2015 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
VEHICLE ELECTRONICS AND ARCHITECTURE (VEA) MINI-SYMPOSIUM
AUGUST 4-6, NOVI MICHIGAN

Technical Challenges Running TARDEC VECTOR Software on ARM Architecture

Mark Russell
U.S. Army RDECOM-TARDEC
Warren, MI

ABSTRACT

TARDEC VEA will begin integrating their Vehicular Integration for the Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance / Electronic Warfare (C4ISR/EW) Interoperability (VICTORY) Enabled Company Transformation (VECTOR) software onto three unique military vehicles in FY2015. One of the main objectives of VECTOR is to evaluate the VICTORY standard. VECTOR will use the aforementioned military vehicles as a platform for integration with the VICTORY software library (libVictory). The feasibility of expediting component integration and enhancing vehicles in theatre will be assessed; VECTOR will attempt to leverage the capabilities of libVictory in order to do so. One of the key deliverables for VECTOR is the capability to port the software applications and middleware configuration items to an embedded low-cost ARM architecture. The VECTOR team selected a unique hybrid system that includes both a single board computer and an Ethernet switch. This research paper will present the rationale for porting VECTOR software to the ARM architecture and explain the details of overcoming technical challenges therein. Due to the fact that the ARM architecture is designed for mobile applications, computing hardware resources and software availability are much more limited than common Intel-based desktop or workstation computers. Consequently, the level of technical risk is greatly increased, and the viability of executing the required full-blown, graphical desktop application is less likely. This paper will capture these shortcomings and will illustrate the difficulties of overcoming the hurdles of limited computing resources. Finally, this paper will provide some quantitative measurements of performance (resource utilization, graphical interface lag times, and update rate thresholds) as observed on the embedded ARM processor (also referred to as the target), and how certain optimizations can improve the system.

INTRODUCTION

The *VICTORY Enabled Company Transformation (VECTOR)* is an initiative created by U.S. Army RDECOM-TARDEC to instantiate the Vehicular Integration for C4ISR/EW Interoperability (VICTORY) software library (libVictory) on three existing military vehicles. libVictory is a software library containing VICTORY-compliant components and is openly available on software.forge.mil under the DoD Community Source License. The three vehicles that are being outfitted with new VECTOR hardware and software include the Family of Medium Tactical Vehicles (FMTV), High Mobility Multipurpose Wheeled Vehicle (HMMWV), and Stryker. The three vehicles have varying levels of VECTOR hardware and software integration. The outcome of these vehicle transformations will help provide invaluable data to vehicle Program Management Offices (PMOs) that document scalable software and hardware architectures. It will also document integration and transition efforts of VICTORY onto their respective vehicles. Ideally, this effort will reduce risk; both in aiding integration of legacy components and providing a common integration package. One of the primary goals of VECTOR is to develop a modular, reusable, and adaptable architecture that shares common interfaces and components. The flexibility and adaptability of VECTOR software would clearly be demonstrated by successfully porting to additional computing

architectures; doing so would be a significant achievement for the program. In particular, the ARM architecture was targeted due to its low-cost, mobile, and embedded nature. Custom-build software items that are required to run on the target include the VICTORY software library (libVictory), and the VECTOR main applications, middleware components and custom adapters. The target will also require the configuration and use of low-level interfaces and drivers. Demonstrating that VECTOR software can execute directly on an ARM single-board computer (SBC) will speak to the inexpensive nature of the VICTORY architecture. In addition, it benefits the project to maximize the use of ARM processors to reduce size, weight, power, and cost (SWaP-C) constraints. These findings could help provide confidence to current and future programs of record that higher-end Line Replaceable Units (LRUs) could execute the same VECTOR software on existing platforms with greater ease. After completing the initial challenge of physically porting all of the related software configuration items to the ARM processor, the resulting process could provide a much more defined developmental and integration roadmap. For the majority of the engineering development, VECTOR code and third party applications and libraries were created on Intel© x86 / 64 hosts. This research will focus on difficulties that were overcome in migrating to ARM. Additionally, this research will address hardware selection, Operating System (OS) selection, SWaP-C savings, and will expound upon application performances on the target hardware.

