Verification and validation of safety applications based on PLCopen safety function blocks

Prepared by:

Mohammed Salah Al-Radhi

malradhi@tmit.bme.hu

Superviced by:

Prof Istvan Majzik

6th December 2017

Background: Programmable Logic Controller

PLC's Are ...

- Similar to a Microcontroller.
- Initially designed to replace relay logic boards.
- Wiring between devices and relay contacts are done in the PLC program.
- Easier and faster to make changes.

Background: PLC system diagram

External input devices







- Pushbutton switch
- Relay contact
- Flow meter
- Level transmitter sensor
- Temperature transmitter sensor
- Proximity sensor
- Pressure transmitter sensor

PLC CPU

- Power Supply
- Micro-controller
- Memory
- Fieldbus
- I/O ports

External output devices

- Indicator
- Digital display
- Actuator
- Solenoid valve
- Contactor
- Motor
- Pump

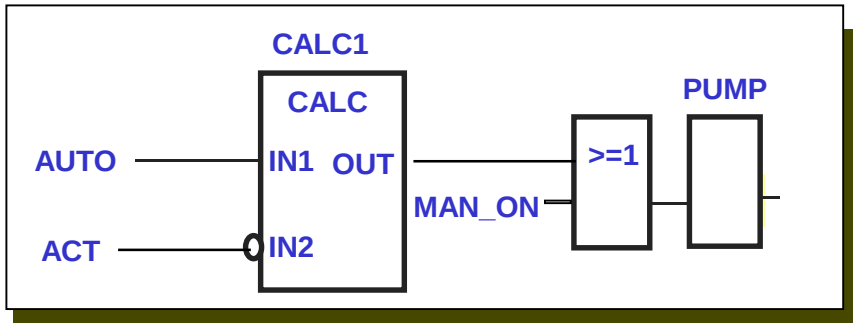




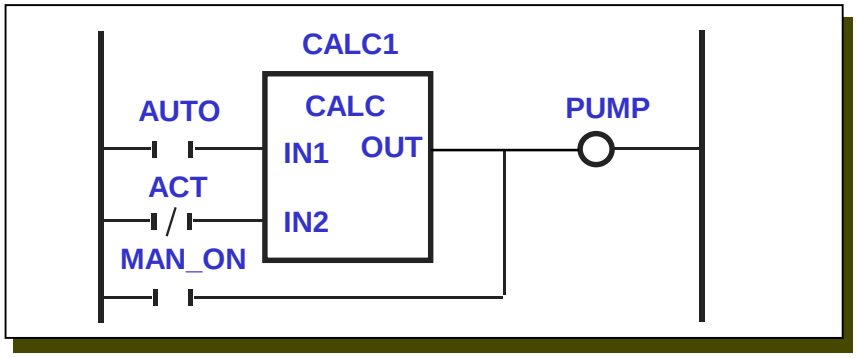


Bakground: IEC 61131-3 Programming languages

Function Block Diagram (FBD)



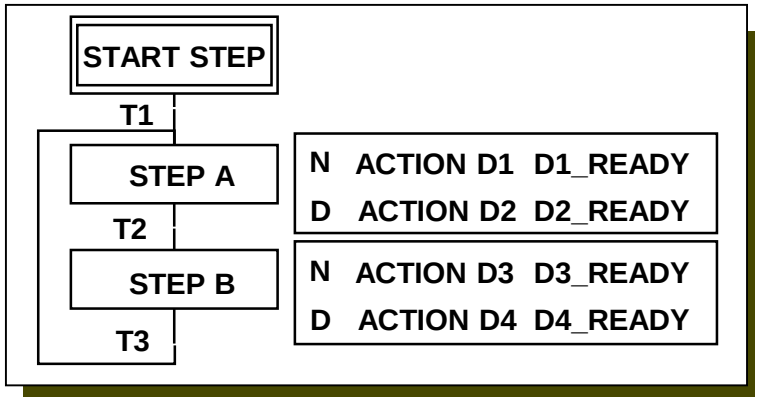
Ladder Diagram (LD)



Instruction List (IL)

```
A: LD    %IX1  (* PUSH BUTTON *)
   ANDN  %MX5  (* NOT INHIBITED *)
   ST    %QX2  (* FAN ON *)
```

Sequential Flow Chart (SFC)

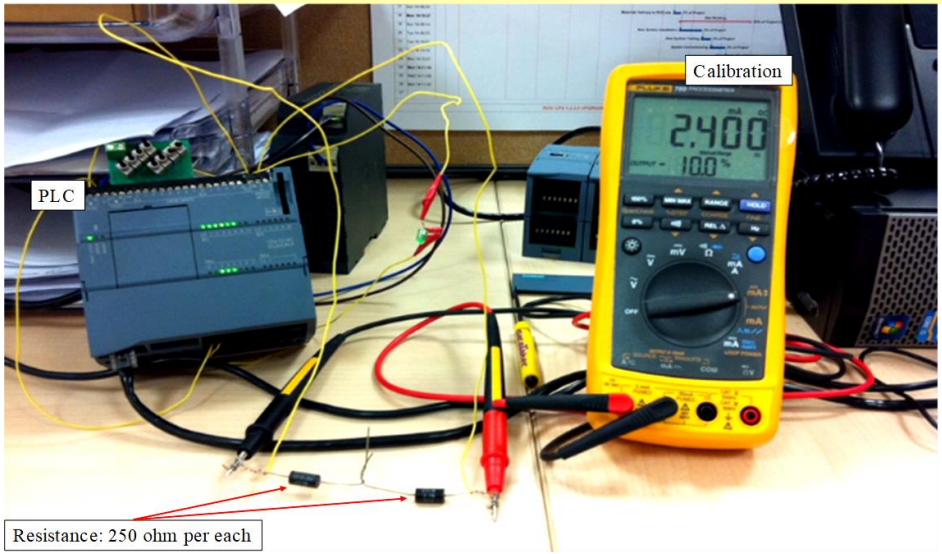


Structured Text (ST)

```
VAR CONSTANT X : REAL := 53.8 ;
Z : REAL; END VAR
VAR aFB, bFB :  FB_type; END_VAR

bFB(A:=1, B:='OK');
Z := X - INT TO REAL (bFB.OUT1);
IF Z>57.0 THEN aFB(A:=0, B:="ERR");
ELSE  aFB(A:=1, B:="Z is OK");
END_IF
```

Bakground: Previous my work (May 2016) in Iraq



Problem and Goal

➢ According to IEC61131-3

- Safety issues are not addressed
- PLCs not distinguish between logic signals and safety signals

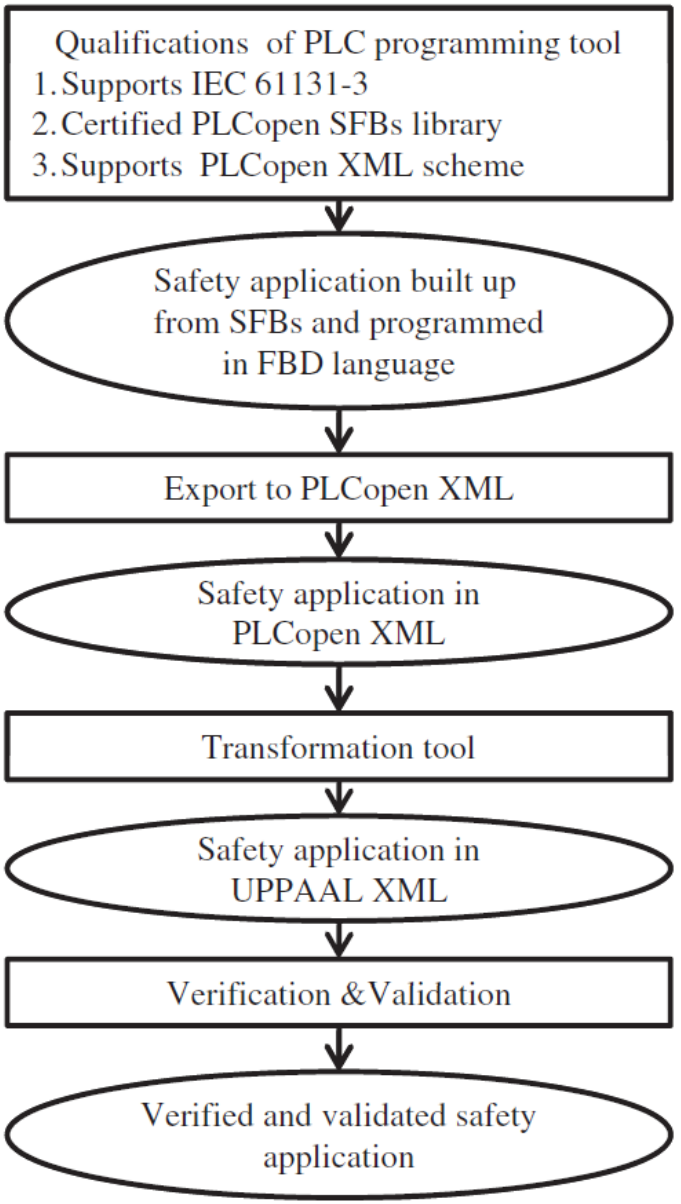
❑ Goal:

- Formal V&V and tests by simulations are needed in the beginning of the design process because of safety aspects.
- Have a tool which allows verification of safety application built up in the PLC safety function blocks (SFB) library.

