

One Click Build Environment and Tool Chain for Real Time Safety Critical Aerospace Embedded System

Tejas Chhaya, Dan Evans, David Dempsey, and Jason Powers
MAHLE Powertrain LLC

ABSTRACT

“Embedded Software Build” is the process of taking all source code files that are part of the embedded system and compiling them into build artifacts, such as executables or binaries for the target embedded hardware. In the case of embedded systems, it includes low level software for a specific embedded hardware device, the application software and the associated application data. To further complicate this, low level software also known as firmware or middleware layers are often hand-coded while the application layer is auto-coded. To ensure build reproducibility, accurately predict build results and achieve error-free embedded software builds both in a lab environment and in the field, embedded software builds are required to be automated. At the very initial stage of the project, we realized that the automation of the Embedded Software Build is highly imperative and extremely crucial, as we're dealing with a Real Time - Safety Critical - Aerospace Embedded System. In this paper, we will discuss the different types of software products that need to be integrated together prior to the build, and the cost-effective solution that MAHLE Powertrain has devised to automate the software build process to create a “One Click Build Environment and Tool Chain”.

CITATION: Chhaya, T., Evans, D., Dempsey, D., and Powers, J., "One Click Build Environment and Tool Chain for Real Time Safety Critical Aerospace Embedded System," *SAE Int. J. Aerosp.* 6(2):2013, doi:10.4271/2013-01-2259.

INTRODUCTION

In general, any software build involves gathering the various parts and types of the software to create a program that could be executable for a targeted electronic hardware platform or device. For example there could be a build targeted for Windows, UNIX, or Linux platform and such a build could be for the operating system itself or for the application that runs on it.

In the case of Embedded Software Build, typically it involves Real Time Operating System (RTOS), Low Level Firmware (Middleware) for a specific embedded hardware and finally the application that runs on this target hardware. At the system level, the final application could be functionally critical or not, but certainly needs a specific format in order to tell the hardware to operate in a specific way for given input conditions and physical values.

Depending on if the application is Real Time - Safety Critical or not, or if it's a Prototype versus Production intent, there are various ways to perform the software build. It becomes important to achieve an error-free embedded software build when the final application resides in a “Critical” or “Production” intent area. For the Real Time

Critical application that falls within the Safety Critical category, for each instance of the software build, it is imperative to have a “Build Automation” to eliminate manual errors while building the final product to ensure the highest level of repeatability and predictability. For safety critical real time embedded applications, quality, error-free, and flawless functioning of the embedded software is paramount. This is particularly true when such embedded systems are an integral part of our lives, for example automobiles, general aviation, aerospace, other modes of public transportation, medical, etc. There are unique development challenges that surround designing and building embedded systems, because quite often software is developed independently of the hardware that it will eventually run on. For these reasons, for embedded software build applications, we took into account the following software components while coming up with an Automated “Embedded Software Build”;

- Real Time Operating System (RTOS)
- Firmware - Low level electronic device specific software hand coded in “C” language. This is sometimes also known as middleware
- Application Layer that is developed in

- Mathworks suite of tools [1], and
- VHSIC Hardware Description Language (VHDL)
- Instrumentation software that allows the user to calibrate the application and communicates with the embedded device through application software of the product

With more than 15 years of embedded systems development experience in aerospace and automotive applications, for this aerospace project, the task we had taken on was relatively large and complex, which included design and development of embedded hardware, firmware, and the propulsion control application. In addition, it also required to meet aerospace grade redundancy and safety aspects as well as long term support for the product for future years. Understanding these requirements, during the concept stage of this aerospace project, we had realized that the robust-flexible-automated Embedded Software Build automation was an imperative and extremely crucial component of the final product delivery. Not to mention, that this is extremely important for any Real Time - Safety Critical - Aerospace Embedded System.

