FACS - A FORTH ANALOG COMPUTER SIMULATOR

Nicholas G. Lordi
College of Pharmacy
Rutgers University
Piscataway, N.J. 08854

Forth provides a natural programming environment for creating special purpose simulation languages. FACS is a block-oriented continuous-system simulation language implemented in both integer and floating-point versions. Forth versions of CSSL's (including FDARE, a Forth equation-oriented CSSL) have the following advantages over CSSL's written in Pascal and Fortran: minimal memory requirements, keyboard alteration of model parameters without recompilation, and extensibility. Only the integer version, programmed in F83 Forth, will be discussed.

In developing a FACS simulation, all state variables should be scaled so that they do not exceed one unit (= 100,000,000) during a simulation run. Scalars, in contrast to state variables, are defined so that 1 scalar unit = 10000. All numbers except integers are entered and displayed in decimal format, using no more than 4 decimal places. This restriction is consistent with the degree of accuracy attainable with reasonable integration step sizes (ca., 0.01). The current version allows output to be displayed or printed as well as low resolution character plotting on a printer.

Table 1 lists the FACS vocabulary included in the integer version. Transcendental functions, which have been implemented using series approximations, are generally accurate to 7 decimal places in the range 0 to 1. Unlike other block-oriented CSSL's, there is no limit on the number of or specific blocks which may be used in a simulation. Special block-defining words are used to define integrators, arbitrary function generators, tables, differentiators, track/store elements, and delays. The integrator block implements the second-order Runge-Kutta algorithm.

Definitions of the three principal FACS data types are listed below. Bartholdi's "TO" solution is used to simplify use of FACS variables. DTO? is a double number version of TO?. #IN converts a decimal number to a scaled integer.

```
: INTEGER (S -- n ) CREATE , DOES> TO? ;
: SCALAR (S -- n ) #IN INTEGER ;
: VECTOR (S n -- d ) 1+ CREATE 4 * DUP HERE SWAP ERASE
  ALLOT DOES> SWAP 4 * 4 DTO? ;
```

The Van der Pol equation simulation outlined in Figure 1 illustrates how FACS is used as well as some of the implementation details. The FACS user should first scale the problem so that state variables do not exceed 1 unit, then construct an analogous block diagram. The first step in developing a FACS program is to set the number of nodes to 3+ the number of blocks required by the simulation. The first 3 nodes are reserved for system use. The user must define 2 vectors: one to store current block outputs, e.g., Z(), and a second which stores initial conditions assigned to integrator blocks, e.g., Z0(). These definitions have been deferred to minimize system memory allocation, thereby avoiding recompilation for large problems. Execution of SET-Y and SET-IC resolve the deferred vectors.

```
: SET-Y (S -- ) DEFINED DROP IS Y() ;
```

Two system parameters must then be set. COMINT is the communication interval, i.e., the number of integration steps executed before output is requested. DT/2 is one-half the integration step interval. TO is used to assign values to scalars (always in decimal format) or integers. Model parameters are defined as scalars (e.g., K); initial conditions other than 0 are assigned (e.g., integrator block 5 is set to 0.5 unit); and the specific blocks for which output is requested are identified by PR-NODE.

Each integrator block must be separately named using the word RK2; thus, only those integrators required by the simulation need be defined. Execution of n1 n2 <name> twice replaces Y(n1) with the integral of Y(n2), since the 2nd-order integrator requires two passes. Each integrator allocates memory space for a flag which identifies the pass and for temporary storage of the previous integrator output.

```
: RK2 (S n1 n2 -- ) CREATE 0 , 4 ALLOT DOES> TO PFA ( stores pfa )
  Y() DT/2 10000 M*/ ( multiplies input vector by scalar )
  PFA @ 0= IF ( 1st pass ) 2 PICK Y() 2DUP PFA 2+ 2! D+
  ELSE ( 2nd pass ) D2* PFA 2+ 2@ D+  0 PFA ! ( restore flag )
  THEN -> ROT Y() ( update output ) ;
```

