# 767 Electric Power System Modeling in SysML

***Authors***:
*John Finn*
*Mohammad Mostafizur Rahman Mozumdar*
*Alberto Sangiovanni Vincentelli*

***University of California, Berkeley***

***Version & Date***:
Ver.02 – April 19th, 2011

***Reviews:***
Ver.01 – 02/09/2011: Preliminary version
Ver.02 – 04/19/2011: First Draft of Complete Power System

*Abstract*

This document provides a preliminary description for SysML modeling of the 767 electrical power system, a part of the design methodology to the MuSyC/DSCS avionics challenge problem. It starts with a short overview of the aircraft's electric power system and then highlights the components that have been modeled. Rational Rhapsody is used as a SysML modeling interface.

## Description of Power System

The Electrical Power System (EPS) is an important subsystem of a typical avionics vehicle management system. The scope of the EPS is to provide electrical power to the different aircraft subsystems. Typically it consists of power generators, switches, contactors and electrical loads. Figure 1 shows a simplified schema of the Boeing 767 Electrical Power System.

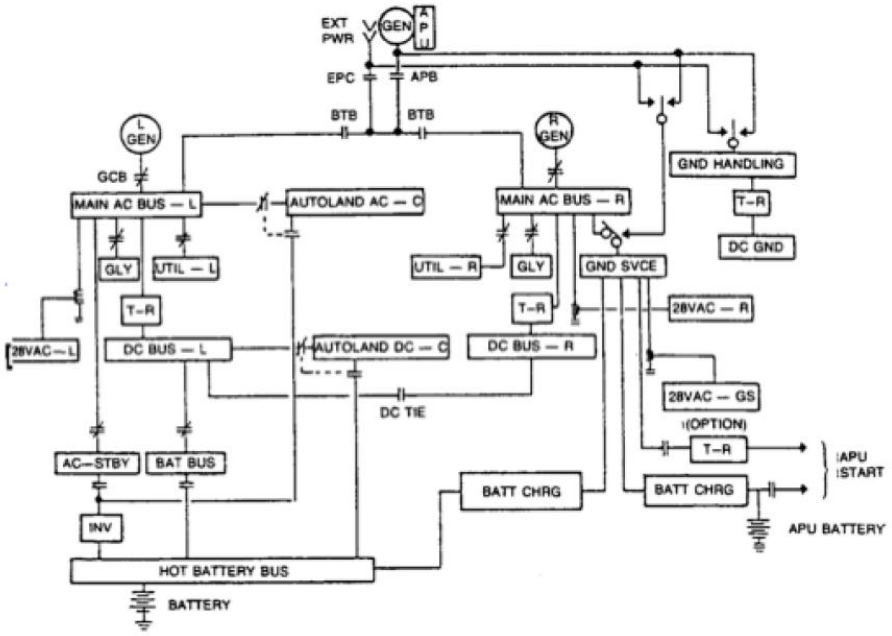


**Figure 1 – Simplified Boeing 767 Electrical Power System**

There are two generators in the aircraft that serve as a primary power sources (shown as L-Gen and R-Gen in Figure 1). Each of them provides power to their respective AC Bus through a Generator Control Breaker (GCB), which is controlled by a local Generator Control Unit (GCU). Next, each AC Bus powers the local DC Bus through a Transformer Rectifying Unit (TRU). The Bus Power Control Unit (BPCU) controls the Bus Tie Breakers (BTB), which in the event of main generator failure, the BTB allows for the other generator, Auxiliary Power Unit (APU) or the External Power (EXT) to compensate for the lost generator. Similarly, the DC Tie allows one DC Bus to compensate the other in the event of a TRU failure. Lastly, the Left DC Bus or the onboard batteries can power the Battery Bus. This primitive functionality was incorporated into SysML as described in the next section.

## Modeling Electrical Power System in SysML

The EPS can be viewed as an interaction between three entities, the pilot, control system and power system. The pilot has the ability to start/stop the right or left engine. The control system controls the power system based upon its current status and commands from the pilot. Finally, the power system generates power for the various aircraft loads from passenger lighting to anti-ice devices for the wings. This functionality is illustrated in the SysML use case diagram of Figure 2.

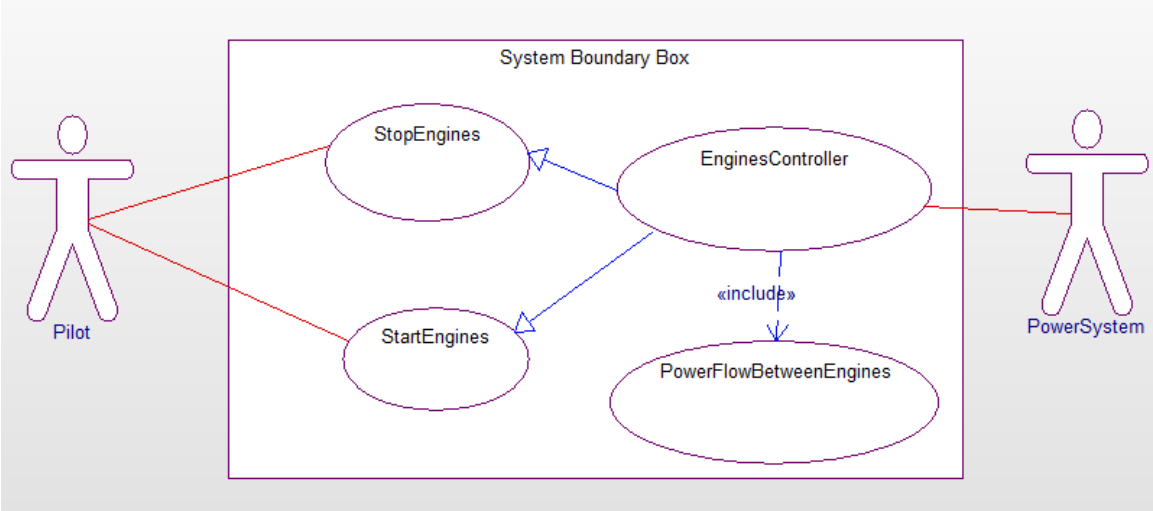


**Figure 2 - Use Case Diagram of EPS**

## System Architecture

Figure 3 illustrates the top-level view of the EPS using a block diagram, which consists of the pilot, control system, left/right side of the power system, Bus Tie Breaker, DC Tie Breaker, External and APU Power Units, AC/DC Standby Buses, Battery Bus, and the Battery. All these components contain ports by which they are connected with each other. Using ports, components can communicate with each other by means of events, which can carry specific data values.

As shown in the figure 3, the pilot sends the start/stop commands to the control system. The engine control system consists of two GCUs (one for each engine) and one BPCU (shown in figure 4). The GCU sends the start/stop command to each of the engines and receives (measures) the current generator voltage level. Additionally, the BPCU controls the BTB. In the current implementation, each side of the power system contains blocks for engine, AC Bus and DC Bus (shown in figure 5).

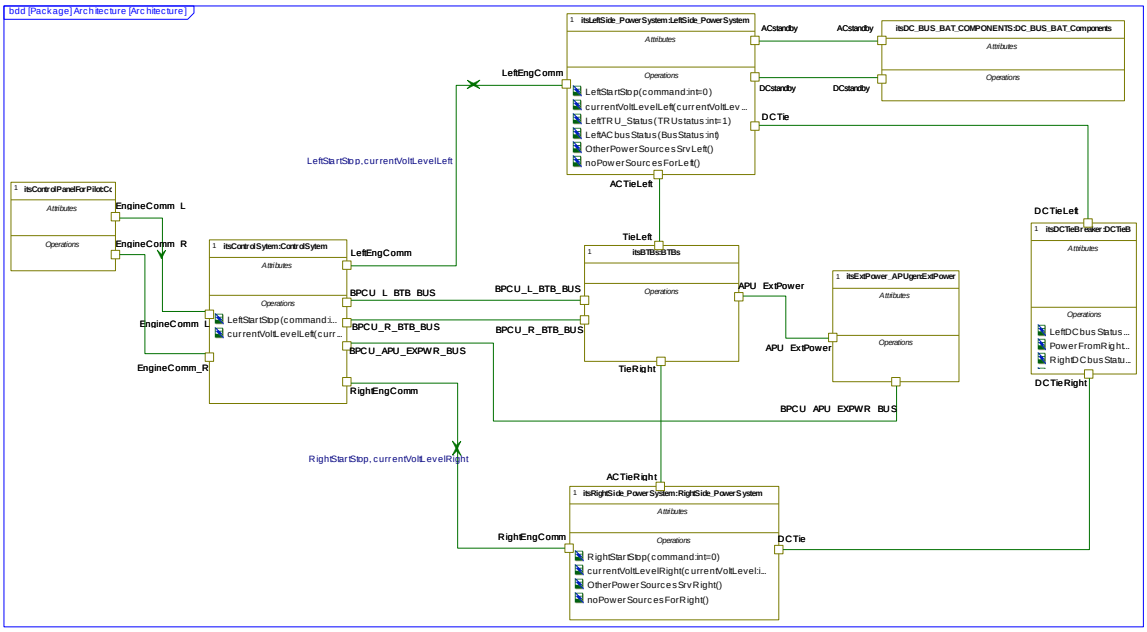


**Figure 3 - System Architecture**

## Control System

As mentioned earlier, the control system in Figure 4 has three components, a GCU for each engine and a BPCU for the BTB. Each GCU is connected to its corresponding engine through a flow port by which it sends and receives data and events. The GCU can start/stop its corresponding engine based on commands from the pilot. In addition, the GCU monitors the generator's output voltage, and if this voltage is outside a specified range, the GCU will shut down the engine. If an engine is shut down by the GCU, the GCU signals the BPCU to configure the BTBs to compensate for the lost engine. The configuration of the BTBs depends on the status of the available power sources, which are EXT, APU or the other generator (in that priority order).