SOFTWARE BUILD

In its basic form, the “Build” is the conversion of source code into binaries. In other words it is a process of integration taking all the source code files, which make up an application, and compiling them into build artifacts, such as executables for a targeted embedded hardware. Automating the software build process is a fundamental step in maturing the software development process to ensure the following;

- Accurately predict build results
- Ensure build reproducibility and repeatability
- Eliminate the risk of human error in the build process
- Minimal development down time due to build issues
- Immediate success or failure feedback for developer's new check-ins providing flexibility for the incremental builds
- Embedded sanity checks for system resources
- Ensuring the system's runtime works by ensuring that the system is in a build-able state

It is relatively easy to work with small software projects that can be easily compiled into a small number of executables. When dealing with large and complex software projects and attempting to build the software manually it becomes time consuming, cumbersome, error prone and a daunting activity. In addition to compiling the individual components of an application, it is required to link them together, which is the process of telling the individual compiled components where to find things they require or depend on to run. To further complicate the issue, without a software build process while working in a team environment, the integration of new code and stability of the whole

application becomes unpredictable. Quite often this leads to delaying the integration until a later date which might lead to a major re-engineering effort or architecture changes due to fundamental differences between new and existing code. Finally, a serial approach to code building could lead to improper or partial testing and become a bottle-neck in delivering the final embedded product.

Key Aspects of Build Process

Automated Build, Continuous Integration, and Build Management are three key aspects of any “Software Build Process” with the aim of ensuring software build integrity, repeatability and reproducibility.

Automated builds

Automating the software build process is a fundamental step in maturing the software development process. The most important goal is to ensure build completeness and consistency to ensure the whole system is built with all the selected components. This way the build result is accurately predicted, build reproducibility is ensured and development downtime due to build issues are reduced or eliminated.

Continuous integration

Continuous Integration is the process of building the application on every new source code check-in and is an integral part of the Software Configuration Management, which allows small and frequent releases.

Build management

Build Management incorporates the integration of all the components of a software application into an installable software product. Build number, Build date, Build version, Build machine, Build personnel, Branch numbering, QA testing, Error reporting, Build reports, Build notification, etc. are some examples that are part of Build Management.

The most important goal of the Build Management is neither completeness nor consistency, which shouldn't be misunderstood as unimportant aspects - but rather, the most important goal of Build Management is repeatability. This ensures the ability to recreate any past instance of the software build and reproduce the original output with bugs that can be compared to the new output without the bug to ensure that the said bug is fixed. On the other hand, when the bug is introduced in a new software build, in order to debug it, it helps to go back to the past known output without the bug and rebuild the associated instance for that output instead of rewriting the whole code in order to fix the bug. The understanding of the bug itself is very important, but to return to a known-good and proven output is extremely important to be sure that all the tested components or features are unchanged and one is only touching the part that contains the error.

Thus, Automated Build, Continuous Integration, and Build Management are the key aspects of a build process, when the software development teams are looking for

increased efficiencies and improved quality of the software product.

BUILD PROCESS OVERVIEW

Automated Builds and Integration

A build process such as the one devised by MAHLE Powertrain consists of following various key elements,

1. Proprietary Real Time Operating System
2. Low Level Software that was “hand coded”
3. Application Layer Software, where the application was developed using Mathworks Tools and the output code was automatically generated [2]
4. Proprietary scripts that package software together, invoke Compiler and Linker to make build products