The simulation model (e.g., VDPOL) is defined using MODEL: (a redefinition of ":"). The general format for block program statements is: scalar out-node in-node1 in-node2 ... word. Figure 2 shows an example of displayed output. SIMULATE <model name> selects the named model for simulation by vectoring to the system defined dummy word "model".

```
: model NOOP ;
: SIMULATE (S -- ) DEFINED DROP ['] model 2+ ! ;
: RUN (S n -- ) RESET INITIALIZE CONTINUE MESS1 ;
: INITIALIZE (S -- ) DT/2 TO DT' 0 TO DT/2 model model
   DT' TO DT/2  CR output ;
: CONTINUE (S n -- )  0 DO HALT COMINT 0 DO model TIME
   model TIME  LOOP output LOOP ;
```

DISPLAY selects output in tabular form on the current output device; alternatively, PLOT would provide low resolution character plotting of selected blocks. RUN controls the simulation: it sets all initial conditions (RESET), it calculates the output of all other blocks by calling "model" twice with DT/2 temporarily set to 0 (INITIALIZE), and executes the simulation (CONTINUE) n*COMINT times (in this case, 500). The 2nd-order integrator requires that "model TIME" be executed twice. TIME increments Y(0) by DT/2 (node 0 is reserved for the time function). HALT suspends a run when any key is pressed, continues if any key is pressed again except for CR which aborts the run. Each time "output" is called, results will be displayed (or plotted). When a run is completed, execution of n CONTINUE will provide further output. Parameter changes may be made or initial conditions altered without recompilation.

In FACS, one may easily collapse complex problems into single words, which can subsequently be used as blocks in more complicated simulations. For example, FIN-DIF (Figure 3) is a block-defining word (really a special purpose integrator) which solves the 1-dimensional diffusion problem using the difference-differential equation method. It is used in the form n FIN-DIF <name>, where n is the number of stations. Execution of RECT/SPHER/CYLIN n1 n2 <name> (where n1 is the output and n2 is the input boundary node) solves Fick's Second Law in rectangular, spherical, or cylindrical coordinates, depending on which word is first executed. DX and D/DX2 are user modifiable parameters.

TABLE 1 - FACS VOCABULARY

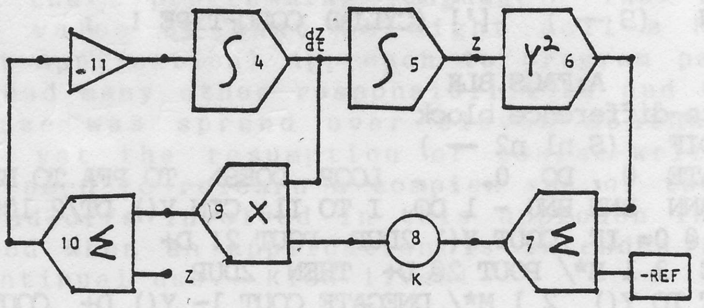
Data Types		System Control		I/O	Block-Defining Words	
INTEGER	NODE	ASSIGN-IC	INITIALIZE	DISPLAY	DELAY	RK2
LABEL	SCALAR	ASSIGN-Y	RUN	PLOT	DIF	TABLE
MATRIX	VECTOR	CONTINUE	SIMULATE	PR-NODE	FIN-DIF	T/S
		HALT	TIME	TO	FUNCGEN	

Blocks							System Nodes
ABSV	GAIN	INV	OFFSET	POT	RAND	STOP	T
DIV	MULT	SQR	SQRT	SUM	Y**X	10**	+REF
ARCT	COS	EXP	LN	LOG	SIN		0REF
BANG	CMP	DEAD	LIMIT	SWITCH	+CLIP	-CLIP	-REF

