Proceedings of the 1985 Winter Simulation Conference
D. Gantz, G. Blais, S. Solomon (eds.)

A SIMULATION OF AN IMBEDDED SOFTWARE SYSTEM FOR GLOBAL POSITIONING SYSTEM NAVIGATION

Jon Vavrus
Computing Applications & Software Technology
5450 Katella Ave., Suite 103
Los Alamitos, California 90720

ABSTRACT

A set of software was developed to allow the execution of a set of real time navigation code on a host computer. This program (STM) allowed the debugging and testing of the real time software to be done in an environment containing a large variety of software tools. The program simulated Global Positioning System (radio ranging), and Inertial Measurement System inputs for the target software. It was found that the simulator performed adequately and enhanced our ability to deliver the working real time code.

INTRODUCTION

A program was devised to provide a simulation of an embedded system environment. Running in this environment was a set of software designed to do Global Positioning System (GPS) navigation in conjunction with inputs from an Inertial Navigation System (INS). The environment consists of a multi-tasking, real-time system on a National Semi-conductor 32016 microprocessor based system. Whereas the simulator runs on a standard mini- or main-frame computer.

The software running in this environment performs the navigation function for a moving vehicle (plane, boat, truck, etc.). It does this by using input from both GPS satellites and from an onboard INS unit. In order to service the various input data flows and output data needs, the navigation software was designed to consist of several tasks running at different rates. These task invocation rates range from 1 hertz for the input of GPS data to 100 hertz for the output of certain forms of receiver loop aiding information.

Thus the simulator had to be capable of running different tasks at different rates. Plus it had to have the capability of providing these tasks with several different types of data (GPS, INS, altimeter, etc.) at several different data rates.

BACKGROUND

CAST was contracted to develop navigation software for use in an integrated GPS/INS environment. This software would run in a multi-tasking real-time environment in hardware built specifically for the application.

At an early stage we realized that the task of testing and integrating this software would present serious difficulties. The "usual" approach would have been to integrate the major program components using special purpose drivers written by the navigation program software developers themselves. Some limited testing, on a host computer, of the fully integrated program would then be done, this would be followed by integration on the target hardware system.

There are several problems with such an approach. In general the testing tends not to be as thorough as it should be, especially during the stage at which the fully integrated program is tested. This lack of testing stems from two primary causes. The first is that the same person often writes both the test software and the software to be tested. This situation can lead to matching bugs in both programs. The second cause is the sheer difficulty involved in adequately simulating a diverse set of consistent inputs for the integrated program.

In view of this it was decided to initiate a separate project aimed at writing a simulator of the navigation program's environment. This simulation needed to create varied sets of consistent, accurate inputs for the navigation program as well as trigger the navigation program's various tasks at the appropriate rates. It was also necessary to provide for the comparison of the outputs from the navigation program with a set of values representing what they were supposed to be (a truth model) and to output the results of the comparison (see Figure 1).

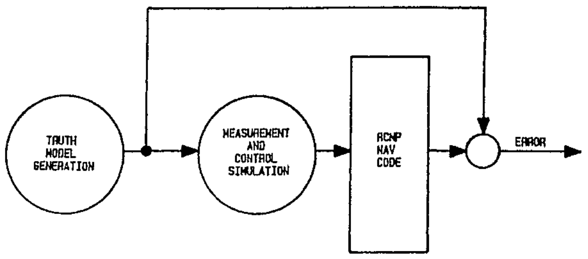

Figure 1: STM Functional Overview

OVERALL PROGRAM DESCRIPTION

The "Software Test Module" (STM) was designed to meet these criteria. Initial testing and integration of the navigation program was going to take place on an Amdahl V6 computer running U.T.S. Final testing would be done in a National Semi-Conductor microprocessor on a commercially available board attached to a VAX-11 processor. This implied that coding would have to be done in a portable manner. Thus both the navigation program and STM were written in standard FORTRAN-77.

For a variety of reasons STM was designed as a pair of stand-alone programs. This allowed all the CPU intensive tasks to be placed in one program and the actual navigation code drivers placed in the other (see Figure 2). The medium of communication between the two programs consists of a set of files containing simulated input data for the navigation software along with information which can be used as the truth model for comparison with the navigation program outputs.

