ORDERING AND INITIALIZATION ALGORITHMS FOR CONTROL SYSTEM MODELS

D.G. Chapman, Member, A.J. Heise
Manitoba HVDC Research Centre
Winnipeg, Manitoba

F.L. Alvarado, Senior Member
University of Wisconsin
Madison, Wisconsin

N.J. Balu, Senior Member
EPRI
Palo Alto, California

Abstract - A recently developed multiterminal HVdc stability program utilizes a control modeling approach which defines all HVdc control models from user supplied data. The models are built up through combination of basic control system building blocks such as transfer functions, limiters, adders and transducers.

Successful implementation of this modeling approach required the development of algorithms to place the control system building blocks into the correct solution order and to then initialize the control model to a predetermined steady state solution.

The algorithms developed are quite general in applications. They are examined and demonstrated through the use of control system examples.

INTRODUCTION

In a recent research project a building block approach was taken in the modeling of HVdc systems for load flow and stability studies. In the development of the stability program a capability for description of the dc system control models through user provided data was established.

User described models can be provided in many ways, ranging from model descriptions in FORTRAN code which must be linked to the program (with parameters supplied separately) through to models which have both the structure and the model parameters contained in the input data. In all cases there are requirements to ensure that the models are initialized correctly and that they will execute correctly.

Early in the program design process a decision was made to use a sequential solution for the control models, based on the desire to create highly modular code (for portability to other programs and other dc solutions) and on the results reported in [1,2,3].

The algorithms described here, for ordering and initialization of general control system models were developed to permit easy specification of these models through input data while ensuring correct initialization.

87 SM 550-7 A paper recommended and approved by the IEEE Transmission and Distribution Committee of the IEEE Power Engineering Society for presentation at the IEEE/PES 1987 Summer Meeting, San Francisco, California, July 12 - 17, 1987. Manuscript submitted August 27, 1986; made available for printing April 17, 1987.

The general description of the programs and of the modeling system is given in [4]. The whole system of definition of control system models, by the program users, from input data alone, is known as User Defined Control or UDC. The UDC concept arose from the TACS concept of the Electromagnetic Transients Program [5], in that the control models were to be defined by input data alone, with no predefined models available.

In the UDC concept the output from a block may be used as an input to any number of other blocks. These inputs can be in the control signal path, such as the input to an integrator, but they can also be limits on the control signal or inputs to logical functions affecting the signal. In general, only transfer function parameters and non-linear curves are fixed real numbers, while all other inputs may be block outputs. To obtain a dynamic solution of the whole system it is necessary to ensure that, as far as possible, all input quantities to a given block have been calculated before the block output is calculated. Thus, a solution ordering algorithm must consider ordering of all block inputs.

The ordering algorithm incorporated into the UDC concept is quite powerful, being able to handle systems involving feedback loops in any arbitrary configuration.

Once the blocks are placed into solution order the whole system must be initialized to steady-state conditions. The UDC concept utilizes initialization algorithms which permit initialization of arbitrary control systems to steady state values with a minimum of user specification.

The range of building blocks is given in Appendix A. It can be seen that the blocks available range from simple processing blocks, such as single input blocks, through blocks to extract information from the ac and dc solutions, through more complex blocks, some involving logic, to the interface block with the HVdc converters.

INPUT PROCESSING

Considering the large number of possible building blocks and the wide range of possible input data requirements, one of the first decisions made in the definition stage was to provide a "Free Format" input capability. In this form of input each building block has a predefined set of required and optional input items and these data must be provided in the correct order. However, as long as the elements are separated by commas, and the whole block description terminated by a colon, the block description may be laid out in whatever fashion the user considers appropriate. Comments may be imbedded in the model description by surrounding them by asterisks. Inputs to a block may be specified as real numbers or as the name of another

0885-8977/88/0700-1189$01.00©1988 IEEE

block, implying that the output value of that block will be used during the calculation of the system response.