HARDWARE SELECTION

VECTOR systems engineers engaged in an Analysis of Alternatives (AoA) process to determine the best possible hardware solutions early in the project acquisition phase. There were many ruggedized processing units that were available from various defense manufacturers. Engineers first selected an Intel-based mission processor to host VECTOR applications, however, the VECTOR project also desired an embedded low-cost processor to benchmark against the dedicated mission processor. Combining this need with the requirement to have a ruggedized networking switch guided the team's market research to investigate multi-function computing and networking systems. The new vehicle system architecture requires a managed switch for all VICTORY data and streaming video network traffic. An explicit requirement of VECTOR is to reduce SWaP-C constraints whenever possible; procuring a computing appliance that combines a managed network switch with a single board computer (SBC) satisfies this requirement. Finding suitable appliances that conform to ruggedized military standards greatly limits purchasing options, as there is not much market penetration for these uniquely hardened and consolidated systems. Further adding complication to the hardware selection process is VECTOR project's optional requirement to run 18 VICTORY software components simultaneously within one device. The search was narrowed down to two competing products, which are labeled option A and B. Option A is an ARM-based product, and Option B is a similar product, but with higher-end components running on Intel architecture. Table 1 below summarizes the comparable features of each.

Feature	Option A	Option B
SBC Processor	Dual-core ARM Cortex-A9 800 Mhz	Dual-Core Intel Core i7 2.2 GHz
SBC RAM	1 GB DDR3	8 GB DDR3
SBC Storage	16 GB Flash (8 GB usable)	128 – 1000 GB SSD
Power	~23W average	~90W average
Weight	6.5 lbs	20 lbs
Size	~ 10.5" x 7.5" x 3.0"	~ 6.6" x 6.75' x 6.25"
Switch	Managed 16-port Gigabit	Cisco 5915 18-port Embedded Services Router
I/O Interfaces	Gig Ethernet, CAN, AIO, DIO, RS232/422, USB, VGA, DVI, RS-170, GPS, IMU	Gig Ethernet, RS232, Audio, USB, VGA, HDMI, DIO, PS2,
Supported OSes	Linux	Linux, Windows 7 & Embedded
Unit Cost	~ $8,500.00	~ $23,700.00

Table 1 - Hardware Comparison

The system engineers chose Option A because of its smaller size and weight, efficient power consumption, lower cost, and extensive feature set. It fulfills the project's wishes for a light weight embedded platform. This device will now be referred to as "VICTORY-in-a-box", as it provides switching capabilities, VICTORY Shared Processing Unit (SPU) functionality, and legacy I/O adaption. VECTOR requirements state that hardware shall be capable of communicating with an inertial navigation unit (INU), Controller Area Network (CAN) device, Global Position Service (GPS) device, Commander's Remote Weapon Station (CROWS), Solider-Machine Interface (SMI), threat sensor, and pan-tilt-zoom (PTZ) camera. When legacy components are used that do not have VICTORY compliant interfaces, the VECTOR solution is to provide adaptation of legacy Inputs/Outputs (I/O) with the SPU to run the libVICTORY software. Figure 1 below is a graphical depiction of the how the VICTORY-in-a-box device

could be connected to various hardware components. The green circles represent "adapter software" for legacy or non-VICTORY-compliant interfaces.

