

Verification of flight software written in C using ACSL

Rance S. Edmunds
Sandia National Laboratories
P.O. Box 5800, Div 9132
Albuquerque, New Mexico 87185

ABSTRACT

This paper documents a technique that has been used to interface the ACSL simulation package to C modules on a VAX computer. Using the technique documented in this memo, it should be possible to verify major portions of flight software (e.g. guidance and control algorithms) in a dynamic simulation environment. This verification can be accomplished fairly early in the development cycle, and if the flight software algorithms are written in C to begin with, then the time and effort required to convert them from FORTRAN (or ACSL code) to C can be avoided entirely. Also, when the C code is moved to the target flight computer, the debug time should be greatly reduced.

INTRODUCTION

Up until fairly recently, most flight software has been written in assembly language. This software could only be checked out by running the assembled code on the target computer or on a computer that could be made to emulate the target computer. More recently, advances in compilers, and improvements in flight computer speed and memory, have permitted some flight software to be written in higher level languages such as Pascal, Ada, and C. With these higher level languages, compilers are also available for general purpose (GP) computers permitting portions of the code to be checked out on the GP machine before going to the target computer.

Checking out portions of a flight code on a GP computer offers certain advantages. First, the target computer is sometimes unavailable early in the code development cycle. In this case flight code checkout can begin earlier using a GP computer. Second, many flight software algorithms (e.g. guidance and control algorithms) must be initially tested on a GP machine, to ensure that they perform as intended in a dynamic environment. If these can be tested using actual flight code, then the verification is more complete.

Traditionally, guidance and control algorithms have been written in FORTRAN or in a simulation language such as CSSL. After the algorithms had been checked out and verified, they were then coded in assembly language, or some other higher

level language, for checkout and use in the flight computer. Another method has been used in developing guidance, navigation, and control system software for several programs currently underway at Sandia National Laboratories. Software algorithms for the systems under study are written directly in C, the higher level language used for the SANDAC flight computer. They are then tested using the ACSL simulation package to simulate the dynamics of the vehicle, actuators, and sensors.

This testing does not, of course, replace the need for testing the flight code with the target computer, but it does move a major portion of the flight software verification to an earlier date in the development cycle. This significantly reduces the debugging time required when testing the flight software on the target computer.

This paper presents the techniques needed to accomplish the interface of C modules with the ACSL simulation package on a VAX computer. Interfacing ACSL with modules written in Pascal, and Ada should be possible using similar techniques.

EXAMPLE PROBLEM

Before discussing the interface technique, a brief description of the example problem used to illustrate the method will be given. Figure 1, shows a block diagram for an elementary single axis simulation of a rigid body spacecraft and associated attitude controller. The angular acceleration of the spacecraft is:

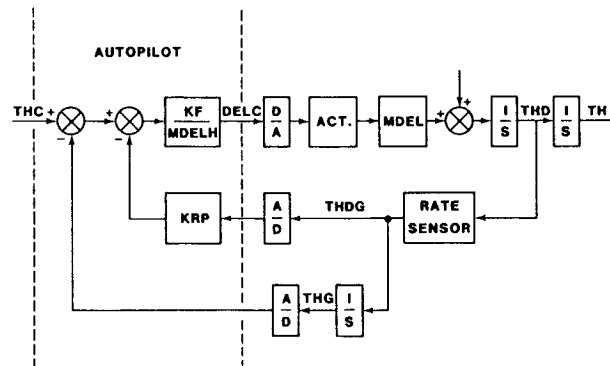


Figure 1. Control System Block Diagram

THDD = MDEL*DEL

THDD = spacecraft angular acceleration
MDEL = constant relating actuator deflection to angular acceleration
DEL = actuator deflection

The actuator and rate sensor are both assumed to have linear second order transfer functions:

$$\frac{\text{THDG}(S)}{\text{THD}(S)} = \frac{\text{WG*WG}}{S*S + 2*ZG*WG*S + \text{WG*WG}}$$

$$\frac{\text{DEL}(S)}{\text{DEL}(S)} = \frac{\text{WA*WA}}{S*S + 2*ZA*WA*S + \text{WA*WA}}$$

THD= spacecraft angular rate
THDG= measured value of THD
WG,ZG= rate sensor natural frequency and damping ratio

DEL= commanded actuator deflection
WA,ZA= actuator natural frequency and damping ratio

The angular position (attitude) measurement is obtained by integrating the rate sensor output.

The controller (or autopilot) uses conventional rate plus position feedback to obtain the actuator command:

$$\text{DEL} = \text{KP}*(\text{THC}-\text{THG}) - \text{KR}*\text{THDG}$$

KP = position gain
KR = rate gain
THC = commanded spacecraft attitude
THG = measured spacecraft attitude

ACSL will be used to simulate the dynamics of the spacecraft, actuator and rate sensor. The actuator command will be computed using C code.

ACSL CODE AND C INTERFACE

The ACSL code for this sample problem is listed in appendix A. Appendix B provides a listing of the associated C code. Figure 2 shows some sample output for a commanded attitude of 0, and an initial attitude of 0.1 .

