DEVELOPMENT AND TESTING OF AUTOMATICALLY-GENERATED ACS FLIGHT SOFTWARE FOR THE MAP SPACECRAFT

James R. O'Donnell, Jr., Ph.D., Stephen F. Andrews, David C. McComas, David K. Ward
NASA Goddard Space Flight Center
Greenbelt, Maryland 20771 USA
James.R.ODonnell.1@gsfc.nasa.gov

Abstract

Using integrated analysis and design tools for the development of spacecraft attitude control systems (ACS) can make the process much more efficient, requiring less time and effort than before. The integrated toolset used for the development of the ACS of the Microwave Anisotropy Probe (MAP) spacecraft includes the ability to automatically generate software from the MAP high-fidelity (HiFi) simulation. By using this automatically-generated code to provide portions of the MAP ACS flight software, that part of the development effort also became more efficient. However, because components of the HiFi simulation were being used to generate flight software, special consideration needed to be given to these components during all aspects of the ACS analysis, design, and testing cycle. An additional benefit of the integrated analysis and design toolset used for MAP is that it allowed the opposite to be done; actual flight software could be run in the HiFi simulation environment, increasing the level of testing possible.

Key Words: attitude control systems, flight software testing, simulation

Introduction

By integrating the attitude determination and control system (ACS) analysis and design, flight software development, and flight software testing processes, it is possible to improve the overall spacecraft development cycle, as well as allow for more thorough software testing. One of the ways to achieve this integration is to use code-generation tools to automatically generate components of the ACS flight software (FSW) directly from a high-fidelity (HiFi) simulation. In the development of the Microwave Anisotropy Probe (MAP) spacecraft (see Figure 1), currently underway at NASA's Goddard Space Flight Center, approximately 1/3 of the ACS flight software was automatically generated. In this paper, we will examine each phase of the ACS subsystem and flight software design life cycle: analysis, design, and testing.

In the analysis phase, we scoped how much software would be automatically generated and we created the initial interface. The design phase included parallel development of the HiFi simulation and the hand-coded flight software components. Everything came together in the test phase, in which the flight software was tested, using results from the HiFi simulation as one of the bases of comparison for testing. Maintaining good configuration control was an issue for the HiFi simulation and the flight software, and a way to track the two systems was devised. Finally, an integrated test approach was devised to support flight software testing at both the unit- and build-test levels that used the HiFi simulation to generate data for performance verification.

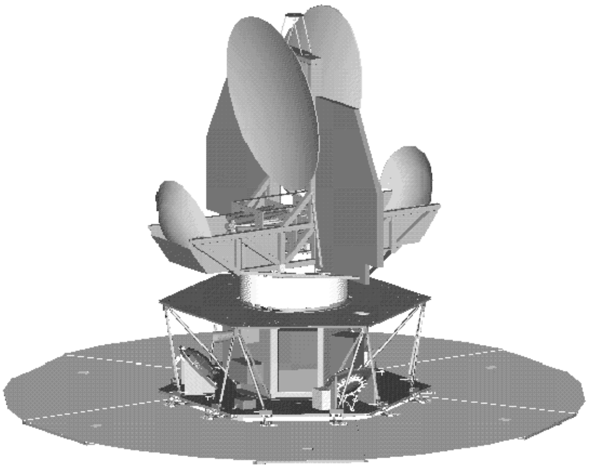


Figure 1: The MAP Spacecraft

Another benefit of the simulation and code-generation application used on the MAP project is that it supported bringing flight software and test data into the HiFi simulation environment. This capability was used to incorporate the flight software Kalman filter into the HiFi simulation and also to import flight software test data for comparison and performance verification.

We will conclude our discussion with a summary of the lessons learned thus far using automatically-generated code for the MAP project.

Analysis Phase

In the early stages of the MAP project, it was necessary to make the initial decisions regarding the use of automatically-generated flight software, including whether or not to use it, which code-generating tool to use, and how much of the system's flight software was to be generated automatically. These decisions were a part of the initial system engineering design of the spacecraft and the project.

System Engineering

The decision to use automatically-generated code using the AutoCode module of ISI's MatrixX integrated toolset was made in an attempt to address some of the lessons learned from previous in-house spacecraft developments at Goddard. Following the development of the XTE and TRMM spacecraft, a need was seen to limit the manual interfaces required to design and develop an ACS subsystem. The design system used for those spacecraft was characterized by a large duplication of effort, with three separate teams—the analysts, the flight software developers, and the developers of the hybrid dynamic simulator (HDS) used to test the flight software—designing the same system independently.

