



The paper is organized as follows: Section II provides the background containing brief descriptions of SGPC controller and SCADE specification language. Section III describes formal model development. Validation of SCADE model using simulation and MTC is described in section IV. Section V describes formal verification of SGPCS model by specifying and proving safety properties. Section VI describes code generation and integration and finally section VII gives the brief report of experience gained and future work.

## II. BACKGROUND

### A. Process Control Systems in Nuclear Reactor

The objective of SGPCS (Steam Generator Pressure Control System) [7, 8] is to regulate the steam generator (SG) pressure by controlling the total amount of steam drawn from the SG. The heat input to the SG through Primary Heat Transport system is a function of reactor power, which is controlled independently. When the energy in the steam drawn from SG equals the energy released from reactor fuel into the primary coolant, the primary fluid temperature and steam temperature (and hence the pressure) are maintained at steady values. The steam pressure regulation is achieved by controlling the valves in various steam lines taking steam to either the turbine (turbine governor valves TGVs) or dumping into condenser (condenser steam dump valves CSDVs) or discharging to atmosphere (atmospheric steam discharge valves ASDVs). So the objective of the SGPC system is to generate the control signals for valves specified above. The control signal is generated by applying PID control algorithm to the error signal calculated based on the difference between measured SG pressure and operational pressure setpoint (OPSP). The operational pressure setpoint is function of reactor power and is calculated using no load setpoint (NLSP) and  $\Delta T$ , the measured differential temperature across SG, which is a measure of reactor power. The main control signal is generated by applying PID control algorithm to the error, which is then split into three ranges to derive control signals for the three types of actuating valves i.e. TGVs, CSDVs and ASDVs. Since these controllers perform critical functions, the input measurements are triplicated /quadruplicated to provide redundancy.

### B. SCADE Specification Language

The SCADE [4, 5] modeling notation supports Dataflow Equation and Safe State Machine for modeling the system. A dataflow equation describes the logical relationship between input and output variables. Safe state machine describes the control flow in the system in terms of state transitions. Set of dataflow equations are evaluated in parallel unless there is a data dependency between them. Internally the SCADE models are represented in SCADE textual language. SCADE textual language is based on the dataflow formalism, which is similar to the Synchronous dataflow language Lustre [3].

Synchronous languages [2] are based on synchrony hypothesis, which assumes that the program is able to react to an external event, before any further event occurs. In dataflow language a program is described as network of operators

transforming flows of data. SCADE model is graphically represented as network of operators. For example a basic counter that counts up from 0 can be expressed in SCADE textual language as

$$N = 0 \rightarrow \text{pre}(N) + 1;$$

In above equation N is a variable that stores the value of counter at a given instance of time. The operator  $\rightarrow$  is called the “followed by” operator and it is used for initialization. Operator “pre” is used to refer to the value of variable at previous instant of time. Therefore the above equation specifies that the initial value of variable N is 0 and thereafter its value is one more than the previous value.

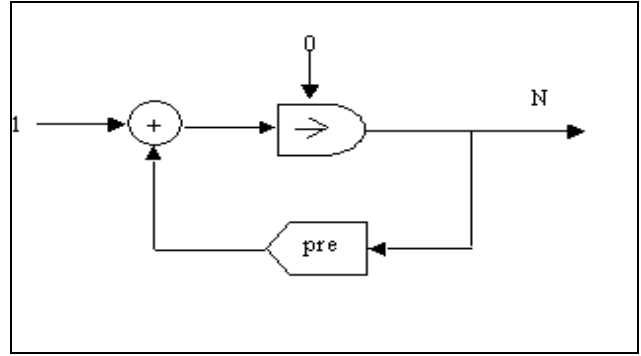


Fig. 1. SCADE Operator Diagram

SCADE model corresponding to above equation is shown in Fig.1. A network of operators in SCADE can be encapsulated as a new reusable operator that is called a node or SCADE operator. The requirements of the system to be implemented are mapped in the form of network of SCADE operators, where each operator provides a specific functionality independently or by interaction with the other SCADE operators.

The set of dataflow equations contained in a SCADE operator are all evaluated simultaneously when that operator is instantiated.