Figure 1 Van der Pol Equation

$$\frac{d^2Z}{dt^2} + k(Z^2 - 1)\frac{dZ}{dt} + Z = 0$$

BLOCK DIAGRAM



FACS PROGRAM

```
11 NODES
11 VECTOR Z()          11 VECTOR Z0()
SET-Y Z()              SET-IC Z0()
0.001 TO DT/2          100 TO COMINT
0.5 SCALAR K
5 0.5 ASSIGN-IC        5 4 11 PR-NODE
RK2 DZ/DT              RK2 Z
```

```
MODEL: VDPOL  4 11 DZ/DT  5 4 Z  6 5 SQR  7 6 -REF SUM
              K 8 7 POT  9 4 8 MULT  10 9 5 SUM  11 10 INV  ;
```

Figure 2

```
SIMULATE VDPOL                                  Blocks
DISPLAY 5 RUN            T        5 (Z)     4 (dZ/dt)   11 (d^2Z/dt^2)

                         0        0.5000     0           -0.5000
                         0.2      0.4896    -0.1031      -0.5288
                         0.4      0.4582    -0.2104      -0.5413
                         0.6      0.4052    -0.3186      -0.5383
                         0.8      0.3307    -0.4247      -0.5198
                         1.0      0.2354    -0.5254      -0.4836
```

Figure 3

```
27 LIST 28 LIST
Scr # 27         A:FACS.BLK
  0 \ special functions - finite-difference block              9May86ngl
  1 0 INTEGER I1   0 INTEGER BN0   0 INTEGER BNN
  2 : CIN    (S -- n )    BN0 I1 + BNN BN0 - + ;
  3 : COUT   (S -- n )    BN0 I1 + ;
  4 : POUT   (S -- n )    PFA 2+ I1 1- 4 * + ;
  5 0. SCALAR X0   1. SCALAR DX   0.01 SCALAR D/DX2
  6 VARIABLE CORD-TYPE
  7 : CALC-RDR    (S -- )
  8   COUT 1+ Y()  COUT 1- Y() D-  DX 10000 M*/  10000 I1 DX *
  9   X0 + M*/  ;
 10 : (RECT)    (S -- d )    NOOP  ;
 11 : (SPHER)   (S d -- d )   CALC-RDR D+  ;
 12 : (CYLIN)   (S d -- d )   CALC-RDR 1 2 M*/ D+  ;
 13 : RECT    (S -- )     ['] (RECT) CORD-TYPE !  ;
 14 : SPHER    (S -- )    ['] (SPHER) CORD-TYPE !  ;
 15 : CYLIN   (S -- )     ['] (CYLIN) CORD-TYPE !  ;

Scr # 28         A:FACS.BLK                                     9May86ngl
  0 \ finite-difference block
  1 : FIN-DIF   (S n1 n2 -- )
  2   CREATE 0 , DO  0. , ,  LOOP  DOES>  TO PFA TO BN0
  3   TO BNN  BNN BN0 - 1 DO  I TO I1  CIN Y() DT/2 10000 M*/
  4   PFA @ 0= IF  COUT Y() 2DUP  POUT 2! D+
  5   ELSE 2 1 M*/ POUT 2@ D+  THEN  2DUP
  6   COUT TO Y()  2 1 M*/ DNEGATE COUT 1- Y() D+  COUT 1+ Y()
  7   D+  CORD-TYPE @ EXECUTE  D/DX2 10000 M*/  CIN TO Y()
  8   LOOP  PFA @ 0= PFA !  ;
  9 \ A 2nd-order Runge-Kutta integrator which simulates a 2nd-order
 10 \ partial differential equation by solving n equations of the
 11 \ form (e.g., in rectangular co-ordinates):
 12 \
 13 \
 14 \
 15 \
ok
```

$$\frac{dU_i}{dt} = \frac{D}{\Delta x^2}\left(U_{i-1} - 2U_i + U_{i+1}\right)$$