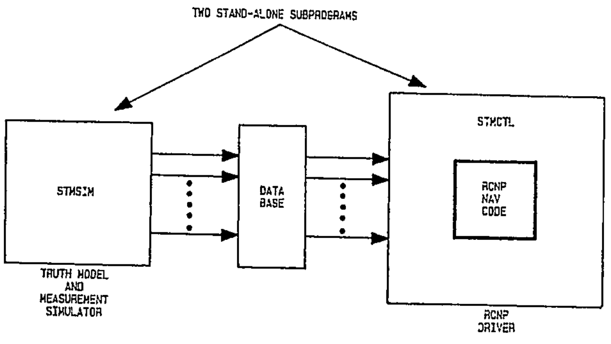
TWO STAND-ALONE SUBPROGRAMS
STMSIM
DATA BASE
STMCTL
RCNP NAV CODE
TRUTH MODEL AND MEASUREMENT SIMULATOR
RCNP DRIVER

Figure 2: STM as Two Stand-Alone Programs

This separation of function allows repeated runs of the navigation code using identical input data with only minimal (simulation) code and CPU overhead. This allows the turnover from one debugging (of the navigation program) run to another to be relatively short, thus greatly increasing the usability of the program.

A further benefit of the two program design is the increase in modularity that it provides. If different data simulation models are needed they can be implemented without affecting the code that drives the navigation program. Similiarly if the navigation program interfaces are modified the changes can be implemented without affecting the simulation models.

Due to scheduling constraints, an important benefit from our point of view was the ability to implement both programs in parallel. This allowed STM to be done much sooner than it otherwise would have been.

Finally, a dual program implementation had some direct impact on the manner in which the program was to be used in the VAX environment. It allowed us to actually load only the driver program into the National Semiconductor board, the data simulation being done in the VAX itself. This minimized any effects that might be caused by the presence of the simulator's code.

The measurement simulation half of the program (STMSIM) is a relatively standard analytical program. This program handles the creation of the simulated input data and the computation of the truth models to which the navigation output is to be compared.

The task driver half of the program (STMCTL) handles the driving of the navigation software itself. It must handle formatting of the input data into the correct form (the form which is expected by the navigation code), along with the capturing of the output from the navigation program tasks after they have been run. It also provides for the comparison of the output from the navigation program to the truth model generated by STMSIM.

INPUT SIMULATOR PROGRAM DESCRIPTION

The stand-alone input data simulation and truth model calculation program (STMSIM) is designed to take input from a set of text files, and produce a set of files containing all the information needed by the navigation software driver program (STMCTL). A decision was made that input should only come from files and not directly from the user. This decision was based upon the desire to be able to file away entire test cases for later use. Such test cases would be run to check out the navigation software after any modifications have been made.

The input information needed from the operator can be grouped into three different areas: control information, scenario design, and mission profile (see Figure 3). The control information consists of: the test mode in which to run (which is used to determine the set of data to actually be produced) and information which determines the rate at which data should be simulated. The scenario design information consists of such things as: satellite orbits, vehicle characteristics, atmospheric models, etc. The mission profile consists of a set of "course leg" parameters which specify the maneuvers through which the movement of the user vehicle is to be simulated.

FILE	FUNCTION
CTL.DAT	1. PROGRAM CONTROLS 2. INITIAL CONDITIONS 3. SENSOR CHARACTERISTICS 4. SYSTEM PARAMETERS
PROF.DAT	MISSION PROFILE
SAT.DAT	SATELLITE ALMANAC/EPHEMERIS AND IONO PARAMETERS
ANT. DAT	ANTENNA SILHOUETTE
OUTG.DAT	GPS OUTAGE PROFILE

Figure 3: STM Simulator Input Files

The STMSIM program is designed as 7 separate modules, (see Figure 4) each of which is only loosely coupled to the others. The functions of each of the modules are described below:

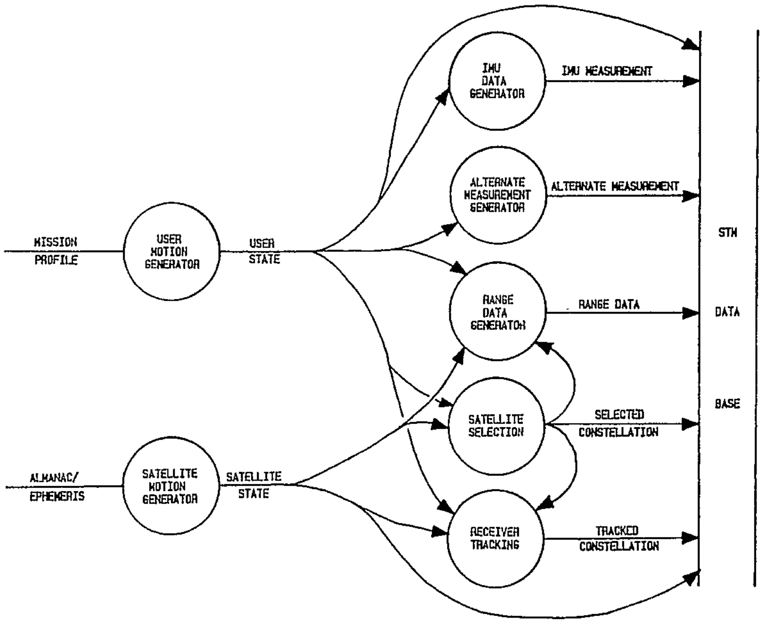

Figure 4: STMSIM Functional Breakdown

Executive: This is the overall controller for the STMSIM program.

User Motion Generator: This module is responsible for generating the true vehicle position, velocity, acceleration, attitude, attitude rate, and clock bias. This user state is then output as part of the truth model. Since the simulation is designed to test the navigation program and not as an exact simulation of the real world, a simplified model of user motion was employed.

This model consists of simulating the motion of the vehicle as a series of "course legs". During each of these course legs the vehicle will experience a combination of: constant acceleration, constant angular rates, circular turns, and climb/dive maneuvers. This motion is specified by the user through a series of vehicle states (pitch, heading, roll, altitude, speed) and times. The simulator then uses a combination of the above maneuvers to enable the vehicle to smoothly change from its current state to the specified state at the specified time.

IMU (Inertial Measurement Unit) Data Generator: This module uses the true user vehicle acceleration and angular rate (as supplied by the user motion generator) to produce simulated IMU output. When calculating the IMU output various user specified error sources are included (eg. gyroscope and accelerometer biases). The resulting simulated IMU output is used as input to the navigation program.

Alternate Measurement Generator: This module is responsible for simulating output from any non-GPS/ non-INS sensors which may be on the user vehicle (eg. altimeters). This module is necessary since the navigation software has the capability of utilizing such sensor information as input.

Satellite Motion Generator: This module is responsible for generating the true positions, velocities, accelerations, and clock biases of all the GPS satellites. These satellite states are then output as part of the truth model. Unlike the user motion generator, this model must be accurate. The reason for this is due to the fact that the navigation software also calculates these same values based on sets of ephemeris data. Thus STMSIM uses the same model of eliptical orbits (ICP-GPS-200) that the navigation software employs.

Satellite Selection: This module determines which satellites are actually visible from the user vehicle. It will also select an optimal navigation constellation (including backup satellites) based on geometric considerations. This information is output as part of the truth model.

Range Data Generator: This module is responsible for generating the true satellite to vehicle ranges (distances), range rates, and range accelerations. All of this is output as part of the truth model.

This module also simulates the GPS information that is used by the navigation software as input. This information consists of pseudoranges and delta-ranges for both the L1 and L2 frequency channels. The pseudoranges are calculated from the true ranges by taking into account the following factors:

Ionospheric delays
Tropospheric delays
Time of signal transit
Rotation of the Earth during transit

NAVIGATION SOFTWARE DRIVER PROGRAM DESCRIPTION

The driver program (STMCTL) takes the input information produced by the data simulation program (STMSIM) and formats it correctly for the navigation software tasks and routines. After a navigation task or routine has run to completion, its output is saved in a file along with a selection of other important (navigation software) internal variables. The navigation output data is then

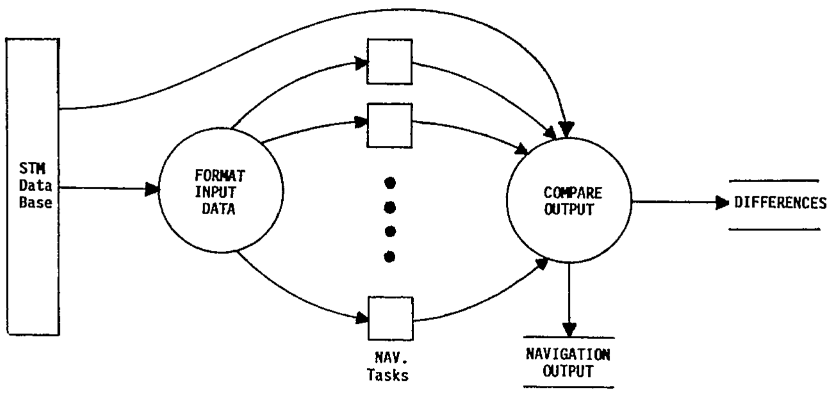

Figure 5: STMCTL FUNCTIONAL BREAKDOWN

compared with the truth model and the result output to another file. Also provided are some post-processing utilities which can be used to produce plots of the navigation program output or plots of the difference between this output and the truth model.