The call to the C code is contained in a discrete block, with a procedural defining the variables used in the actual argument list (see Appx. A):

CALL AP(THC,THG,THDG,DEL)

This a FORTRAN subroutine call, and is translated as such by the ACSL translator. In order to successfully interface the FORTRAN call to C modules the following facts must be understood. FORTRAN passes its arguments by reference, not by value.

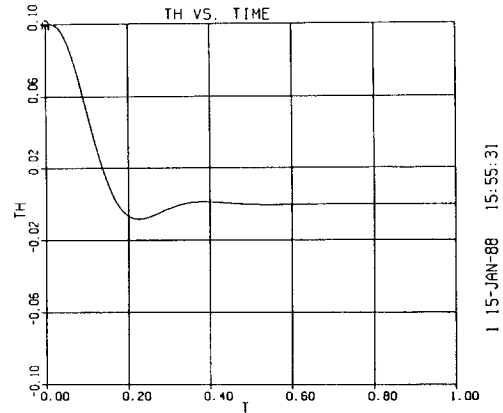


Figure 2. Sample Output

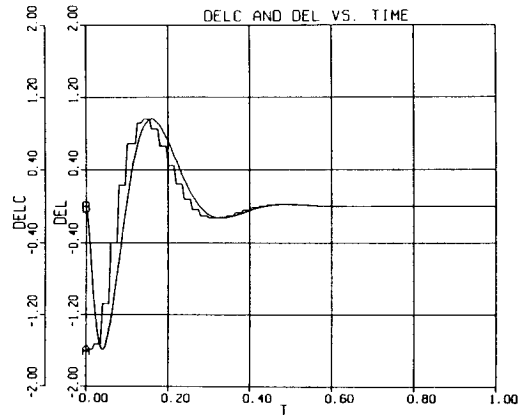


Figure 2. Sample Output

In other words, the subroutine call passes the addresses of the variables contained in actual argument list, not the values of the variables. Also, ACSL operates with a default variable type of real (single precision). So for this example, the call passes the addresses of the four real variables contained in the argument list.

The C code equivalent to a subroutine, is a function of type void. The function declaration statement associated with the above call is:

```
void ap(p_thc,p_thg,p_thdg,p_delc)
float *p_thc,*p_thg,*p_thdg,*p_delc;
```

Note that the arguments are declared to be pointers to float (single precision real), to be consistent with the FORTRAN and ACSL convention. The symbol * in the type statement, identifies a pointer variable. More specifically, *p_thc is the value of the single precision real variable to which p_thc points. The p_ prefix is a convention the author has found to be helpful in keeping track of pointers but has no special meaning in C.

Suppose that the call included an integer (IFLAG) and a real array CIC(3,3). Then an example CALL from ACSL would be:

```
INTEGER IFLAG
ARRAY CIC(3,3)
CALL AP2(IFLAG,CIC,THC)
```

The associated C function statement would be

```
void ap2(p_iflag,t_cic,p_thc)
int *p_iflag;
float t_cic[3][3],*p_thc;
```

In this case p_iflag is a pointer to an integer variable, and p_thc is a pointer to single precision real variable. Since t_cic is an array, C expects to have a pointer passed to it and the * prefix is not used in the type declaration. FORTRAN identifies the starting address of CIC as CIC(1,1), and C identifies the starting address of the associated array as t_cic[0][0]. So far so good. But, FORTRAN and C index the elements of arrays differently. FORTRAN stores (and accesses) two dimensional arrays by column, whereas C stores them by row. Not so good. There are two ways to deal with this.

Either the FORTRAN call can send the transpose of the array CIC to the C module, or the C module interface can take the transpose. In the above example, it has been assumed that the C function will operate on the transpose of CIC (This is the meaning of the t_ prefix on the cic[3][3] array. Note however, that t_ has no special meaning in C).

Referring to Appx. B, it can be seen that two C modules have been used. The first module "ap" is strictly for interfacing the rest of the C modules (in this case a single module) with ACSL. The second module "actuat" represents a simplified example of C "flight" software code. The interface module is required so that when the module "actuat" is incorporated into completed flight software, that no changes will have to be made to it. The module "ap" is not intended for use as part of the flight software.

The interface module can be used to convert pointers to values (since C by default calls by value), change single precision real numbers to double precision (if the C module works with its default double precision real numbers), and reorder arrays (if arrays having more than one subscript are used).

VAX/VMS COMMANDS

The following VAX/VMS commands are recommended for translating, compiling, linking, and running the sample code of Appx. A and B (ACSL run time library level 8R).

