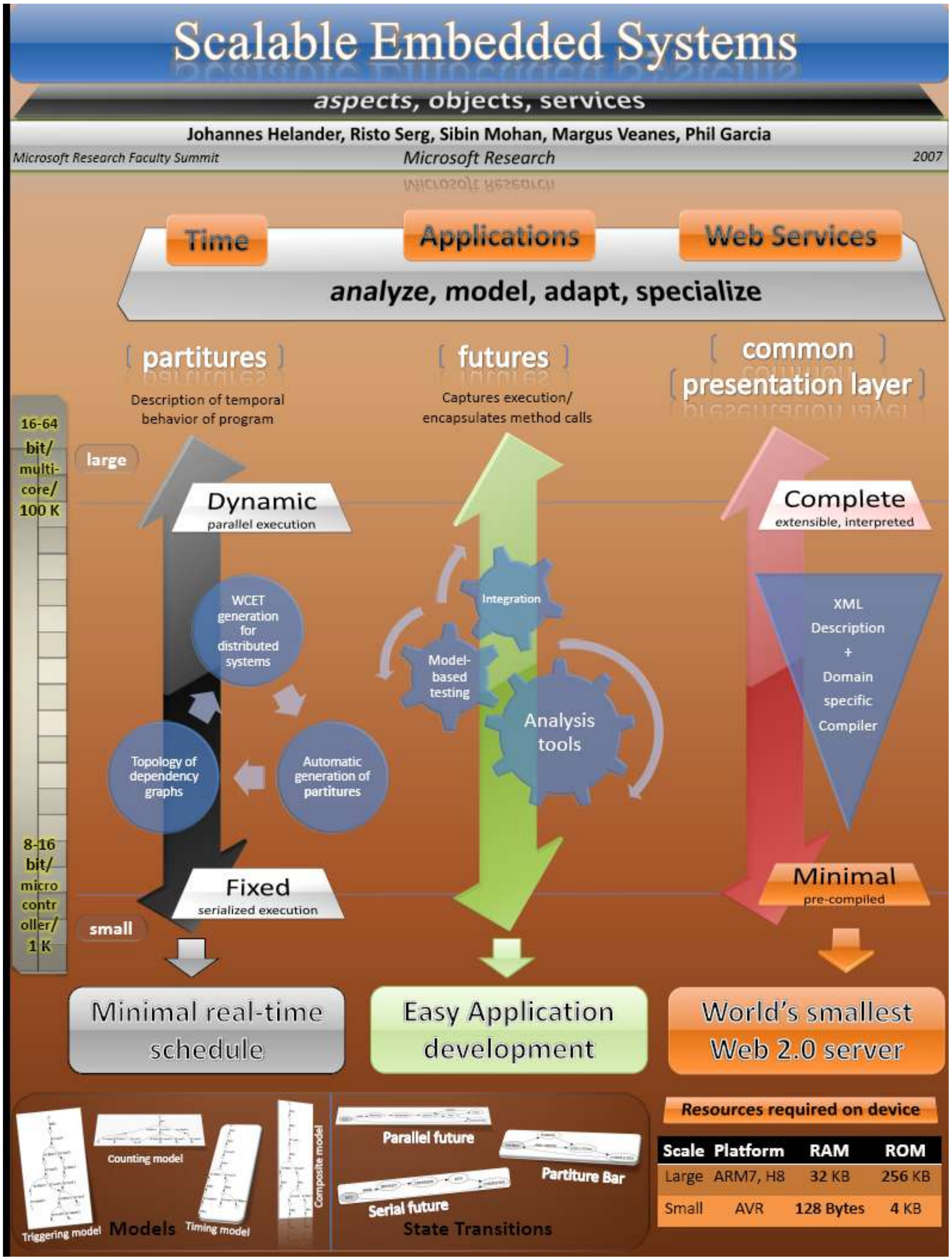
Some Trust Scenarios
- Casual interactions
- Robots
- Wiki
- Discovery (Multicast client)
- Protecting sensor data
- Smart walking stick