The UDC code first reads the input data, one block at a time. During this reading process the program rejects blocks which are unrecognizable, which are incompletely defined and which have bad data.

The block description data is stored in input order in a large linked list, for later processing by the ordering and initialization algorithms.

ORDERING

Once the input data have been completely read it is necessary to determine the block solution order, for use during the dynamic simulation. The solution of the control model is sequential rather than simultaneous, so an algorithm to provide the solution order is essential.

Any block in the control system has one output value, referenced in the system description by the block name, calculated at each solution time step. With the exception of the block which controls a dc converter, no block has more than one output. This one output may be used as an input to an unlimited number of other blocks. The number of inputs used by any one block depends on the block type and subtype and can range from one upwards, without limit.

Ordering starts with SOURCE, FROMAC and FROMDC blocks, blocks which have no inputs from other blocks but which produce an output based on data obtained from the ac or dc system or calculated from basic parameters. These blocks are assigned a solution order in the same sequence as the input order of the block.

Next, the ordering algorithm orders blocks which have all inputs ordered. As mentioned above, these inputs may be in the signal path or may be used for limits or in logic functions. The list of unordered blocks is repeatedly scanned until either all blocks are ordered or no further blocks can be ordered because one or more of the blocks representing inputs to each of the remaining unordered blocks have not been ordered.

In the first case, the process is complete and the program proceeds to initialization. In the second case, either one or more blocks which are inputs to some unordered block have not been described (an error), or a loop exists.

For the purposes of this paper, a loop is defined as a collection of blocks where the input to at least one of the blocks depends, ultimately, upon the output of the same block. A loop does not exist if the only feedback is through the dc network.

The program must recognize that a loop exists and must then order the blocks within the loop so that, during the dynamic solution, a one time step delay occurs at one of the inputs to the loop (since the solution is sequential and all input quantities to the first block in the loop cannot be calculated before the block is evaluated). Also, for initialization purposes, the identities of blocks making up the loop must be recorded. Initialization of a loop implies that the inputs to and output from a loop must be consistent, so that the loop is in steady state. As will be described below, an iterative process is required to arrive at this steady state. This process requires that the blocks making up the loop be evaluated repeatedly.

LOOP FINDING

Since the UDC program allows for the description of the control system to be input with little regard to ordering of the blocks, the loop finding algorithm must be capable of working forward through the list of blocks creating a new list of the blocks contained within the loop itself. The algorithm is an application of graph theory [6].

The loop tracing process begins when the ordering algorithm stalls, with unordered blocks remaining which cannot be ordered. Firstly, an input/output linked list is established for all blocks. This is a set of linked lists; one storing the indentity of each input block for a given block and the other storing the identities of blocks to which the given block provides an input. Then, to speed up the loop tracing process, the lists are reduced in size by removing selected input and output links.

For example, it is evident that any block making up the loop must have at least one input link and one output link. Therefore, any blocks without output links are removed from the linked lists. This process of removal is carried out repeatedly until only blocks with output links remain.

Next, the tracing algorithm attempts to find a loop starting at the first block in the loop. Since, at this time, the first block is unknown, the algorithm establishes a "leading" block, a block which has been ordered but has at least one unordered block on its output list. This unordered block is identified as the "first" block in the loop. Therefore, any block which has been ordered and which has an output list which contains only ordered blocks is removed from the input/output lists.

Now, the tracing process, through the remaining blocks, begins at a leading block (there may be several), where the leading block is chosen in input order. The first block on the output list of the leading block is taken as the first block in the loop. Following the top connection on the input/output list of each block, as it is encountered, the algorithm searches for a link back to the first block. As each block is encountered all links to previously encountered blocks are also checked.

If a link back to the first block is established, a loop has been found. The first block in the loop is given a solution order, the complete input/output linked lists recreated, and processing continues until either another loop is encountered or all blocks are ordered.

If an inner loop exists, this loop might be found before the outer loop. This inner loop is ignored by removing the feedback pointer from the input/output lists and continuing the tracing.