Motivation to implement SFB library

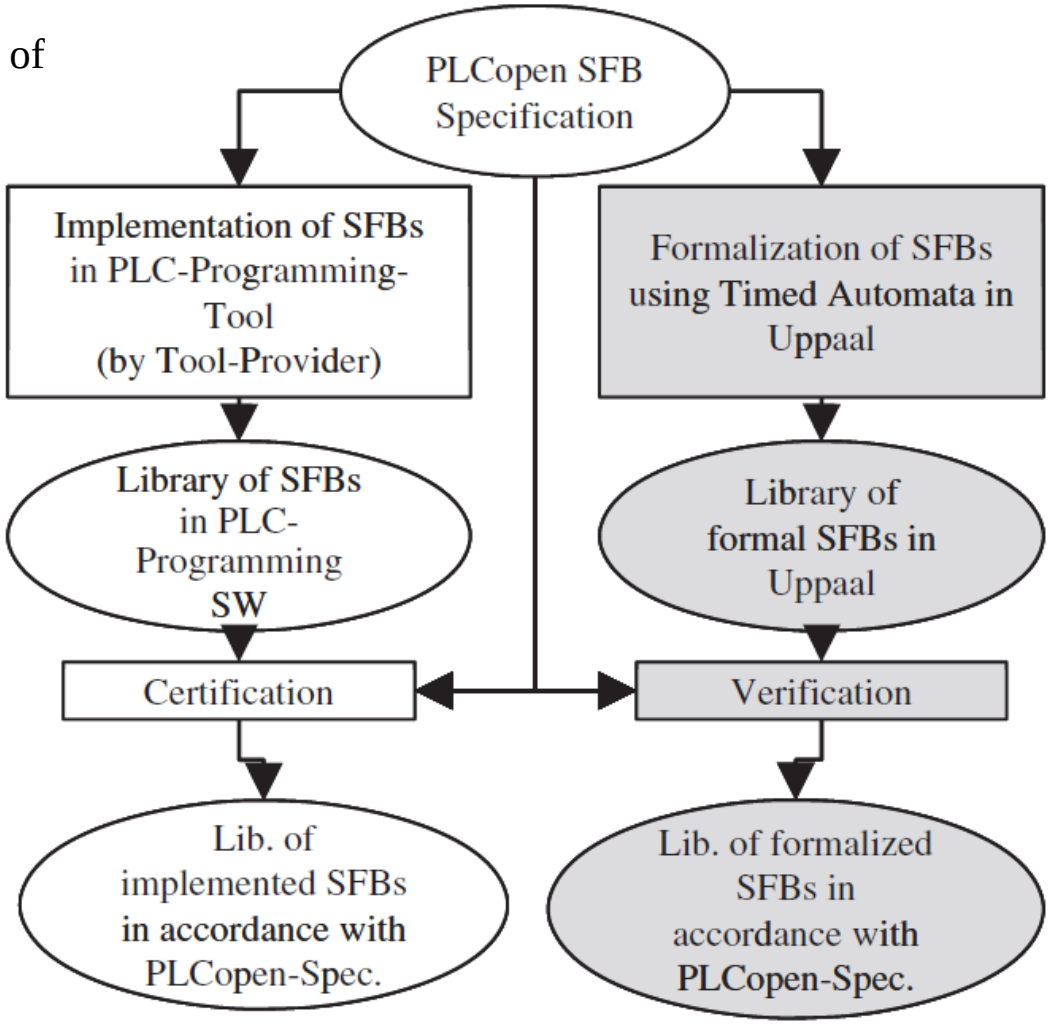
- XML interface is used to allow automatic transformation of a safety application into UPPAAL model checker.

- UPPAAL is an integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata.



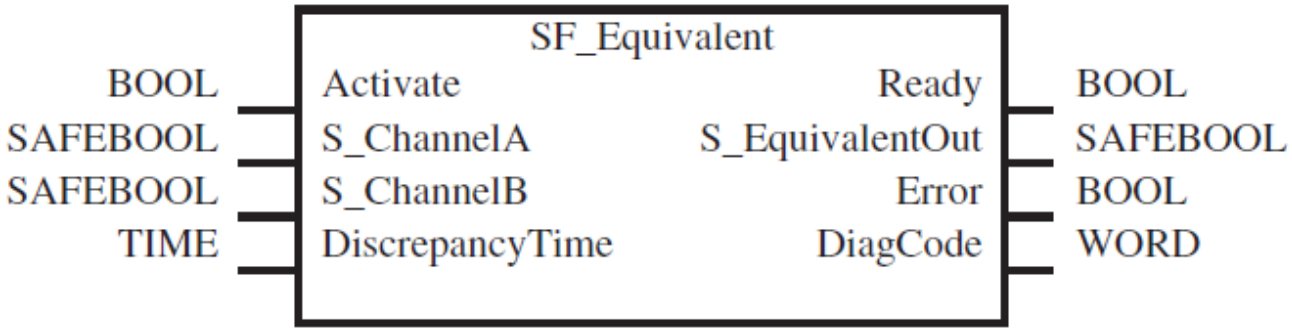
Approach of verification of a safety application

- The conformance of these SFBs is normally certified by some organization like TUV (German organization that certifies the safety of products of all kinds).



Description of PLC open specifications

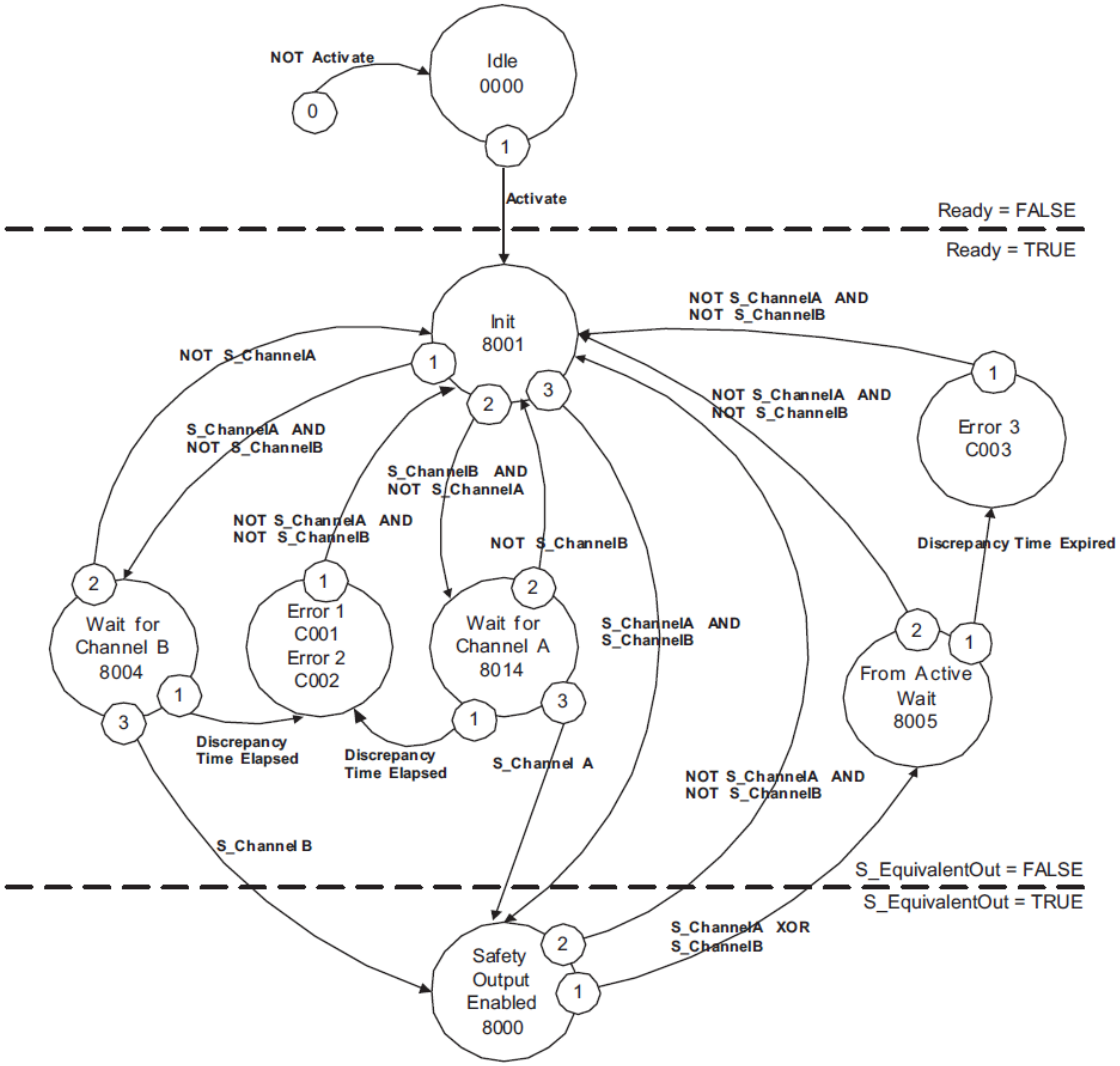
- SFB: *SF_Equivalent*
  - This function block converts two equivalent SAFEBOOL inputs to one SAFEBOOL output with discrepancy time monitoring.
  - If one channel signal changes from TRUE to FALSE, the output immediately switches off (FALSE) for safety reasons.
  - Discrepancy time: is the maximum period during which both inputs may have different states without the function block detecting an error.
  - Error: when both inputs do not have the same status once the discrepancy time has elapsed.