## III. DEVELOPMENT OF THE MODEL FOR SGPCS IN SCADE

The processing logic of the SGPCS can be categorized as Signal Processing Logic and Control Logic

This is shown schematically in Fig. 2. For all the important processes parameters, triplicate or quadruplicate measurements are provided. Signal processing logic performs the validation of input signals and generation of “representative” (good) signal for each input parameter from multiple measurements rejecting where necessary any faulty measurements, to ensure reliable operation even in case of some sensor failures. Control logic involves the implementation of process control algorithm for SGPC system. The “representative” signals from signal processing logic are the inputs to the control logic and it generates the required control signal for Turbine Governor Valves (TGVs), Condenser Steam Dump Valves (CSDVs) and Atmospheric Steam Discharge Valves (ASDVs). The formal model has been developed in SCADE only for the Application Control

Logic since this component could be cleanly substituted in old implementation without any other modification. The equations expressing relationship between output and input variables were grouped and implemented graphically in SCADE as different nodes, according to the functionality they provide. In this way the SCADE model is modularized and these modules can be reused in other implementation. The representative signals for all the discrete and analog inputs that are used in the model are read from the interface of the signal processing logic already implemented in the existing code.

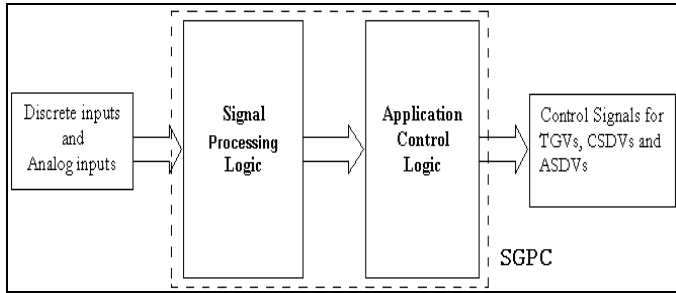


Fig. 2. SGPCS Inputs and Outputs

Control Logic is divided into nodes. The nodes, which generate actual control signals are Calc\_NLSP, Calc\_Control\_Signal and Calc\_ASDV\_Control\_Signal. The function of these nodes is to generate “No Load Set Point (NLSP)”, “Control Signal for TGVs and CSDVs” and “Control signal for ASDVs” respectively as shown in Fig.3.

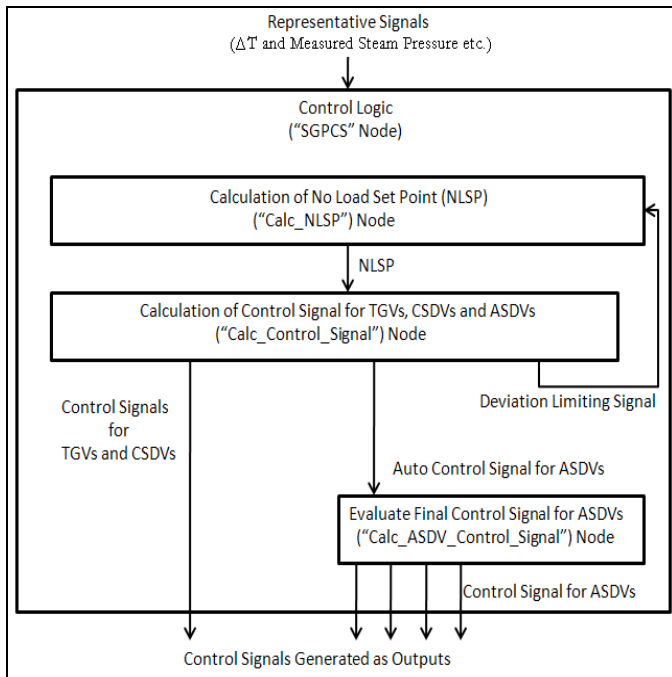


Fig. 3. SCADE Nodes and Data Flow

Calc\_Control\_Signal is the most complicated node because this node implements the algorithm for computation of

pressure set point (OPSP) for SG and calculates the control signal by applying PID control algorithm to the error signal. The brief overview of node Calc\_Control\_Signal is depicted in Fig.4.

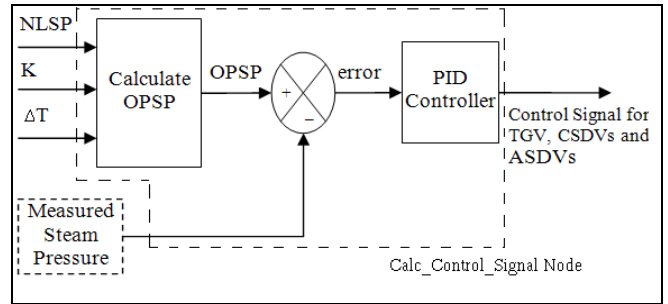


Fig. 4. Schematic of SCADE Node Calc\_Control\_Signal

The lower level nodes realize the control functionalities like PID control within Calc\_Control\_Signal, lookup tables and range conversion algorithms. Fig 5 shows the hierarchical architecture of the SCADE model of SGPCS, along with the coupling among nodes and number of instances of each node.

