PropFACS USER MANUAL Version 1.1

Propeller Forth Analog Computer Simulator

Nicholas G. Lordi March 2011

Forth provides a natural programming environment for creating special purpose simulation languages. One example is PropFACS, a block-oriented continuous-system simulation language implemented in PropForth version 4.0a (authored by Sal Sanci), designed to run on the Parallax Propeller microcontroller. It is a version of FACS, originally written in F83 Forth. This document describes the characteristics and use of PropFacs with examples drawn from the analog computer literature of the 1960's. It emulates how computers were used to simulate systems expressed in differential equation form in that era.

PropFACS has the following characteristics.

This version emulates a 48 amplifier + 24 potentiometer analog computer.
It is integer based, requiring scaling of model parameters.
It consists of blocks, each performing a distinct mathematical operation.
In addition to the usual mathematical operations, logic and nonlinear blocks are included.
Multiple input options include random, ramp, step, and impulse functions.
Output options are display, store, and plot.
A FACS model consists of blocks which make up a forth word.
The FACS vocabulary includes words which control the simulation, enable definition of model parameters, and provide output.
FACS is extensible, allowing new words to be added to the dictionary as required by specific models.

While it is not necessary to know the forth programming language in detail, one should be aware of some conventions regarding its use. Most important is that reverse polish notation is used, e.g., 1 2 + not 1 + 2. All operations are carried out on a data stack, thus 1 2 + puts 3 on top of the stack. The basic approach to programming in Forth is to define words which are strung together to form new words. PropFACS is a language with words designed to facilitate the solution of differential equations.

The basic components of analog computers were operational amplifiers whose function was determined by the passive components (resistors, capacitors & diodes) which were placed at the input and feedback connections of the amplifiers. FACS blocks are the "analog" of the operational amplifiers in an analog computer. Since analog computers worked with voltages which are usually limited (10 or 100 volts), problems had to be scaled so that the maximum allowed values were not exceeded. The PropFACS scaling factor is 1000000 for state variables, which corresponds to 1.0 in a model. The model parameter (potentiometer settings on an analog computer) scaling factor is 10000. PropFACS models will tolerate overflows up to 10 units, enabling simpler problem scaling. Furthermore, the

precision (3-4 decimal places) is 10 times the precision of analog computers, which was dependent on the quality of the operation amplifiers and passive components.

Since analog computers were parallel computers and the computing time was determined by the time constants of the integrator capacitors, simulations could be run faster than, slower than, or in real time. Model complexity was not a factor. Changes in model parameters could be made during a simulation run and the results immediately seen. This is not the case with digital emulations. The actual run time increases with model complexity. No parameter changes can be until a run is completed.

Note: The current version of PropFACS (1.1) occupies 4814 bytes on top of the PropForth , including the optional word set and eeprom filing system words. 8609 free bytes (in hub ram) is available for models. PropForth is CASE SENSITIVE. Since most PropForth words are in lower case and most PropFACS words are in uppercase, this avoids the necessity to be very careful in not redefining existing words.

FACS VOCABULARY

Utilities Only those words which are required to use PropFACS are listed.

TO An assignment operator for state variables and scalar parameters, which function as both constants and variables. If 'rate' is a parameter, then 345 TO 'rate' assigns the value 345 to 'rate', while executing 'rate' pushes 345 on the data stack.

pretime
postime This word pair allows the user to determine the execution time in microseconds of all words set between them. For example: 'pretime <…. words ……> postime' would place the execution time on the stack. The words are set for an 80 MHz clock and corrected for use in compiled words..

noop No Operation

decimal This word sets the base used by PropFACS. PropForth uses hex as the default base. Decimal is the first executed in defining a model and always executed after a reset to avoid misinterpretation of number assignments and output.

sc Use this word to ensure stack is cleared.

free Use free to determine how much ram memory (bytes) is available after defining models.

forget Executing forget FACS deletes PropFacs .

Note: Some knowledge of forth programming techniques will prove useful, since there will be situations where you need to perform specific action(s) which require definition of new words. This is demonstrated in some of the examples.

Constants and Variables Those words used by specific blocks will be defined in the blocks section.

Y0 A long constant. The state variable scaling factor = 1000000, which is equivalent to 1.0 in a scaled model.

Ym A long constant. The maximum allowed value of the state variable = 10*Y0. An overflow condition results if exceeded.

scale A word constant = 10000, used to scale model parameters, e.g., 5000 is equivalent to 0.5 and other scalars (defined as longs but scaled so 10000 is equivalent to 1).

+REF
0REF
-REF Constants associated with the reference values, Y0, 0 and –Y0. These constants place the node numbers 1, 2, & 3 on the stack, corresponding to Y(1), Y(2) and Y(3), which store the actual reference values .These 3 vector cells are reserved for the three references and can not be used for other blocks.

DT A word variable. The integration step interval in simulated time: default 10.

COMINT A word variable. The communication interval, i.e., the number of time steps which elapse before output is requested: default 100.

FIX Word constant which specifies the number of decimal places used in displaying Output: default 3.

Y() A vector, i.e., an array of longs, which defines the state variables accessed by all Blocks: default 52. (4 elements are reserved for system use)

IC() The initial condition vector which stores the initial conditions assigned to integrators in the model. IC() is predefined with 12 cells. Two are used for each initial condition.

Kx The model parameter vector (longs) stores potentiometer values and other model parameters: default 24.

N() An array of words (2 bytes) which stores nodes associated with state variables for which output is requested: default 10.

B() An array of words which stores logic bits -1 or 0 (true or false) associated with logic block nodes. By convention -1 not 1 is used to represent a true state: default 20.

Pi Leaves 31416 on stack (a word constant = 3.1416).

Ei Leaves 27183 on stack (a word constant = 2.7183).

Defining Words These are words which define other words

vector Defines a one-dimensional array of longs, e.g., 5 vector Z() defines an array of 5 4 byte cells. It is used as follows: 100000 TO 3 Z() ….. 3 Z() leaves 100000 on the stack. Y(), Kx, and IC() are predefined vectors.

array Defines a one-dimensional array of words (2 byte cells). It is used to define the system arrays N() & B().