- 1) Create the ACSL source file SCRAFT.CSL
- 2) Translate, and Compile


```
$ ACSL/NOLINK SCRAFT
```
- 3) Create the C source file AP.C
- 4) Compile


```
$ CC/LIST/STANDARD=PORTABLE AP
```
- 5) Create a library file for your C modules


```
$ LIB/CREATE CLIB
```

If more than one C module is involved, include them in the library also.
- 7) Link the C object code with the FORTRAN object code for SCRAFT


```
$ ACSL/LINK/NORUN/LIBR=CLIB SCRAFT
```
- 8) Run the executable code


```
$ RUN SCRAFT
```

If the C code is modified, it is only necessary to repeat steps 4, 6, 7, and 8. In otherwords, it is not necessary to retranslate and recompile the ACSL code. This can be a considerable savings in computer time for large ACSL codes.

CONCLUSION

The procedure discussed above is not difficult to implement. However, the potential payoff in man months saved on a project could be significant. As a minimum, it should reduce the risk associated with installing and verifying flight software on the target computer.

APPENDIX A, Sample ACSL Program Listing

PROGRAM

```
"This program models an elementary ...
 spacecraft attitude ...
 control system. The purpose is to ...
 demonstrate the ACSL ...
 interface with C modules."
```

INITIAL

```
VARIABLE T,TIC=0.0
CINTERVAL CINT= 0.01 $"comm interval"

ALGORITHM IALG(2)= 5 $"4TH order RK"
MAXTERVAL MAXT(2)= .002 $"max time step"
NSTEPS NSTP(2)= 1 $"number of steps"

CONSTANT TMAX= 1.0 $"sim stop time"
CONSTANT MDEL= 10.0 $"ang accel/act def"

CONSTANT THC= 0.0 $"commanded angle"
CONSTANT THDIC= 0.0 $"initial ang rate"
CONSTANT THIC= 0.1 $"initial angle"
CONSTANT THDGIC= 0.0 $"sensor ang rate IC"
CONSTANT THGIC= 0.1 $"sensor ang IC"

CONSTANT PI= 3.14159
CONSTANT WGHZ= 30.0
CONSTANT WAHZ= 15.0
TPI= 2.0*PI
WG= WGHZ*TPI $"Rate sensor natural freq"
TZWG= 2.0*0.707*WG $"Rate sensor
 damping"

WG2= WG*WG
WA= WAHZ*TPI $"Actuator natural freq"
TZWA= 2.0*0.707*WA $"Actuator damping"
WA2= WA*WA
```

END \$"INITIAL"

DYNAMIC

```
DISCRETE AUTOPI $"Autopilot"
INTERVAL DTAP= .02

PROCEDURAL(DEL=THC,THG,THDG)
CALL AP(THC,THG,THDG,DEL)

END $"PROCEDURAL"
```

END \$"DISCRETE"

DERIVATIVE

```
"Vehicle dynamics"
THD= INTEG(MDEL*DEL,THDIC)
TH= INTEG(THD,THIC)

"Sensor output"
THDGD= ...
INTEG(WG2*THD-TZWG*THDGD-WG2*THDG,0.0)
THDG= INTEG(THDGD,THDGIC)
THG= INTEG(THDG,THGIC)

"Actuator output"
DELD= ...
INTEG(WA2*DEL-TZWA*DELD-WA2*DEL,0.0)
DEL= INTEG(DELD,0.0)
```

END \$"DERIVATIVE"

```
TERMT(T.GE.TMAX)
END $"DYNAMIC"
```

TERMINAL

```
END $"TERMINAL"
END $"PROGRAM"
```

APPENDIX B, Sample C Program Listing

```
/* C code to interface autopilot loop to
 ACSL code
 p_ prefix indicates pointer
 inputs
 thc= commanded position
 thg= measured position
 thdg= measured rate
 output
 delc= commanded actuator position
*/

#define DEBUG 0

void ap(p_thc,p_thg,p_thdg,p_delc)
float *p_thc,*p_thg,*p_thdg,*p_delc;

(
 void actuat();
 float thc,thg,thdg,delc;

/* convert pointers to their values for all
 scalar inputs */
 thc= *p_thc;
 thg= *p_thg;
 thdg= *p_thdg;

/* C function call */
 actuat(thc,thg,thdg,&delc);

/* store output */
 *p_delc= delc;

)

/*
 actuator command module
 inputs
 thc= commanded position
 thg= measured position
 thdg= measured rate
 output
 *p_delc= commanded actuator position
 internal
 MDELH= computed value of MDEL
 KP= position gain
 KR= rate gain

*/

#define TPI (2.0*3.14159)
#define MDELH 10.0
#define WAP (2.0*TPI)
#define ZAP 0.707
#define KF (WAP*WAP)
#define KP (KF/MDELH)
#define KRP (2.0*ZAP/WAP)
#define KR (KP*KRP)

void actuat(thc,thg,thdg,p_delc)
float thc,thg,thdg,*p_delc;
{
 *p_delc= KP*(thc-thg) - KR*thdg;

#if DEBUG
 printf("\nactuat: thc,thg,thdg %g %g %g",
 thc,thg,thdg);
#endif
}
```

```
    printf("\nap: KP,KR,*p_delc %g %g %g",  
          KP,KR,*p_delc);  
#endif  
}
```