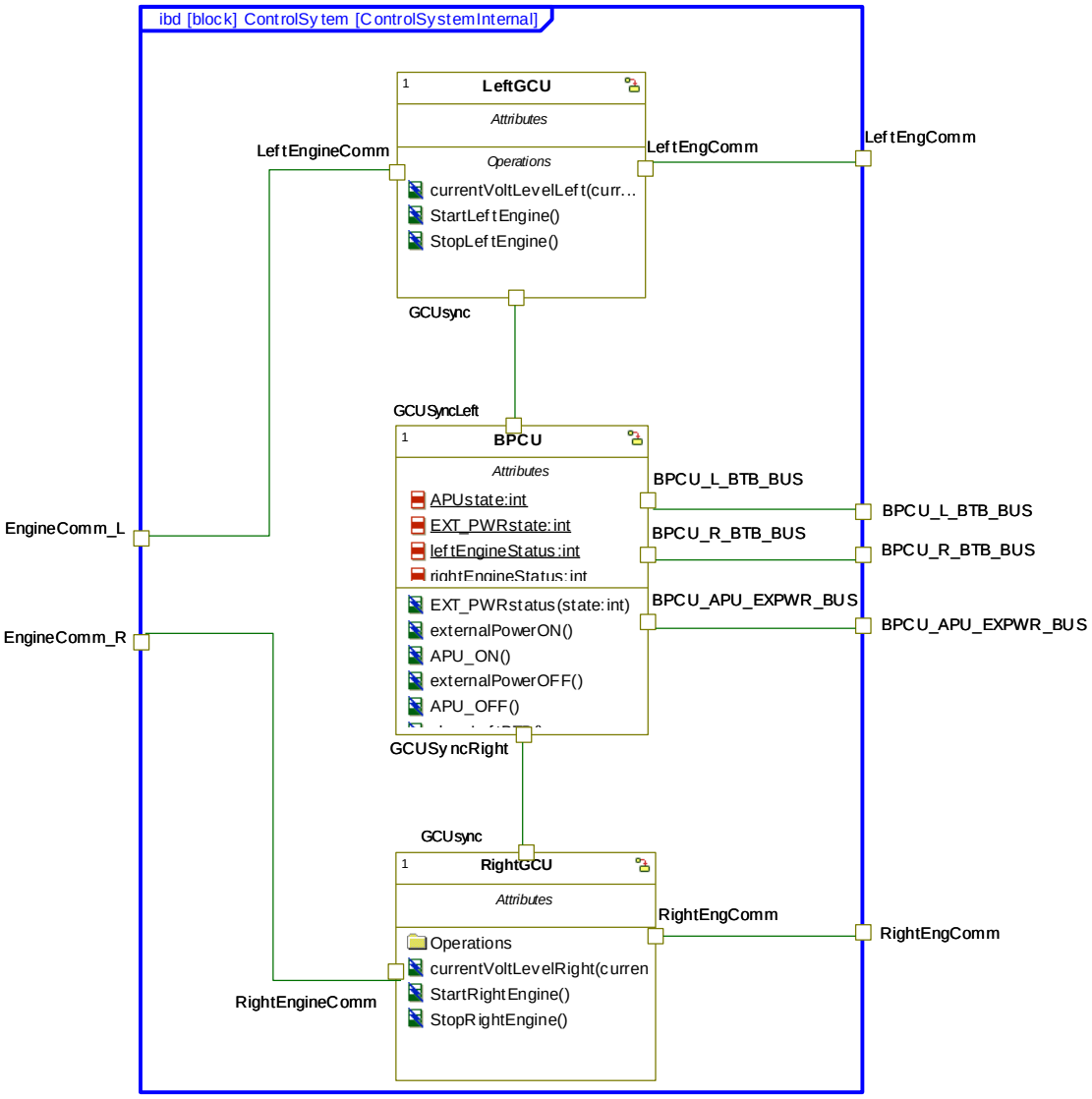


**Figure 4 - Internal Block Diagram of Control System**

## Left/Right Power System

Each side of the power system contains three blocks including the engine, AC Bus and the DC Bus as shown in Figure 5, which illustrates the left side. These blocks communicate through ports, which send and receive events. The engine acts as the generator, which when active, generates an output voltage to power the AC bus, which in turn powers the DC Bus through the TRU. The GCUs control the engines (via the *LeftEngComm* port) based on the pilot's commands and whether or not a satisfactory voltage is being supplied. Additionally, both the AC and DC Buses have ports to their respective Tie Buses in the event of a system failure.

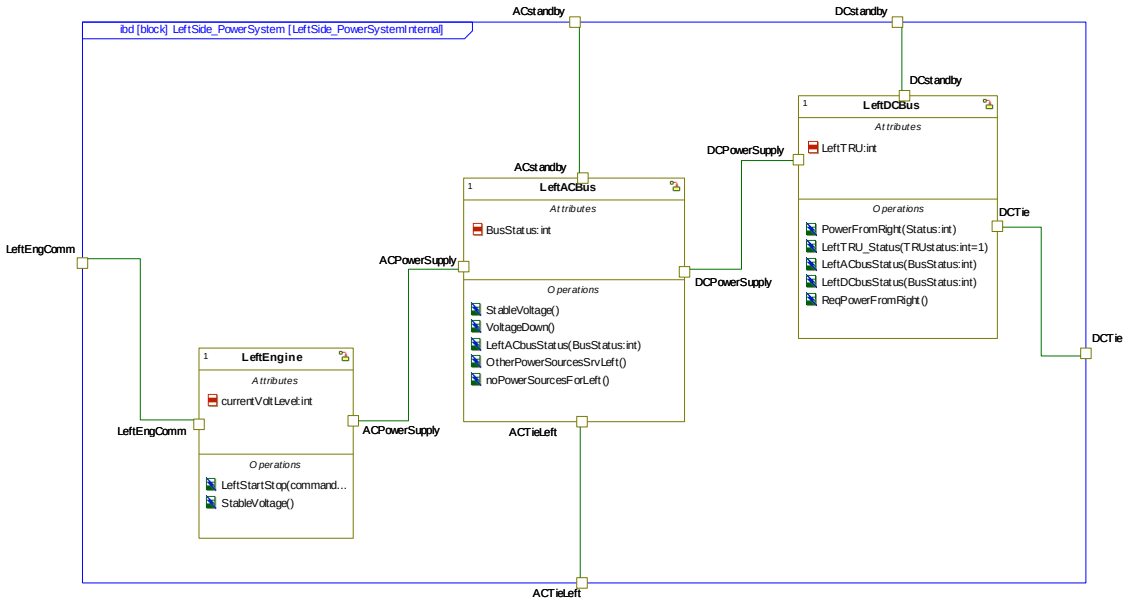


**Figure 5 - Internal Block Diagram of Left Power System**

## Generator Control Unit- GCU

The state flow shown in the Figure 6 (left side) illustrates the operation of each GCU. Initially, the GCU is in the idle state waiting for a start command from the pilot (*StartLeftEngine)*. Once the start command is issued, the GCU will transition to the *StartEngine(Left/Right)* state. On entry to this state, the GCU sends the start command to the engine. For example to send the start command from the left GCU, it uses following action language of SysML,

OUT_PORT(LeftEngComm)->GEN(LeftStartStop(1))

Here, *LeftEngComm* is the flow port between the left GCU and the left side of the power system (see Figure 3), and *LeftStartStop* is the event with argument value of 1 (1= start, 0=Stop). Once in the *StartEngineLeft* state, the GCU will transition to the *LeftEngineStarted* state if the output voltage from the engine is acceptable. If not, the statechart will transition to the *StopEngine* state. While in the *LeftEngineStarted* state and the voltage suddenly become unacceptable or a stop command is received, the statechart will transition to the *StopEngine* state. From here, the engine can transition back to the *StartEngineLeft* state with the *LeftStartStop* event from the pilot.

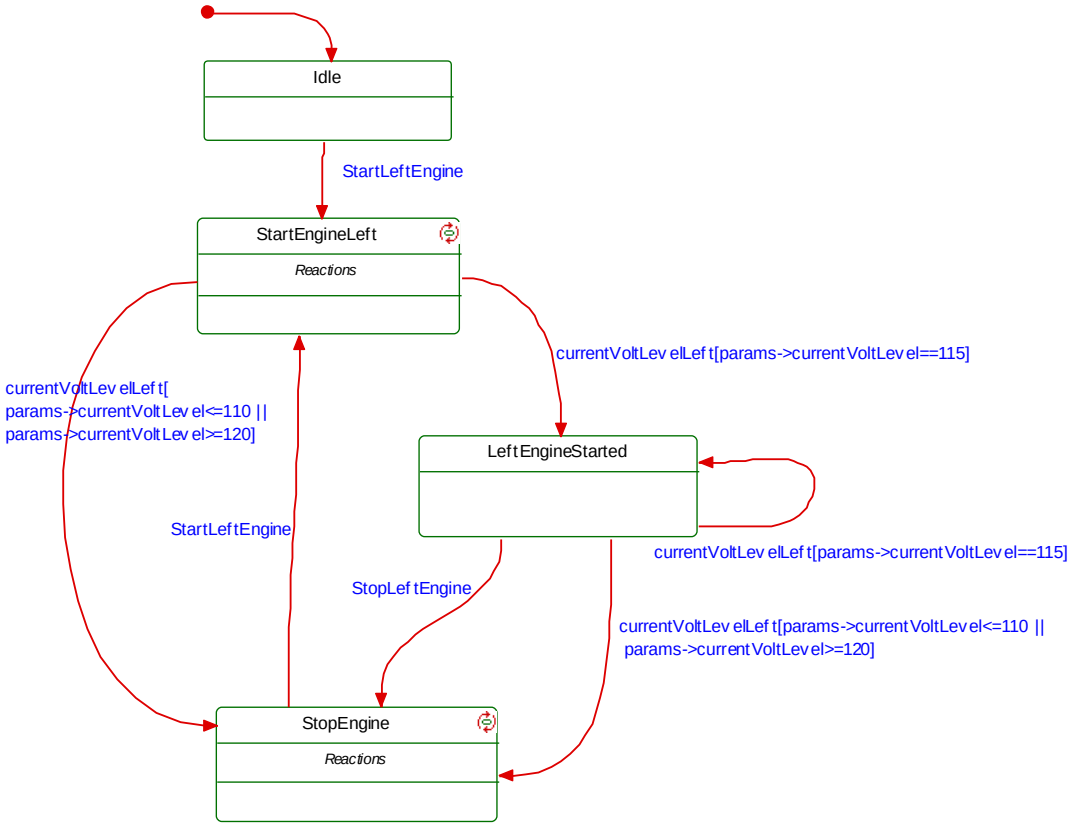


**Figure 6 - Statechart Implementation of Left GCU**

## Engine

Initially, each engine waits for the start command in the *idle* state. Once the GCU sends the start command (for example, LeftStartStop(1) from the left GCU), the engine will make transition to the *EngineStarted* state. Once started, the engine will send its output voltage to the GCU every 1000ms via the *LeftEngComm* port. On entry to the *EngineStarted* state, the engine sends an event to the AC Bus indicating its operating voltage level, which based on its status; this event is either *StableVoltage* or *VoltageDown* via the *ACPowerSupply* port. Lastly, if the GCU sends a stop command (for example, LeftStartStop(0) from the left GCU), the engine will transition to the *idle* state and send *VoltageDown* to the AC Bus.