Input/Output variables declarations of the block

State diagram for SFB SF_Equivalent

- State diagrams: It is used to give an abstract description of the behavior of an SFB.
- For example, the *Ready* output is false only in Idle state, and the *S_EquivalentOut* output is true only in Safety output enabled state.
- The *Error* output is true only in error states.
- The numbers in small circles indicate the priority in transitions where zero is the highest priority.

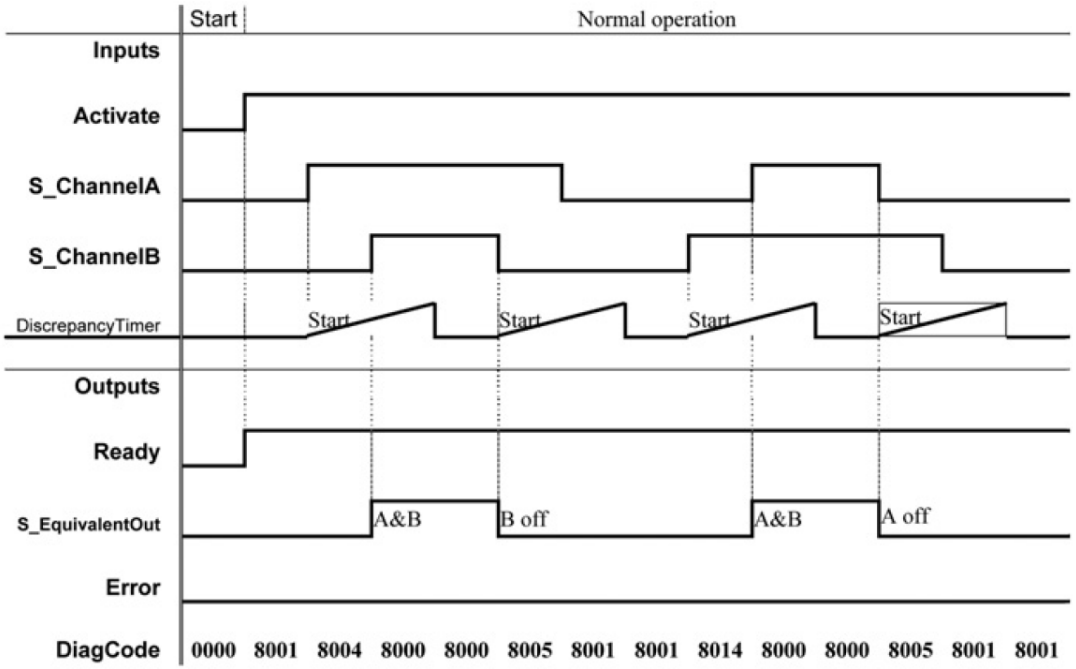


Temporal behavior of the SFB *SF_Equivalent*

1. In the Start operation, all outputs are false by default until the SFB is activated (Activate = true).
2. The Ready output goes immediately to true and the Normal operation begins.
3. The status of the two input channels (S_Channel A and S-Channel B) is observed:

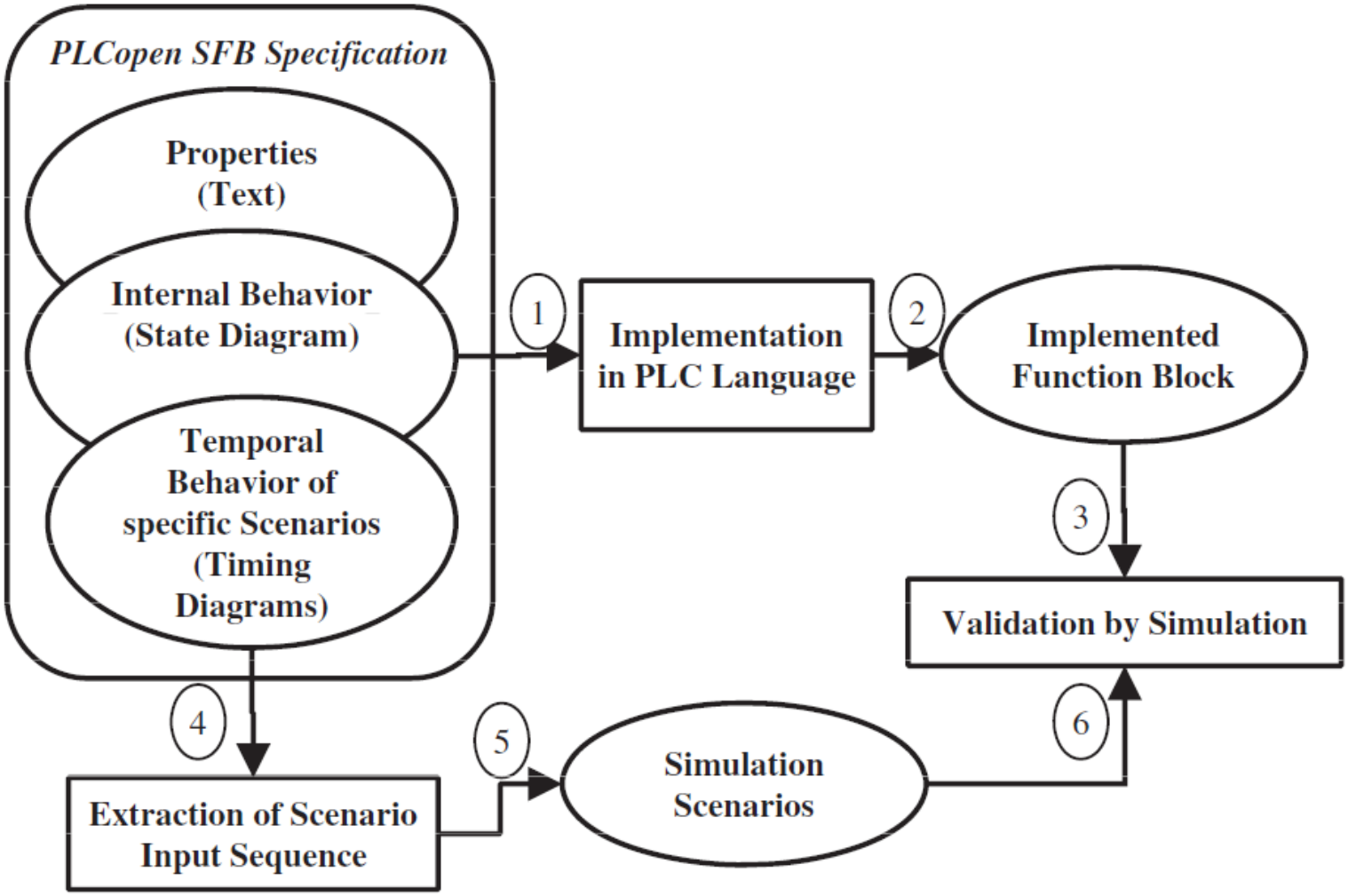
- If both channels go to true consecutively within a specified time (Discrepancy Time), the *S_EquivalentOut* output goes to true indicating a safe state. Otherwise, the output goes to false indicating an unsafe state.
- If the discrepancy time is elapsed and both input channels have different status, then the Error output goes to true.

