2012 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
VEHICLE ELECTRONICS AND ARCHITECTURE (VEA) MINI-SYMPOSIUM
AUGUST 14-16, TROY MICHIGAN

Performance of an Embedded Platform Aggregating and Executing Core Vehicular Integration for C4ISR/EW Interoperability (VICTORY) Services

Mark Russell
U.S. Army RDECOM-TARDEC
Contracted by UBT Technologies
Warren, MI

ABSTRACT

*The **V**ehicular **I**ntegration for **C**4ISR/EW In**t**er**o**pe**r**abilit**y** (VICTORY) Standard adopts many protocols that are traditionally used for developing enterprise application software deployed on general-purpose or server/workstation based computing platforms. This has led to discussions regarding the suitability of the VICTORY Standard for deployment to embedded and resource-constrained platforms. An independent software implementation of VICTORY core services was developed within the U.S Army Tank and Automotive Research, Development and Engineering Center (TARDEC) VICTORY System Integration Lab (SIL). These services were ported from a general-purpose computing platform to an embedded environment. Test procedures were developed and extensive performance tests were conducted to determine the feasibility of operating in this resource-constrained environment. This paper discusses the development procedures, implementation, test procedures, and performance results.*

INTRODUCTION

The Vehicular Integration for C4ISR/EW Interoperability (VICTORY) project is an initiative by the U.S. Army to improve upon current military ground vehicle electronics architecture. The VICTORY technical approach includes the use of shared services with well-defined application interfaces and protocols to achieve interoperability and reductions in size, weight, power, and cost (SWAP-C). A reference implementation of these VICTORY core services was developed at TARDEC. The preliminary stages of VICTORY software development, execution, and testing occurred on Linux-based workstations, specifically Dell (T3500). These machines contain an abundance of computing resources, including open-source libraries, sophisticated hardware, advanced operating systems (OS), and generous amounts of power on demand. In addition, most vehicular ruggedized computing units consume a lot of SWAP-C. The VICTORY SIL team researched alternatives to identify a single board computer that would be capable of running Linux GCC-based VICTORY service executables. The selected platform was an ARM-based development/hobbyist board that is representative of mainstream commercial hardware. This board, called the BeagleBoard xM, contains a Texas Instruments (TI) Cortex A8 32-bit ARM processor running at 1GHz [1]. Additionally, the BeagleBoard xM provides 4 GB of Package on Package (POP) SDRAM memory at 200 MHz. There are many external I/O ports, but of particular interest are the 4 available USB ports, RS-232 Serial port, and a 10/100 Ethernet Interface, which are used to connect peripherals and sensors, for performance testing and service functionality. In this paper, we describe the use of the BeagleBoard xM to execute and evaluate performance of VICTORY core services which form the VICTORY Data Bus (VDB).

SYSTEM CONFIGURATION & SETUP

The BeagleBoard xM's bootable MMC Flash card was partitioned to run a 32-bit ARM-tailored version of the Ubuntu 11 Linux OS. VICTORY executables were developed in C++ on an x86_64 running Red Hat Enterprise Linux (RHEL), and were cross-compiled for 32-bit ARM target on the host. The tool chain used is the GNU/GCC-based cross-compiler called Code-Sourcery. Specifically, the system under test is TARDEC's VICTORY v1.0 implementation [2]. The system provides VICTORY core services and defines that they shall consume various sensor data. The BeagleBoard is responsible for configuring and reading from the various external sensors. In our current system configuration, one VICTORY service reads data from a Global Positioning System (GPS) device attached via RS-232, and another VICTORY service reads data from an RJ45 Ethernet-based Inertial Navigation Sensor (INS) connected on the network. A third VICTORY service subscribes to VICTORY Position and Orientation Data Messages and calculates Direction-of-Travel. Furthermore, the BeagleBoard is implemented as an adapter between additional sensors and interfaces, such as an acoustic shot sensor, and a remote weapon station. Figure 1 below graphically depicts the high-level overview of the system:

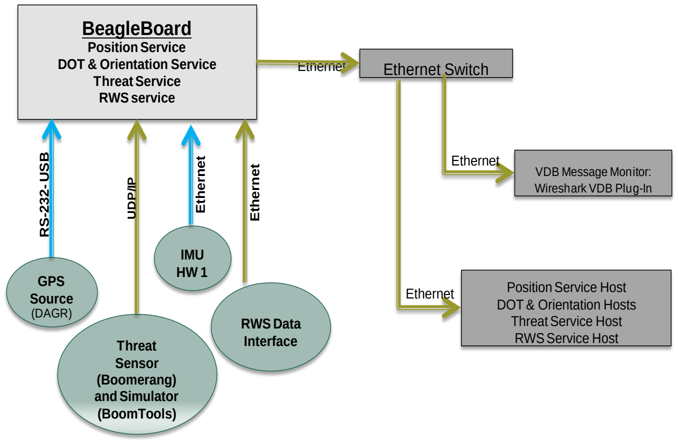

Figure 1: BeagleBoard System Overview

VICTORY requires that each service has a data interface for publishing data and management interface for monitoring and controlling the service [2]. First, the service must provide an interface to its service data; it accomplishes this by publishing its' data to an IGMP multicast group. Interested subscribers are required to join the multicast group and bind to the respective port to receive the data. The advantage of this methodology is that data is being transmitted only to the nodes who wish to listen/subscribe. VICTORY requires that each data message be encapsulated using a VICTORY-specified XML schema; they are known as VICTORY Data Messages (VDM), and are being served over the VDB.

Next, a management interface to the service is mandated. The VICTORY 1.0 Specification requires that each service shall have the capabilities to be managed by an alternate software technology. The technology that is required is by means of web-service based remote procedure calls (RPC). These management capabilities, or features, control many aspects of the service, including simpler functions such as enabling/disabling data transmissions, to more complex methods of rerouting the data via alternate transport locations. Additionally, management functions can also query a subset of the data that are published. For example, the Position Service is capable of publishing/serving a vehicular position point (latitude, longitude, and altitude) by both multicast UDP (data) and web-service based remote procedure calls (management). These management operations are facilitated by Simple Object Access Protocol (SOAP) technologies, and are specifically implemented with an open-source software development toolkit, called gSOAP [3]. The gSOAP toolkit creates web servers and clients that bind XML data types in WSDLs and XSDs to/from C and C++ data types. The data binding provides a type-safe and transparent solution through the use of compiler technologies that optimize the resulting code and ensures precise serialization. The VICTORY service is capable of functioning as a web server that provides a transport layer with an HTTP stack on top of TCP/IP. TARDEC has developed both the gSOAP web-service management server and web-service client to facilitate management operations (gets and sets) of the service.

SYSTEM PERFORMANCE BENCHMARKS

The goal of the experiment was to test and collect data illustrating the efficiency of the system under load. First, statistics were measured on the VICTORY Position service executed on the BeagleBoard; the service was configured to read the GPS via an RS232 connection. Using precise kernel timing methods, the following statistics were obtained: CPU utilization per second, average CPU utilization per hour, and minimum and maximum CPU Utilization. Examined were the units of work (clock cycles) the CPU performed on the process during a one second interval; this measurement is known as a Jiffie [4]. The current process' Jiffie(s) are compared to the entire amount of the CPU's Jiffies in the elapsed second, and a percentage is calculated. The average memory usage per hour was calculated, as well as average time required to construct as well as transmit a VICTORY data message. The results of these tests are shown below:

1. Average CPU Utilization: 1.053%
2. Average Memory Consumption: 0.772%
3. Average System Jiffies: 0.666
4. Minimum CPU Utilization: 0%
5. Maximum CPU Utilization: 3.061%
6. XML Build Time Average: 713132.49 ns
7. Message Transmit Time Average: 953089.1 ns

Second, there were two additional services under test, the Orientation Service and Direction of Travel Service. Both of these services were either as or more efficient than the Position service described above. Running all three service processes in parallel monopolized at maximum 3-4% of the CPU at any one time as displayed in Figure 2 below:

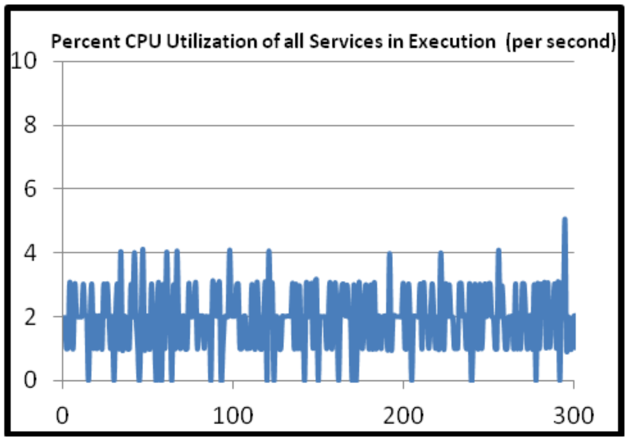

Figure 2: Percent CPU Utilization (%) vs. Time (Secs)