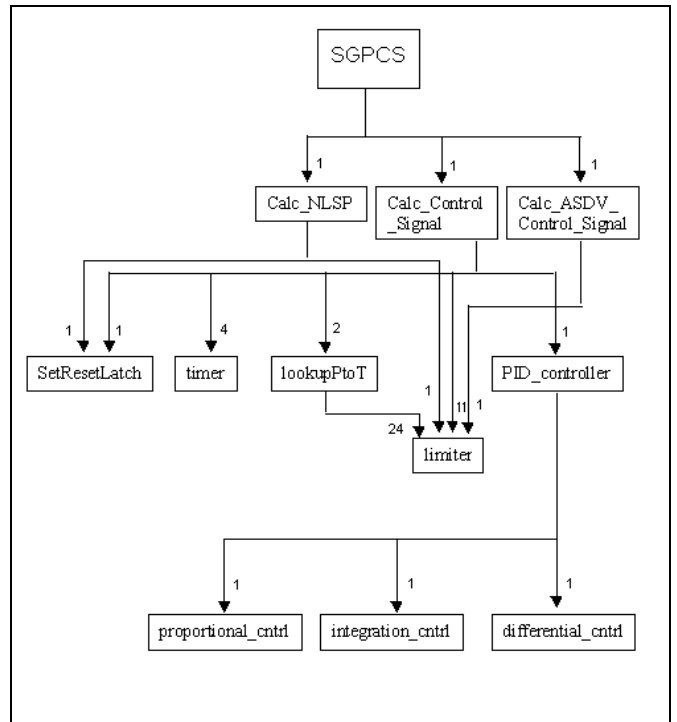


Fig. 5. Hierarchical architecture of SCADE model of SGPCS

#### IV. VALIDATION OF SGPCS MODEL

The validation of SGPCS model is done by simulating the SCADE model. We have to apply a sequence of input vectors to test a given system behavior. The trend of outputs with the applied input sequence is observed and compared with the expected outputs, for analyzing the correctness of the model.

Each test sequence that may expand over several cycles is stored as a separate scenario file. The scenario file stores the value of all the input vectors applied during each cycle. The scenario files are useful to automatically regenerate the previously conducted test whenever required.

To assess how thoroughly the behavior of the model is simulated, we have used Model Test Coverage (MTC). MTC is a SCADE Suite module, which allows measuring the coverage of model during model simulation activity. SCADE MTC analyzer uses MC/DC [6] structural coverage criteria to assess the model coverage. The MTC is done by using the scenario files stored during the model simulation. The use of scenario files ensures that the same test cases are applied during simulation and MTC. Hence the coverage results observed after MTC shows that how extensively the behavior of the model was simulated and points out the part of the model that was not covered during model simulation. We can then design new test cases for covering those parts of the model that are not covered. By applying test cases as per the previously prepared test plan, we could achieve 97 percent model coverage.

From the analysis of the operators that were not tested using the test plan, we designed new test cases for increasing the model coverage. An example for derivation of such a test

case can be illustrated using the Fig.6. Each operator in the figure is labeled with an instance number to distinguish between two instances of same operator. So to identify each operator uniquely during illustration, we have used operator name followed by its instance number e.g. the AND operator labeled with instance number 2 will be depicted as AND2. Similarly the inputs of each operator are identified as I1, I2...In, depicting 1st, 2nd ... nth input respectively. In the figure below MTC shows that the operator AND2 is not covered completely. According to the MC/DC criteria the three input AND gate, having inputs I1, I2 and I3 is said to be covered completely if the following conditions are simulated.

- I1, I2 and I3 are true
- Only I1 is false
- Only I2 is false
- Only I3 is false

The coverage analysis shows that, for the operator AND2 the condition “Only I2 is false” was not simulated. From the Fig.6 it could be determined that to simulate this condition, we should have an input combination where NOT(ASDV\_auto) is true, NOT(ASDV\_HC) is false and ASDV\_CM\_raise is true. This derived input test vector is added to the test plan to increase the model coverage.