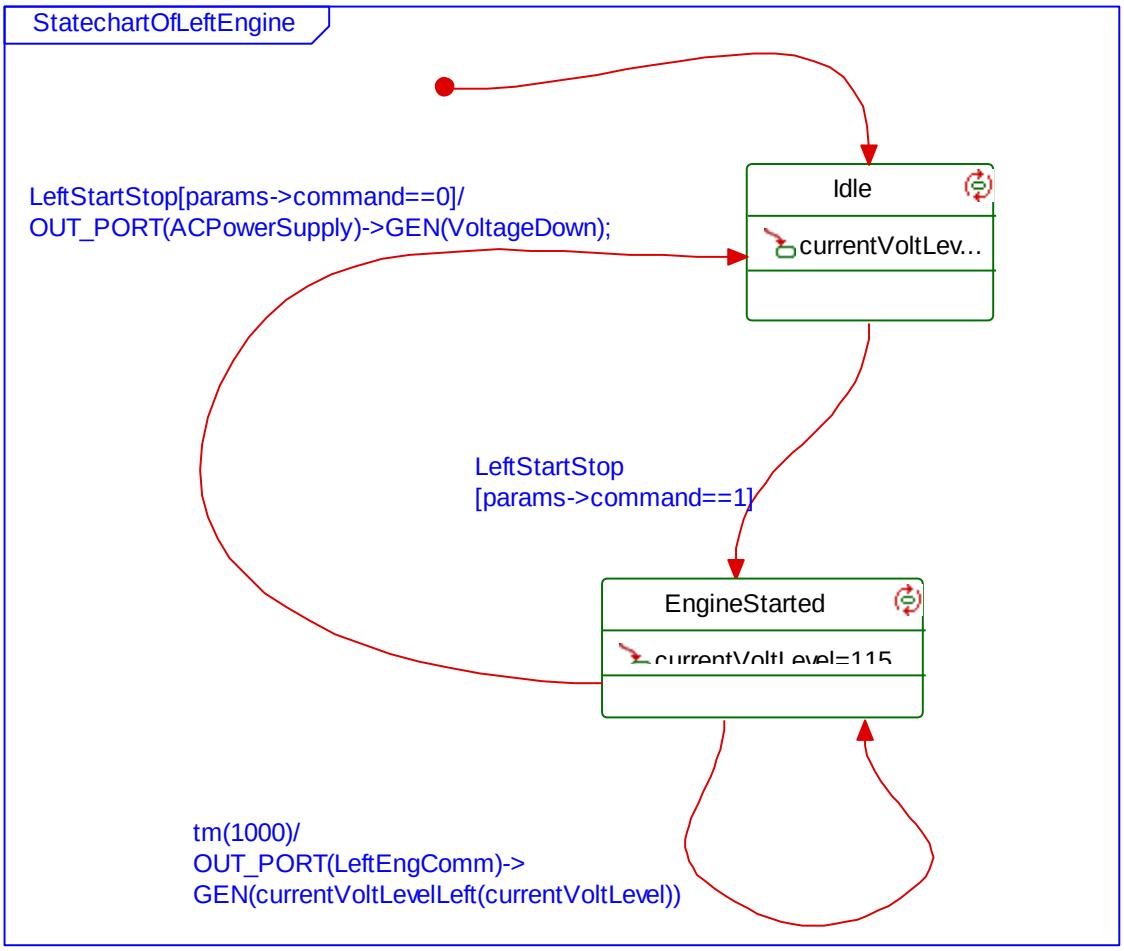


**Figure 7 - StateChart Implementation of Left Engine**

### AC Bus

The state flow in Figure 8 illustrates the Left AC Bus. In this state flow, the Left AC Bus is initially in the *NotInService* state, which implies the AC Bus is not powered. On entry to this state, the AC Bus sends an event to both the DC Bus and AC Standby Bus, which indicates the AC Bus is not in service, and therefore, the DC Bus cannot be in service either, but the AC Standby Bus can be in service from the Battery Bus. Once the left generator produces a stable voltage, the AC Bus will transition to the *InService_PowerFromLeftGenerator* state. If the engine fails, then the *VoltageDown* event generated by the GCU will transition the AC Bus back to the *NotInService* state. When the BPCU closes the BTBs, the *OtherPowerSourcesSrvLeft* event will transition the AC Bus to the *InService_PowerFromOtherSources* state, which implies either the EXT Power, APU or the Right AC Bus is supplying power to the Left AC Bus. When the *NotInService* state is exited, the AC Bus sends an event to both the DC Bus and AC Standby Bus, which indicates the AC Bus is in service, and therefore, the DC Bus and AC Standby Bus can be in service as well.

The same state flow was implemented for the Right AC Bus, except there is no Standby Buses on the Right Side, and therefore, there is no need to report its status to the Standby Bus.

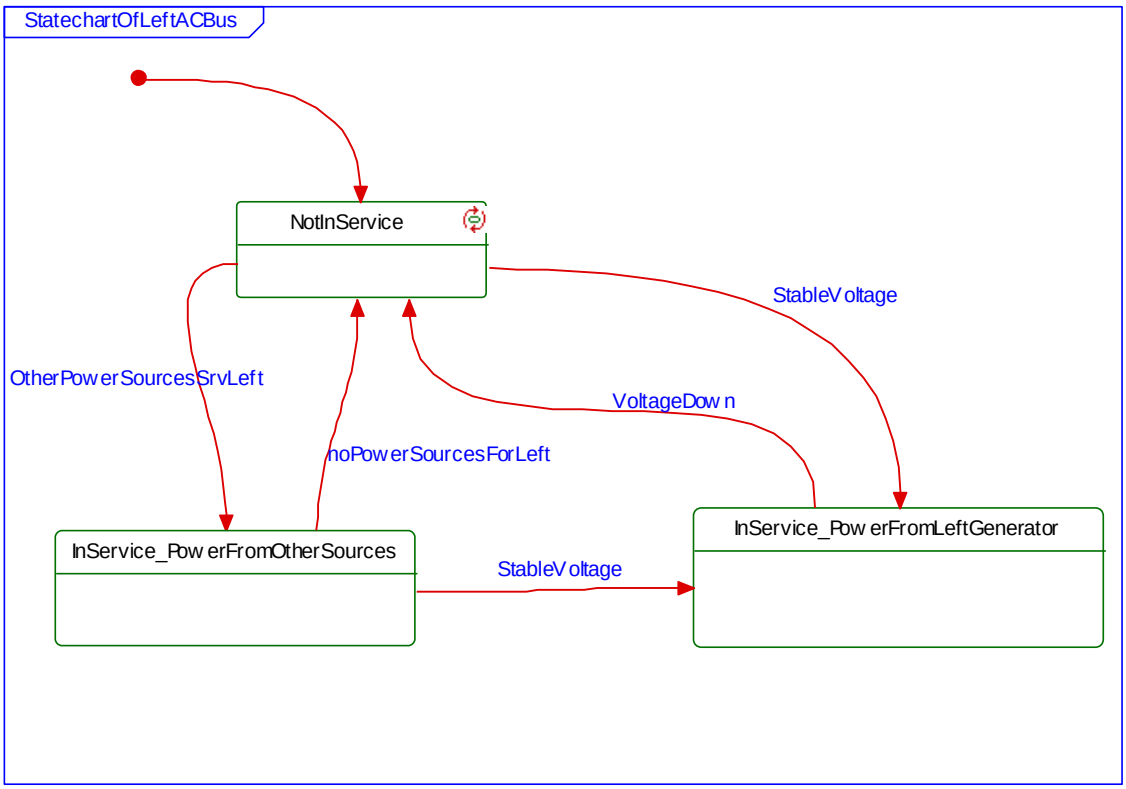


**Figure 8 - StateChart Implementation of AC Bus**

## DC Bus

The state flow in Figure 9 illustrates the Left DC Bus. The functionalities of the DC Bus are implemented using two parallel state machines. The first state machine (left side) represents the actual DC Bus, and the second one updates the TRU status.