ZERO-Y() Sets all Y() vector values to 0.

ZERO-IC Sets all IC() vector values to 0.

ZERO-B() Sets B(() array to 0.

matrix A matrix defining word: 10 3 matrix mata defines mata as 10 rows by 3 columns. **SMAT-IS** resolves the deferred word **SMAT,** which will be used to store output in hub ram.

Note: All vectors and arrays include a 0 cell. 0 Y() is reserved for the time function.

Blocks

Conventions n1 is the output node
n2, n3 … are input nodes
k, if present **,** represents a parameter value (a long)
b1, b2, … represent logic bits 0 :on or -1:off

The following notation (stack diagram) is used : (k n1 n2 n3… ---) where --- represents the block name. An equivalent statement is: 500 3 2 1 <block name>.
All available blocks are listed according to function.

Note: All blocks must leave the stack empty.

Mathematical There are 12 basic mathematical operations.

INV (n1 n2 ---) Y(n1) = - Y(n2)
The sign of the input vector is inverted.

ABS (n1 n2 ---) Y(n1) = Absolute Value of Y(n2)

POT (k n1 n2 ---) Y(n1) = Y(n2) * k If k is <= 1. it is a potentiometer with one side grounded. If k >1 (max 10), it is a potentiometer with a gain of 10 . k is a scalar defined in the vector Kx. If k has a negative value, the POT functions as an implicit inverter.

OFFSET (k n1 n2 ---) Y(n1)= Y(n2) +k where k is a parameter scaled as a state variable. It may be + or -.

SUM (n1 n2 n3 …. ---) Y(n1) = Y(b2) + Y(n3) + …
The output vector is the sum of the inputs. In principle, any number of inputs is allowed.

MULT (n1 n2 n3 ---) Y(n1) = Y(n2)*Y(n3)
Two input sate variables are multiplied.

DIV (n1 n2 n3 ---) Y(n1) = Y(n2) / Y(n3)
Division – note the order of the vectors.

SQR (n1 n2 ---) Y(n1) = Y(n2)*Y(n2)

SQRT (n1 n2 ---) Y(n1) = Square Root[Y(n2)]

SIN (n1 n2 ---) Y(n1) = Sin[Y(n2)]
This block (as well as LOG2 and ALOG2) accesses the corresponding propeller data table at increments of 0.000855 radians (0.049 degrees). Units must be in radians. Results are accurate to a precision of 3 decimal places. Input is restricted to 0 – 90 degrees.

LOG2 (n1 n2 ---) Y(n1) = Log 2 [1 + Y(n2)] or Log2[Yn2] – 1 where Y(n2) is 1 to 0.5. Note that LOG2 is log to the base 2. The propeller table gives values only for inputs 1 – 2, while inputs are restricted to scaled values 0-1. These may be converted to Lg(base 10) or Ln values by multiplying the output by the system constants Lg2 or Ln2, respectively.

ALOG2 (n1 n2 --) Y(n1) = Antilog base 2[1+Y(n2)] - 1

Note: Operational amplifiers in many analog computers also inverted the input signals. In PropFACS, all blocks are non-inverting, with the exception of the explicit inverter as well as potentiometers and gains if assigned negative values .

Integrators Three integrators are available.

EULER (n1 n2 ---) Y(n1) = Integral[Y(n2)] where Y(n1) = Sum of Y(n2)*DT .
This is the simplest and fastest integrator at a given DT value. However, it is the the least accurate, requiring small DT values. It can be used for simple linear models or cases where only approximate solutions are needed

RK2 (n1 n2 ---) Y(n1)= Integral[Y(n2)]
The 2nd Order Runge-Kutta integrator corrects the Euler method, resulting in more accurate results at larger DT values. Use this integrator where greater accuracy is necessary.

INTGR (n1 n2 ---) A deferred word which can be used in place of EULER or RK2 in model definitions, allowing the user to change integrators without redefining a model.
INTGR-IS <name>**,** where name is RK2 or EULER, activates the desired integrator.

Note: The 4th Order Runge-Kutta integrator was the integrator of choice, but is not necessary to achieve useful data in an analog computer emulator

Input Functions

TIME (---) Increments time DT units at each execution, in effect, generating a linear output with a slope of one in 0 Y(). This can be used to activate other input functions.

RND (n1 ---) Y(n1) is a random number 0 – Y0.

RAMP (n1 ---) Y(n1) is linear from X0 to Y0 with a slope = +SLOPE. If > Y0 , Y(n1) = 0. Cannot be used in combination with SQWAVE or TRIWAVE.

X0 These are variables (longs) which the user defines to establish the ramp's
SLOPE characteristics: starting value and slope which is a scalar.

IMPULSE (n1 ---) Y(n1) is a train of impulses, whose period is determined by the variable (long) **PERIOD**. An impulse is defined as a square wave with a width = DT and an amplitude = X0.

SQWAVE (n1 ---) Y(n1) is a train of square waves with a period determined by the value of the variable **PERIOD** and the square wave width by the variable **XT**. These variables are defined in units of time. The amplitude is X0 and XE sets the zero-level. Cannot be used in combination with RAMP or TRIWAVE.

TRIWAVE (n1 ---) Y(n1) is a repetitive triangular wave which may have different positive and negative slopes depending on the values of **+SLOPE** and **–SLOPE** which are user defined scalars. The period depends on the slope values and the range is 0 to Y0 and XE sets the zero-level. Cannot be used in combination with RAMP or SQWAVE.

INPUT (n1 ---) A word used to identify an input in a model. n1 is the output node of the input block (excluding FUNGEN) which is selected prior to a run. This approach allows selection of different inputs (RAMP, SQWAVE, etc.) without redefining models. **INPUT-IS** <name> is used to select inputs.