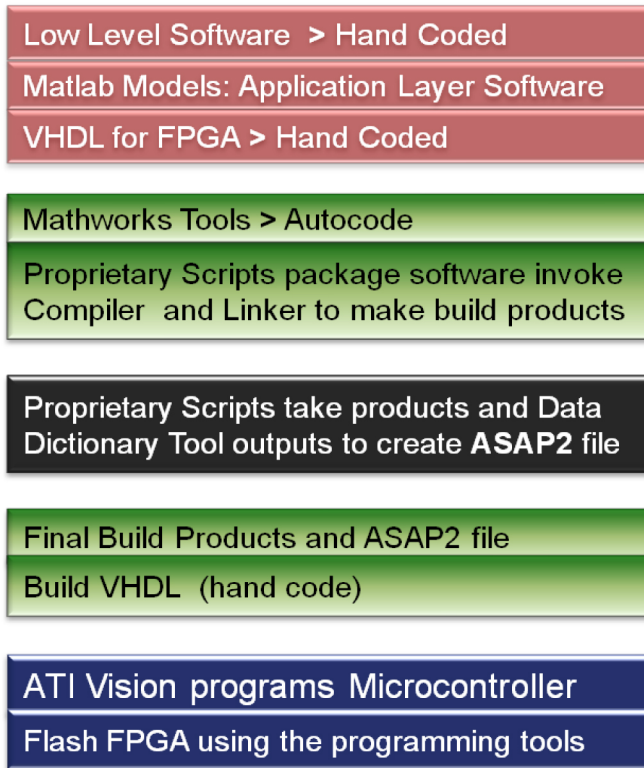


Figure 1. Build Process Blocks

5. ATI Vision inputs build products and ASAP2 file [3]
6. ATI Vision programs Microcontroller through JTAG port
7. Add FPGA and ATI Vision to programming tools
8. Finally, adding an interface for the instrumentation software to calibrate the application

The color coded blocks in Figure 1. Build Process Blocks reflect various key-blocks and their relation to the flow of the

software build within the MAHLE Powertrain's One-Click Build environment and tool chain for real time safety critical embedded systems as shown in Figure 2. Build Process Flow

The build process includes the development of some Simulink Graphical User Interfaces (GUIs) that provided two essential functions. First, a data manager interface was implemented that made use of Simulink Custom Storage Classes (CSCs) to implement a data dictionary that provides a method to define constants including scalar, vector and matrix data objects mainly for use as calibration parameters. These calibration parameters were used during both the simulation and auto coding build process. The defined values are saved in a Matlab M Script file, which are automatically loaded when the Simulink Model is opened. Also, later in the build process these CSC objects are used to automatically generate instrumentation related files such as an ASAP2 file that defines calibration and signal data definitions. Second a schedule manager GUI was developed that provided a means to sequence and schedule particular subsystems in a desired manner while documenting the timing of subsystems.

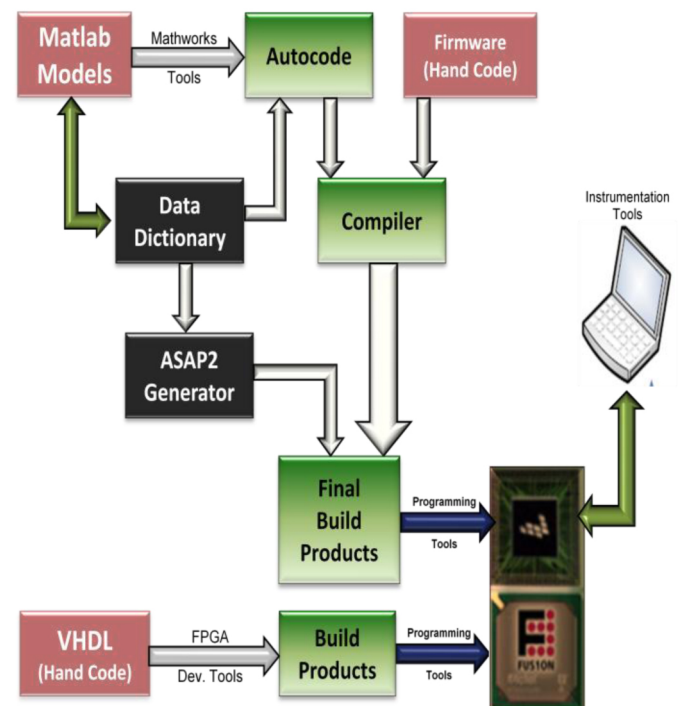


Figure 2. Build Process Flow

Version Control and Software Configuration Management Overview

For software development that involves an elaborate technical concept or requires collaborating with other people within a project team or working on different tasks in parallel, version control becomes extremely important to keep up with changes and reduce errors. It is important to work in unison while working on different tasks without getting in