If no loop is found from the current first block, the next block on the leading block output list is chosen as the first block, and the tracing algorithm continues.

When the output list for a leading block has been exhausted, unsuccessfully, the algorithm chooses the next leading block, from the input order, and begins the tracing process again. When the leading block list is exhausted, each block in input order is then tested as a first block. If this process is not successful then there is an error in the input data.

To ensure that the one time step delay is inserted

into the feedback path of a loop, rather than the forward path, it is necessary to ensure that the correct leading block occurs, in input order, before any other possible leading block. In some cases determination of leading blocks is a subtle process, because inputs to logic and to limits within the loop must be identified and considered.

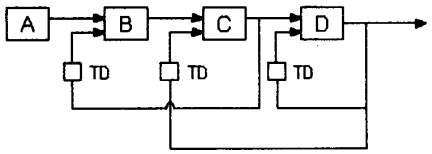


Figure 1: Example of Intersecting Loops

The algorithm has been tested extensively on systems with multiple loops, both nested and intersecting, and has consistently determined the correct relationships for the blocks in the loops. Figures 1 and 2 show two examples of complex systems with an indication of where the loops are broken by the one time step delay (the TD blocks). Note that, in the second case, there are only two time delays required although the system appears to have three loops.

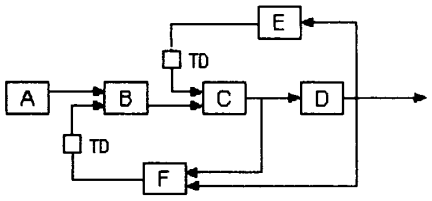


Figure 2: Example of Multiple Loops

The loop tracing algorithm is provided, in pseudo code, below.

Loop Tracing Algorithm

Construct input/output lists for each block

Reduce System

Trace the block connections to find a connection back to the "first" block by:

UNTIL loop found or error detected DO

Designate the "first" block as the "previous" block.

Search for a "leading" block.

Trace the connection from the "previous" block to its output block.

IF no connection exists THEN

Designate the next "first" block as the next output block from the "leading" block.

IF no next "first" block exists THEN

Reconstruct the input/output lists for each block.

Reduce the UDC system.

Search for a "leading" block.

END IF

ELSE IF the new output block is the "first" block THEN

The proper loop has been found - EXIT routine.

ELSE IF an inner loop has been found THEN

Remove feedback pointer for inner loop.

ELSE

Record the new output block as the next block in the loop and designate the output block as the "previous" block.

END IF

END IF

END DO

To Reduce System

UNTIL no further reduction is possible DO

Remove blocks with no outputs.

Remove blocks that have all output blocks ordered

END DO

To Search for "Leading" Block

Search for a "leading" block in the reduced system by:

IF a block exists which has outputs and no input THEN

Designate the first block which has outputs but no input as the "leading" block and designate the first output block as the "first" block.

ELSE if an unordered block exists THEN

Designate the first unordered block as the "first" block.

ELSE

An error in formulation has occured - STOP.

END IF

Figure 3 shows, graphically, the process of loop tracing for a simple example.

The example system, in 3a, consists of blocks A through G containing two loops. Blocks A, B and D can be ordered before the process requires loop detection (3b). An asterisk in a block indicates an ordered block. Shaded blocks identify blocks to be removed from the search.

In 3c, Blocks A, F and G are identified as being blocks that can be removed from the loop search, leaving the system in 3d. The loop tracing algorithm detects the inner loop on E(3e) and remove it,(3f) then identifies the correct loop involving C and E(3g), so C can be ordered.(3h) The feedback pointer for the detected loop is removed and the full system again reduced to contain only blocks C, D, E(3i). The loop around E is detected(3j), E is ordered, the feedback pointers for both loops removed and blocks F and G then ordered(3k), completing the process.