Third, the external sensors that produce synchronous data points will be removed from the system, and will be replaced by simple "synthetic" data. Specifically, this data will be a hard-coded value and will be used to illustrate the overhead the VICTORY-compliant sensors levy on the system. Results from this test are shown in Table 1 below:

Device	Service	Average CPU Utilization	Average Process Jiffies
GPS Sensor	Position	1.053%	.666
Synthetic GPS Data	Position	.375%	.388
INS Sensor	Orientation	.433%	.665
Synthetic INS Data	Orientation	.395%	.388
Obtained via Position & Orientation Data	Direction of Travel	.531%	.659
Synthetic Direction of Travel Data	Direction of Travel	.404%	.388

Table 1: Average CPU Utilization

The new average CPU utilization measurements for the services are: 0.375%, 0.395%, and 0.404% for Position, Orientation, and Direction of Travel Services respectively. When compared with actual obtained sensor data, the only service with a noticeable difference is the Position Service (1.053% versus 0.375%). Thus, when running VICTORY services that consume sensor data, it is found that each sensor accounts for 64.3%, 8.7%, and 23.9% of the process execution, respectively for each service. Running at 4800 Baud, the receive, process, and transmit states for the serial GPS device are the largest bottleneck in terms of system performance. The average number of system Jiffies did appear to remain constant between all services. Consequently, this would mean that the processes shared similar execution and sleep states; implying that the core of the services do approximately the same amount of work. The only variability would be the amount of time the process was using for I/O and sensor related computations.

PERFORMANCE OF WEB-SERVICE-BASED SERVICE MANAGEMENT

The goal of this experiment was to measure the overhead on the BeagleBoard by exercising its web-service interfaces. In this scenario, the example described above in the service management description was again utilized. The Position Service was executed with synthetic position data, thereby factoring out the overhead associated with I/O to external sensors. The test was started by running the Position Service and was connected to the corresponding Position Management Client. To determine how well the web-server technologies perform under heavy utilization, the Position Management Client was utilized to request vehicle position data at increasing frequencies. As a precursor, the time it takes for the client to request and receive a data point is 5 milliseconds. This duration was calculated using timers provided by the Linux real time clock in time.h. Table 2 illustrates the load the service is placed under at various RPC frequencies, and illustrates the fidelity at which the data was sent/received:

Frequency	% Average CPU Utilization	# of dropped Ethernet packets
1 Hz	.54	0
10 Hz	2.77	0
100 Hz	25.22	0
200 Hz	49.14	0

Table 2: Service Performance to Client Loads

Finally, the limits of the Position Service web server were tested. Realistically, there are not many systems where a client needs to obtain data at such high frequencies; however, it is still valuable to understand the limits of the system. After increasing the frequency of the data request, it was found that the client could make RPCs to the service application at a maximum rate of 200 Hertz; simply because it takes the client .005 seconds to request/receive a position data point. Consequently, for this single client and server test, the server was unable to be exercised at frequencies higher than 200 Hertz. It is also noteworthy that the OS statistics did not record a single dropped Ethernet packet. This implies that the gSOAP client/server implementation is a very reliable transportation model. In addition, this test suggests that the VDB provides high-availability, which may help mitigate some of the concerns regarding the fidelity of the VDB request/response model.

POWER CONSUMPTION

The BeagleBoard's 5 Volt power supply was attached to a non-invasive current probe which connected to a digital oscilloscope. The oscilloscope captured the current draw of the system in its basic running state, which includes the following: the Ubuntu OS executing a 2.6 Linux kernel, running a stripped-down X-windows GUI that was being driven to an LCD display, and having no user applications executing. This state shall be referred to as the system having VICTORY services/applications at rest. It shall be used as a baseline and is a benchmark for future tests. Recording the power measurements in this state illustrates that a stable 0.62 Amps were being drawn, implying a 3.10 Watts accordingly.