FUNGEN (n1 n2 ---) Y(n1) = F[Y(n2)] Function Generator simulates the classic diode function generator which provided user defined wave forms and was used to generate log & other functions. It requires a table of slopes defined in the model definition, as follows. S**LOPES** is a deferred word which is part of the definition..

variable <name> 10000 l, 5000 l, 2000 l, 6000 l, …..
SET-SLOPES <name>

Break points are set at equal intervals determined by the period value **FPERIOD**. Also assign the number of break points to the variable **NBPT**. If NBPT is exceeded during a run, the slope will be set to 0. If n2 is 0, the generator will produce a time-dependent waveform, otherwise a transformation of Y(n2). The user has the option of defining tables of any length, available memory permitted. Slopes are longs treated as scalars. **FGAIN** is used to adjust output to desired levels (values must be > 0).

Nonlinear Functions These are functions which include discontinuities.

-CLIP (n1 n2 ---) Y(n1) = only positive values of Y(n2)

+CLIP (n1 n2 ---) Y(n1) = only negative values of Y(n2)

BANG-BANG (n1 n2 ---) If Y(n2) is positive, Y(n1) = Y0, else Y(n1) = -Y0.

DEAD (k1 k2 n1 n2 ---) DEAD-ZONE sets Y(n1) = k1 if Y(n2) >k1 and <k2.

LIMIT (k1 k2 n1 n2 ---) The LIMIT block sets Y(n1) to k1 if y(n2) < k1 or to k2 if Y(n2) >k2, otherwise, Y(n1) = Y(n2).

STOP (n2 n3 ---) Terminate the run if Y(n2) >Y(n3) – both are inputs. A reset is executed when a run is terminated.

DELAY (n1 n2 ---) Y)(n1) = Y(n2) is delayed DN*DT time units, where **DN** is a variable. The Delay Line block is emulated as a FIFO shift register with each cell delayed DT time units. The shift register is defined as a vector with DN cells. **DELAY-IS** <name> activates the deferred vector **DLY()**. **CLEAR-DLY** clears the Delay Line.

Logic and Logic-to-Analog Interfaces

T/H (n1 n2 b1 ---) This Track/Hold block must precede an integrator. If B(b1) is true, Y(n1)=0, otherwise Y(n1) = Y(n2). The output of the integrator is held at its current output value when Y(n1) = 0.

CMP (n2 n3 b1 ---) The comparator block sets B(b1) true if Y(n2)>Y(n3) else B(b1) is false. This block must precede a SWITCH block.

SWITCH (n1 n2 n3 b1 ---) If B(b1) is true, Y(n1) = Y(n3) else Y(n1) = Y(n2). +REF , etc., can be substituted for Y(n2) or (n3). If on (-1) or off (0) is substituted for b1, SWITCH functions as an on/off manual spdt switch.

PULSE A pulse generator which produces on/off bits at user selected widths and intervals. PULSE is used with switches and/or logic blocks to control aspects of specific simulations. The required parameters are START (a scalar), WIDTH, & INTERVAL, which must be set as part of the model definition.

RNDB (--- b1) Produces random on/off bits in B(b1).

OR (b1 b2 b3 ---) B(b1) = B(b2) or B(b3)
AND (b1 b2 b3 ---) B(b1) = B(b2) and B(3)
NOT (b1 b2 ---) B(b1) = invert B(b2)

Output

dout (x --- 'x') The basic output routine which displays state variables on the terminal in decimal format, i.e., 'x.xxxx'.

tout (---) Outputs time 0 Y() in decimal format.

NODES (n ---) Sets number (n) of nodes in model.

ZER0-NODES (-- -) Clears current node assignments. Used in OUT-NODES.

OUT-NODES (n1 n2…. ---) Enter specific nodes for which output is requested other than time, which is automatically included. Nodes should be specified in reverse order then you wish displayed.

(display) (---) Tabulates time and outputs in columns identified by OUT-NODES On the active display.

(mstore) (---) Stores 1 row of data in SMAT.

MREAD (n1 n2 ---) Reads (n1 rows – n2 columns) and displays data stored in SMAT.

xplot (n1 ---) Sends x-value, Y(n1), to xy-plotter.

yplot (n1 ---) Sends y-value, Y(n1) to xy-plotter.

(typlot) (---) Enables vector-time plots on plotter: 1 or 2 variables specified by the nodes
(typlot2) stored in 1 N() and 2 N() if (typlot2) is selected.

(xyplot) (---) Enables plotting y- versus x-vectors.

WAIT A parameter (word) set to no. millseconds delay in plot output. This is used to adjust simulation output to plotter requirements: default 0.

XSCALE, YSCALE Values (longs) used to adjust plotter x- and y-scales.

XLEVEL, YLEVEL Values (longs) used to set zero-position in plots.

NOTE: Plotting is done on a TV screen, using special programming and hardware. Use of the plotter will be described in an addendum to this document. Analog computer results were obtained using oscilloscopes, strip chart or xy-recorders. The FACS plotter allows the user to visualize the data during a simulation run, rather than just stare at lists of numbers.

SET-OUTPUT Use: SET-OUTPUT <output> activates <output> as the current output mode: <output> is (display), (mstore), (typlot), etc.

System Controls

MODEL: An alias for the forth directive **':'.** MODEL: <name> …… blocks ; defines <name> as a simulation model.

SIMULATE SIMULATE <model> activates <model> as the current simulation object.

ASSIGN-IC (…..y3 n3 y2 n2 y1 n1 ---) Clears IC() vector assigns initial conditions to it,e.g., IC(1) = y1, IC(2) = y2, IC(3) = y3. Initial conditions are only applied to integrators.

ASSIGN-REF (---) Assigns values to reference blocks: +REF, 0REF, and –REF.

RESET (---) Clears vectors Y() & B(), executes ASSIGN-REF, resets time to 0, and applies all initial conditions.

INITIALIZE (---) Initializes model block outputs to starting values. This happens automatically in an analog computer. The emulation requires the model to be run with DT = 0, in order to set blocks other than integrators to their correct initial states.

HALT (-- -) A running simulation will pause when any key is pressed and continue when another key is pressed. HALT is part of the CONTINUE definition.