Demo: Object Hierarchy



Commercial: http://www.microsoft.com/whdc/system/pnppwr/wsm/Mgmt_WS.mspx

Academic: source code at http://research.microsoft.com/invisible



Scalable Embedded Systems

aspects, objects, services

Johannes Helander, Risto Serg, Sibin Mohan, Margus Veanes, Phil Garcia

Microsoft Research Faculty Summit *Microsoft Research* *2007*

Time
Applications
Web Services
analyze, model, adapt, specialize
[partitures]
Description of temporal behavior of program
[futures]
Captures execution/ encapsulates method calls
[common presentation layer]
16-64 bit/ multi-core/ 100 K
large
Dynamic
parallel execution
WCET generation for distributed systems
Topology of dependency graphs
Automatic generation of partitures
Integration
Model-based testing
Analysis tools
Complete
extensible, interpreted
XML Description + Domain specific Compiler
8-16 bit/ micro controller/ 1 K
small
Fixed
serialized execution
Minimal
pre-compiled
Minimal real-time schedule
Easy Application development
World's smallest Web 2.0 server
Triggering model
Counting model
Timing model
Composite model
Models
Parallel future
Partiture Bar
Serial future
State Transitions

Resources required on device

Scale	Platform	RAM	ROM
Large	ARM7, H8	32 KB	256 KB
Small	AVR	128 Bytes	4 KB

Appendix B: Movies and Pictures

The following movie demonstrates the Aggie Orb II in action. If you are looking at the Microsoft Word version of this document you can double-click on the icon to watch the video.

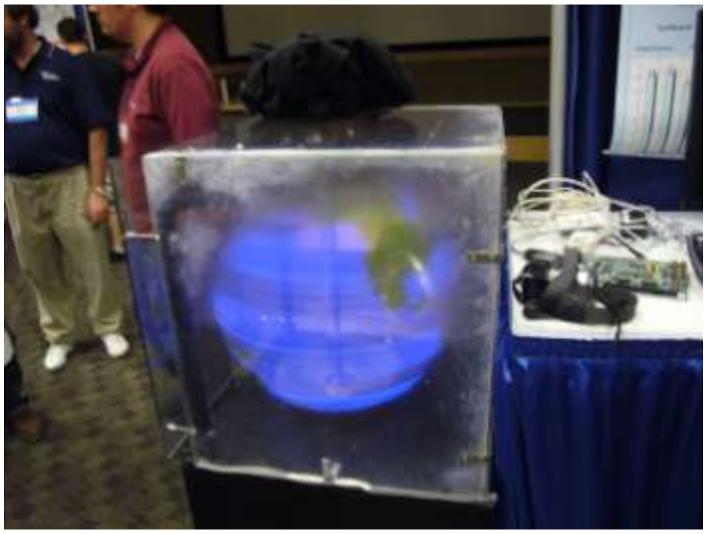
OrbDemo_July07-1.wmv

The next movie shows the Rubik Cube solver. Double-click on the icon to watch the video.

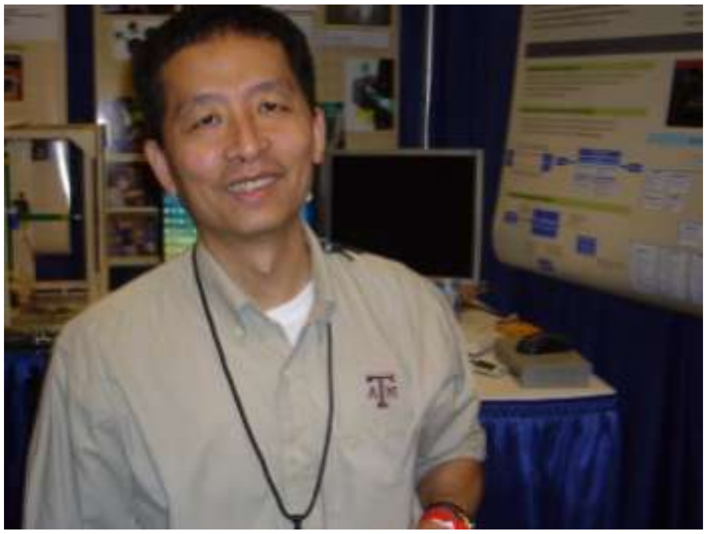
RubikCube.wmv

Here are a few pictures from the DemoFest booth.

Struggles of New College Graduates
in Their First Software Development Job



Complete
Analysis tools
Easy Application development
World's smallest Web 2.0 server



TUDIO