➢ A diagnostic code output (DiagCode) indicates the active internal state of the SFB in hexadecimal numbers.



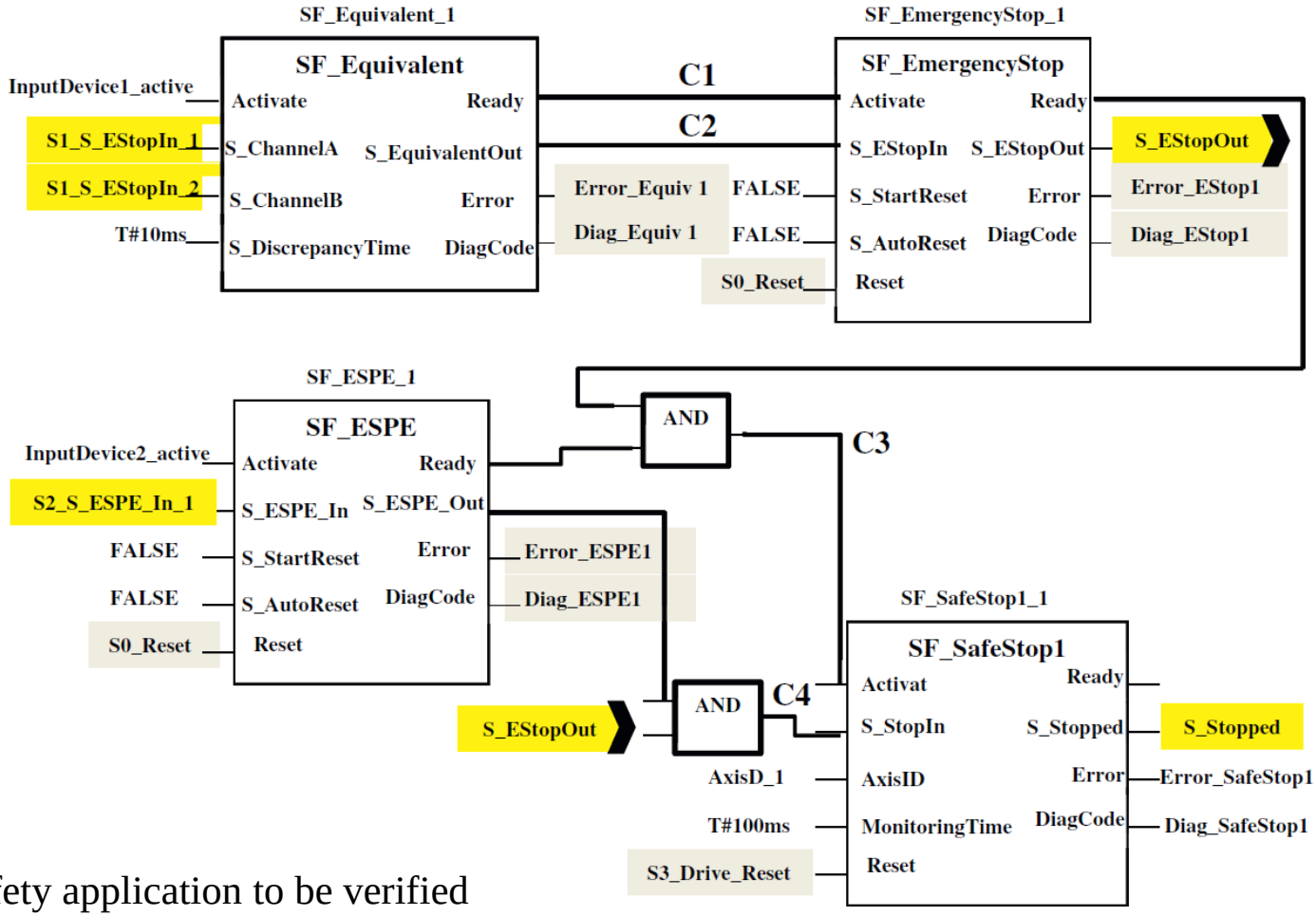
Implementation of SFBs in a PLC programming tool

- All the Input/Output variables of the SFB are declared.
- The state diagram is transferred into an IL code to allow direct verification of SFB



Example: Emergency stop with safe stop equivalent monitoring

- *SF_EmergencyStop* handles the Emergency Stop condition for the application, and ensures the correct restart.
- *SF_ESPE* handles the output signal switching device of electro-sensitive protective equipment (ESPE), and ensures the restart inhibit.
- *SF_SafeStop1* initiates and monitors a stop of a drive system in accordance with stop category1.

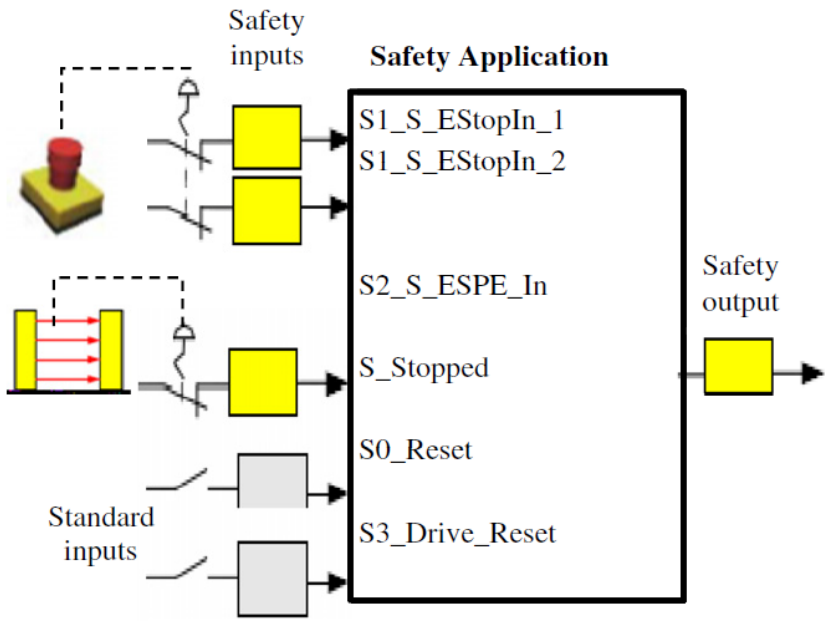


Safety application to be verified

Example: Complete description of a safety application

- Emergency stop with safe stop & equivalent monitoring

  - Three safety inputs and one safety output
  - The two inputs *S1_S_EStopIn_1* and *S1_S_EStopIn_2* are connected to the emergency stop push button through 1oo2 switch
  - If both go true within specified discrepancy time, it indicates that the switch is released and no emergency stop is requested.
  - This information passes via SF_Equivalent and *SF_EmergencyStop* to *SF_SafeStop1*.