This method relied on written documentation to describe any changes in the control algorithms that must be reflected in all three systems. One person was dedicated to the development of the HiFi simulation, which was not useful in the linear analysis of the system, and another person was a dedicated "documentation engineer", needed to keep the flight software and test simulation teams informed of changes. This system was prone to manual implementation errors and misunderstandings, which resulted in the FSW team not always initially implementing the algorithms as the design team had envisioned them.

By the preliminary design stage for the Medium Explorers (MIDEX) program, of which MAP is the second mission, tools existed that would make design, analysis and development an integrated process, allowing a reduction in manpower and a reduction in development time, consistent with the philosophy of the MIDEX program. There was also an interest in reusing software and developing reusable model/software libraries for quicker mission designs in the future. The integrated analysis and design toolset selected, MatrixX from Integrated Systems, Inc., was selected because it possessed the desired capabilities. The MatrixX components used for MAP include a linear analysis tool (XMath), a graphical environment for developing and executing nonlinear simulations (SystemBuild), an automatic code generation product (AutoCode), and a documentation generation product (DocumentIt).[1]

Scoping

For maximum gains in the efficiency of the design process, it is desirable to do as much as possible using the MatrixX integrated toolset. In theory, this would include designing, analyzing, simulating, performing code generation, and documenting the entire ACS subsystem. However, since this was the first time that these techniques and this product was used on a flight project at Goddard, it was decided to limit the scope of the portions of the system that would be "AutoCoded" in order to minimize risk.

It was decided early in the process not to AutoCode portions of the HiFi simulation to be used both in the flight software and in the HDS, for fear of an error going undetected by being replicated in each. It was decided to use AutoCode for the control law algorithms and system momentum calculations only. This limited risk by not automatically generating code for any flight software component that required a direct interface to ground commands or to the spacecraft sensors and actuators. Further, because the control laws had a high algorithm-to-code ratio and a clearly defined interface to the rest of the system, they provided a good test of the code-generation method. A final benefit to using AutoCode primarily on the spacecraft control laws is that the controllers are good candidates for reuse on future missions.

It is interesting to note that, in the early, analysis phase of the MAP spacecraft development, one component of the ACS subsystem—the Kalman filter used for onboard attitude determination—was identified as a good candidate for reuse and also "going the other way". Flight software implementing a Kalman filter existed and had been tested and flown on other spacecraft, so it could be reused for MAP. Because MatrixX's SystemBuild simulation component supports the use of existing software within its simulations, it would be possible to include the flight Kalman filter inside the HiFi. As will be discussed later in this paper, the ability to move flight into the HiFi simulation, enables it to be more completely and thoroughly tested.

Initial Interface Design

During the analysis phase, it was necessary for the analysis and flight software teams to begin discussing the interface design. It was necessary to design the HiFi such that one or more of its pieces would interface in a compatible way with both the other parts of the HiFi and the flight software. At this point in the project, it was necessary to consider two things:

Input/Output Interface: In SystemBuild, the inputs and outputs to a given portion of the simulation determine what information is available each cycle, as well as what information that block may provide. To support the flight software as well as the HiFi simulation, this list of inputs and outputs might need to be augmented with other points of interest.

Simulation Parameters: As opposed to inputs and outputs, which normally can change each control cycle, simulation parameters are quantities such as control gains of spacecraft mass properties. While it is desirable to be able to change these parameters both within the simulation and during flight, they do not change very often. It was decided that SystemBuild's %VAR capability, which allows a variable to be assigned as a parameter for an element of the simulation, would provide a way of implementing these parameters in the HiFi and would map into flight software tables, which provide the same function on-orbit.

Design Phase

Previously, the early design phase of a spacecraft ACS, particularly could done without worrying about too many outside considerations. For example, while the linear analysis and low- and high-fidelity simulations are being developed, they can normally be focused exclusively on satisfying the needs of the design process and the analysts. In the development of the MAP ACS, however, because of the use of AutoCode and DocumentIt, it was necessary to design the system from the start with these considerations in mind. This was particularly true because the SystemBuild HiFi simulation forms the cornerstone for everything else.

SystemBuild is a graphical block diagram editor used to construct the desired simulation. Its two basic element types are "blocks", which represent the functional elements of a simulation, and "SuperBlocks", which can be used to group other blocks and/or other SuperBlocks into hierarchies. SystemBuild comes with a wide assortment of blocks that can implement linear and nonlinear systems, continuous and discrete systems, as well as a variety of user-definable blocks that can be used to include arbitrary functionality into a simulation.