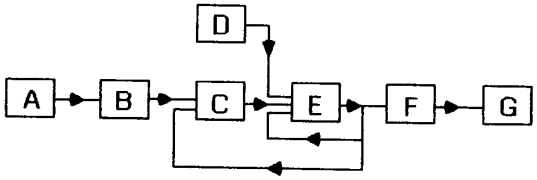


Figure 3a: Test System

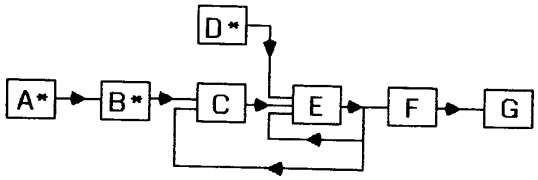


Figure 3b: Blocks Ordered Before Loop Encountered

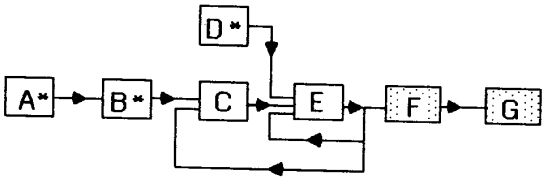


Figure 3c: Reduce System

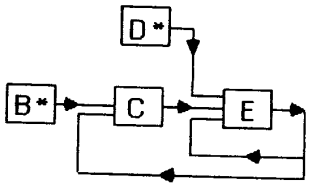


Figure 3d: Reduced System

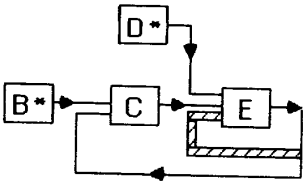


Figure 3e: Identify Inner Loop

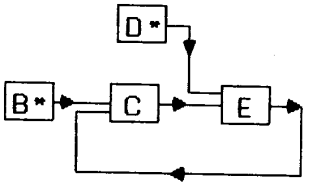


Figure 3f: Delete Inner Loop

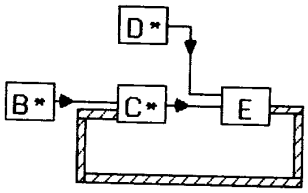


Figure 3g: Identify Required Loop

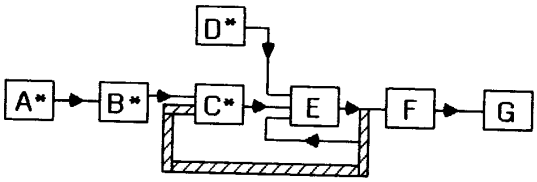


Figure 3h: Order C, Remove Outer Loop

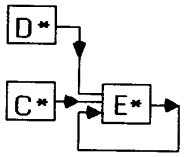


Figure 3i: Reduced System Before Finding Inner Loop

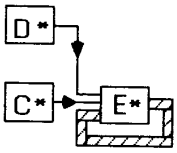


Figure 3j: Find Inner Loop, Order E

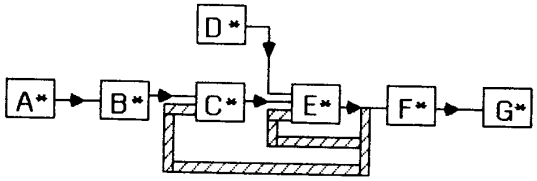


Figure 3k: Remove Loop Feedback Pointer, F, G Ordered

INITIALIZATION

There are three ways to initialize a control model without requiring specific code to be written to calculate an initial condition. Firstly, from a flat solution, the control system might be run dynamically, in hopes that the system would come to a steady state. This process requires the correct feedbacks from the controlled system, so that, for HVdc systems, a full stability solution would be required, with no guarantee that a steady state solution of the control system is possible.

Secondly, the initial values for reference signals and for integrator outputs might be calculated

algebraically by setting up separate initialization logic and the control model would incorporate these calculations. The UDC system permits this type of initialization but this process results in larger models, because of the extra calculations required by the extra blocks, and non-linearities are not easily handled.