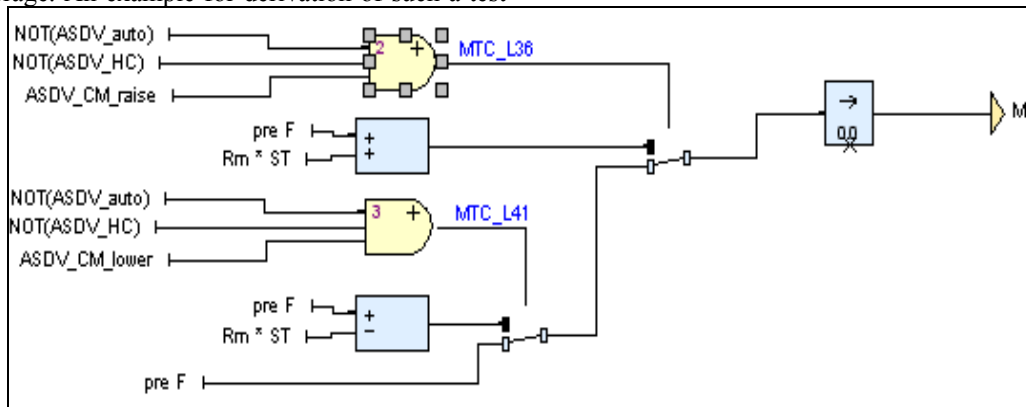


Fig. 6. Derivation of Test Case from the Result of MTC

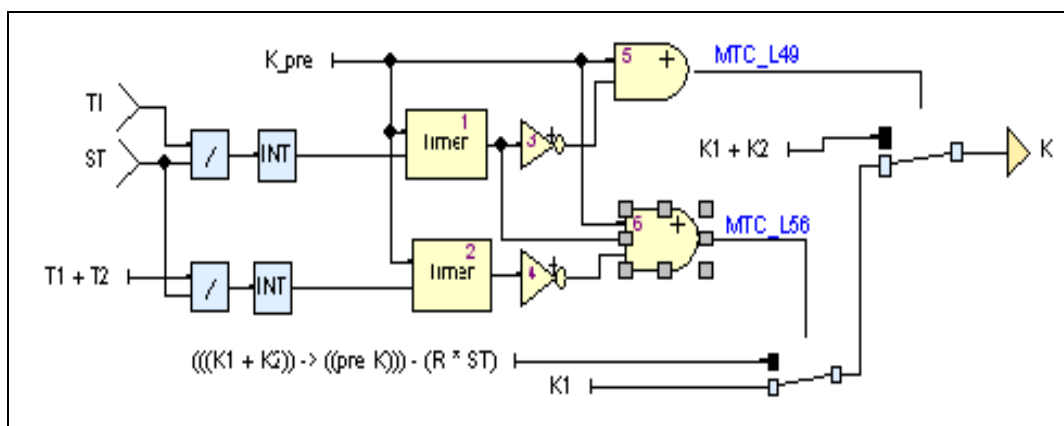


Fig. 7. Part of Calc\_Control\_Signal node

We have also observed that some conditions required to completely cover the operator according to MC/DC criteria were physically infeasible. Fig.7 shows one such condition. The MTC in this figure shows that the operator AND6 is not covered because the condition “only I1 is false” was not simulated. To simulate this condition, input I1 for operator AND6 should be false keeping inputs I2 and I3 as true. I2 for the AND6 is output of operator timer1. The output of timer1 operator becomes true only if input is true for specified time interval or more; otherwise its output is false. So it can be determined that if I1 is false then output of timer1 will be false and hence I2 will also be false i.e. whenever I1 is false, I2 will also be false. So we cannot design test case, which simulates the condition “only I1 false” for the operator AND6.

Infeasible scenarios as described above had to be left out and hence it was possible to achieve 99.8 percent model coverage.

### V. FORMAL VERIFICATION OF REQUIREMENTS OF SGPCS

The verification of the SGPCS model was carried out by using the formal verification technique called model checking. It involves expressing a system requirement as a property and checking whether property is satisfied by the model. Each of the properties to be verified was modeled as an “observer node” using built-in SCADE verification operators. The composition of “observer node” with the system model was used for model checking using the SCADE Design Verifier.

Some of the important system properties of SGPCS that were formalized and verified are explained below. Each property is first stated in English as it appears in specification

followed by brief explanation. Later the property is stated formally using SCADE notation and explanation is provided whenever required.

#### A. Property1

*If manual\_crash\_cool or auto\_crash\_cool signal is true then No Load Set Point (NLSP) should decrease till lower saturation limit (100.0) is reached.*

The signals **manual\_crash\_cool** and **auto\_crash\_cool** are two discrete inputs, which are used to maneuver the pressure setpoint (NLSP) during some emergency condition. If at least one of these inputs is true then NLSP is decreased till the limiter clamps the value of NLSP to 100.0

Formal Specification of the *property1* in SCADE is shown in Fig.8. In figure the block “verif::Implies” implements the mathematical implication ( $p \rightarrow q$ ). The first input denotes ‘p’ and second input denotes ‘q’. The output of the block is true if ( $p \rightarrow q$ ) holds. The output of this block is assigned to the Boolean output *property1*.