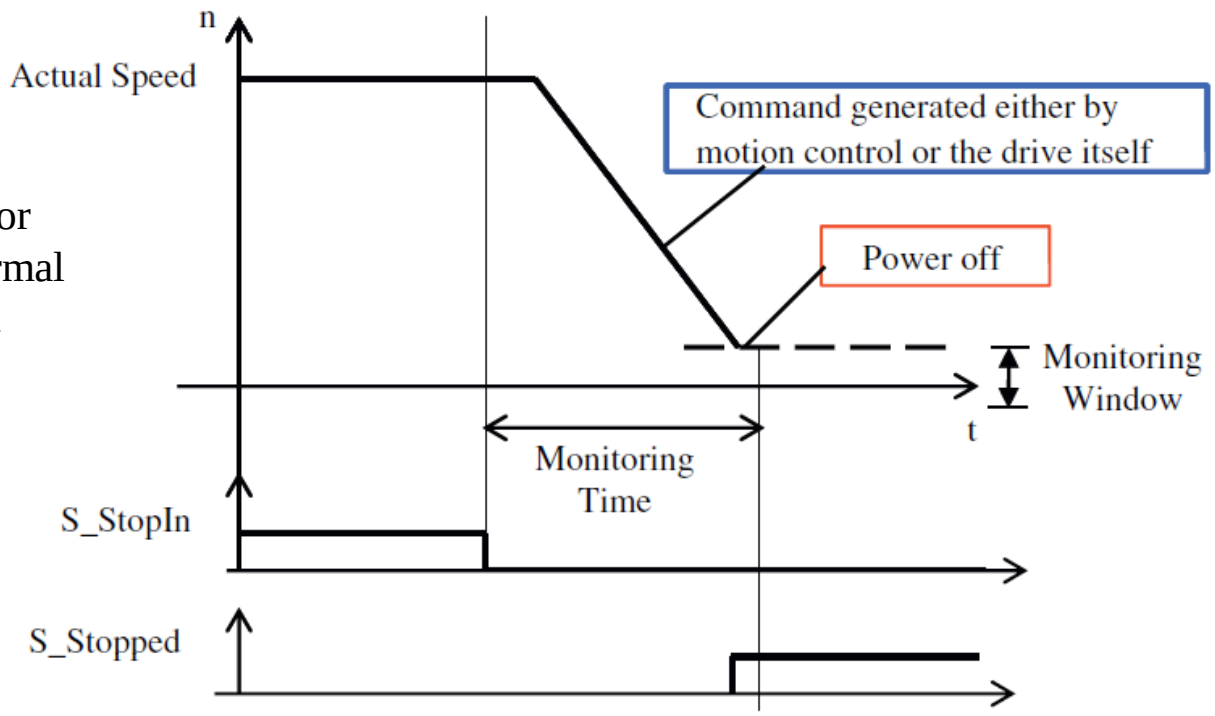


Graphical overview of the safety application interface

Example: Complete description of a safety application

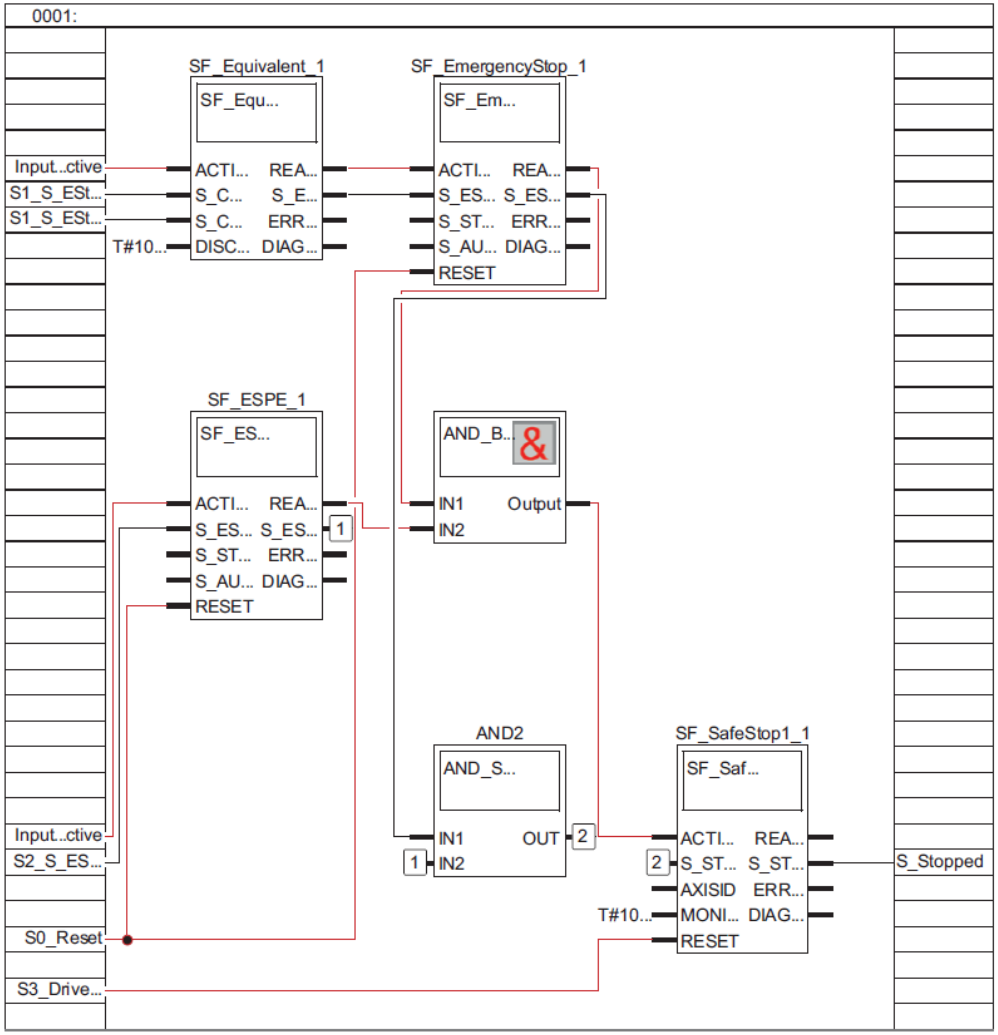
- If there is no stop request (*S_Stop* = true), then it indicates a normal operation and the safety output *S_Stopped* goes to false.
- In case of false *S_StopIn*, it waits for acknowledgment from the drive system within a specified monitoring time.
- This acknowledgement signal informs the safety application that the motor is stopped.
- When received, *S_Stopped* output goes to true and brings the system to a safe state.

- Shows the timing diagram for *SF_SafeStop1* in case of normal operation and safety request



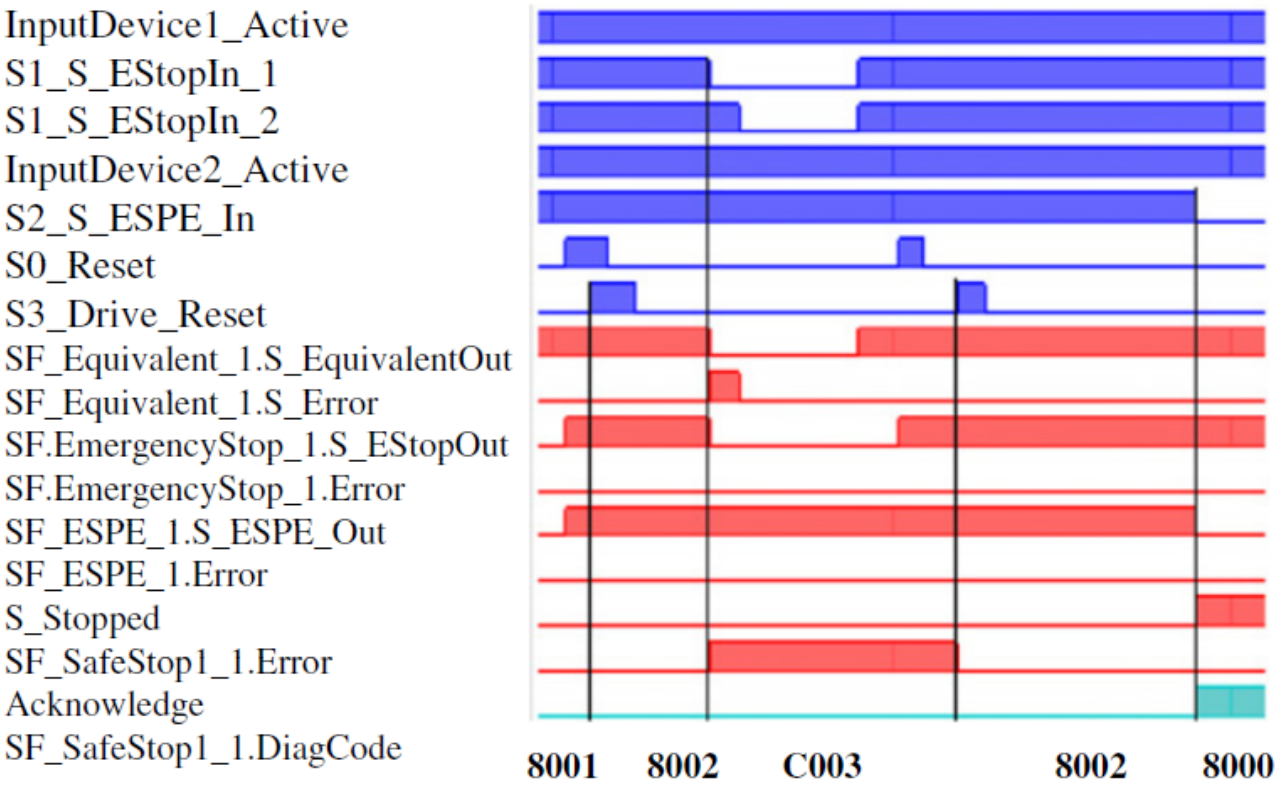
Example: Implementation of a safety application

- Using the FBD editor, the four SFBs are inserted and declared. The safety Input/Output variables are also declared to allow connecting to the SFBs.