CONTINUE (n ---) When executed, simulation will run for n communication intervals. It may be used after a run is completed to continue executing the simulation.

CHECK (n1 ---) Execute n1 CHECK to check "wiring" accuracy. This word will list output node and corresponding state value for all nodes from 4 thru n1.

CHECK-RUN (n --) Unlike CHECK, n CHECK-RUN dynamically checks wiring accuracy, where n is the number of runs. Compare results to n RUN for equivalency. This word differs from RUN, in that it repeats execution of model with DT = 0, correcting for errors in positioning of blocks in the model definition. See EXAMPLES section for a description of its use.

RUN (n ---) Simulation will run for n communication intervals. Combines RESET, INITIALIZATION, and CONTINUE operations.

PRUN (n ---) PRUN is a version of RUN which is used if the xy-plotter is selected as the output mode, i.e., setting (typlot) or (xyplot) as outputs. It includes a 10 sec. delay, enabling the user to activate the plotter.

System Messages

MESS1 " SIMULATION RUN COMPLETED "

MESS2 " EXECUTION TERMINATED "

+ERR " OVER/UNDERFLOW "

0ERR " ZERO NOT ALLOWED "

-ERR " NEGATIVE NOT ALLOWED "

Note**:** The error messages execute the forth word reset, terminating the run. However, besides clearing the stack, reset also changes the base from decimal to hex, the default PropForth operating mode. Execute 'decimal' before making changes in parameters.

Loading PropFACS

Assuming you have PropForth Vs.4.0a running on your propeller, i.e., the core, the optional propForth word set and fs.f (eeprom filing system), copy the entire source file and paste into your running forth system on the pc terminal emulation. PropFACS will write into the next available location in your eeprom (at least 64K). Then, in forth, execute fsload propfacs.f .

SIMULATION EXAMPLES

Note: It is assumed you are running PropFORTH using a pc terminal emulation set at 57600 baud.

Example #0 ***Measuring Execution Time***

The words *pretime* and *postime* can be used to measure model execution times in microseconds. For example, execution of *pretime postime* in interpretation mode outputs 1360 usec. Add 13 to give the actual value. Executing test, where : test pretime postime ; outputs 0 (compilation mode).

As an example of propForth's performance, let us execute the following word:

```
: test pretime  10000 0 do loop  postime ;    result: 28 msec
```

Bean has reported an execution time of 230 msec for the FOR-NEXT loop in Embedded Basic.

Example #1 ***Simple Zero & First Order Differential Equations***

This first example uses the simplest differential equations to describe the steps in developing a PropFACS model. We will use the resulting models to compare the functioning of the three available integrators, demonstrate how multiple runs can be made, and make timing measurements.

The equations are: dx/dt = -k and dx/dt = -kx

Given a set of differential equations, the steps involved in constructing an analog model involve preparing a block diagram; identifying the connections (nodes), initial conditions, and parameters; and scaling the problem. In this example, each model has 2 blocks (an integrator and potentiometer), and one parameter set initially at 0.5. There is one initial condition: 1.0. Time is scaled in arbitrary units, let us say seconds. K is set to 5000 (0.5) and IC to Y0 (1.0).

Note: The minimum number of nodes required in a given model is:

4 reserved (time, +ref, 0ref & -ref) + number of all other blocks.

We do the following steps in order to define the model.

Step 1: Define models.

```
decimal                    \  Do this first.
6  NODES                    \  Number of nodes used must be specified.

MODEL:  ZORDER     1 Kx  4 +REF  POT    5 4 EULER  ;
MODEL:  FORDER1    1 Kx  4 5 POT     5 4  EULER ;
MODEL:  FORDER2     5 4 EULER   1 Kx  4 5 POT ;
MODEL:  FPRDER3     5 4 RK2   1 Kx 4 5 POT  ;
```

Step 2: Set IC, model parameters, and any required system parameter values.

```
 sc                          \  Do this to ensure a clear stack and avoid problems with ASSIGN-IC.
Y0 5 ASSIGN-IC               \  Set the initial condition value to node 5.
-5000 TO 1 Kx                \  Set model parameter value: negative for implicit inverter.
100 TO DT                    \  Set integrator time step (= 0.01 sec.).
10 TO COMINT                 \  Set communication interval (= 0.1 sec. ).
```

Step 3: Set output mode, input mode (if required) and integrator (if not specified in model).
Select model for simulation.

```
 sc
 5  4  OUT-NODES        \  output will be displayed in columns: time -  node 4 (dx/dt) -  node 5 (x)
OUTPUT-IS  (display)
SIMULATE  ZORDER
```

Step 4: Check model definition for consistency.

```
 6 CHECK          \ a static check listing node values at 0 time.

10 CHECK-RUN  \ a dynamic check -  should be equivalent to Step 5 results
```

Step 5: Run the simulation, in this case 10x, equivalent to 1 second in scaled time.

```
 10 RUN
```

Observations: Changing the integrator does not change the output, suggesting that the integrators are functioning correctly.

Changing the order of the blocks in the model definition (ZORDER) does not change the output.

Changing DT does not change the output.

Note that I have included two definitions of the first order model, in which the position of the POT block (same node numbers) is changed. If we do the following experiments:

```
1000 TO DT     1 TO COMINT
SIMULATE FORDER1
10 RUN                              \  #1
SIMULATE FORDER2
10 RUN                               \  #2
SIMULATE FORDER1
10  CHECK-RUN                        \  #3
```

Observations:	Output - #1	0	-0.500	1.000	Output - #2	0	-0.500	1.000
		0.1	-0.500	0.950		0.1	-0.475	0.950
		0.2	-0.475	0.902		0.2	-0.451	0.902
		0.3	-0.451	0.857		0.3	-0.428	0.857

Both runs #1 & #2 produce the same integral (x) output. However, the derivative values (= k*x) in run #1 are displaced one time step down. Run #3 produces the same output as RUN #2. If we used smaller DT values and larger communication intervals (COMINT), we might not notice the difference. These results show that correct ordering of blocks in a model definition is important. If we use CHECK-RUN, block ordering is not relevant, since this command runs the model twice, the second time setting DT to 0. Of course, execution time is increased in doing this. However, good programming practice requires that the blocks be placed in a logical order. Indiscriminate block placements in complex models would lead to confusion in examining the model definitions.