High Fidelity (HiFi) Simulation Development

Figure 2 shows a drawing of the top-level SystemBuild SuperBlock of the MAP simulation, which gives an idea of the pieces of the MAP system that are being modeled, as well as how the simulation is organized. At the top level, everything has been organized into three SuperBlocks, named "ACS", "ACE", and "Models". These SuperBlocks are used for the following simulation elements:

ACS: The ACS SuperBlock contains elements of the simulation that correspond to the aspects of the MAP flight software contained in the main processor, including those elements from which flight software will be automatically generated (see the Code Generation section, later in this paper). Other than FSW, the ACS SuperBlock also models some of the ground-based commanding.

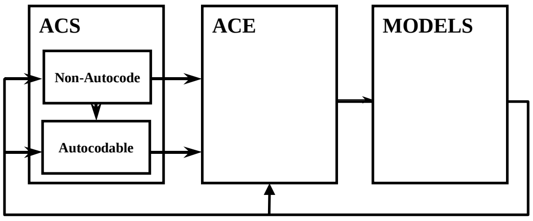


Figure 2: MAP High Fidelity Simulation

ACE: The ACE SuperBlock models the needed elements of the MAP Attitude Control Electronics (ACE). In the MAP spacecraft, the ACE is used to implement the independent Safehold Mode and to provide the interface to most of MAP's sensors and actuators. Because the MAP simulation uses engineering units and does not go to the level of counts and voltages, it was not necessary to model those interfaces in the ACE SuperBlock. Instead, the two functions that are modeled are the independent Safehold and the interface that takes the thruster commands from the ACS SuperBlock and converts them into an appropriately sized pulse width.

Models: In the Models SuperBlock, the actual physics of the MAP spacecraft and environment are modeled. This includes the models of the spacecraft attitude, position, and velocity, environmental disturbance models, and models of MAP's sensors and actuators.

Flight Software Interface Considerations

HiFi Simulation: Figure 3 shows the ACS SuperBlock, divided into the portion to being AutoCoded (the Autocodable SuperBlock) and the portion that wasn't (the Non-AutoCode SuperBlock). The inputs and outputs of the Autocodable SuperBlock provide the main interface between the AutoCode and non-AutoCode portions of the MAP ACS flight software. Very early in the design of the HiFi, this interface was clearly defined between the developers of the HiFi and the flight software.

SystemBuild uses %VARs in order to parameterize aspects of a simulation. These are parameters that can be included within SystemBuild blocks that are tied to XMath variables, allowing them to be set and changed when a simulation is begun. In the MAP ACS flight software, parameter tables are used to fulfill the same function. The initial stages of the design set the standard of using the %VARs in the Autocodable SuperBlock as a single flight software table, which allowed them to be consistently set and changed in either the HiFi or the flight software.