Thirdly, the program can, itself, calculate reference values and initial conditions based on the existing structure. Here, the new algorithm provides some powerful methods to ensure good initialization.

If it is assumed that all signals in the control system are constant in steady state then it is possible to initialize a system by working forward from known values towards the control system output. Generally, the system outputs are quantities which must agree with the steady state solution of the overall ac and dc systems. Often internal reference values must be adjusted to ensure that the output quantities are correct. Also, integrators appear as open circuits during initialization since the input is usually zero while the output is non-zero. The new algorithm provides a means for automatic adjustment of both reference values and integrator initial values to ensure that desired values are obtained at user chosen points in the control system. This adjustment is possible using the idea of a "subsystem".

In the control model it is possible to extract a self contained part of the system where the first block of this subsystem must be adjusted to give a desired value at the last block of the subsystem. All blocks evaluated from the first to the last make up the subsystem.

A subsystem is defined by input data, using a category of blocks known as SUBSYS blocks. Since initialization of a subsystem may require some iteration, as do the loops previously discussed, the program attempts to treat subsystems as special cases of loops. During the ordering process, the first block in the subsystem is analogous to the first block of the loop, the last block of the subsystem to the feedback block in the loop. A ficticious loop is created by providing a temporary feedback pointer from the last block to the first block, if a leading block exists, or to the second block of the subsystem, if no leading block exists.

During the ordering process the loop tracing algorithm must find a loop corresponding to the previously defined subsystem. Here, ordering of the leading and first blocks in the input data is extremely important, as the whole ordering process will fail if the identical loop is not found. Future work on this algorithm will concentrate on improved subsystem loop detection.

It should be noted that the loop tracing algorithm, as well as the loop initialization algorithm, can handle subsystems within loops and loops within subsystems.

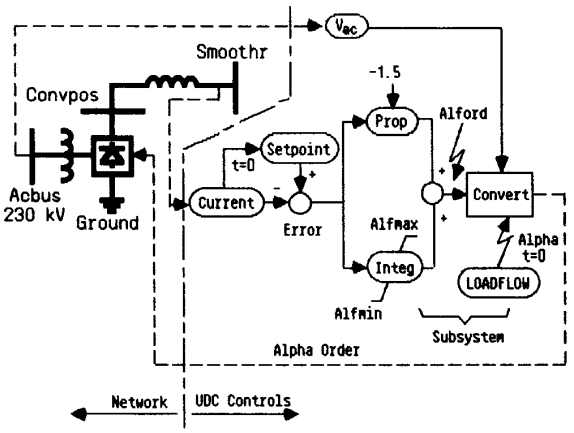
To initialize a loop the blocks making up the loop must be evaluated iteratively until the loop output is consistent with the feedback value required to give this output.

The process begins by providing the first block in the loop with the uninitialized output from the last block in the loop (normally 0.0). The blocks making up the loop are then evaluated and the newly derived value from the last block is compared to the previous value. If the two values match, within tolerance, the loop is considered to be initialized and the initialization process proceeds. If the two quantities do not match, the loop is provided with a new input equal to the average of the previous input and the last output and the blocks in the loop are evaluated again.

Unless an intermediate block in the loop provides some constraint within the loop the loop will be solved, to tolerance, relatively quickly. If convergence fails it is likely that the control system is poorly defined and the system cannot possibly operate at the stated steady state conditions with that particular control system.

A similar but more complex iterative process is used to initialize subsystems. This process can be demonstrated with reference to a simple example of a current controller, shown in Figure 4.