The general rule is that one should start with the integrator of highest order. All blocks whose output depends on integrators (e.g., FORDER) should be placed after the integrators. This is not the case in the ZORDER model, so the POT block works no matter where it is placed. CHECK-RUN can be used to establishing that wiring of a model is correct. We will demonstrate this in the next example.

Suppose we wish to determine the effect of increasing DT on the accuracy of the simulation results. Since PropFACS is extensible (presuming you know the elements of forth programming), we can define a word to accomplish this:

```
: DRUN   0 do  10 RUN   DT 2* TO DT  COMINT 2/ TO COMINT  loop ;
```

Note that COMINT is decreased so that we can observe output on the same time scale.

```
100 TO DT       40 TO COMINT
SIMULATE FORDER2
4 DRUN       \  will produce 4 successive sets of data at different DT-values.

SIMULATE FORDER3   4 DRUN
```

We can use this approach to vary model parameters:

```
: KRUN   0 do  20 RUN   1 Kx   1000 + TO 1 Kx  ;

100  TO DT   10 TO COMINT   -10000 TO 1 Kx
SIMULATE FORDER3
9 KRUN
```

Let us estimate the time required to run these simulations.

We define a new test version of RUN:

```
: TEST   TO RUNO   pretime   RESET   INITIALIZE   RUNO CONTINUE   MESS1
                   postime   ;

100 TO DT  10 TO COMINT   -5000 To 1 Kx
SIMULATE FORDER2
```

Output has been set to (display): 10 TEST - 160829 usec

Then SET-OUTPUT noop : 10 TEST - 73665 usec

SIMULATE FORDER3 10 TEST - 86128 usec

SET-OUTPUT (display) 10 TEST - 173293 usec

I leave it to the reader to reflect on the significance of these numbers.

Note: All further examples will be presented in PropFACS form, using the template we introduced in the first example. They were selected to show the scope of problems that PropFACS will handle, as well as illustrate the application of different blocks.

Example #2 ***The Van Der Pol Equation***

This is a nonlinear 2nd degree differential equation of the form:

$$dz^2/d^2t + k(1-z^2)dz/dt + z = 0$$

```
: (VDPOL) ;          \ It is useful to start with a  word that can be used to mark the location
                     \ of the model definitions in the dictionary.  Executing forget (VDPOL)
                     \ will clear the model definitions from the dictionary.
 decimal
 12 NODES
 MODEL:  VDPOL    4 11 INTGR  5 4 INTGR  6 5 SQR  7 6 –REF SUM  1 Kx 8 7 POT
       9 4 8 MULT   10 9 5 SUM  11 10 INV  ;
 sc
 200000  ASSIGN-IC
 5000 TO 1 Kx          .
 50 TO DT
 20 TO COMINT

INTGR-IS  EULER
sc
11  4  5  OUT-NODES          \ 11 - acceleration  4 - velocity    5  - z
OUTPUT-IS  (display)
SIMULATE VDPOL

11 CHECK                     \ Initial states of all blocks are listed.   Are they correct?

 My Results:
    4 – 0    5 – 200000   6 – 40000   7 –  -960000   8 –  -480000   9 – 0   10 – 200000   11 – -200000

 10 CHECK-RUN                \ Do both runs match?   If not, the block order has to be adjusted.
 10 RUN
```

Example 3 ***Generating Sin, Cos, Square and Triangular Waves***

Sin & Cos are generated by solving: $\mathbf{dz^2/d^2t = - \omega z}$. A BANG-BANG block coverts the Sin to square waves which are integrated to produce triangular waves. The frequency is determined by ω, the angular velocity.

```
: ( WAVGEN) ;
decimal
12 NODES
MODEL:  WAVGEN   4 8 INTGR   5 4 INV  1 Kx 6 5 POT   7 6 INTGR   1 Kx 8 7 POT
      9 4 BANG-BANG  2 Kx 10 9 POT  11 10 INTGR   ;
sc
Y0 7 ASSIGN-IC
5000 TO 1 Kx
2500 TO 2 Kx
100 TO DT    10 TO COMINT

INTGR-IS EULER
OUTPUT-IS (display)
11 10 7 4  OUT-NODES          \ 4 – Sin   7 – Cos    10 – Square   11 - Triangular

SIMULATE WAVGEN

11 CHECK

10 CHECK-RUN

50 RUN

INTGR-IS RK2                  \ repeat using RK2 instead of EULER integrator
50 RUN
```

Note: If you are interested in using this model to provide real-time signals and the RUN function & model are reduced to a minimum (e.g., no output), the maximum practical frequency is likely to be 10 cps.

Example 4 *Using Input Functions*

```
\ This word is defined to exercise the input functions.
: TEST6  0 do  4 INPUT   4 Y() . space  loop ;

        decimal
        7 NODES
        RESET
        INPUT-IS RND
        TEST6

        \ Repeat using STEP instead of RND
             500000 TO X0                    \ Set required parameters.
             500 TO PERIOD
             100 TO DT
             INPUT-IS STEP
             RESET
             TEST6a

        \ Repeat using Impulse
             500000 TO XO
             500 TO PERIOD
             100 TO DT
             INPUT-IS IMPULSE
             RESET
             TEST6a

        \ Repeat using RAMP
             0 TO XO
             10000 TO SLOPE
             100 TO DT
             INPUT-IS RAMP
             RESET
             TEST6a

        \ Repeat using SQWAVE
             500000 TO XO
             500 TO PERIOD
             -200000 TO XE
             100 TO DT
             INPUT-IS SQWAVE
             RESET
             TEST6a

        \ Repeat using TRIWAVE
             100000 TO +SLOPE
             200000 TO -SLOPE
             -500000 TO XE
             100 TO DT
             INPUT-IS TRIWAVE
             RESET
             TEST6a
```