#### B. Property2

*If NLSP\_lower and NLSP\_raise both signals are true then NLSP should decrease till lower limit (100.0) is reached.*

The signals **NLSP\_lower** and **NLSP\_raise** are discrete inputs used to lower and raise the value of NLSP respectively, however if both the signals are true simultaneously then **NLSP\_lower** signal must have a higher priority over **NLSP\_raise**.

Formal Specification of the *property2* is shown in Fig.9.

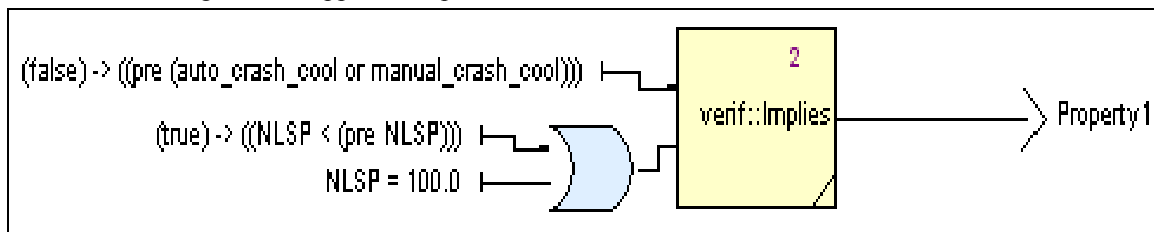


Fig. 8. SCADE Observer for Property1

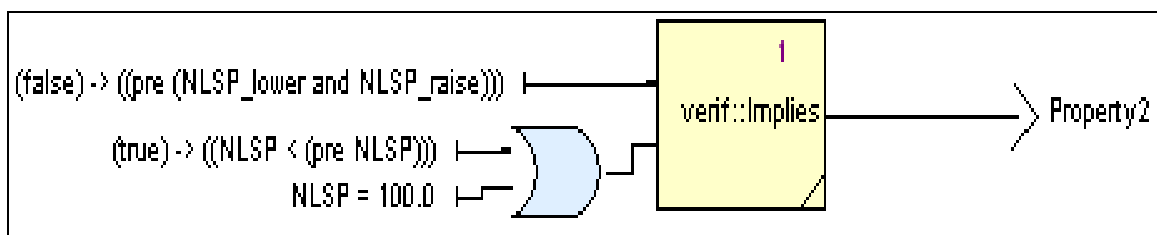


Fig. 9. SCADE Observer for Property2

C. Property3

If the *loss\_of\_electric\_load* or *turbine\_trip* signal is true and reactor power exceeds 20% Full Power then Anticipatory Action (AA) should start and should get completed in time  $T1+T2$ , where  $T1$  and  $T2$  are predefined constants. AA lowers the operational set point (OPSP) in proportion to reactor power based on the following equation.

$$OPSP = NLSP - K * \Delta T$$

The variation of  $K$  before, during and after AA is shown in Fig.10, where  $K1$  and  $K2$  are positive constants.

The signals **loss\_of\_electrical\_load** and **turbine\_trip** are the discrete inputs to the SGPC model, which decide the condition for start and termination of AA

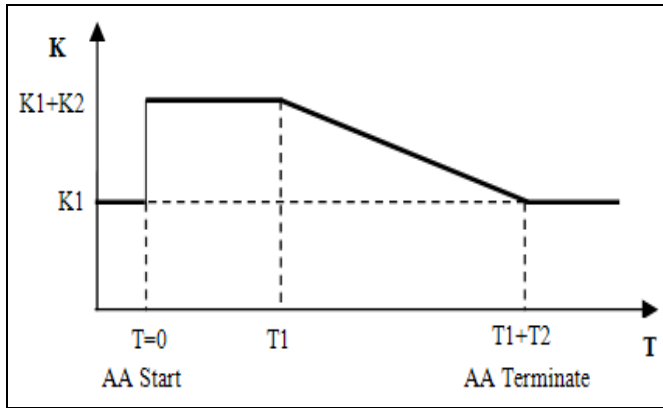


Fig.10 Timing diagram for Anticipatory Action

In SCADE, the time is measured in terms of number of execution cycles. The number of execution cycles in time  $T1+T2$  can be determined by dividing time  $T1+T2$  by

sampling time (0.175 sec). The value of  $T1$  and  $T2$  is 1 and 2 respectively. Hence the number of execution cycles is  $(1+2)/0.175 = 17.14 = 17$ . However Fig.10 shows that the value of  $K$  at the first and last cycles is equal to  $K1$ . Therefore the number of cycles during which AA will be present is two less than the calculated number of cycles in time  $(T1+T2)$ . Hence AA should appear for  $(17-2)=15$  cycles”.