Example: Implementation of a safety application

- The simulation results showed that the safety functions are successfully achieved for the test-cases.

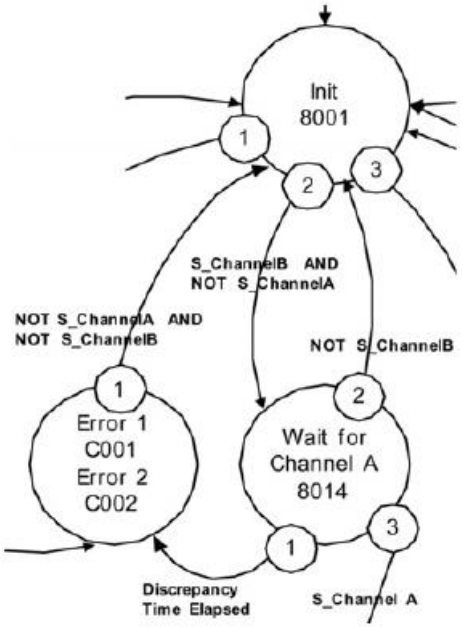


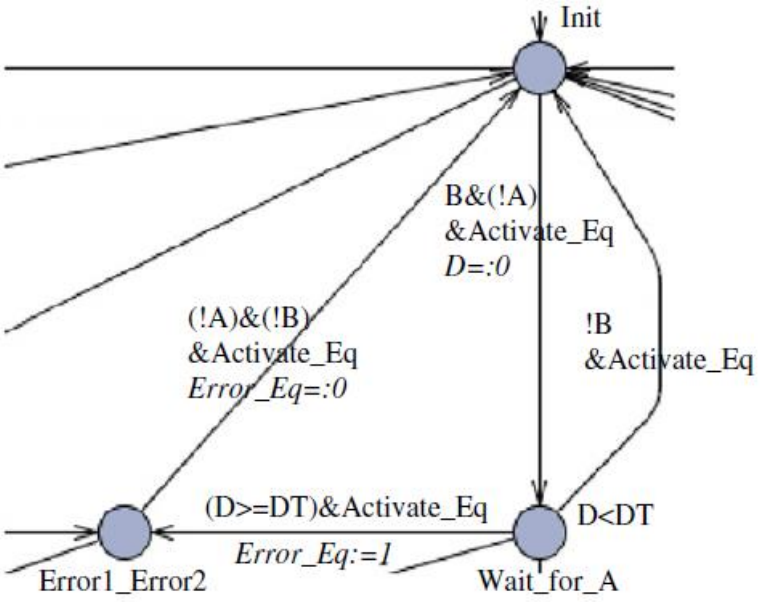
Screen shot of the timing diagram of the input and output variables

Approach to transform a state diagram to timed automaton

- In the left side: state diagram
  - The state diagram changes its state from Init to Wait for Channel A in case of a true *S_ChannelB* and false *S_ChannelA*.
  - It keeps a wait state until either an expiration of specified discrepancy time, or a change in *S_ChannelA*/*S_ChannelB* status.
  - Taking into account the priority number defined with every transition.

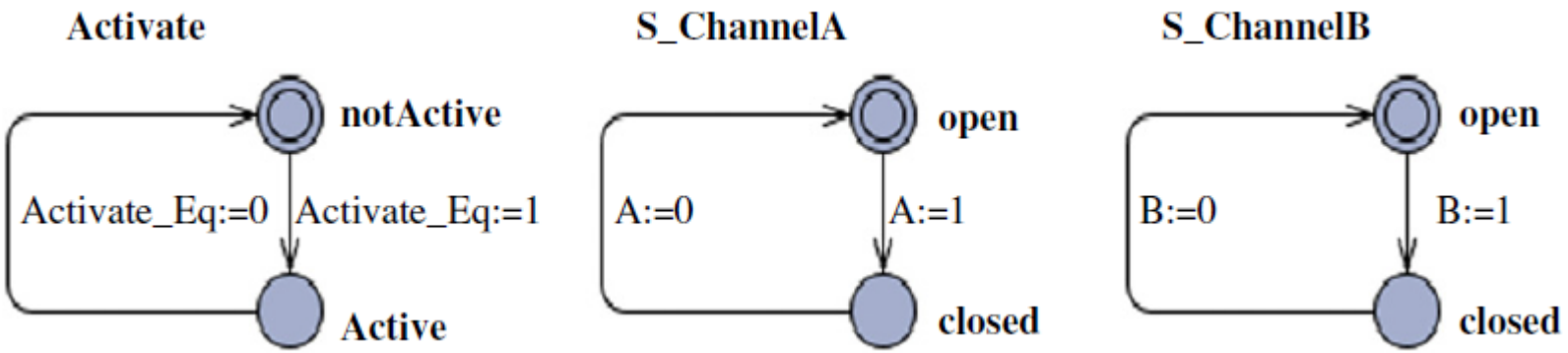
- In the right side: timed automaton
  - The clock D gets set to zero every time the system switches from location {Init} to location {Wait_for_A} on input event (B &(!A) & Activate_Eq).
  - The value of D is checked on another transition from {Wait_for_A} to {Error1_Error2} for the constraint (D = = DT).





System of timed automata

- Three timed automata.
- Every one consists of two locations and two edges between them.
- It simply describes the unconditional true/false variation of the input variable.
- On every edge, anew value of the input variable can be assigned.



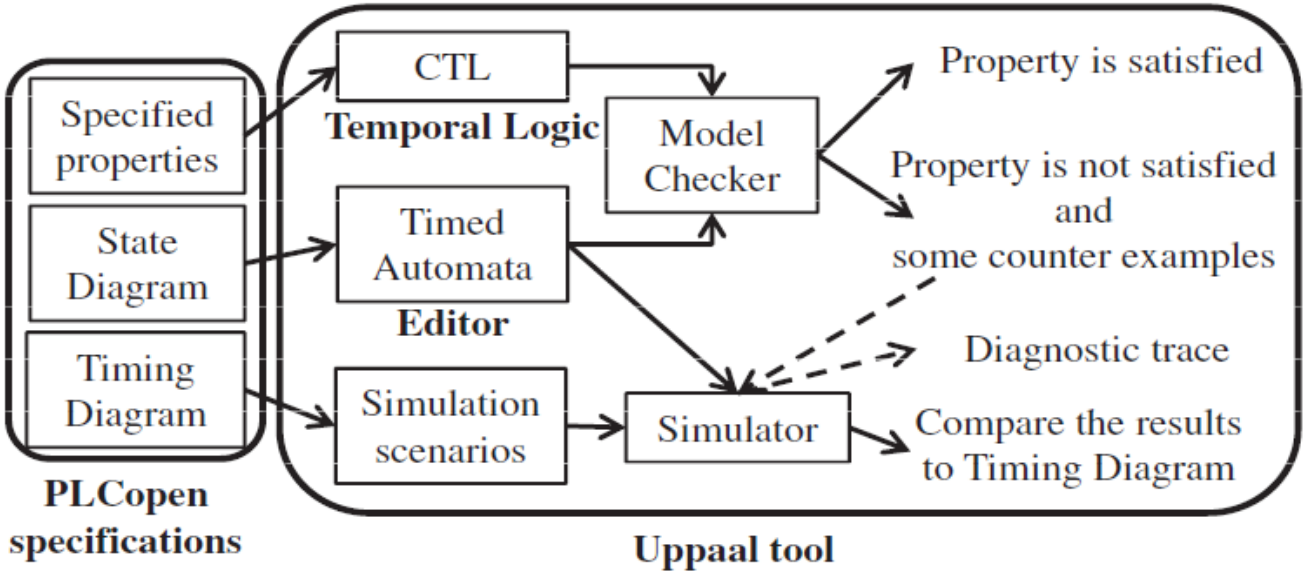
Timed automata for Activate *S_ChannelA* and *S_ChannelB* input variables of SF_Equivalent.

Approach for formalization of SFBs in UPPAAL

- To formalize an SFB:
  1. State diagram is transferred to a timed automaton using the graphical editor of UPPAAL.
  2. Depending on this timed automaton and the input sequences scenario extracted from the timing diagram, a simulation is executed to validate the temporal behavior using the graphical simulator.
  3. Thereafter, the textual properties are translated into temporal logic formulae. Using the verifier, the properties of the safety function can be verified.