Example 5 ***Transfer Functions***

This example illustrates an application in which input functions are used to characterize the behavior of linear systems defined as transfer functions in Laplace transform form. For example:

G(s) = x(s)/y(s) = (as + b)/(s2 + cs + d) or sx = -cx + ay – (1/s)(dx – by)

where a, b, c, and d are parameters; s represents the derivative; 1/s represents an integral; and all initial conditions are 0.

```
: (TRANSFUN) ;
decimal
12 NODES
MODEL: TRANSFUN   4 INPUT  1 Kx 5 4 POT   2 Kx  6 4 POT    7 13 INTGR   3 Kx 8 7 POT
       4 Kx 9 7 POT  10 6 9 SUM  11 10 INTGR   12 11 INV   13 12 5 8 SUM   ;

0 TO 1 Kx                \ a
-2000 TO 2 Kx            \ b
1000 TO 3 Kx             \ c
2000 TO 4 Kx             \ d
100000 TO SLOPE
0 TO XO
100 TO DT
50 TO COMINT

INTGR-IS RK2
OUT-PUT IS (display)

INPUT-IS RAMP
7  4  OUT-NODES

SIMULATE TRANSFUN
```

Example 6 ***Multiple Dosing (A problem in Pharmacokinetics)***

This example demonstrates an approach to determining optimum drug dosing, using a simulation model of drug absorption-elimination. The model is represented by the A → B → E were A is the amount of drug at the absorption site, B is the amount in the body, and E is the amount eliminated We assume first order kinetics: ka is the absorption rate constant and ke is the elimination rate constant.

-dA/dt = kaA dB/dt = kaA – keB dE/dt = keB where D (the dose) = A + B + C

This represents the "One-Body Compartmental " model.

```
: (1BODY) ;
        decimal

        \ This word is defined to enable multiple dosing.
        \ ( n1 n2  --- )    Y(n1) = Y(n2) at integer values of  0 Y()/PERIOD else Y(n1) = 0.

        :  DOSE    0 Y() 0= if  dup  0 TO  swap Y() then Y()  over Y() + TO swap Y()  ;

        13 NODES

        MODEL:  1BODY  4 IMPULSE   5  4 DOSE  5 6 RK2   1 Kx 6 5 POT  7 6 INV  8 7 10 SUM  9 8
                    RK2  2 Kx 10 9 POT  11 10 RK2  12 11 INV   ;

        750000 5 ASSIGN-IC  \ Set initial dose to 75 milligrams  ( 1000000 = 100 mg.)

        100 TO DT
        20 TO COMINT        \ Set for 0.2 hr communication interevals.

        -14000 TO 1 Kx      \ ka = 1.4 hrs-1
        -2300   TO 2 Kx     \  ke = 0.23 hrs-1  ( 3 hr Biological Half Life)

        60000 TO PERIOD     \  Impulse Input (6 hour period)
        750000 TO XO        \  Impulse amplitude = 75 mg.

        OUTPUT-IS (display)
        12 9 5  OUT-NODES \ Outputs at nodes 12  (Xe),  9 (Xb), and 5( Xa) should add up to the total
                          \ administered dose.
        SIMULATE 1BODY
```

Example 7 ***Correlation – Using the DELAY Block***

\ One use of the delay block is correlation, where a signal is correlated with a delayed signal.
\ The delay block may also be used as a buffer for temporary data storage, to be "played back"
\ at a later time in the same or another model.

```
: (CORRELATE) ;
decimal

12 NODES

\ These models will output an exponentially weighted average at node 10 ( the integrator). The
\ magnitude  of the output when it reaches a steady state is a measure of the degree of
\ correlation between  the two signals.

\  A cross-correlation of two different input functions.
MODEL:  CORRELATE   4 STEP   5 RND   6 4 DELAY  7 6 5 MULT  1 Kx 8 7 POT
        9 8 11 SUM  10  9 RK2   2 Kx 11 10 POT  ;

\ Autocorrelation of an input function.
MODEL: AUTOCORRELATE   4 INPUT  6 4 DELAY  7 6 4 MULT  1 Kx 8 7 POT  9 8 11 SUM
      10  9 RK2   2 Kx 11 10 POT  ;

INPUT-IS RND

100 vector D()    \ Define the delay line.  Number is made larger than needed to allow user
                  \  to vary actual number in the model

20 TO DN          \  Number of elements in delay line.
DELAY-IS D()      \ Activate the delay line.
CLEAR-DLY         \ Clear the delay line.

100 TO DT
10 TO COMINT
5000 TO 1 Kx      \ Time constant of integrator is 0.5 (scaled)
-5000 TO 2 Kx

4 6 10 OUT-NODES
OUTPUT-IS (display)
SIMULATE AUTOCORRELATE

10000 TO PERIOD
100000 TO XO
SIMULATE CORRELATE
```

EXAMPLE 8 ***Using the Function Generator***

The steps required to use the general purpose function generator FUNGEN follow.

Step 1: Outline the curve to be generated on a grid with vertical spacing set to DT and horizontal spacing to units of the output vector. Mark off the breakpoints (x-axis) at equal intervals set equal to FPERIOD, i.e., an integer number of DT-values. Mark the y-axis values which intersect the break-point lines.

Step 2: Calculate the slopes the straight lines connecting the intersection points which should approximate the desired curve, i.e., (Y(n) – Y(n-1)*scale/FPERIOD where scale is a number used to produce values between 0 - 10000. You should use the least number of breakpoints required to approximate your curve. Use scaled Y()-values and FPERIOD. In this example, the output is generated as a function of time.

Step 3: As part of the model definition, define a named (your choice) table of these slopes and assign the number of break points to the variable NBPT.