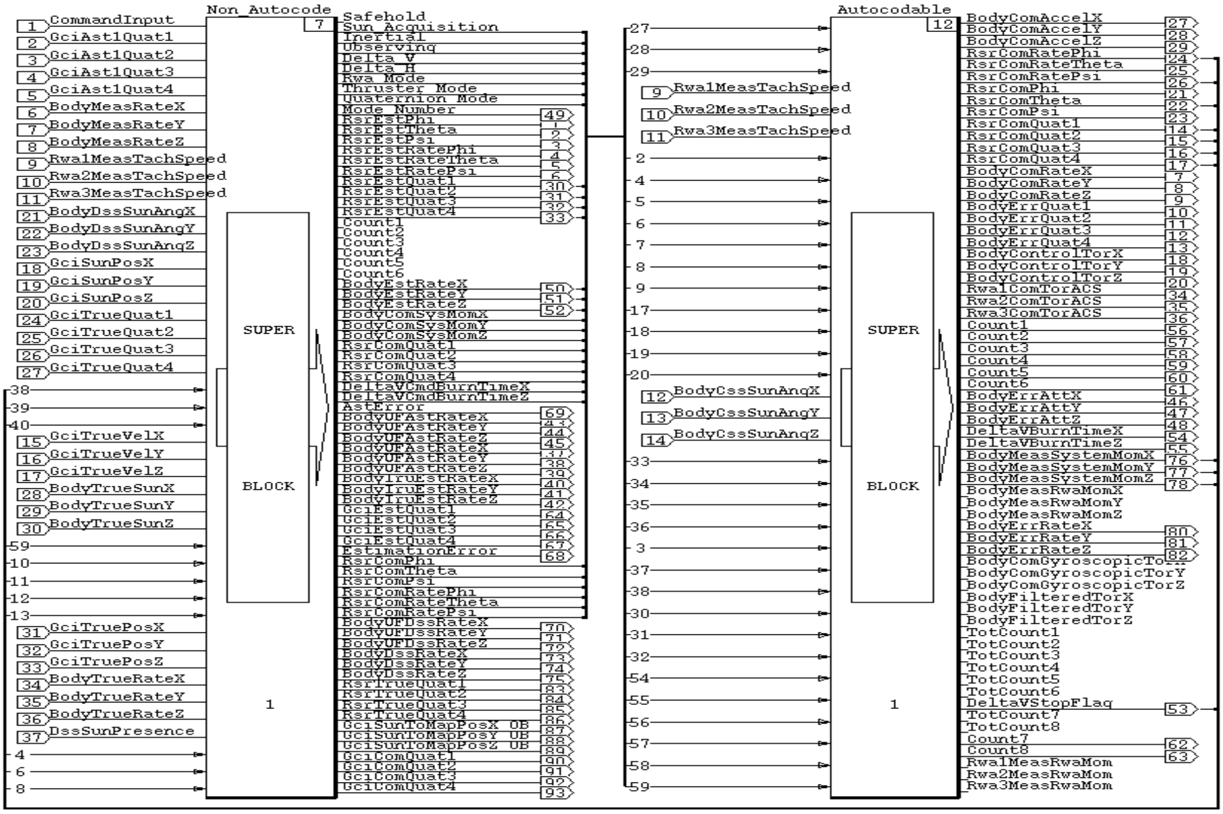


Figure 3: MAP HiFi ACS SuperBlock Showing AutoCode Section

Naming Convention: A small but important aspect of developing the HiFi such that its AutoCoded portions would be compatible with the flight software involved the names of the inputs, outputs, and parameters. The naming convention, determined in cooperation with the flight software developers, made the task of interfacing the AutoCode flight software more straightforward.

Exception Protection: The final consideration that needed to be applied to those parts of the HiFi to be AutoCoded was including protection from mathematical. Normally, exceptions are not a concern in a simulation; either the simulation tool will handle them or they will cause an error and exit. However, in flight such errors are unacceptable.

Having identified the potential problem, though, the solution was easily implemented. In addition to protecting from divide by zeros by checking the divisor before performing the division, the HiFi AutoCode blocks also implement common flight software practices such as checking in a small delta range about a value rather than checking for exact equality.

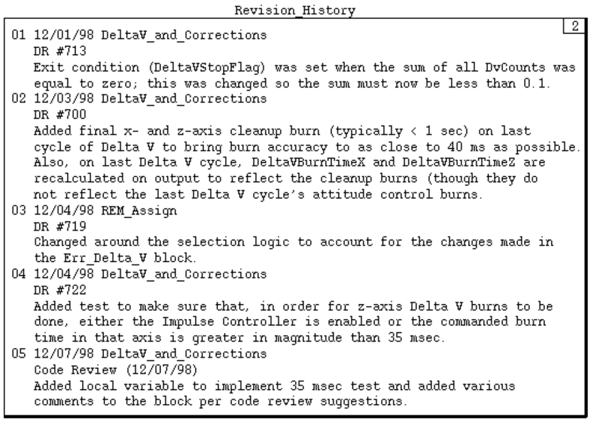
Configuration Control

When developing flight software, the issue of configuration management is very important. Because the success of the mission and the safety of the spacecraft depends on the quality of its software, it is important to be able to track all changes. Because of the use of AutoCode to generate some of MAP's flight software, the configuration control issue now applies to parts of the HiFi simulation.

When development of the MAP HiFi simulation began, there was not a release version of a configuration management tool that could be used along with SystemBuild. Also, because of the way that SystemBuild saves its simulations, it was not a simple task to develop a configuration management plan. The plan that was devised and used included the following points:

Baseline Simulations: A series of baseline simulations were developed that exercised all of the different modes and operating conditions of the MAP HiFi. Then, whenever a new version of the HiFi was set to be released, these baseline simulations were rerun to ensure that the only changes that occurred in the performance of the simulation were those expected based on the changes made in the HiFi itself.