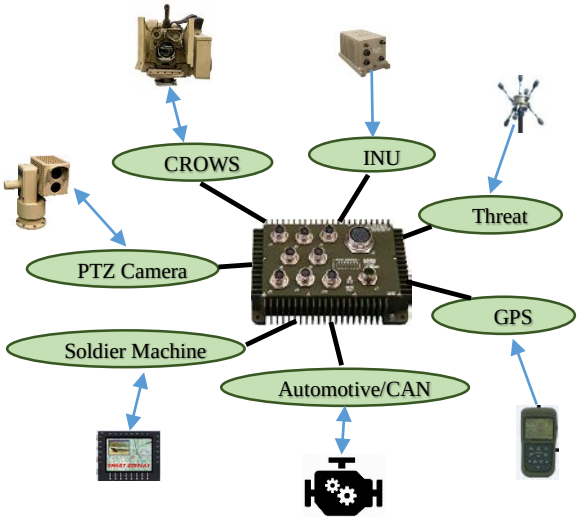

Figure 1 – VICTORY-In-A-Box Device Connectivity

Executing VECTOR software on both the mission processor and the VICTORY-in-a-box device will demonstrate the scalable nature and can provide excellent side-by-side comparisons for project stakeholders.

FEASIBILITY OF ARM ARCHITECTURE

The first ARM processors were introduced to the market roughly 30 years ago in 1985. Technically speaking, ARM is a family of instruction set architectures that is built around a reduced instruction set (RISC) architecture. One of the largest advantages to the RISC architectures is the fewer numbers of transistors needed for the reduced number of instructions. Fewer transistors means less power use, less heat, and less cost. This feature makes ARM processors particularly appealing to mobile and embedded devices where resources are at a greater premium. Contrast this ARM technology to a complex instruction set (CISC) architecture, i.e. Intel© x86, where processors typically run much hotter and consume more power. The ARM Cortex-A9 processor built into the VICTORY-in-a-box device is very representative of hardware that is built into cellular devices. However, this strength also happens to be one of ARM's weaknesses as well. The ARM-based system used for VECTOR is constrained by available resources including limited voltage, limited computing horsepower, and limited storage. Concurrently executing multiple complex VECTOR software processes will present a greater challenge. An additional downside of using the ARM architecture is its' more limited support for software, particularly when it comes to open source and enterprise Operating Systems.

OPERATING SYSTEM SELECTION

A study was conducted early in the software development phase. The purpose was twofold: first, to conduct an early use case analysis to drive the initial VECTOR requirements and architecture; and second, to select one or more Operating Systems to support the VECTOR hardware. Two requirements that steered a majority of the Operating System analysis included preferring commercial off the shelf (COTS) products as well as ensuring compatibility with multiple hardware architectures. Four major types of Operating System types were initially considered, including Linux variants, BSD Unix, Windows variants, and Real Time Operating Systems (RTOS). BSD Unix was quickly ruled out because of its' limited hardware support. RTOSes were also ruled out because they tend to be largely proprietary and the project had no hard real-time requirements. Continuing further, the Linux flavors that were candidates were RedHat, CentOS, Debian, and Ubuntu. Alternatively, the Windows candidates were Enterprise 7, Embedded Standard 7, Enterprise 8, Embedded Standard 8, and RT. After an investigation of Operating System suitability, all Windows variants were ruled out largely in part because of their lack of cross architecture compatibility. The study determined that all Windows candidates, with the exception of RT, ran exclusively x86/x64 systems, while RT runs only on ARM. The study made additional arguments against selecting Windows as the VECTOR software is not currently Windows compatible, as Windows' limited POSIX support is not well suited for embedded platforms. The study identified that Linux has many advantages for VECTOR, including its level of POSIX compliance, compatibility with existing VICTORY software, availability of standard packages/libraries, excellent hardware support, and the availability of Security Technical Implementation Guides (STIGS). For these reasons, Linux was the winner for Operating System type, but it was still necessary to select one of four Linux variants (flavors). RedHat and CentosOS do not support alternative architectures at all, thus the two remaining viable choices were Ubuntu and Debian. The Linaro engineering organization has successfully created Ubuntu ports onto many different ARM architectures. However, at the time of the study, ARM boards were not officially supported by Ubuntu's producer, Canonical, with the exception of two development boards [1]. Therefore, Debian was

chosen because of its official support for a wide range of architectures, including ARM (target) and Intel (host) architectures. The only criterion that did not pass with Debian was a security requirement stating that the Operating System shall be a trusted Operating System. This is a hurdle that could be overcome. All but one other Linux distributions also fell victim to meeting this requirement. The final decision to make was selecting Debian version 7.2 or version 8.0. This decision is not trivial and will be explained later in this research.