Initially, the Left DC Bus is in the *LeftDC_NotInService* state. On entry to this state, the Left DC Bus informs the DC Standby Bus and DC Tie Breaker that its status is "unpowered" via the *LeftDCbusStatus(0)* event. The *LeftACbusStatus(1)* event transitions the state flow to the *CheckTRU* state, which checks the status of the Left TRU. If the Left TRU is functioning properly (i.e. LeftTRU = 1), then the Left DC Bus is powered from the Left TRU, which is signified by the *InServiceFromTRU* state. In the event of TRU failure (i.e. LeftTRU = 0), the Left DC Bus transitions back to the *LeftDC_NotInService* state, and sends the *ReqPowerFromRight* event to the DC Tie Breaker. The same event occurs if the TRU fails while in the *InServiceFromTRU* state. When the DC Tie Breaker receives the request, it can close the DC Tie Breaker,

which will allow the Right DC Bus to power the Left DC Bus. In this case, the DC Tie Breaker will send the *PowerFromRight(1)* event, which signifies the DC Tie Breaker is closed and the Left DC Bus will transition to the *InServiceFromRight* state. If the Right DC Bus goes down, the *PowerFromRight(0)* event will transition back to the *LeftDC_NotInService* state. On entry to either the *InServiceFromTRU* or *InServiceFromRight*, the DC Bus informs the DC Standby Bus and DC Tie Breaker that its status is "powered" via the *LeftDCbusStatus(1)* event.

On the right side of state flow, the Left TRU's status is determined. Initially, the Left TRU is functioning properly, which is designated by the *LeftTRU_ON* state. This state sets the LeftTRU variable mentioned above to one. The *LeftTRU_Status(0)* events transitions the Left TRU to the failed state, *LeftTRU_OFF* state, which sets *LeftTRU* to zero.

The same state flow was implemented for the Right DC Bus, except there is no Standby Buses on the Right Side, and therefore, there is no need to report its status to the Standby Bus.

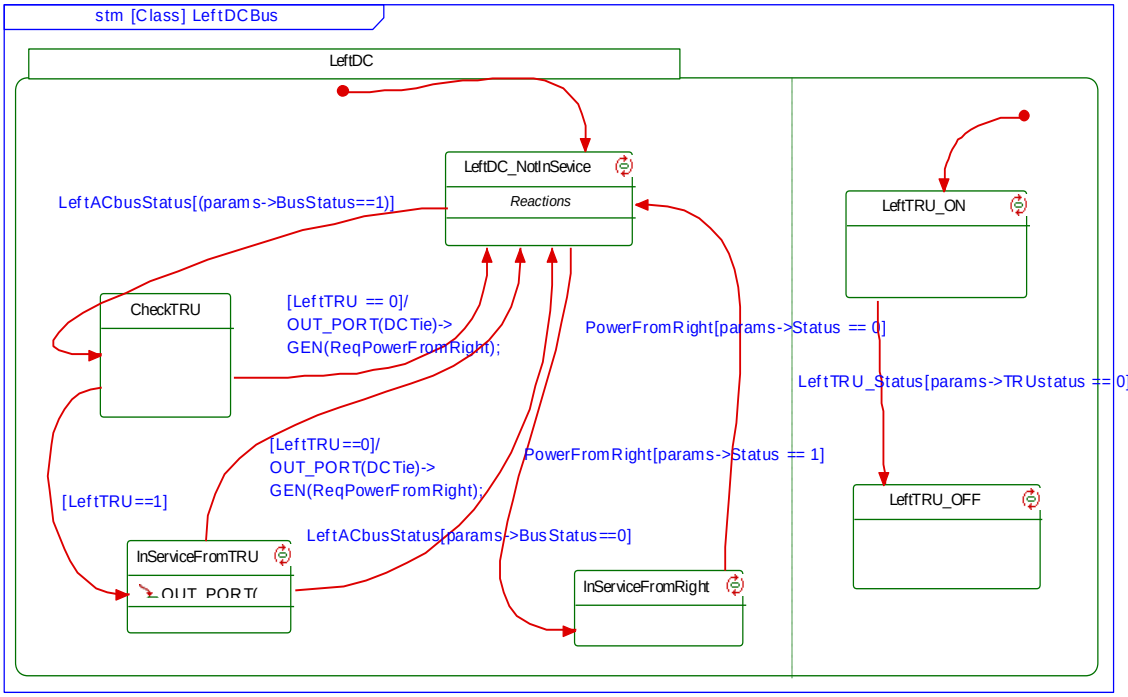


**Figure 9 - StateChart Implementation of DC Bus**

## DC Tie Breaker

The DC Tie Breaker state flow is shown in Figure 10. The two state flows on the right side, update the status of the Left and Right DC Buses. If the Right Bus is powered, then the value of *RightDC* is one, otherwise zero. *LeftDC* is updated

identically. These updates are triggered by the *LeftDCbusStatus({0,1}) and RightDCbusStatus({0,1})* mentioned in the DC Bus section above.

The state flow on the left allows the DC Tie Breaker to open and close based on the requests and the status updates from Right and Left DC Bus. Initially, the DC Tie Breaker is open. If the DC Tie Breaker receives the *ReqPowerFromRight* event, the state flow transitions to the *CheckRightDC* state, which checks the status of the Right DC Bus. If it is powered (i.e. *RightDC* = 1), then the state flow transitions to the *SendPowerFromRight* state which sends the event *PowerFromRight* to the Left DC Bus. However, if the Right DC Bus is unpowered (i.e. *RightDC* = 0) while in either the *SendPowerFromRight* or *CheckRightDC* state, the DC Tie Breaker transitions back to the *Open* state. The DC Tie Breaker behaves identically if the *ReqPowerFromLeft* is received while in the *Open* state.

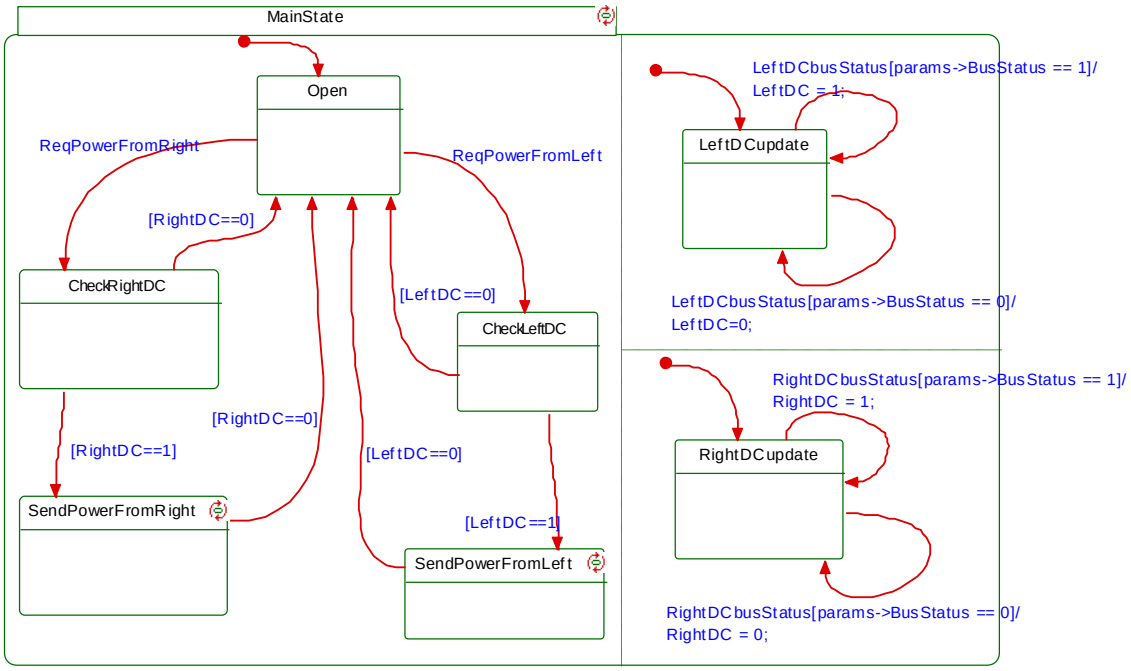


**Figure 10 - StateChart Implementation of DC Tie Breaker**

## BPCU

The BPCU state flow implementation of Figure 11 consists of three parallel state machines. The two statecharts on the right get the status of the APU and EXT Power. These statuses are updated by events from the corresponding APU/EXT block within the Architecture of Figure 3 via the *BPCU_APU_EXPWR_BUS* port. Figure 12 shows the APU status update state flow, which sets the variable *APUstate* according to the status of the APU, which is either ON (*APUstate* = 1) or OFF (*APUstate* = 0). Figure 13 shows the EXT Power status update state flow, which sets the variable *EXT_PWRstate* according to the status of the EXT Power, which is either ON (*EXT_PWRstate* = 2), OFF (*EXT_PWRstate* = 0) or AVAIL (*EXT_PWRstate* = 1). The

AVAIL state indicates EXT Power is available, but not being used while the ON state indicates the EXT Power is being used to power the AC Bus(s).