We will use an approximation of the bell-shaped curve in this example.

```
: (FUNGEN) ;
variable BELL 18 l, 70 l, 262 l, 730 l, 1520 l, 2200 l, 2200 l, 1000 l, 200 l, -200 l, 1000 l,
                    -2200 l, -2200 l, -1520 l, -730 l, -262 l, -70 l, -18 l,

SLOPES-IS BELL
18 TO NBPT

5 NODES
MODEL: TEST8 4 0 FUNGEN ;

\ FGAIN is used to adjust the output amplitude to the desired output range.
500 TO FPERIOD
2000 TO FGAIN
100 TO DT
5 TO COMINT

MODEL: TEST8 4 0 FUNGEN ;
4 OUT-NODES
OUTPUT-IS (display)
SIMULATE TEST8
```

Example 9 ***Using the Output Modes***

All examples have used the (display) output mode. There are two other output modes available: (mstore) which stores data in a matrix in hub ram and (typlot) or (xyplot). Version 1.0 of propFACS included words to store data in the eeprom. These words were eliminated from this version and the eeprom reserved for forth programs as well as FACS model definitions. Data storage in the eeprom was limited to only words and took much longer than (mstore) to store data. Analog computers did not produce lists of numbers but plots on strip-chart or flat-bed recorders. The plot routines in this version require special hardware and programming to produce plots on a TV-screen. While examples will be shown here, a complete description of the technique used to produce the plots will be detailed in an addendum to this document.

Using (mstore): The following example uses 1200 bytes of hub ram for storage.

```
100 3 matrix mata      \ Defines a matrix mata with 100 rows and 3 columns of longs.
SMAT-IS mata           \ Activates the system matrix SMAT.

OUTPUT-IS (mstore)
 7 5 4 OUT-NODES
SIMULATE <model>
100 RUN                \ Stores output from nodes 7, 5, and 4 is SMAT.

 100 3 MREAD           \ Displays unscaled data stored using SMAT.
```

Stored data is volatile and will be lost if the system is terminated.

Using (..plot): While it would be possible to use special PC programming to plot data, one of my objectives in this project was to eliminate the use of the PC as the display device. In order to do this, I determined that the most efficient approach was to use a two propeller system. The first propeller is assigned to run the forth base and the second to act as a terminal function with keyboard, TV and SD card connections. This was accomplished by merging OBC's terminal program with his modification of mpark's full screen editor. With this system, it became possible to run forth using the terminal screen, code words on the editor screen, load them into forth for testing, and saving the code on an SD card. I found that it was not possible to include a graphics screen without significantly reducing the plot resolution and using two TVs. My solution to this problem was two programs and 1 TV.

I used the Propeller Platform as the Forth Engine and a specially designed platform module containing the second propeller which controlled the keyboard, TV, and SD card. Simple serial communication between the two propellers was controlled by the terminal software. With the exception of two pins used for communication, all other pins on the two propellers are separately available. Two eeproms, which could be individually selected to allow either of two programs to run on the 2nd prop: the terminal-editor and plotting program. The latter was based on Eric Bells GreyTV object (130x192 resolution) which turned out to be ideally suited for this project. However, its needs a 24960 byte buffer which limits further options such as SD storage.

The PropFACS software includes special words to control the plotting mode which were described in the vocabulary section under output modes.

Using the VDPOL model as an example: (Figure 1)

```
SIMULATE  VDPOL
OUTPUT-IS (typlot)     \ Setup plotter to plot a selected state vector as a function of time.
5 TO 1 N()             \  Select z as output.
90 TO YSCALE           \  Adjust scale to allow for + & - values.  (180 is maximum for + values)
90 TO YLEVEL           \  Set y-axis zero-level to middle of screen.
0 TO WAIT              \  Delay between plotting points set to 0.
```

Setup the following command BUT DO NOT RUN.

```
120 PRUN               \  This runs the maximum sample size using a special plotting run routine.
```

At this point, select the 2nd eeprom which stores the plotting program Execute 120 PRUN, press the reset on the board and then F2 on the keyboard. The plot display will show on the TV and plotting will shortly begin.
You have 10 seconds to accomplish these tasks.

Note: Plotting keys are F1 – Clear Screen F2 - Set TYPLOT mode F3 – SET XYPLOT mode
F4 - Set TYPLOT2 mode F5 – Set STRIP CHART MODE F6 – Add GRID F7 – Exit Strip Chart.

The plotting routine is substituted for the terminal program in receiving data from the running simulation.
The data is transmitted by PRUN as hex bytes which the program transforms into integers which have values between -90 to 90 in this example. The integers identify the vertical locations on the screen which are 'turned on' at increasing horizontal positions (1 – 120). (Actually all values are increased by 5 to avoid edge problems.)
At this time I have no methods to store the images, other than to take snapshots of the screen.

Multiple Plotting: WAVGEN example showing combined plots of sin, cos, square and triangle waves.

```
SIMULATE WAVGEN
OUTPUT-IS (typlot2)
90 TO YSCALE
90 TO YLEVEL
\ The following commands are combined in a single word which is executed according to the above instructions.
: TEST
4 TO 1 N()  7 TO 2 N()   \ Plots both sin and cos.
120 PRUN
10 to 1 N()  11 TO 2 N()  \  Adds square and triangle waves.   Press F2 again to prepare screen for adding plots.
120 PRUN  ;
\ Execute TEST     (Figure 2)
```

How to use (xyplot): WAVGEN example - ploting cos(x) vs sin(x). (Figure 3)

```
SIMULATE WAVGEN
OUTPUT-IS (xyplot)
4 TO 1 N()             \ Select Sin as x-axis.
7 TO 2 N()             \ Select Cos as y-axis
60 TO XSCALE           \ Set output for four quadrants
60 TO XLEVEL
90 TO YSCALE
90 TO YLEVEL
100 TO WAIT            \ Set delay to 100 msec to ensure synchronization between plotter and data generation.
```

120 PRUN \ Follow the same procedure as described for (typlot) except press F3 to display
\ 4-quadrant display.

Note: COMINT and DT determine the range of data plotted in all plotting modes. ..SCALE and ..LEVEL determine the y-axis positioning and amplitude of the plotted data.

Plotting Results

Note: The following examples show ty-plotting of single & two variables, the addition of a grid to a plot and an xy-plot.