VICTORY-IN-A-BOX DEVICE INITIAL SYSTEM DIFFICULTIES

The VECTOR software engineers soon learned that there was a customized version of the Ubuntu 11 Operating System installed onto the target as the factory default. This version of Ubuntu had many patched Linaro updates applied to it that allowed it to run on the board's unique architecture. Unfortunately, this distribution was released in 2011 and is not a long-term support (LTS) product. Because Ubuntu 11 software packages are no longer available on their trusted repositories, additional dependencies required of libVictory and other VECTOR modules could not be installed. Consequently, the software could not compile. Therefore, it was decided the engineers would follow the recommendation of the study to install the Debian Operating System on the target. Choosing between the current stable or newest unstable version of Debian also came with its own set of challenges. Note: at the time of this work, the stable version of Debian was named "Wheezy" (version 7), and the unstable version was named "Jessie" (version 8). There were a lot of hang-ups in getting a working Linux system for the target. It is imperative to first find a Linux kernel that is compatible with the Linux userland, which are user applications that run outside of privileged kernel space. Many permutations of system configurations failed. Most failures involved kernel boot failures or non-functioning hardware / peripherals. For example, the combination of a Jessie kernel and a Jessie userland resulted in strange 2-colored video output from the VGA port. It was obvious the custom driver in the kernel was not setup correctly. There were a lot of ARM-based kernels available for Jessie, but they only somewhat resembled the actual VICTORY-in-a-box hardware. Each kernel attempted resulted in a kernel panic or a malfunctioning hardware device driver. Only after using the original stock Ubuntu kernel shipped with the target in combination with the Jessie userland was progress made. Fortunately for the project, the ability to execute all user applications proved that the userland had backwards compatibility with our older kernel and was able to make the system operational. At the end of the day however, Jessie did not complete its' booting process. Jessie replaces the older UNIX System V init-based boot stages with "systemd". Systemd is a basic building block for Operating Systems. It has containers for groups of processes called Control Groups (cgroups). Cgroups are used in controlling, managing, and throttling groups of processes with one interface. Because the older Ubuntu kernel does not have built-in support for cgroups, the boot process would end abruptly at a single-user command-line only mode as the root user. Users were required to manually start the required startup scripts, the X server, and associated VECTOR applications to continue. This type of system user interface would not pass the most basic of security certification tests. Therefore Jessie was abandoned in place of the older Wheezy version of Debian. Finally, Wheezy allowed for a full booting process that completes the System V five stage init process and automatically launches the X-windowing interface. The difficulties encountered with Debian Wheezy will be discussed in the next section.

SOFTWARE INTEGRATION STUMBLING BLOCKS

In addition to overcoming hurdles with the configuration of the VICTORY-in-a-box Operating System, there were many obstacles in generating complete and working application binaries for the target. There were three complex issues that required troubleshooting to overcome. Two of these issues involved external libraries that are not maintained by TARDEC. The three stumbling blocks are listed below; the first being TARDEC's own libVictory:

1. Finding a suitable compiler toolchain for the board hardware and Operating System proved to be more challenging than expected. The first steps in compiling libVictory involved creating target binaries by cross-compiling on a build host. One mandatory dependency that libVictory requires is the use of the newer C++11 coding standards. LibVictory requires C++11 threading and time-related features, which were introduced in to the GNU compiler toolchain (GCC) in version 4.7.3. The first arm toolchain used, arm-linux-gnueabi-4.7, did not have the necessary C++11 support available yet because it was not revision 3. The second arm toolchain, arm-linux-gnueabihf-4.8, which did have C++11 support, had a newer version of the low-level libc than the libc resident on the target. For these reasons, the resulting binaries did not run on the target and the project was forced to switch to the stable "Wheezy" release of

Debian. Initially, when engineers built libVictory in Debian Wheezy, the same problem C++11 existed as described above. The compiler did not have the necessary C++11 updates and features. Luckily for libVictory software, there was an update for the native GCC compiler on Debian's repository. This update provided the necessary C++11 features that were added as of 4.7.3, and thus created a working binary directly on the target.