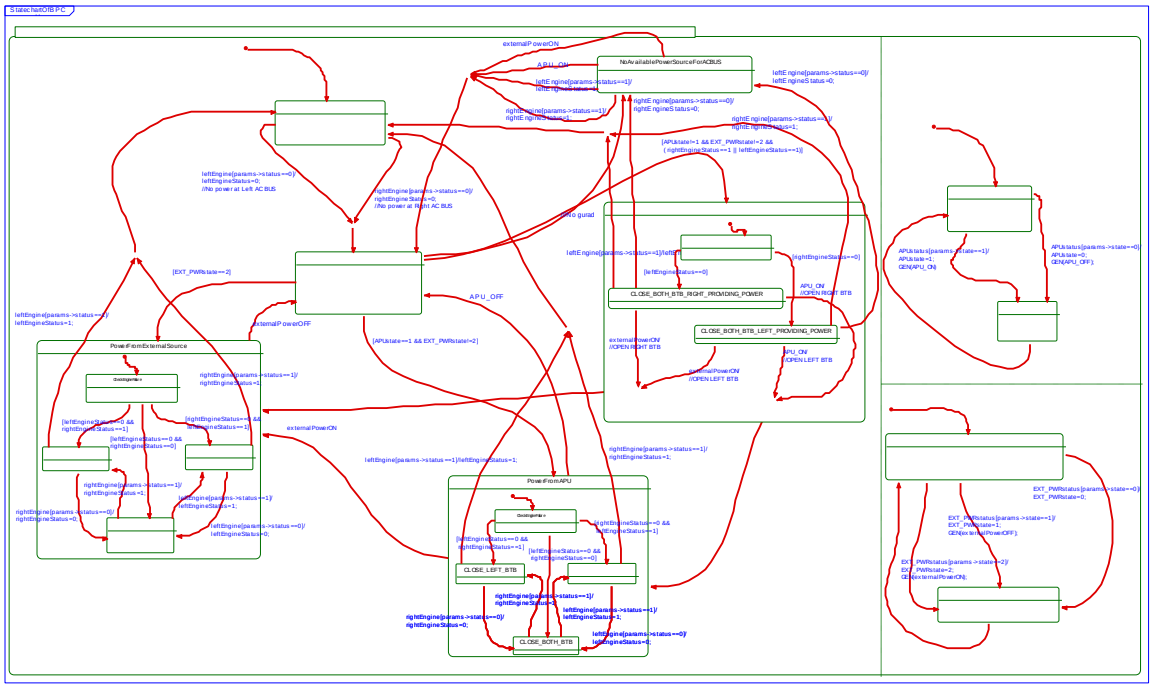


**Figure 11 - StateChart Implementation of the BPCU**

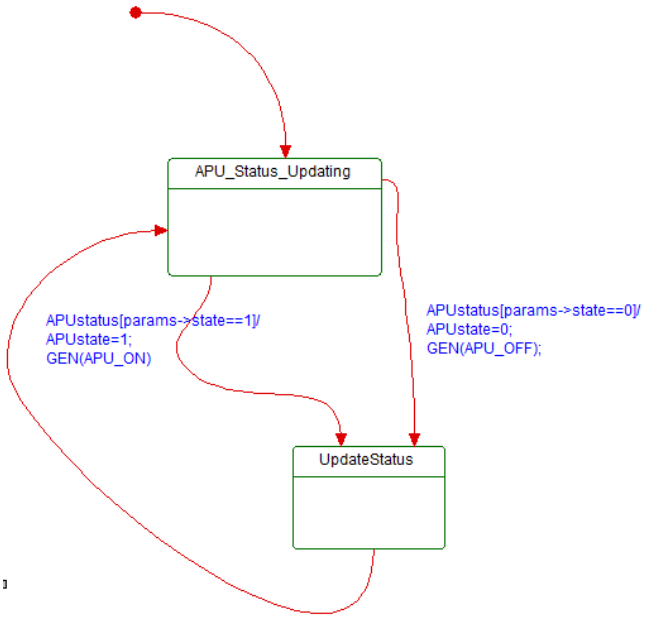


**Figure 12 - StateChart Implementation of the APU Status Update within the BPCU**

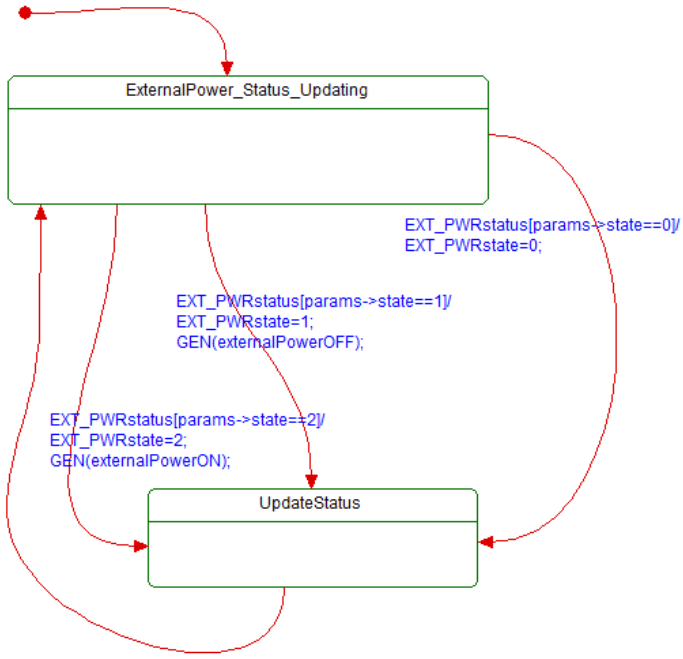


**Figure 13 - StateChart Implementation of the EXT Power Status Update within the BPCU**

The large state flow on the left side of the entire BPCU state flow can be divided into five states including three "super states" each indicating the flow of power and the BTB configuration. Recall the priority to power a given AC Bus is EXT Power, local engine, APU and opposite engine. The initial state, *Both_BTB_Open*, signifies both of the BTBs are open and the local engine powers each AC Bus. In the event either engine fails, the state flow will transition to the *AlternativePowerSource*, which is a kind of "junction state" that enforces the priority mentioned above and based on the statuses of the alternative power sources, determines which source will power the lost channel. First, if no alternative power source is available, it will transition to the *NoAvailablePowerSourceForACBus* state, which indicates the lost channel will remain unpowered. Next, each of the "super states" represents alternative power sources which will be explained in the following paragraphs.

As far as the BTBs are concerned, the APU and EXT Power have the same BTB configuration for each of the given scenarios. Figure 14 shows the APU "super states" of the BPCU state flow. The state flow of the EXT Power has modeled in similar fashion. The initial state, *CheckEngineFailure*, checks the status of each engine to determine which BTBs to close. An engine status of *one* indicates proper operation while *zero* indicates failure. If only the Right Engine is lost, the Right BTB will close as indicated by the *Close_Right_BTB* state. Similarly, if only the Left Engine is lost, the Left BTB will close as indicated by the *Close_Left_BTB* state. However, if both engines failed, both BTBs will close as indicated by the *Close_Both_BTB* state.

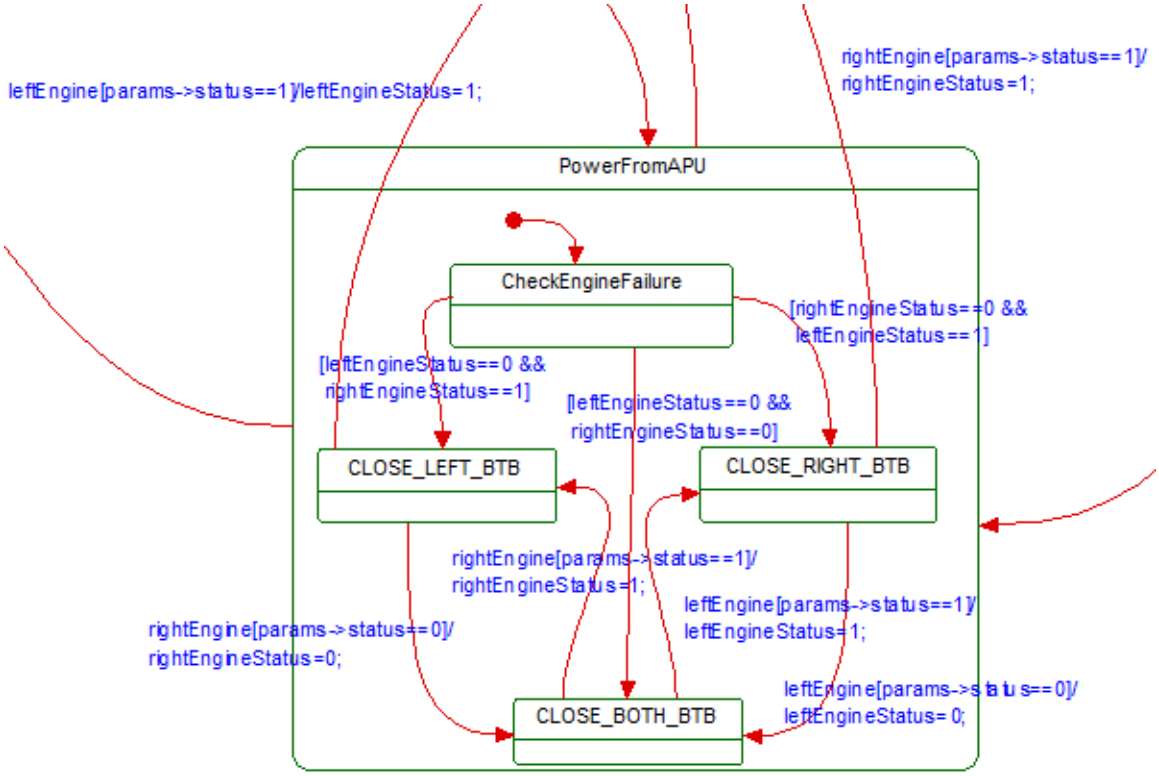


**Figure 14 - StateChart Implementation of the APU Alternative Source within the BPCU**

Figure 15 shows the "super state" in which one engine compensates for the other. In this case, both BTBs must close regardless of which engine fails. The only thing that changes in this scenario is the direction of power flow. If the Left Engine fails, the Right Engine provides power to the Left AC Bus and vice versa if the Right Engine fails.