Formal Specification of the *property3* in SCADE is shown

in Fig.11. In figure the block  $\frac{I1}{I2} \rightarrow O1$  specifies an if-else condition as “ $O1 = \text{if } C1 \text{ then } I1 \text{ else } I2$ ”. The block `verify::AfterNthTick` is used to express that the output equals input, except for the first  $N$  ( $N=1$ ) cycles during which the output is true. The block `verify::AtLeastNTicks` express that the output equals the input as soon as the input is true for  $N=15$  cycles, before that the output is false. Similarly the block `verify::HasNeverBeenTrue` is used to express that the output becomes false as soon as its input becomes true for the first time, after this cycle the output remains false.

D. Property4

Anticipatory action is discontinued after a reactor trip or when both initiating conditions i.e. *loss\_of\_electrical\_load* and *turbine\_trip* are not true.

This property specifies the condition for the termination of anticipatory action. Thus if required condition for anticipatory action does not exist or if it disappears then anticipatory action will terminate.

The formal specification of the *property4* in SCADE is shown in Fig.12.

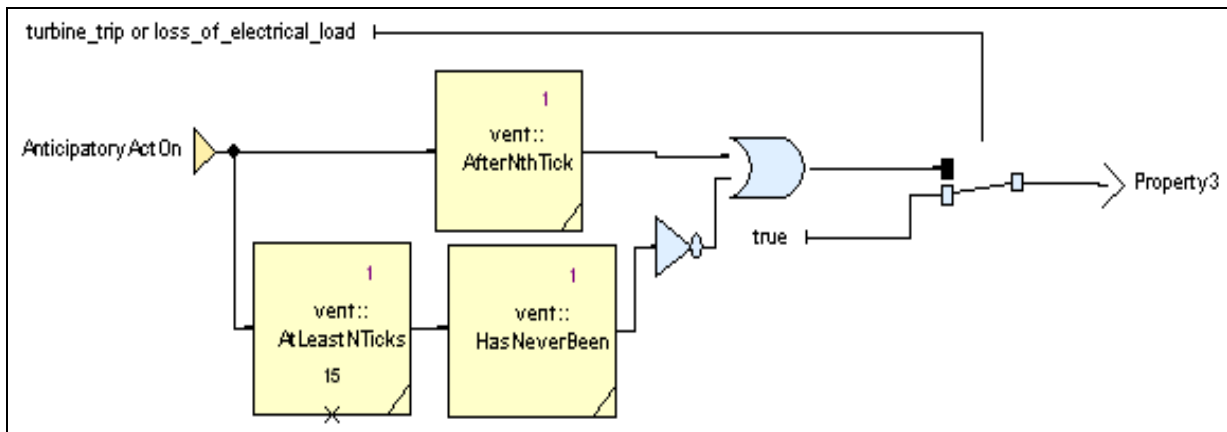


Fig.11. SCADE Observer for Property3

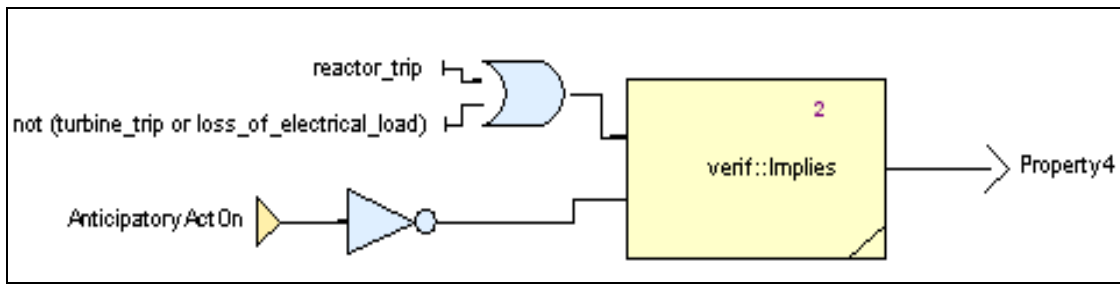


Fig.12. SCADE Observer for Property4

### E. Property5

If ASDV is in computer manual mode (i.e. ASDV is neither in auto mode nor in hand controller mode) and

- ASDV\_CM\_raise is true and ASDV\_CM\_lower is false then ASDV\_control\_signal will increase.
- ASDV\_CM\_raise and ASDV\_CM\_lower both are true then ASDV\_control\_signal will remain unchanged.
- ASDV\_CM\_raise is false and ASDV\_CM\_lower is true then ASDV\_control\_signal will decrease.