Revision History: As blocks were changed in the HiFi, a list of the changes was maintained. Initially, this list was kept in a separate text file. After discussions with members of the Applied Physics Laboratory (APL)[2] who were using automatic code generation tools from The Mathworks' Matlab family of products, the revision history was kept within the simulation itself, using SystemBuild's Text block (see Figure 4).



```
                          Revision_History
01 12/01/98 DeltaV_and_Corrections                                        2
   DR #713
   Exit condition (DeltaVStopFlag) was set when the sum of all DvCounts was
   equal to zero; this was changed so the sum must now be less than 0.1.
02 12/03/98 DeltaV_and_Corrections
   DR #700
   Added final x- and z-axis cleanup burn (typically < 1 sec) on last
   cycle of Delta V to bring burn accuracy to as close to 40 ms as possible.
   Also, on last Delta V cycle, DeltaVBurnTimeX and DeltaVBurnTimeZ are
   recalculated on output to reflect the cleanup burns (though they do
   not reflect the last Delta V cycle's attitude control burns.
03 12/04/98 REM_Assign
   DR #719
   Changed around the selection logic to account for the changes made in
   the Err_Delta_V block.
04 12/04/98 DeltaV_and_Corrections
   DR #722
   Added test to make sure that, in order for z-axis Delta V burns to be
   done, either the Impulse Controller is enabled or the commanded burn
   time in that axis is greater in magnitude than 35 msec.
05 12/07/98 DeltaV_and_Corrections
   Code Review (12/07/98)
   Added local variable to implement 35 msec test and added various
   comments to the block per code review suggestions.
```

Figure 4: HiFi Revision History Text Block

By keeping a revision history within each SystemBuild SuperBlock, it was easy to tell at a glance what had changed and when. Also, using "SystemBuild Access" tools, it was possible to extract the revision histories from throughout the HiFi and assemble a report covering the changes through the simulation.

Parameter Database: One final technique that was needed to control the configuration of the HiFi simulation, the flight software, and the HDS, was a way to make sure that each of these systems was configured in the same way. In previous projects, parameter changes—such as spacecraft properties or controller gains—would be changed in each system manually, and propagating these changes to each system was a time-consuming and inefficient task.

For the MAP project, a flight software parameter database was developed that was used to populate each MAP system with the parameters that it needed to run. Where these parameters were the same for different systems, the values in the database were linked so that a change in one would automatically be reflected in the others. Reports for each system (HiFi or HDS initialization scripts, flight software initialization files) could be generated from this common database.

Testing Phase

Flight software testing at Goddard goes through a number of stages. The first level of testing, unit testing, is usually done by the flight software developer, and is a low-level series of tests used to show that each software component or module works. After unit testing, build testing and then acceptance testing is done to verify that all of the software components work correctly together and provide the functionality that will allow the spacecraft to meet its requirements.

Using automatically-generated code as part of the flight software for MAP introduced some additional requirements and opportunities for the testing phase of the project.

Unit Testing

Ideally automatic code generation would make the time to translate the algorithms into flight code negligible. This can only be achieved if the translation process were guaranteed to be perfect. Since this is not the case, unit testing must be performed. The MAP team employed two phases of unit testing. The first phase treated the automatic code as a black box. Nominal inputs were fed into the automatic code and the outputs were verified. The goals of black box testing were to verify that the automatic code was properly integrated into the manual code and to verify the automatic code's nominal functional behavior. The second phase of unit testing, known as white box testing, "opened" up the automatic code to isolate and test individual code paths. The goals of the white box testing were to test all code paths and to test boundary conditions.

Since white box testing is laborious and since the automatic code was being inspected, we didn't introduce white box unit testing until after several releases had been made. This strategy allowed the algorithm-to-flight code translation process to be as efficient as possible when the number of changes were more frequent without sacrificing the quality of our unit testing on the final product. The black box testing environment was established during the first software build and was rerun for every build. Since this effort was nearly automated it didn't slow down the flight software build process.

Figure 5 shows the black box unit test environment. The HiFi outputs simulation results, simulation inputs, and the automatic code to the unit test platform. The unit test driver is linked with the FSW controller classes and the automatic code. Five HiFi test cases, one for each operational mode, were used as the test suite. This data was run through the FSW and the FSW results were compared to the HiFi results. This testing has consistently shown that the automatically generated code accurately represents the SystemBuild design.