2. The graphical warfighter machine interface (WMI) display screens host a visually intensive application. The WMI merges many technologies together into one large graphical-based Qt windowing environment. The software was written for and tested on Intel workstations, and comes bundled with a handful of dependent third-party libraries. For an understanding of just how large and complex this software is; it takes developers 30 minutes to compile the WMI application from source using an Intel workstation with four cores simultaneously. First, a week was spent trying to cross-compile the third party libraries, but with very limited success. Those who may be familiar with cross-compiling applications will tell you that the practice can be a black art; the process of configuring Linux Makefiles is never the same from one application to the next. There are often unmet dependencies as well. This task was no exception and the roadblocks continued. The next course of action was to begin compiling natively on the target itself. This certainly eases integration and level of difficulty, but a large margin of performance is sacrificed to accommodate the much slower computing speed used in compiling. One benefit of operating natively on the target itself is the ability to install pre-installed Debian packages that are available on the Debian repository. The WMI application required third party libraries including video decoding capabilities (codecs), mapping tools, image manipulation, geospatial data abstraction, distributed messaging systems, cartographic projections, and Qt. Some of these were available as Debian packages, while others had to be compiled from the source distribution. Potentially the greatest challenge was compiling the Data Distribution Service (DDS), which is a highly configurable, high-performance publish/subscribe messaging system. The WMI application is highly dependent upon the Prismtech's commercial Vortex OpenSplice DDS library for its intra-node communications. Without a working DDS implementation, the WMI would not be useable. More specifically, the application is built around Vortex OpenSplice version 5, and ARM architecture support was not introduced until version 6. Version 6 represents a major upgrade from version 5 and is not backwards compatible. Naturally, version 5 did not compile for the target, as there are some optimizations built in that are written in assembly for Intel-only architecture. The solution to this problem involved the non-trivial route of replacing the Intel assembly with ARM assembly. Surprisingly, the software successfully built and ran; it is potentially one of the only version 5 Vortex OpenSplice implementations running on an ARM system.

3. At the backbone of libVictory software's management feature set lies Genivia Inc's gSOAP library. gSOAP is full-featured web services toolkit that features not only data serialization capabilities, but also well-formed eXtensible Markup Language (XML) remote procedure calls (RPCs) over Transmission Control Protocol (TCP) via its built-in webserver. The intent is for VICTORY system designers to build full-featured systems in XML only. gSOAP will then automatically convert XML into C or C++ data structures. This feature allows for system designers to create a working model without having any knowledge of how to program. The problem with this method is that the summation of all of the VICTORY services into data structures creates a very large digital footprint. gSOAP additionally produces large amounts of auto-generated code, including conversion functions, typedefs, new and delete operations, and other helper functions. The resulting object code that is created from gSOAP is over 20 Megabytes in size. Having large executable sizes in today's modern workstations is a moot point, simply because of the inexpensive nature of computing resources. However, in the context of embedded computing, the limited size in this scenario was a stumbling block. In attempts to compile libVictory natively on the target hardware itself, the GCC compiler would experience a crash. Even when enabling 16 Gigabytes of external swap space, GCC would still crash and issue internal errors. GCC even suggested filing a bug report against it. The resources needed to compile the gSOAP library were simply too great for the target to handle. Meanwhile, the gSOAP library itself was able to be cross-compiled on a workstation and copied over to the target. This allowed the target build to run to completion. However, having a multi-stage and multi-host compilation process does not make for a suitable source control and content management practice. Only after combing through the gSOAP documentation was a solution to this problem found. Within the gSOAP toolkit is a program called soapcpp2. Soapcpp2 takes input from XML files and serializes the data into C or C++ format. If the option of '-f N" is passed to soapcpp2, the tool will split into

N XML serialized implementation files. For example, if –f 40 was given for N, there will be 40 resultant object files instead of 1. In the example of building libVictory, soap object files will now be broken into 500 Kilobytes chunks. After making this change, the gSOAP library and libVictory built seamlessly on the target.