The final control signal for ASDVs is calculated based on the mode of operation. ASDVs can be in one of the three modes: *auto*, *hand control* or *computer manual*, which are selected from a three-position switch. Hence in the model the mode is determined by two input variables **ASDV\_auto** and **ASDV\_HC**. If input **ASDV\_auto** is true then mode is auto, if input **ASDV\_HC** is true then mode is hand controller and if

both inputs are false then mode is computer manual. Inputs **ASDV\_CM\_raise** and **ASDV\_CM\_lower** are used to raise and lower the ASDV\_control\_signal respectively.

The formal specification of the *property5* in SCADE is shown in Fig.13. The property could not be proved and a counter example was generated. The property could not be proved because the specification does not place any restriction on mode of ASDV in the instant before coming to computer manual mode. Therefore if in the previous instant the mode was other than computer manual the control signal values before and after mode change do not conform to the property specification. The model was later corrected to implement the bumpless transfer functionality.

The time required for verification of these properties was found to be less than 10 seconds on a XEON Server with 4GB RAM.

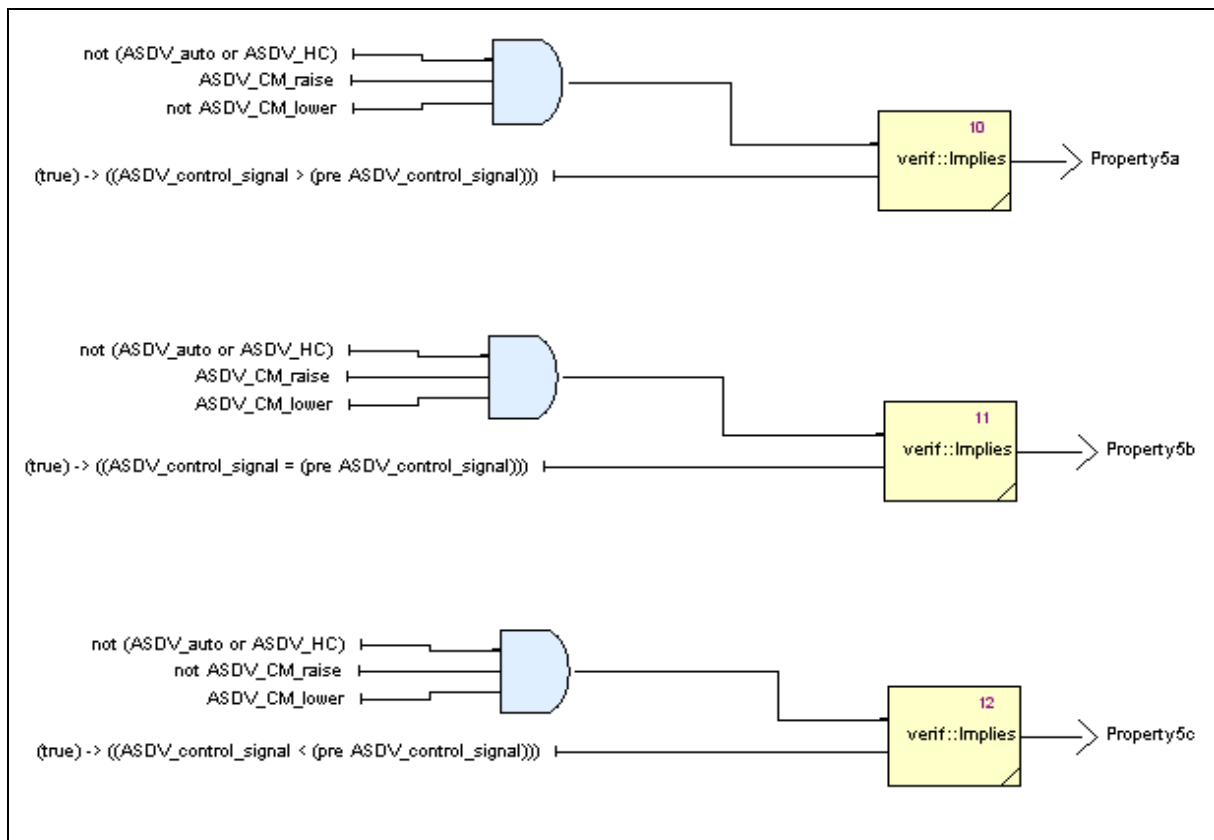


Fig.13. SCADE Observers for Property5a, Property5b and Property5c

## VI. CODE GENERATION AND INTEGRATION

SCADE environment provides IEC61508 SIL3 certified automatic code generator (KCG). The verified model was used to generate the C code automatically. The use of certified code generator ensures that every specification in the model is correctly reflected in the code, which eliminates the need for unit testing.