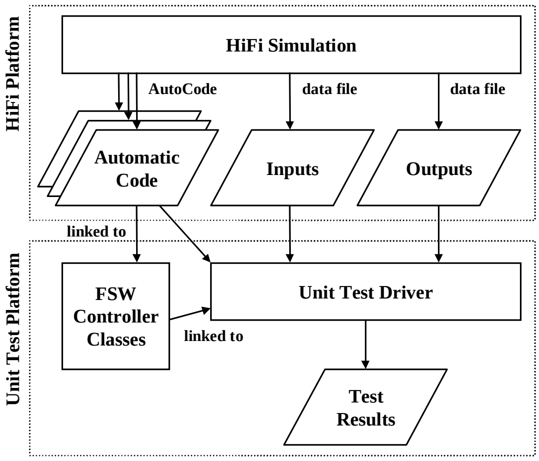


Figure 5: Test Procedure for AutoCoded Software

The white box testing consisted of a unique test driver being written for each section of the automatic code being tested. This testing proved to be very important and uncovered several problems that were then able to be corrected. The unit test driver calls the hand coded FSW controller functions and not the automatically generated code. Driving the test from this level verifies that the manual code and the automatic code are properly integrated. Since XMath scripts managed and documented the HiFi test cases, the FSW developers didn't require much of the analyst's to get necessary information. Defining common controller interfaces in both the HiFi and the FSW and having a standardized data file format were the drivers that empowered the unit tests. These activities could occur with or without automatic code generation, but were used in this case because of it.[3]

Build and Acceptance Testing

Once build testing was begun, there were fewer special considerations needed because of the use of AutoCode. However, there were considerable benefits from the use of the MatrixX integrated toolset that greatly enhanced the efficiency of the build testing effort.

In previous projects, data from flight software testing was generally used to verify the correct performance of the spacecraft ACS by plotting the data and then running a comparable HiFi simulation for comparison. For the MAP project, the general outline of the process remained the same, but the integrated nature of the toolset improved the process considerably. In the past, the people performing the flight software tests were a different group than the analysts verifying the test performance. Typically, one of the analysts was responsible for running most or all of the HiFi simulations, which created a potential bottleneck for testing. Finally, comparing the test and HiFi simulation data was often a matter of looking at two sets of plots.

Telemetry Import and Manipulation

By using various parts of the MatrixX integrated toolset, along with some tools based on the underlying Unix operation system, it was possible to automate many of the above steps for flight test verification. The first step of this was to import the telemetry from flight tests into the XMath environment. This was done by writing an XMath command that in turn automatically wrote and executed a Unix shell script to reformat the data to make it compatible with XMath. Then the XMath command imported the data and saved it in its own format.

Once the flight test data was available within the XMath environment, it became possible to automatically tie in the other MatrixX components. XMath tools were written to produce a number of standard plots. Also, it was possible to analyze the test data to learn about how the corresponding test was run. This, in turn, made it much easier to set up HiFi simulations for comparison, as discussed in the next section.

Automatic Simulation Generation

Once the flight test data was in a format compatible to XMath, it was possible to write XMath commands to analyze flight test data; looking for such things as mode transitions, commanded attitude changes, initial position, velocity, time, and attitude. In order to allow more than just the primary developer to run HiFi simulations, the HiFi had originally been set up to run based on a standard set of XMath script files. Several of these files set up the nominal state of the system, and were produced from the flight parameter database mentioned earlier, and were common to all HiFi simulations. The main script file was set up specifically for each simulation, and included all off-nominal parameter values and other simulation-specific initial conditions and system changes and events.

For flight test verification, it became possible to produce HiFi simulation script files *automatically*, based on values within the test data itself. This meant that just about anyone could run HiFi simulations, and that flight software testers could run their own comparisons (though flight software performance verification still required analyst review). Further, plots could be produced that put flight test and HiFi simulation data on the same plot, thus making performance verification much easier (see Figure 6).

Flight Software Testing Using the HiFi Simulation

One additional benefit of using the MatrixX package other than automatic code generation, though related to it, was SystemBuild's ability to include user-written C code into its simulation though what it calls a User Code Block (UCB). This capability was used on the MAP project to run the actual flight onboard Kalman filter within the HiFi simulation. Running the flight filter in the HiFi allowed the existing filter flight code to be tested before it made it into the actual flight software (see Figure 7). Also, because of the greater degree of control possible with the HiFi versus the HDS testing environment, it allowed the flight filter to be tested and stressed in ways that could not be done in the testing lab. Because the HiFi runs faster than real time, it also allowed more testing to be done.

There were other benefits to being able to use and test the flight Kalman filter within the simulation environment. By conducting tests to cover failure scenarios, it is possible to generate backup values for the filter to be used in the event that the failure occurs in-flight. Also, because the actual code is being run in the simulation, it makes performing comparisons between flight software tests and HiFi simulations easier. Finally, an incidental benefit of using the flight software Kalman filter in the HiFi as a UCB is that it sped up the simulation.