The amount of additional power consumption was almost negligible after starting the core VICTORY services / processes. The average current grew to only 0.02 Amps to 0.64, while power consumption rose merely 0.1 Watts to 3.2. Viewing a small snapshot in time while running the services, Figure 3**Error! Reference source not found.** and Figure 4 below depict the current draw as measured on the BeagleBoard:

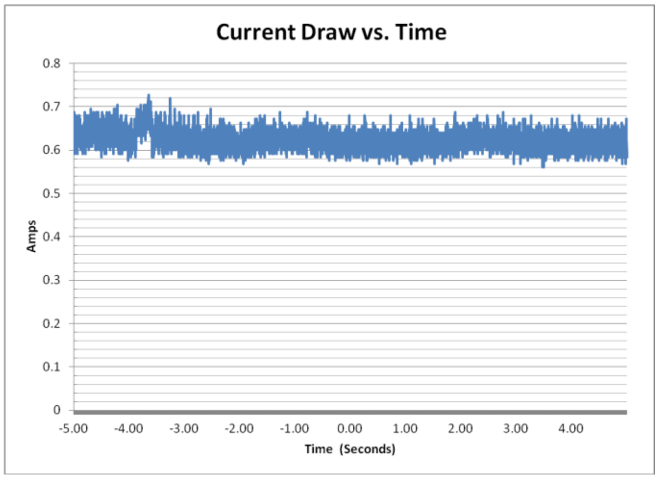

Figure 3: Current Draw(Amps)- VICTORY Services at Rest

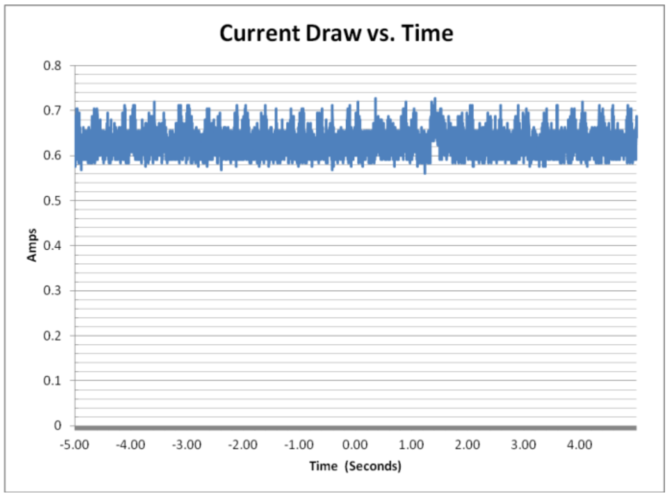

Figure 4: Current Draw(Amps) with VICTORY Services in Operation

Therefore, the electrical draw on the system incurs a very small usage on the overall system when running the services.

As a comparison, the same tests were performed on our development workstations, which are Dell Workstation PCs running 6-cores. The same tests were executed while running the same VICTORY services; this resulted in an observed 0.85 Amps RMS. This consequently computes to a total average power consumption of 72.12 Watts. Thus, the magnitude of 72.12 Watts versus the 3.15 Watts measured on the BeagleBoard represents a savings of 68.97 Watts, and is 95.6% more efficiently to run the very same services!

CONCLUSION

Running multiple VICTORY-based services on the ARM-powered BeagleBoard demonstrated power savings, efficient execution, and VICTORY-service functionality. For a small, light-weight, low-powered system, it performed well beyond expectations. The most demanding experiments that were performed showed little overhead on the system. In summary, executing multiple VICTORY data services, and reading multiple VICTORY-compliant sensors at the same time resulted in the following performance measurements for the system:

- 0.64 Amps / 3.15 Watts Power Consumption at run-time.
- Roughly 0.77% System Memory Utilization per Service
- 1.05%, 0.433%., 0.531% average CPU utilization for Position, Direction of Travel, and Orientation Services, respectively.
- Less than 1.7 milliseconds processing time (Building and Publishing Full Featured VICTORY XML Messages).
- A delta of 68.97 Watts between workstation and BeagleBoard, which is 95.6% more efficient.

Therefore, it is proven that boards based on this type of architecture are an excellent candidate for running VICTORY services while providing significant reductions in SWAP-C for the VICTORY 1.0 project.

REFERENCES

[1] BeagleBoard-xM Rev C System Reference Revision 1.0, BeagleBoard.org, 2010. http://beagleboard.org/static/BBxMSRM_latest.pdf

[2] Vehicular Integration for C4ISR/EW Interoperability (VICTORY) Standard Specifications Version 1.0, VICTORY Standards Support Office, 2011.

[3] Van Engelen, Robert, gSOAP 2.8.9 User Guide, Genivia Inc., 2012. http://www.cs.fsu.edu/~engelen/soapdoc2.html

[4] Kerrisk, Michael, The Linux Programming Interface, No Starch Press, San Francisco, 2010