EXECUTION OF VECTOR APPLICATIONS

The FMTV vehicle's applications place the largest load of all three vehicles onto the VICTORY-in-a-box device. There are four main processes that run concurrently on the target. The most demanding software is the WMI application, followed by the FMTV commander application, and lastly the separate CAN and GPS adapter software utilities. The WMI application is the graphical front-end touch-screen user interface. The underlying framework of this application is built upon the Qt library and it provides window management, graphical layout, custom widgets, and responds to user interactions. The application provides a full range of vehicle information that is obtained through communications with libVictory. This information is displayed graphically with custom vehicle widgets, and is typically layered on top of a tactical map. The WMI application also provides a client display for a video stream that is originated from a network-attached PTZ camera.

The aforementioned VECTOR software "adapters" that execute simultaneously with the WMI software are small snippets of code that adapt legacy device data into VICTORY messages. The FMTV runs the GPS adapter that converts serial messages into VICTORY messages. Likewise, the CAN adapter converts 16 unique CAN messages into VICTORY messages. Finally, VECTOR's FMTV commander application is the executable that ties all these pieces together. It instantiates libVictory "readers", whose function is to listen for incoming VICTORY messages and forward them onto an interested party. In this particular scenario, the commander listens to incoming GPS and CAN messages and then frequently forwards them onto the WMI application, and graphical updates are made as needed. The GPS VICTORY message updates the vehicle position and it's placement on the tactical map moves accordingly. Meanwhile, the CAN messages read many types of automotive data, including fuel level, for example. As the vehicle continues to operate, the on-screen fuel gauge updates in near real-time.

The vehicle test-benches in TARDEC's lab provided a means to measure resource usage of these applications in execution. The table below provides some quantitative measurements of software in execution. Note: with multi-core processors, the summation of CPU utilization can be greater than 100%, each CPU can contribute up to 100% each.

Process	CPU %
WMI	115%
WMI (with Streaming Video Client)	145%
CAN Adapter	20%
GPS Adapter	6%

Table 2 - Process CPU Utilization

The video streaming capabilities were also tested. The video feed comes from a network-based PTZ camera and is streamed at 640x480 in x264 (MPEG-4) format. The WMI application provides two metrics for frame rate: D fps and R fps. D fps is the number of frames decoded per second and should typically match the output frame rate of the source stream. The R fps is number of frames rendered per second and can be manually throttled back in a configuration setting if desired. D fps for the video feed was 30, and R fps was 22. 22 fps is not perfect, but is more than adequate for the project needs.

The interaction with the WMI GUI was better than expected. The smart display touch screens respond nearly instantaneously to map zoom or move gestures and button clicks. There were virtually no noticeable visual or tactical performance differences between the Intel-based mission processor and the VICTORY-in-a-box device, both of which run VECTOR applications.

CONCLUSION

The VECTOR program examined the feasibility of running a full-featured suite of VICTORY-based applications and middle-ware programs ported to ARM architecture. It was important to prove that running VECTOR (and therefore VICTORY) software could be done very inexpensively and with limited resources. Despite many technical difficulties in Operating System selection, software compilation, and software integration issues, the VECTOR applications executed without noticeable deficiencies. Now, vehicle programs desiring to become VICTORY compliant have an affordable and rapid entry point with the VICTORY-in-a-box solution. Using this hardware could represent significant cost savings for vehicle integrators. The VECTOR program has proven that the same objectives can be accomplished with a much lower price point with the ARM-based computing appliance system than the higher-end Intel-based system.

The VICTORY-in-a-box hardware provides many required VICTORY components and offers rapid adaptation of legacy equipment. Thanks to VECTOR, Program Management Offices will now have evidence that VICTORY-based systems provides a means for simpler acquisition, accelerated developmental phases, and streamlined hardware and software integration efforts.

REFERENCES

[1] Ubuntu Server, https://web.archive.org/web/20130209054025/http://www.ubuntu.com/download/server, 2013.