Summary of Lessons Learned

The ACS for the MAP spacecraft has been under development for over three years, with just over a year until the scheduled launch. In that time, a number of lessons have been learned concerning the use of automatic code generation techniques.

AutoCode Scope: Using a relatively small scope for automatically generated code for this first mission was a good idea; there is certainly more that could have been done, but it will be much easier to do more on the next mission with what we have learned on the MAP spacecraft.

AutoCoding HDS: Even though the original design philosophy considered AutoCoding the HDS software as well as the flight hardware, it was decided that the HDS models be developed independently, to reduce the possibility of an error slipping through the cracks because it was replicated in both the flight software and in the test environment.

Configuration Management and Maintainability: Because MatrixX is a commercial product which has seen at least one major upgrade during the lifetime of the MAP project, it is currently an open issue of how to deal with upgrades since AutoCode output may vary from version to version. This affects maintainability, and may require reused software to be completely retested from mission to mission.

Learning Curve: Because of the learning curve needed the first time this toolset were used, using AutoCode and MatrixX may not have saved as much development time for MAP. However, it has already been seen at Goddard that, using MatrixX and the MAP HiFi as a base, it is possible to develop an initial HiFi for a new project in a fraction of the time necessary in the "FORTRAN HiFi" days. It is on future projects that significant gains can be made using these tools.

XMath Environment: XMath proved to be very useful, as it was possible to import flight test data and automatically create HiFi comparison simulations, run simulations, and plot test and verification data on the same plot. This allowed all of the members of the testing team to perform their own test verifications.

Development Process: One of the more dramatic changes in the ACS FSW development process is that many people are performing multiple roles. The analysts have participated in every activity, performing analysis, writing requirements, reviewing code, writing test procedures, and verifying test performance. Developers helped to write the requirements, reviewed the AutoCode, developed code, and supported the performance verification. The tools brought the team closer together but also allowed them to work more independently and efficiently. Many participants have worked as system engineers with a specialty.

References

[1]Ward, D. K., et. al., "Use of the MatrixX Integrated Toolkit on the Microwave Anisotropy Probe Attitude Control System", 21st AAS Guidance and Control Conference, Breckenridge, CO, 1999.

[2]Salada, M. A., and Dellinger, W., "Using MathWorks' Simulink® and Real-Time Workshop® Code Generator to Produce Attitude Control Test and Flight Code", 12th AIAA/USU Conference on Small Satellites, 1998.

[3]McComas, D. C., et. al., "Using Automatic Code Generation in the Attitude Control Flight Software Engineering Process", 23rd Software Engineering Workshop, Goddard Space Flight Center, 1998.