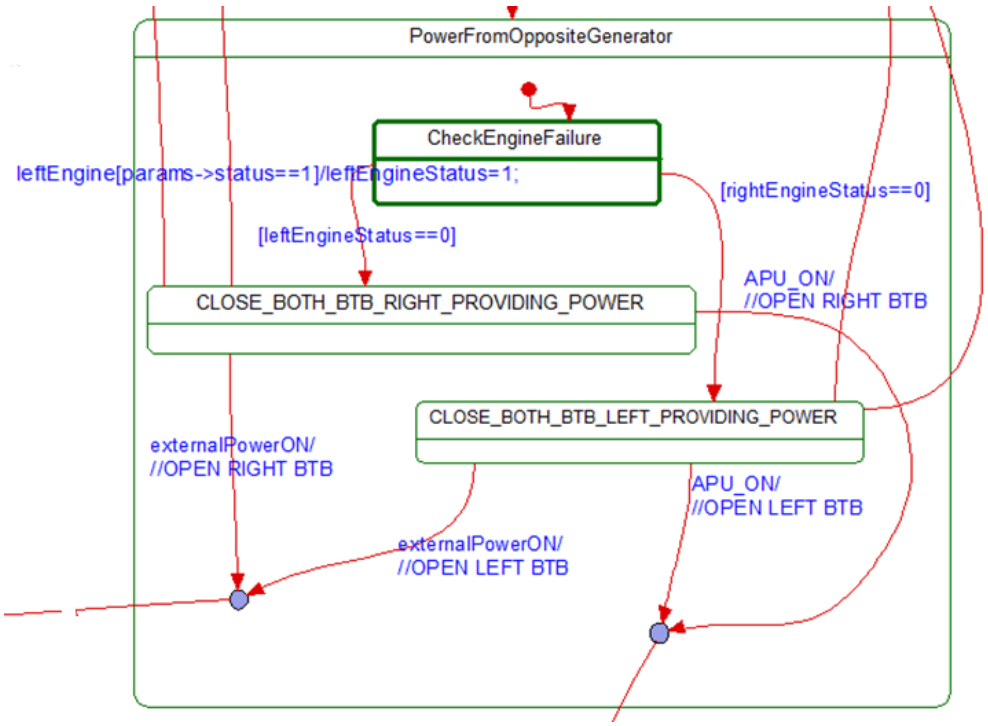


**Figure 155 - StateChart Implementation of the Other Engine Alternative Source within the BPCU**

When either the Left or Right BTB is command to be opened or closed, the appropriate event (i.e. *openLeftBTB*, *closeLeftBTB*, etc.) is sent from the BPCU to the Bus Tie Breaker in the Architecture (Figure 3) via the corresponding port( *BPCU_L_BTB_BUS*, *BPCU_R_BTB_BUS)*.

The last thing to note about the BPCU state flow is that there are transitions between the five main states mentioned above. The transitions correspond to the event when a source of higher priority is suddenly available in which there is a transition from the lower priority source to the higher priority source.

**Bus Tie Breakers Internal Block Diagram**

The Bus Tie Breaker (BTB) internal block diagram is shown in Figure 16. As previously mentioned, the BTB consists of two contactors, the Right BTB and the Left BTB, shown in Figure 16. The ports *BPCU_L_BTB_BUS* and *BPCU_R_BTB_BUS* receive open/close commands from the BPCU as mentioned in the BPCU section. The *TLeft(TieLeft)* and *TRight(TieRight)* either send power to or receive from the corresponding AC Bus. Lastly, the *APU_ExtPower* port receives power from the APU, EXT Power or both, which can be distributed to one or both of the AC Buses.

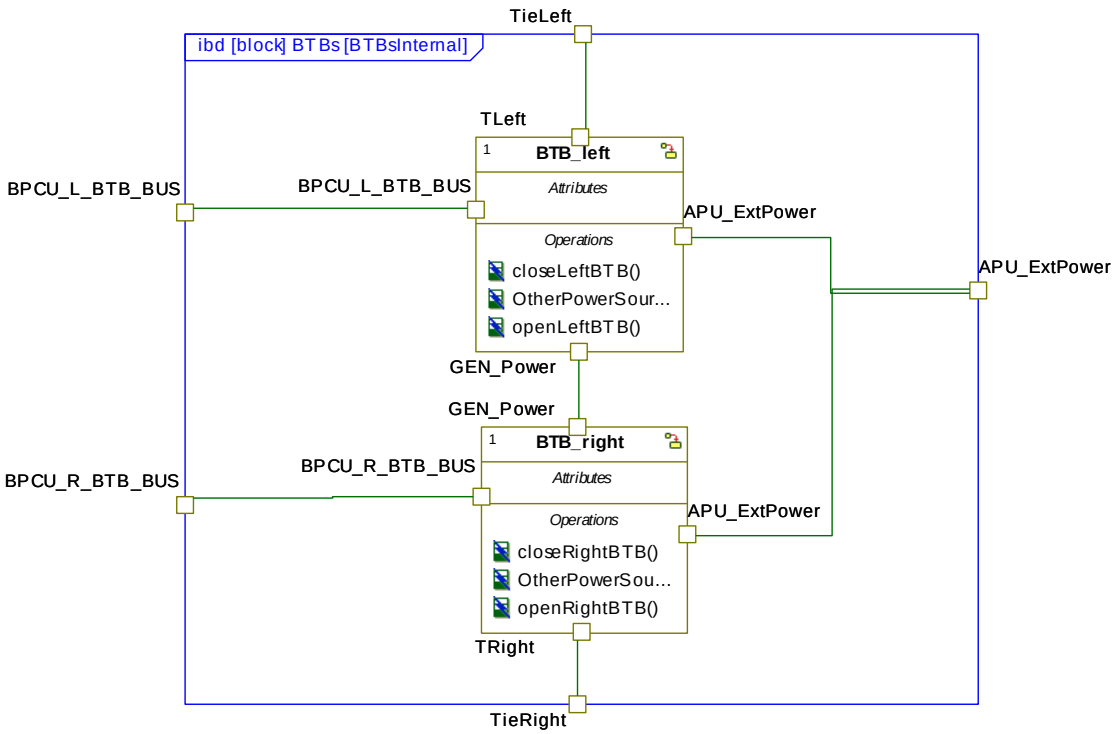


**Figure 166 – Internal Block Diagram of the Bus Tie Breaker**

## Right/Left Bus Tie Breaker State Flow

Figure 17 illustrates the simple state flow diagram of the Left BTB. Initially, the Left BTB is open. It will close to the corresponding state on the *closeLeftBTB* event. Likewise, the *openLeftBTB* event, will transition back to the initial state. The Right BTB has an identical implementation.

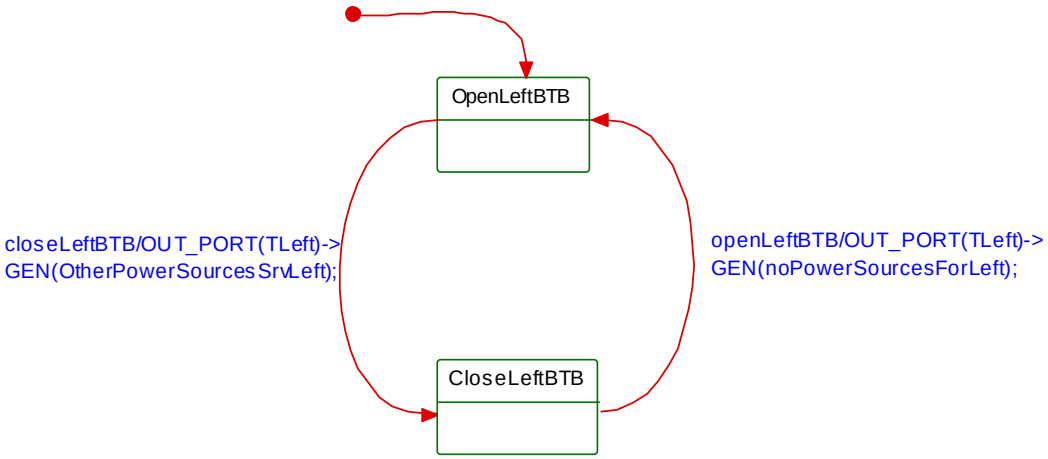


**Figure 177 – Right/Left Bus Tie Breaker State Flow**

## External Power and Auxiliary Power Unit Internal Block Diagram

Figure 18 shows the internal block diagram of the ExtPower_APUgen block in the Architecture (Figure 3). The *BPCU_APU_EXPWR_BUS* port allows the BPCU to receive the status of both the APU and EXT Power. The *APU_ExtPower* port allows the Bus Tie Breaker (in Architecture) to receive power from the APU or EXT Power.