The design of the navigation software is such that the individual navigation tasks are fairly loosely coupled. This allows the STMCTL program to run these various tasks sequentially, rather than in parallel. In our design the only functionality this costs us is the possibility of locating timing bugs in the navigation program. Previous experience has taught us that attempting to simulate the exact nature of the executive (as far as scheduling multiple tasks is concerned) is not worth the effort. A significant amount of time is required to develop such simulation code and when completed it will still fail to point out the vast majority of timing bugs that may be present in the system.

Since tasks are invoked sequentially and run to completion, simulation of different task rates is handled by invoking high rate tasks several times sequentially. For example, a 20 hertz task will be invoked 20 times (running to completion each time) before a 1 hertz task is invoked. This system limits the timing information available from the simulator to the relative lengths of different tasks, such as task A is twice as long as task B.

In order to run, the navigation code is itself linked in with STMCTL. This has the benefit of allowing the simulator to get easy access to the COMMON blocks used by the navigation code, and also allows for straightforward invocation of tasks and routines (through subroutine calls). Being able to access the navigation program COMMON blocks allows the simulator to format input data and to look at output data and internal variables. The sole means of inter-task communication is through shared memory either common blocks or an executive call. In either case, the simulator can observe all such communication.

As well as being able to drive the complete navigation program in either of two modes (with and without IMU information), STMCTL was designed to drive various subsets of the navigation code. This enables STM to be used to test various functional modules within the navigation code. It also tested the documentation of the various internal interfaces in the navigation code, due to their implementation by two different people. Testing of the following navigation program modules is provided:

Transmitter (satellite) State Computation (NTSC)
NTSC & Transmitter Selectioni
NTSC & Pseudorange Computation (NPRC)
NTSC & NPRC & Receiver Aiding
Navigation State Propagation & IMU Preconditioning

This provides for testing 6 out of the 10 major components of the navigation program. The modules that were left out (Controller (NCON), Kalman Filter (NKFL), Library (NLIB), and Alternate Measurement Computation (NAMC)) were either too simple (NCON and NAMC) to warrant inclusion as a separate test case, or better tested at a different level (NKFL and NLIB).

OPERATIONAL SCENARIO

In general, STM is used as a navigation code debugging tool, not, as the name implies, a navigation code tester. A user creates a set of input data (simulated navigation input and truth model) by running the STMSIM half of the program. The user then links STMCTL with the navigation code and runs the combined STMCTL/navigation program.

At this point, the STMCTL/navigation program can be debugged like any other host computer program. Tools such as debuggers (sdb on Amdahl U.T.S.) and/or imbedded print statements are commonly used for this purpose. This mode of operation enables the programmer to come as close as possible to actually debugging the program in the target system, without actually putting up with the target system's lack of debuggin tools.

After the navigation program is up and running, STM has another function, that of a code tester. This is done on a VAX with an attached National Semiconductor microprocessor board. The navigation code is cross-compiled and linked with the cross-compiled STMCTL code. The STMSIM half of the program is then run on the VAX itself, followed by running the STMCTL/navigation program in the attached microprocessor board. In this way a check for any bugs inserted by the cross-compilation process is performed.

CONCLUSIONS

We feel that the design and inplementation of the STM simulator succeeded in currting out several months of hardware integration time. In addition, the use of different programming teams for STM and the navigation code aided the process of tracking down bugs in the external interfaces of the navigation program. This dual programming team also allowed for a more efficient utilizatjon of manpower. Both of the above greatly added to our ability to meet our development schedule.

Overall, the extra expense and effort put into the simulator at the beginning was a good investment. The expense was more than made up for during testing and integration of the navigation software.

JON VAVRUS received a BS degree from the California Institute of Technology in 1979. In the past, he has done work in Very Long Baseline Interferometry and Multi-processor operating system design for the Jet Propulsion Laboratory in Pasadena, California. At present he is working in the field of integrated Global Positioning System (GPS)/Inertial Navigation System (INS) systems for Computing Applications Software Technology (CAST) in Los Alamitos, California.

Mailing Address:
CAST
5450 Katella Ave., Suite 103
Los Alamitos, CA 90720
(213) 594-8883