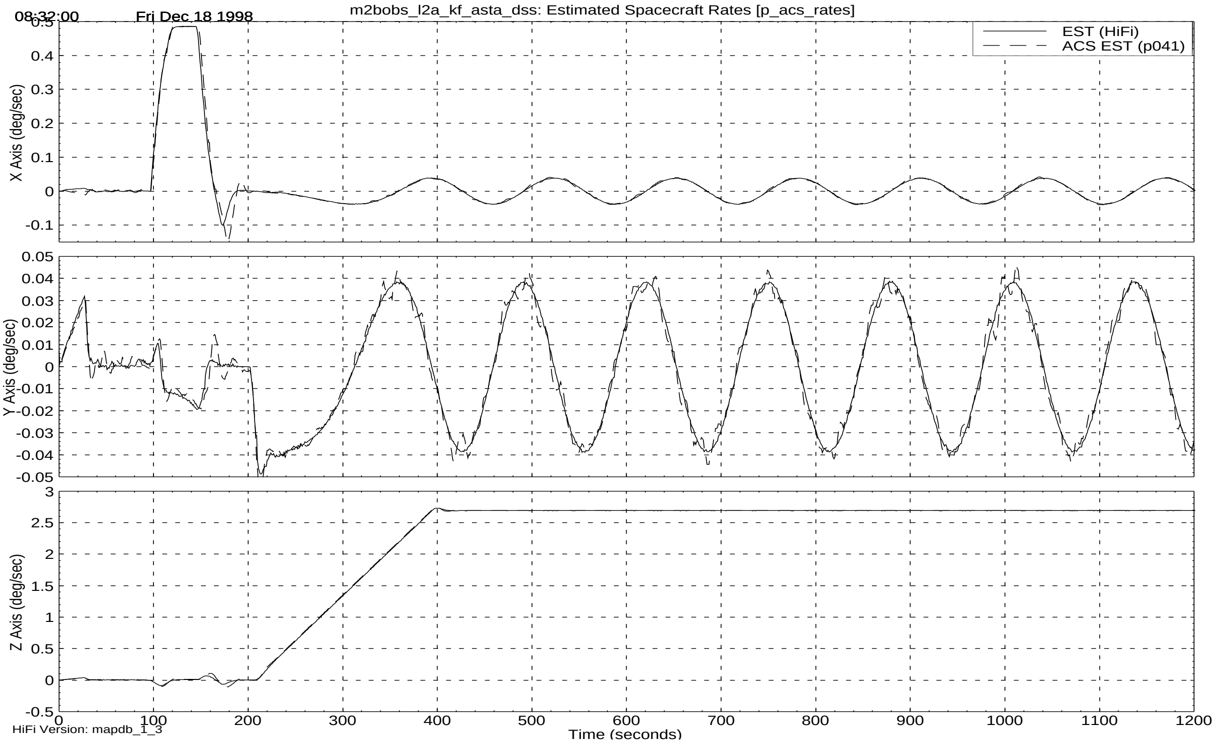


Figure 6: MAP HiFi vs Flight Software Test Data for Science Mode Simulation

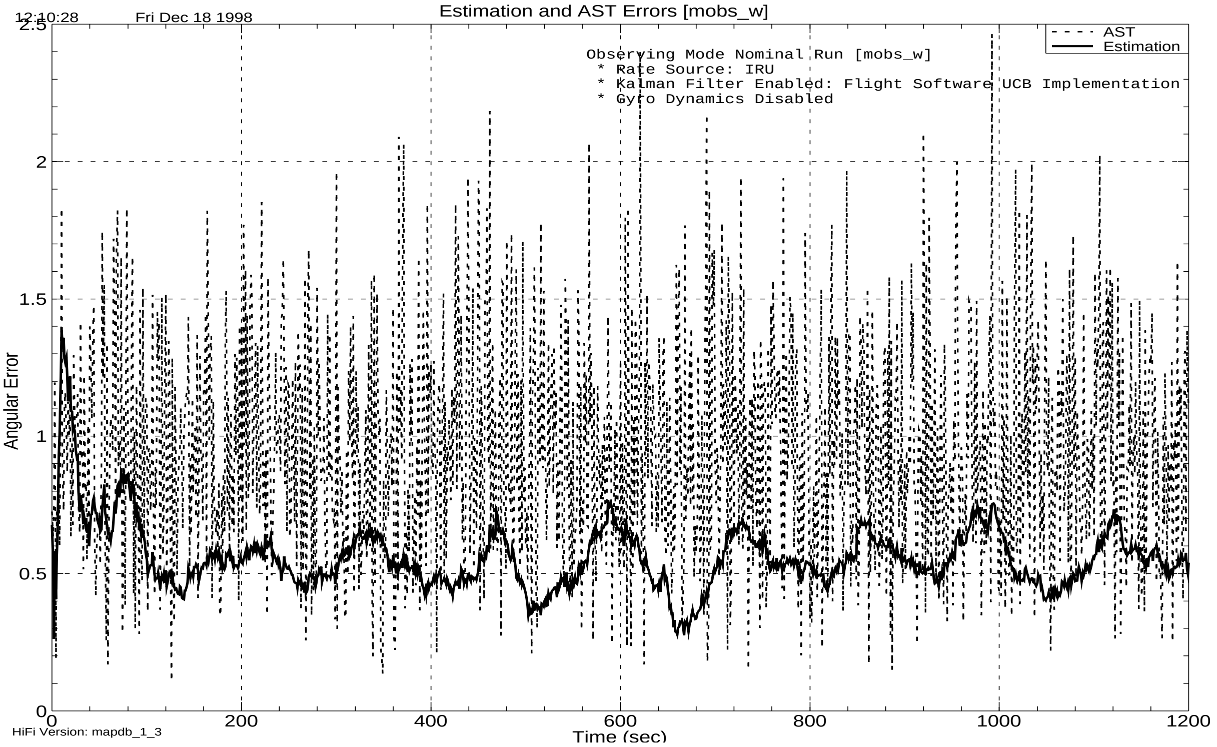


Figure 7: MAP Flight Kalman Filter Estimation Error from HiFi Simulation