The proportional plus integral controller in this model must be initialized so that the HVdc converter has a firing angle equal to the load flow value. Also, the current reference must be set to a load flow value. The UDC data which implement this model are also given in Figure 4.




```
Current, FROMDC, ILINE, P, Convpos, Smoothr, 1 :
Setpoint, SETUP, VALUE, P, Current :
Error, BINARY, MINUS, P, Setpoint, Current :
* Here is the start of a subsystem *
Integ, SUBSYS, LINTEG, P, 15.0, Error, -1.0,
    Alfmin, Alfmax, 0.1, 0.001, Convert, 5.0, 30. :
Prop, BINARY, MULT, P, Error, -1.5 :
Alford, BINARY, ADD, P, Prop, Integ :
Convert, INTERF, P, Convpos, Ground, Acbus, 230.,1,
COMFAIL, 0.0, Vac, LT, 0.8,
ALPHA, Alford,
INITIAL, Alpha :
Vac, FROMAC, VMAG, P, Acbus, 230. :
Alfmin, SETUP, VALUE, N, 5.0 :
Alfmax, SETUP, VALUE, N, 25.0 :
ENDUDC :
```

Figure 4: Example UDC System

In the figure the block type and subtype are in upper case while block names and the names of points in

the network are capitalized.

In this example the current reference is developed using the second initialization method. It is known that the reference must equal the initial value, so block Setpoint was established to extract the initial current from the load flow, using a FROMDC block, and retaining this value for all time (the nature of a SETUP block).

The initial value of the integrator, Integ, must be determined to ensure that the dc converter is controlled by Convert to the load flow firing angle Alpha. To do this Integ is defined as a SUBSYS block, the initial block in a subsystem ending at the INTERFace block Convert. The output of block Integ will then be varied, by the program, until either the desired value is obtained at the input to Convert or there is an convergence failure.

The detailed definition of the Integ block is:

Integ, SUBSYS, LINTEG, Print, Initial Value, Input, Gain, Solution Minimum, Solution Maximum, Increment, Tolerance, End Block, Convergence Minimum, Convergence Maximum:

while the definition of the End Block Convert is:

Convert, INTERF, Print, Converter Name,
COMFAIL, logic for commutation failure,
ALPHA, Input for alpha order,
INITIAL, ALPHA:

The program creates a fictitious input to Integ, creating a loop involving all blocks between the SUBSYS block and the End Block. The loop finding algorithm is then used to record all blocks making up the fictitious loop. Here ordering of the input data is important because the algorithm is looking not for a general loop but for a specific loop starting from the SUBSYS block and ending with the End Block. Thus, at the present time, it is necessary to ensure that, in the input data, all blocks defining inputs to the Subsystem (such as limits, logic, and other signals) be defined after the SUBSYS block.

Once the blocks making up the subsystem are known and recorded, the system can be initialized. In this case, the "loop" is solved using the Initial Value from the block description, and the output is recorded. Next, the Initial Value plus the Increment is used as a second input and a second output calculated. A linear interpolation/extrapolation scheme is then used to calculate a new input in an attempt to reach the desired value at the End Block (the load flow value of Alpha).

The iteration process continues until either the desired value is obtained, to within the Tolerance value (successful completion) or the same value is obtained at the output for two different, successive inputs (internal limit reached), or the iteration limit reached (failure of the algorithm), or the Convergence Limits reached (likely a poor system description).

The advantage of the subsystem approach to initialization is that it is possible, in a minimum number of blocks, to write a general model capable of being initialized to the load flow solution.

In the extensive testing to date the solution initialization process has worked extremely well, handling non-linearities such as inverse cosine functions without difficulty.

Other Refinements

Although no problems have been encountered in initialization in the testing to date, it was foreseen that some additional initialization facilities might be needed. For this reason a block has been provided within the UDC set which permits an initial value to be given to any desired block. This value is provided once during the initialization step and could be overwritten during iteration of loops and subsystems, but it might, in complex situations, assist the loop iteration process through provision of a feedback value closer to the expected steady state value than the default value of zero.

FUTURE WORK

The present UDC system has been configured for HVdc applications. Extensions are possible into interface blocks specific to excitation and governing systems or other machine controls. These extensions would be simple to provide.

More complex extensions are also being considered. All the current UDC blocks are single output blocks. Multi-output block capabilities would permit the modeling of, for example, state space controllers or "computer based" controllers within one block. For the latter case a link to a user supplied FORTRAN routine is being investigated.