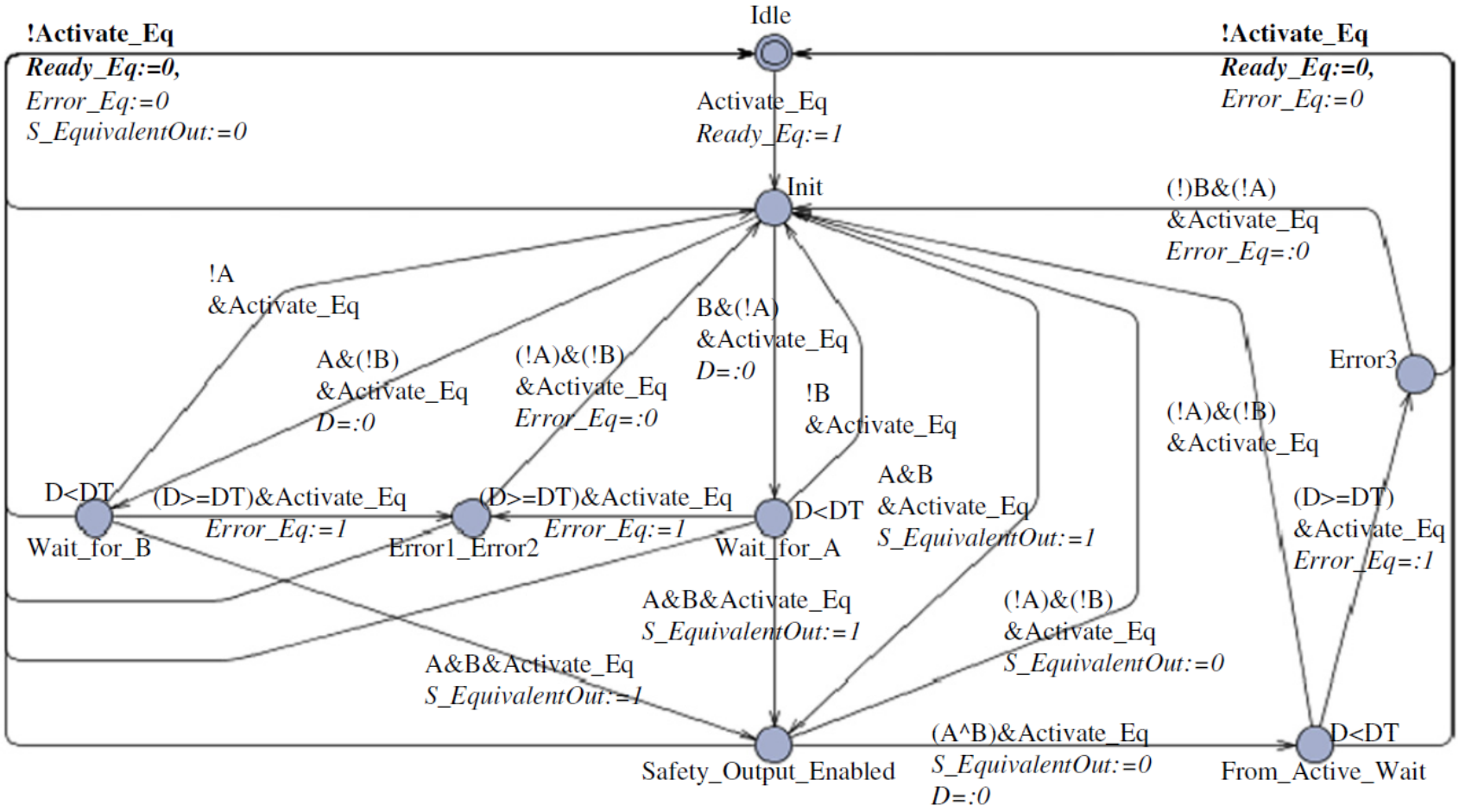
CTL and UPPAAL model checker

- Computation Tree Logic (CTL) is a branching temporal logic used in model checking.
- UPPAAL model checker can be used to determine the satisfaction of a given real time property with respect to a timed automaton.
  - If the property is not satisfied, a diagnostic trace can be generated that indicates how the property may be violated.
  - UPPAAL also provides a simulator that allows a graphical visualization of possible dynamic behaviors of a system description, i.e. a symbolic trace.



Formalization of SF_Equivalent

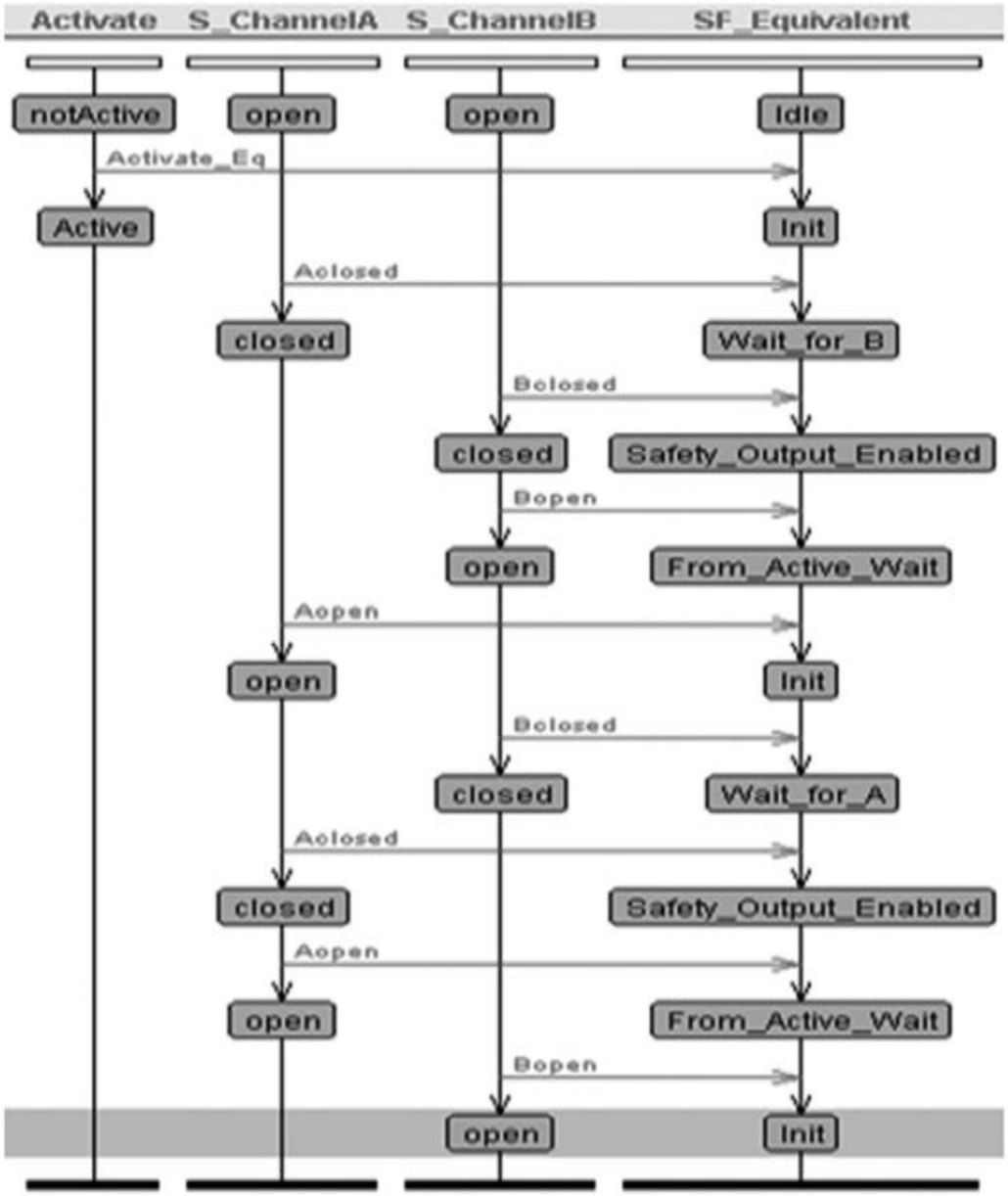
- ✓ The Input/Output variables are declared globally to allow sharing by other automata.
- ✓ The state diagram of the SFB is graphically translated to the state transition timed automaton



UPPAAL timed automaton for state diagram

UPPAAL timed automaton for state diagram

- Simulation scenarios extracted from timing diagram with message passing synchronization between automata.