The SGPC module developed using SCADE was integrated with prototype system in the lab. This was done by replacing old handwritten SGPC code by the one developed with SCADE.

Two identical hardware based controller setups were available. One was loaded with the old handwritten SGPC code. This implementation is in use for several years. The other controller was loaded with automatically generated code from the SCADE model. There was no discrepancy observed in the output of these two systems during testing.

The final size of the executable generated was ~163 KB whereas the size of old executable was ~167 KB, which was comparable

## VII. CONCLUSION AND FUTURE WORK

This methodology is suitable for the systems whose output behavior involves cyclic executions of *read*, *compute* and *output*. Software based controllers used in embedded control applications typically fall in this class of systems. It has been our experience that it is quite easy to adopt the methodology in the design life cycle of such software and comply with the recommended practices of IEC60880. This is because the SCADE environment supports a formal language and has tool support for testing the model conforming to MC/DC structural coverage and formal verification. The availability of IEC 61508 SIL3 certified code generator has reduced the code generation effort to a *push button* and there is no effort required for low level testing. It must be emphasized that once the model is validated and code is generated from the model, it is not recommended to do any manual changes in the generated code. The availability of certified code generator is a great advantage over the other traditional development methodologies (e.g. UML) involving modeling language having less formal semantic foundation. The lack of semantics in such modeling language hinders complete automated code generation and hence no certifiable code generation is possible for the entire modeling language.

During formal verification, it was observed that most of the time was spent on reviewing property expressed in SCADE with corresponding English specification. Although the tool supports an easy to use interface to the back end verification engine, the user is expected to know the limitations of formal verification of systems involving integer, real and nonlinear arithmetic. The verification engine supports two kinds of verification; debug mode, which involves bounded model checking [9] and proof mode involving exhaustive verification. A property which is shown to be true in debug mode (bounded

by given depth) should not be assumed to be true under all possible cycles of execution. The proof strategy can take enormous amount of time (may not terminate in a bounded time) if the property is not verifiable within certain search depth because the system may have very large state space. The verification interface provides strategy options for terminating the search if the verification is not successful within a specified amount of time.

SCADE supports predefined operators for specifying verification conditions. However formal logics such as LTL are more expressive and it is possible to express properties much more succinctly in these notations. For example, the property 3, in section V.C, could have been specified as  $(s \rightarrow (p \mathbf{U}_{15} q))$ , where  $s$  is the initiating event (turbine trip or loss of electrical power),  $p$  is the Boolean condition for anticipatory action and  $q$  is the Boolean condition signifying the end of anticipatory action. The sub-expression  $p \mathbf{U}_{15} q$  in the verification condition demands that  $p$  is true for 15 ticks and henceforth  $q$  is true and the complete property states that if  $s$  is true and remains true then  $p$  shall remain true for 15 ticks and subsequently  $q$  shall remain true. It is felt that such properties are difficult to express using standard SCADE blocks. SCADE does not support a tool that can translate specifications in such logic as explained above into equivalent SCADE property observers. In future we plan to develop such a property synthesis tool and integrate in the SCADE environment.

## ACKNOWLEDGMENT

We are thankful to Mr. G.P. Srivastava, Director E&I Group and Mr. B.B. Biswas, Head RCnD, BARC, for their encouragement during this work. We are thankful to Mr. C.K. Pithwa, Head, CCDS, RCnD, for providing us the initial specification of SGPCS from which the formal model has been developed. Thanks are also due to Mrs. Ratna Bhamra for helping us to understand the initial specifications.

## REFERENCES

- [1] Unified Modeling Language <http://www.omg.org/technology/documents/formal/uml.htm>.
- [2] N. Halbwachs, Synchronous programming of reactive systems. Kluwer Academic Pub, 1993.
- [3] N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud, The synchronous dataflow programming language Lustre. Proceedings of the IEEE, vol. 79(9). September 1991.
- [4] SCADE Tool <http://www.esterel-technologies.com/>.
- [5] SCADE Language Reference Manual, Ver 6.1, Esterel Technologies Ltd., France
- [6] SCADE User Manual, Ver 6.1, Esterel Technologies Ltd., France
- [7] S. Glasstone and A. Sesonske Nuclear Reactor Engineering, Chapman & Hall, New York.
- [8] Software Requirement Specification for DPHS-PCS Rev 2 USI 63000 Internal Document RCnD, BARC, Feb. 2005
- [9] A.Biere, et.al., Bounded Model Checking Vol. 58, Advances in Computers, 2003 (Academic Press)