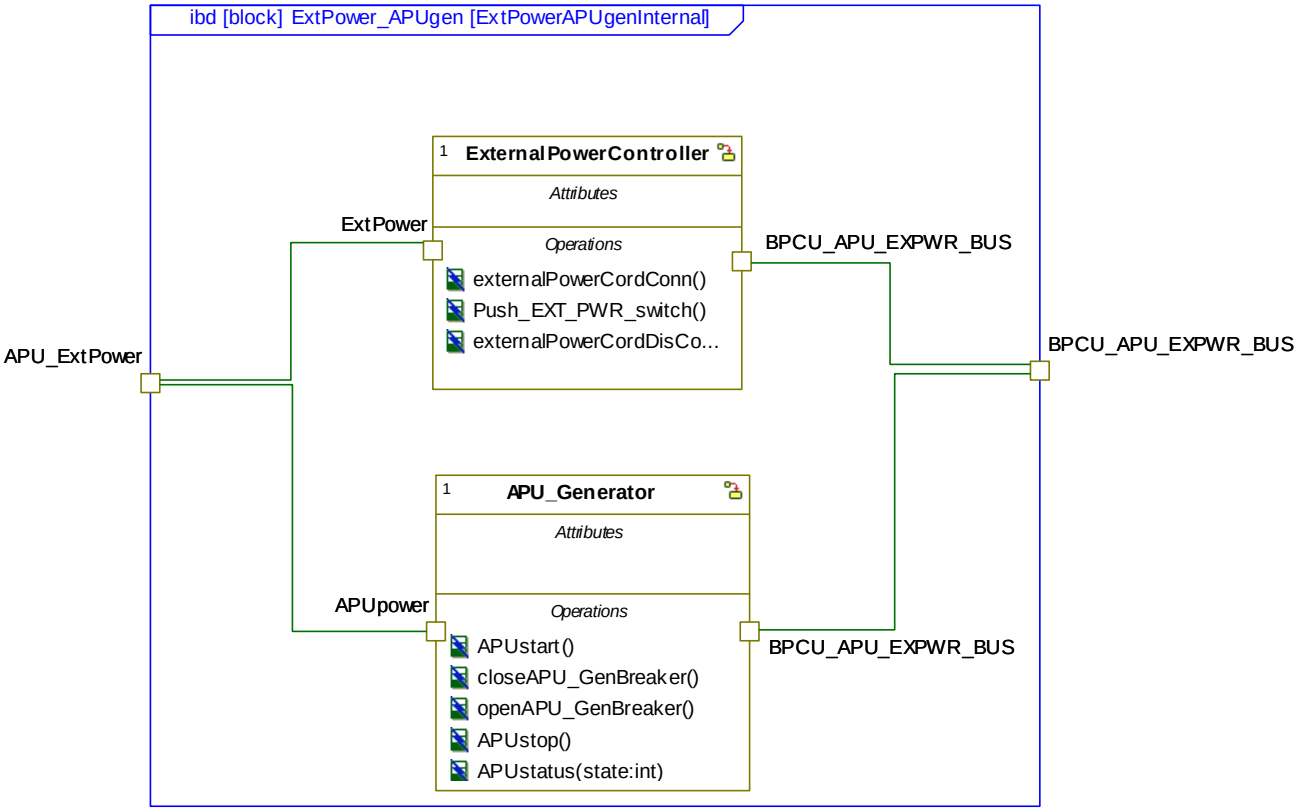


**Figure 188 – APU and External Power Internal Block Diagram**

## External Power

The transitions of states for External Power can be seen in Figure 19, in which it updates the BPCU with the EXT Power status via the *BPCU_APU_EXPWR_BUS* port. Initially, the state flow is in the *NoExternalPower* state, which indicates External Power is not available and *EXT_PWRstatus* is set to zero and sent to the BPCU. The event, *externalPowerCordConn* signifies the external power is available, but is not supplying power to any of the aircraft's buses because it requires a pilot command to supply power to the aircraft loads. This event reports *EXT_PWRstatus*=1 to the BPCU. The state flow will transition back to the *NoExternalPower* state if EXT Power is removed by the *externalPowerCordDisConn* event. When in the *AVAIL* state, If the pilot pushes switch to supply external power (*Push_EXT_PWR_switch*) to the required part of the power system buses, the EXT Power will be ready to supply

power to the aircraft loads and *EXT_PWRstatus*=2 is sent to the BPCU. The EXT Power state flow transitions to the *ProvidePowerToEntireSystem* state. If the pilot sends the *Push_EXT_PWR_switch* event in this state, EXT Power will transition back to the *AVAIL* state and send the BPCU *EXT_PWRstatus*=1.

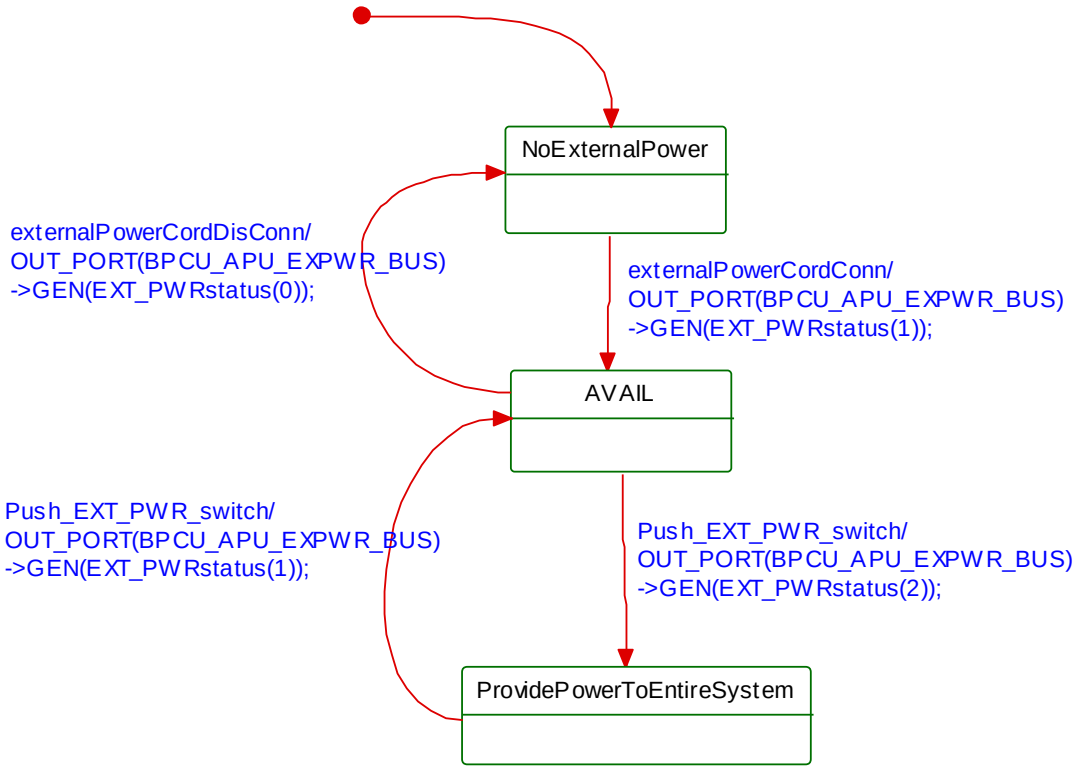


**Figure 19 –External Power State Flow**

## Auxiliary Power Unit

The APU state flow can be seen in Figure 20. This state flow has the responsibility to update the BPCU with the APU status via the *BPCU_APU_EXPWR_BUS* port. Initially, the state flow is in the *Off* state, which indicates the APU is not available and *APUstatus* is set to zero and sent to the BPCU. The event *APUstart*, starts the APU by transitioning the state flow to the *On* state and sending *APUstatus*=1 to the BPCU.

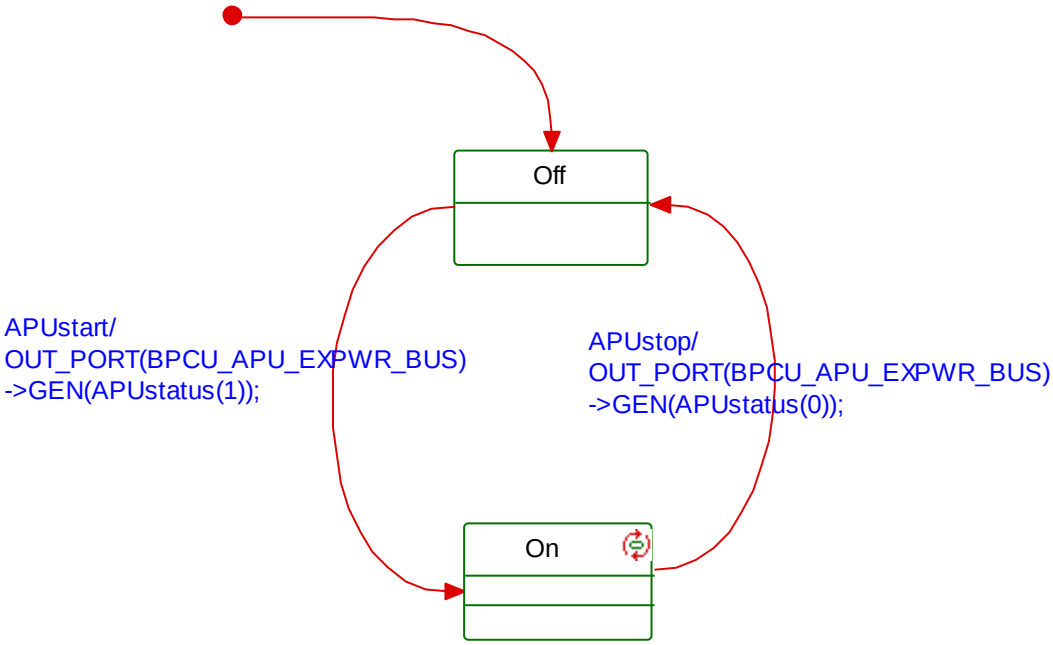


**Figure 20 –APU State Flow**

**AC/DC Standby Bus**

The AC/DC Standby Bus has three modes. Mode 0 is off, which indicates the Standby Buses are unpowered. Mode 1 is AUTO, which powers the AC/DC Standby Buses from the Left AC and DC Buses, respectively. However, if the AC/DC Buses are unpowered while in Auto Mode, the Battery Bus will power the DC and AC Buses (via an inverter). Mode 2 indicates the Battery Bus must power the Standby Buses.

Figure 21 depicts implementation of the AC/DC standby Bus, which has four parallel state flows on the right side and one on the left side. On the right side, the top two get the status of the Left AC and DC Buses (one for powered and zero for unpowered) via the *LeftACbusStatus* and *LeftDCbusStatus* events. The third updates the operating mode, which is set by the pilot via the *StandbyMode* event. The last one gets the status of the Battery Bus via the *BatteryBusStatus* event.

On the left state machine, initially it is in the *NotInService* state. If both the Left AC and DC Buses are powered and mode is in AUTO, then the state flow transitions to the *InServiceFromLeft* state. On entry and exit of this state, an event, *ACDCbusPower*, is sent to the Battery Bus, which indicates whether or not the Left Side is providing power to the Standby Buses. However, if any of these variables change, the state flow will transition back to the *NotInService* state. If the Battery Bus is powered, and the mode is BATT, then the state flow will transition to the *InServiceFromBatBus* state. It will also transition to this state if in the AUTO mode and either the Left AC or DC Bus is unpowered and the Battery Bus is powered. However, if the mode changes to OFF or the Battery Bus becomes unpowered it will transition back to the

*NotInService* state. Or, if the mode becomes AUTO and both the Left AC and DC buses become powered, it will transition to the *InServiceFromLeft* state.