While the UDC routines are presently implemented in a stability program the whole UDC solution package is highly modular and has been used to build a "stand alone" control system simulation package (configured to permit investigation of HVdc controls without the cost of a full stability run). The UDC routine could also be applied conveniently and simply to other power system analysis programs.

CONCLUSIONS

This paper has described a powerful set of algorithms that permitted the establishment of a new control system modeling program. This program permits arbitrary control systems to be described completely by input data. Ordering, for a sequential solution of the system, and initialization, for steady state operation, are handled by the program efficiently and effectively with minimal information required to ensure initialization. While the code is presently designed for HVdc control modeling it is applicable to other types of models.

ACKNOWLEDGEMENTS

The work described in the paper was wholly supported by EPRI under RP1964-2. The authors wish to acknowledge assistance provided in this project by the advisors, E. Gulachenski, G. Rogers, R. Lee, D. Smith, T. Hatcher, J. Luini, J. Reeve and the assistance at the contractors of D. Allan, C. Hasselfield, B. Davies.

REFERENCES

1) B.Stott, "Power System Dynamic Response Calculations", Proceedings of the IEEE, Vol. 67, No. 2, February 1979, pp 219-241.

2) "Power System Dynamic Analysis - Phase 1", EPRI Final Report EL484, 1977.

3) J.Arrillaga, C.P.Arnold and B.J.Harker, Computer Modeling of Electrical Power Systems, John Wiley & Sons, 1983.

4) "Methodology for Integration of Multiterminal HVDC Links in Large AC Systems - Phase II: Advanced Concepts", EPRI Final Report EL4365, 1986.

5) L. Dube, H.W. Dommel, "Simulation of Control Systems in an Electromagnetic Transients Program with TACS", Proceedings of the 1977 Power Industry Computer Application Conference, IEEE 77-CH1131-2-PWR, pp 266 - 271.

6) M.J. Augenstein, A.M. Tenenbaum "Data Structures and PL/1 Programming", Prentice-Hall, Inc., 1979.

APPENDIX A

THE UDC BLOCKS

The UDC blocks are identified by type, the broad classifications given below, and by subtype, the identifier which specifies the exact block to be used. The subtypes available are not provided here but are described in detail in [4].

Information source blocks

There are three types of sources available. The first, type FROMAC, provides values such as ac bus voltage, ac line current and frequency to the control model. The second, type FROMDC, provides dc network and converter quantities such as voltages, firing angles, power flows and converter operating modes to the control system. The third, type SOURCE, provides sinusoidal, ramp, square wave functions to the control system.

Information processing blocks

Here there are many types of blocks. These include Type UNARY, single input blocks providing inversion, table lookup, trigonometric functions and algebraic functions.

Type BINARY, two input blocks providing addition, subtraction, multiplication and division.

Type MULTIP, multiple input blocks providing addition, addition with scaling, minimum and maximum calculation.

Type LIMIT, providing lead-lag functions, with and without limits, and simple limits.

Type SBLOCK, providing unlimited and limited transfer functions of up to 10th order.

Type LOGIC, providing logic functions including AND, OR, NOR, Flip-flops, logic function evaluation, logic controlled switch and logic controlled timer.

Type OTHER, a general set of blocks providing such things as deadbands, hysteresis, tap changers, line parameter modification.

Type SETUP, providing set points, integrators and limited integrators. The set point subtypes keep fixed output values after initialization.

Type SUBSYS, similar to the SETUP blocks but with the ability to be to have the initial output value calculated by the program. The BREAK subtype provides a means for specifying a desired value at some point on the model.

Control Output Interface

Lastly, the control model links to the HVdc solution through an interface block, of type INTERF, which controls the mode of operation of a specific converter. Each of the input variables may take control, dependent upon the evaluation of the logic function associated with the input variable. The first true function takes control. Also, the initial value may be specified explicitly or taken from the load flow solution.