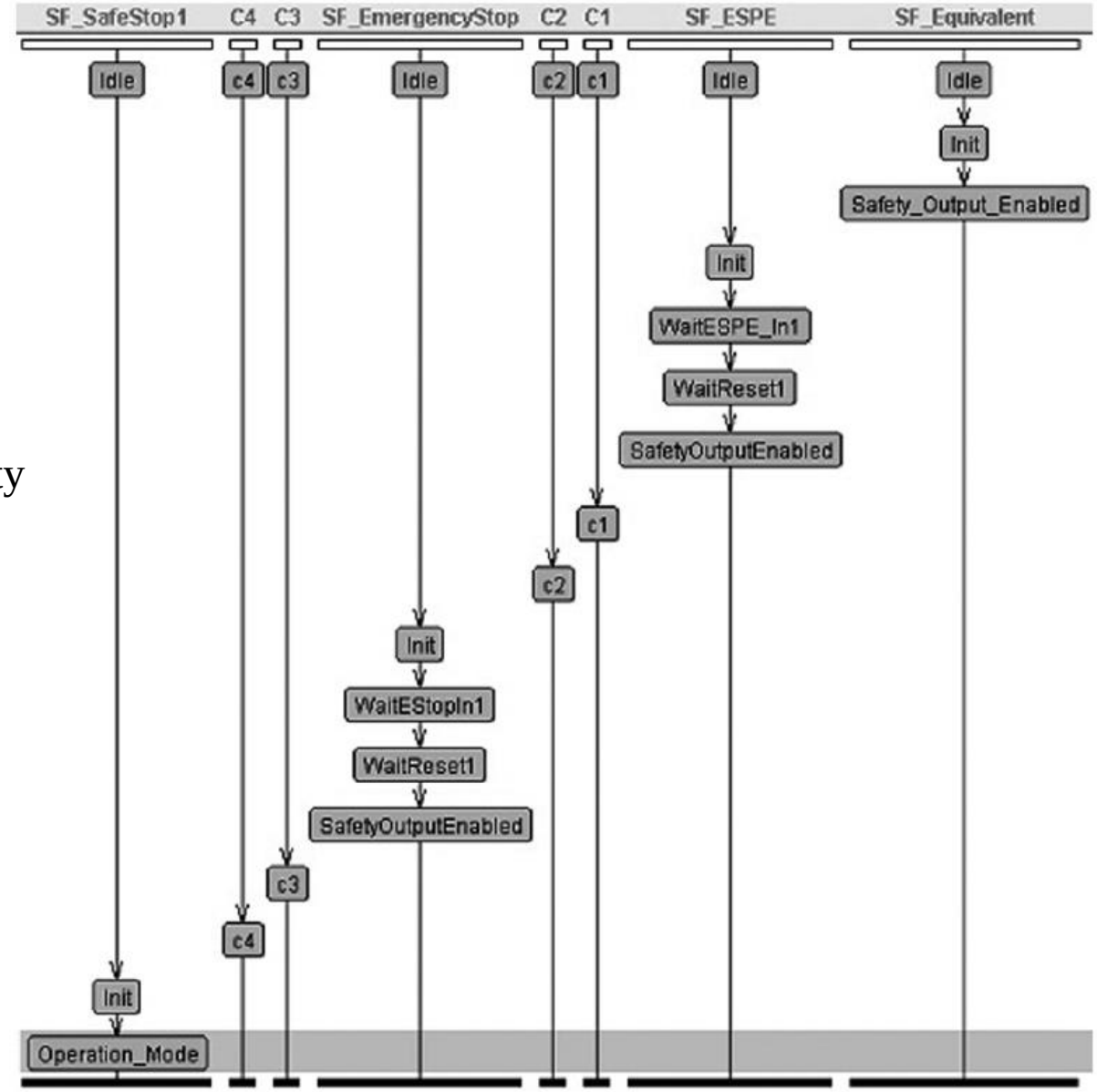


Validation

- The Inputs automaton is used to assign inputs to the safety application.
- Inputs automaton should take the highest priority in execution,
  - But that is not possible because inputs automaton runs unconditionally.
  - It has always active transitions, which means no lower priority automaton gets the chance for execution.
- To tackle this problem, Inputs automaton is given the lowest priority taking into account the initial conditions needed to start execution.
- The safety application is in normal operation state if all safety input variables are true, taking into account the required activation and reset input variables.
- Starting from SF_Equivalent at right side, the timed automata are executed sequentially with respect to priorities until reaching *Operation_Mode* state in *SF_SafeStop1*.
- According to simulation results, it is shown that the system is functioning

Validation

Simulation scenario of the safety application



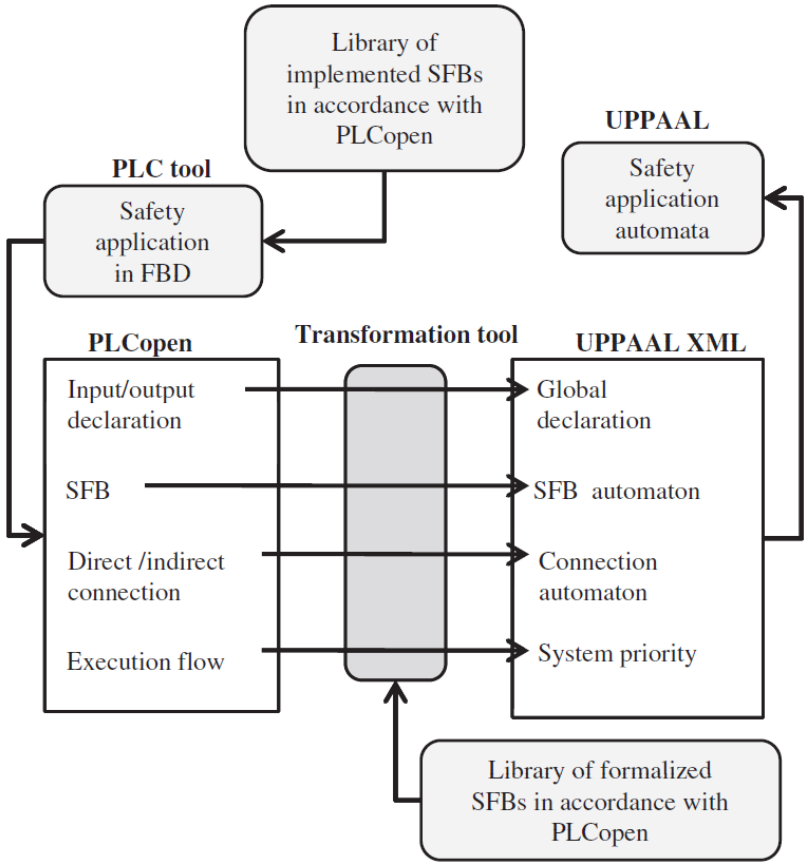
Verification

- **Property 1:** The safety application is in normal operation state if all safety input variables are true, taking into account the required activation and reset input variables.
- **Property 2:** After the activation of *SF_Equivalent* and *SF_ESPE*, the *SF_SafeStop* will keep activated.
  - If the inputs are true, then the output will be always true.
- **Property 3:** Issuing the Emergency stop (via *SF_EmergencyStop*) or interrupting the light beam in the light curtain (via *SF_ESPE*) stops the drive in accordance to stop category 1.
  - If there is a safety request to stop the drive by missing one of the safety inputs (*A*, *B* or *ESPE_In*), and the control system gives acknowledgement of stopped drive, then a safety output *S_Stopped* is issued to indicate safety stop. The other one indicates that it is not possible for the drive to stop while no safety request is needed.
- **Property 4:** The stop of the electrical drive within a predefined time is monitored (via *SF_SafeStop1*).
  - If an acknowledgement signal (Acknowledge=1) is received within specified monitoring time (ToMT), a safety stop is monitored (S_Stopped=1).

Tasks of transformation tool

- The transformation has four tasks:
  1. Transform the Input/output variables declaration of the safety application to global declaration in UPPAAL XML.
  2. Transform all SFBs presented in PLCopen XML to their corresponding automata in UPPAAL XML.
  3. Formalize a connection automaton for every direct or indirect connection between SFBs.
  4. According to the position of every SFB and the order of every connection, the execution flow is extracted from PLCopen XML.

- After transformation, UPPAAL XML file can be read by UPPAAL tool.



Summary

- Formalize an approach for the verification of safety applications built from PLCopen safety function blocks (SFBs).
- This approach was presented and illustrated using an example.
- This assumption is normally verified by certification of SFBs by independent organizations like TUV.
- The SFBs have been implemented directly in IL and verified using Model Checking.

Thank you very much for your attention !