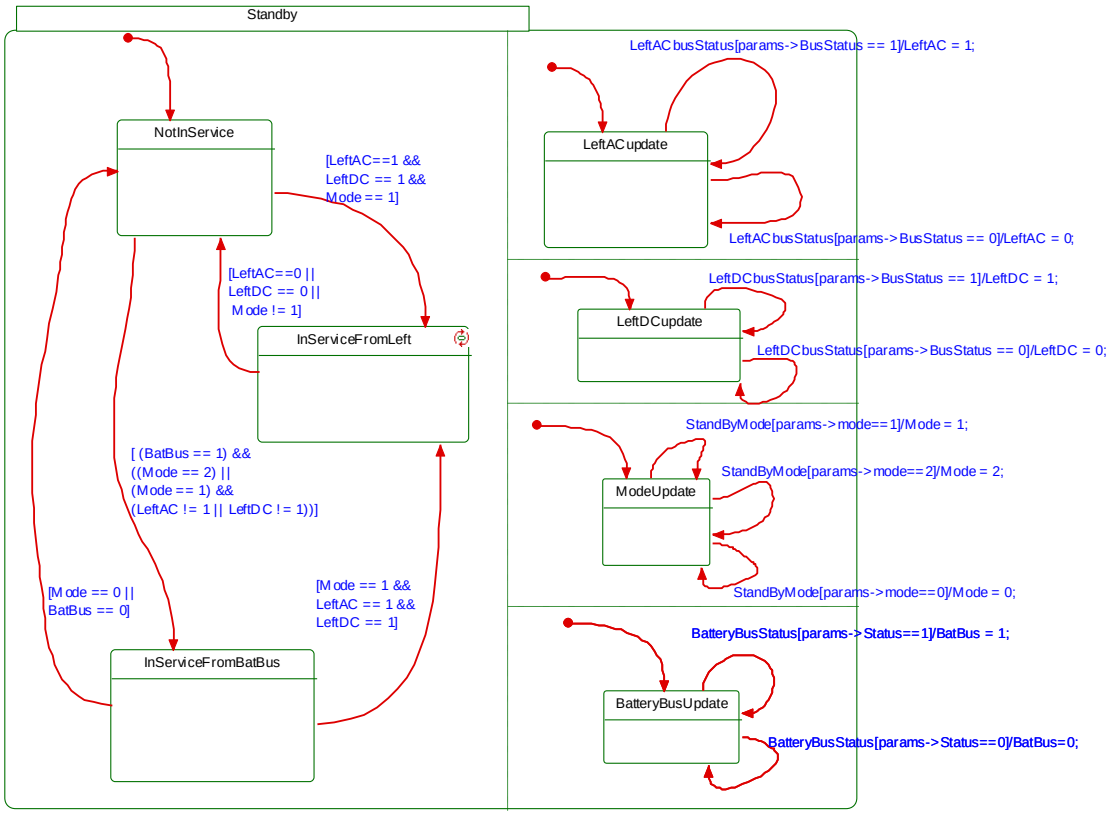


**Figure 191 –AC/DC Standby Bus**

## Battery Bus

The Battery Bus can either be powered from the Left DC Bus or the Battery, but this depends on the Standby Mode previously discussed. The state flow is shown in Figure 22, which has three update state flows on the right side. The first one gets the Standby Mode from the pilot via the *StandbyMode* event. The second update gets the value *ACDC* from the AC/DC Standby Bus, which is one if the Left AC and DC Buses power the Standby Bus. This update is triggered by the *ACDCbusPower* event from the Standby Bus. The last update state flow gets the Battery Status from the Battery via the *BatteryStatus* event. *Batt* = 1 if the Battery can supply power, and *Batt* = 0 otherwise.

Initially, the Battery Bus is not in service. If the Battery can supply power (i.e. Batt = 1) and Standby Mode is either OFF or BATT, then the Battery will power the Battery Bus, and the state flow will transition to the *InServiceFromBatt* state. The state flow will also transition to this state if in the AUTO mode and the Left AC/DC Buses are not powering the Standby Buses (i.e. ACDC = 0). While in the *InServiceFromBatt* state and the Battery becomes depleted (i.e. Batt = 0), it will transition back to the

*NotInService* state. Once in either the *InServiceFromBatt* or *NotInService* states, if Mode becomes AUTO and Left AC/DC Bus is providing power to the AC/DC Standby Buses (i.e. ACDC = 1), then the Left DC Bus provides power to the Battery Bus, which is indicated by transitioning to *InServiceFromLeft*. However, if the mode changes or the Left Side no longer powers the Standby Buses, the state flow will transition back to the *NotInService* state.

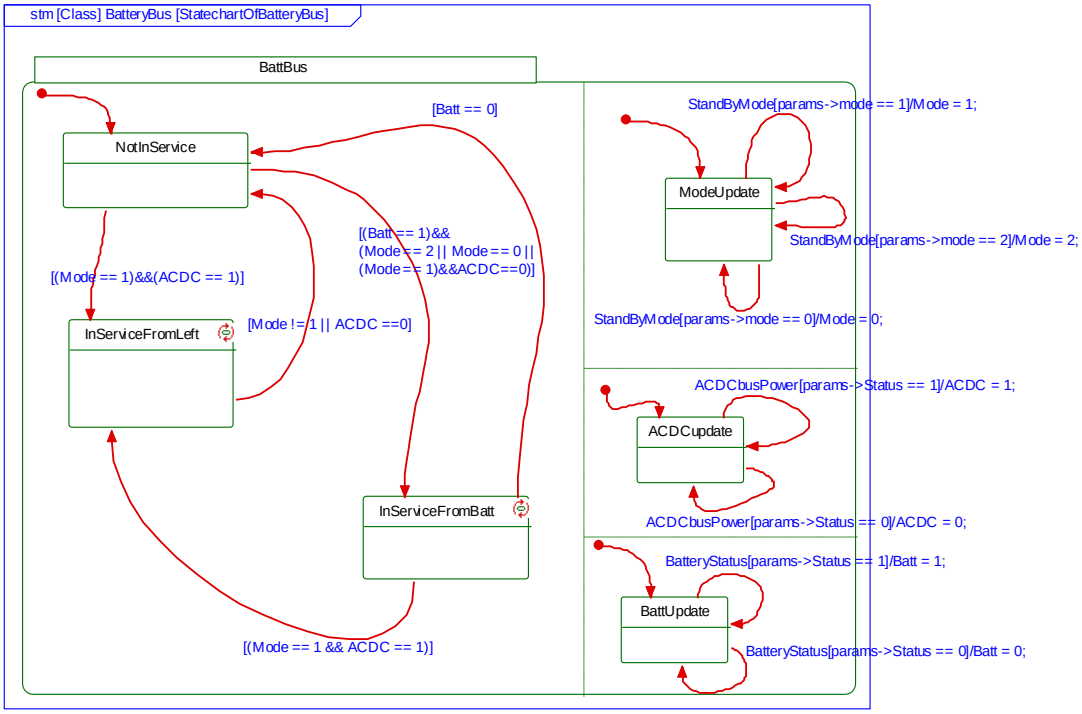


**Figure 202 –Battery Bus**

## Battery

The Battery model (shown in Figure 23) has two update state flows. The first one gets the status of the Battery Bus via the *BatteryBusStatus* event. This event sets *BattBus* to one if the Battery powers the Battery Bus otherwise it is zero. If this is the case, the Battery cannot be charged because the chargers would be supplying power to a load, which could cause them to fail prematurely. The Left AC Bus powers the Battery Chargers, and therefore, the second status update gets the status of the Left AC Bus from the Battery Bus via the *BattBusServiceFromLeft* event. So if the Left AC bus is powered, the variable *LeftAC*=1, otherwise it is zero.

Initially, the battery is Full as indicated by the *Full* state (*charge* = 100). If the Battery Bus is providing power (i.e. BattBus = 1) then the Battery will transition to the *Discharging* state, which decrements *charge* every 1000ms. If *charge* becomes zero, the Battery will transition to the *Depleted* state. If the Battery Bus changes to not supplying power and the Left AC Bus is powered (i.e. BattBus = 0 and LeftAC = 1) while in either the *Discharging* or *Depleted* states, then the Battery can transition

to the *Charging* state where *charge* is incremented every 1000ms until it is 100. At this point, it transitions back to *Full.* However, if the Battery suddenly starts to supply power while in the *Charging* State (i.e. BattBus = 1), then it will transition back to the *Discharging* state.

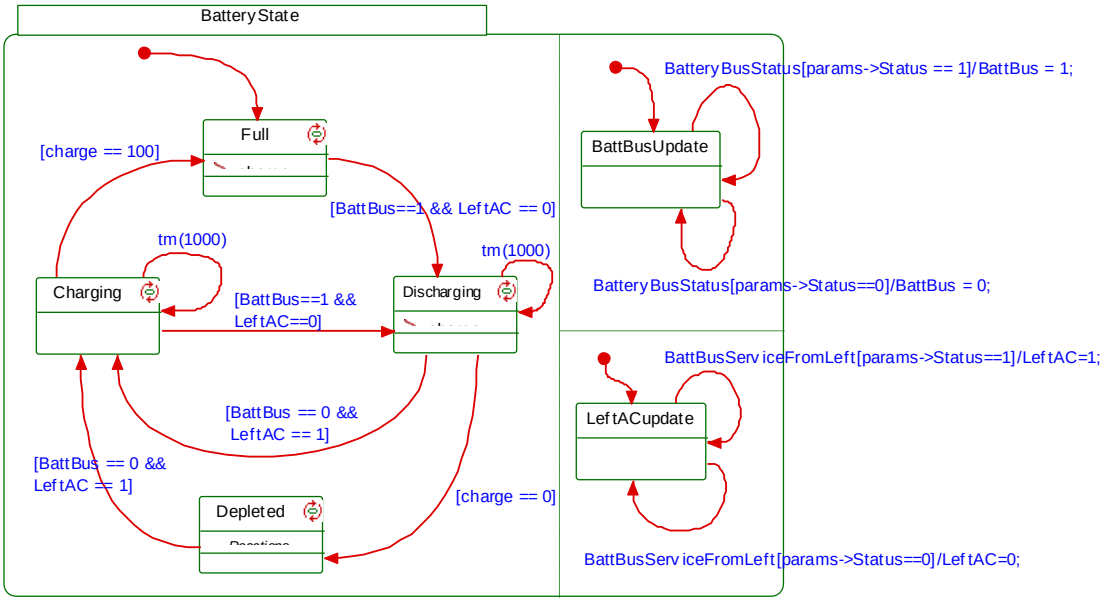


**Figure 213 –Battery**