Figure 1 – Example 2 Van Der Pol Eq.

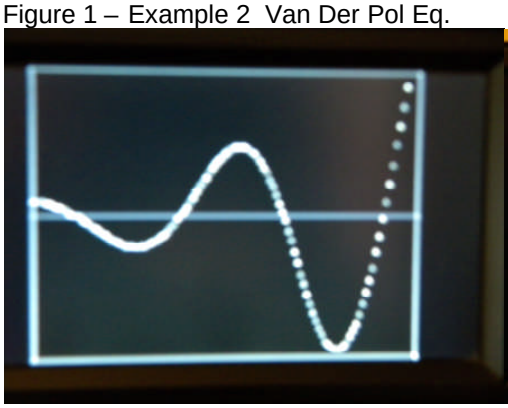


Figure 2 – Eample 3 Sin/Cos Waves

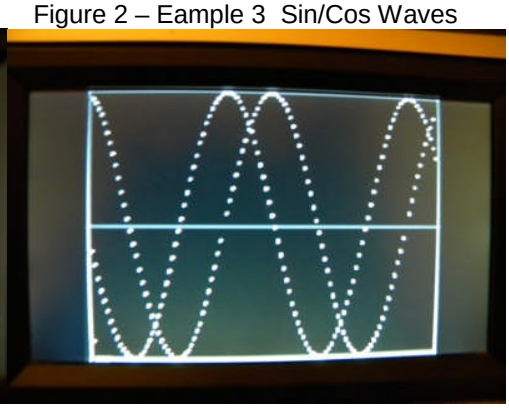


Figure 3 – Example 3 xy-plot Sin vs Cos

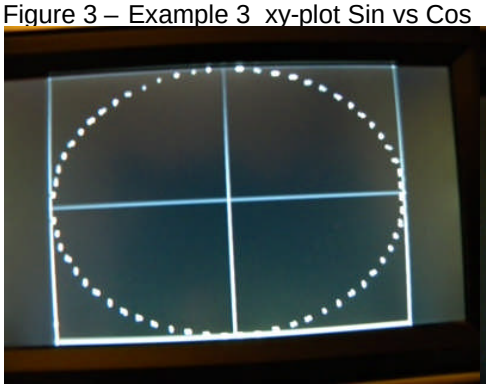


Figure 4 – Example 5 Transfer Function

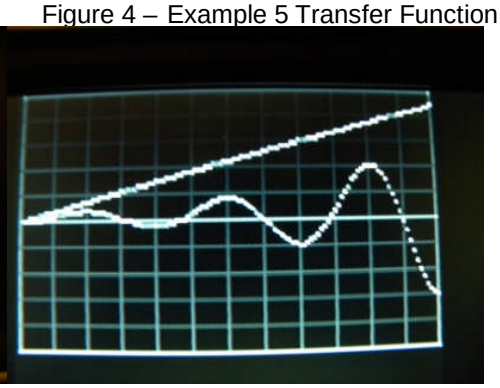


Figure 5 – Example 6 Multiple Dosing

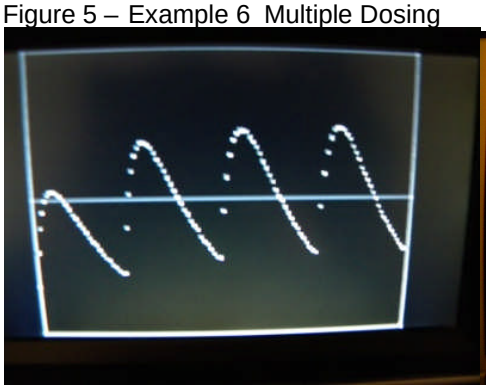


Figure 6 – Example 7 Autocorrelation

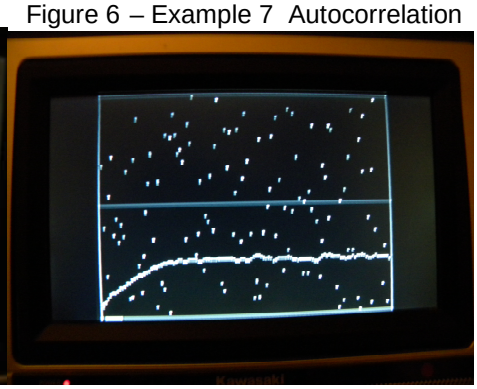
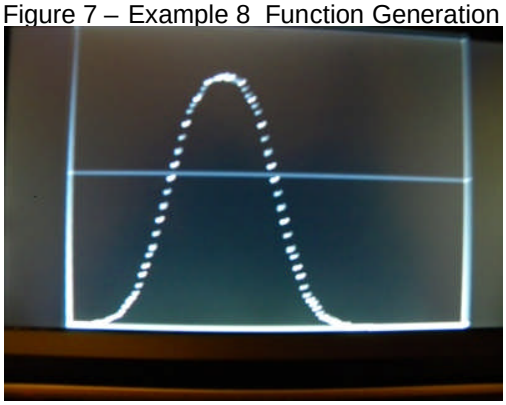


Figure 7 – Example 8 Function Generation



COMMENTS

I hope that the power of FORTH is evident in this application. The ease of extending the language and, if necessary, reducing the source (e.g., eliminating unneeded blocks not required by an application) to free additional memory are important advantages.

The principal limitation of this PropFACS version is that no methods are provided to save simulation results as data or plots. While this could be done on a PC, for example,using ViewPort to generate plots. However, I feel that external memory which can be accessed by both the Forth engine and the terminal/editor – graphics programs is a possible answer. This memory can act as the buffers for these programs, allowing room for the SD-object, so plots (and data) can be saved on the SD card.

SOURCES

"Simulation of Mechanical Systems: An Introduction"
Joseph Edward Shigley, McGraw-Hill, 1967

"Random-Process Simulation and Measurements"
Granino A. Korn, McGraw-Hill, 1966

"Handbook of Analog Computation"
Electronic Associates Inc., 1964

EAI Applications Reference Library
Various, 1964-1969