JOURNAL OF AEROSPACE INFORMATION SYSTEMS
Vol. 11, No. 10, October 2014

Control Software Synthesis and Validation for a Vehicular Electric Power Distribution Testbed

Robert Rogersten*
KTH Royal Institute of Technology, 100 44 Stockholm, Sweden
Huan Xu†
University of Maryland, College Park, Maryland 20742
Necmiye Ozay‡
University of Michigan, Ann Arbor, Michigan 48109
Ufuk Topcu§
University of Pennsylvania, Philadelphia, Pennsylvania 19104
and
Richard M. Murray¶
California Institute of Technology, Pasadena, California 91125

DOI: 10.2514/1.I010167

Modern aircraft increasingly rely on electric power, resulting in high safety criticality and complexity in their electric power generation and distribution systems. Motivated by the resulting rapid increase in the costs and duration of the design cycles for such systems, the use of formal specification and automated correct-by-construction control protocols synthesis for primary distribution in vehicular electric power networks is investigated. A design workflow is discussed that aims to transition from the traditional "design and verify" approach to a "specify and synthesize" approach. An overview is given of a subset of the recent advances in the synthesis of reactive control protocols. These techniques are applied in the context of reconfiguration of the networks in reaction to the changes in their operating environment. These automatically synthesized control protocols are also validated on high-fidelity simulation models and on an academic-scale hardware testbed.

I. Introduction

NEXT-GENERATION aircraft are moving away from hydraulically and pneumatically powered systems into electrically powered systems [1]. As dependence increases on electric power, however, the electric power generation and distribution systems become more critical to the safe operation of aircraft [2]. Because of this increased reliance on electric power, these systems on next-generation aircraft need to be highly reliable and fault tolerant. In current practice, the design of an aircraft electric power system is constructed in an ad hoc manner, and it is either borrowed from legacy designs or created "by hand" using designer experience and knowledge. The entire process from running simulations to software testing (i.e., testing control logic) and hardware testing is both time consuming and costly, as mistakes are found throughout. Moreover, unexpected system failures at the hardware level require returning to the design phase to make changes. As the design stage progresses, the more expensive changes must be in redesign.

The difficulty in design of large-scale complex systems partly lies in the lack of a formal structure to verify the correctness of a system. Methods for diagnostic and prognostic analysis of electric power system testbeds have been considered in [3,4], and simulations using hardware in the loop have been performed on aircraft [5]. Recently, [6] has focused on a design methodology linking topology and control synthesis to real-time simulations. Most of this past work, however, has focused on the verification of predesigned and hardwired control strategies on hardware platforms. We propose a formal design methodology for an electric power system that integrates the use of formal methods [7,8] in order to guarantee correctness. The overall design flow is shown in Fig. 1. The first step in the methodology is translation of specifications. System requirements, including safety and performance properties and customer requests, are typically given in English text-based form. To apply formal methods to establish the correctness of a system, specifications must be translated into a formal specification language (e.g., linear temporal logic is used in this paper) [9,10] that is mathematically based and unambiguous. While the details of the translation are not covered within the scope of this paper, this process is critical to overall system design. Once abstracted and specified formally, we then proceed to the control synthesis layer in the methodology. In this step, we take the abstract model and formal requirements and automatically synthesize a control protocol. The Python-based Temporal Logic Planning toolbox, TuLiP [11], is used to construct a controller that is guaranteed to be correct with respect to the system requirements. If no such controller exists, then specifications or the model can be modified. Instead of constructing a system by hand and then verifying its correctness (i.e., design and verify), we specify and synthesize a control protocol. TuLiP has been used to synthesize controllers in past work on aircraft electric power systems [12] and vehicle management systems [13,14].

Once a control protocol is synthesized, the next step in the methodology is the simulation and hardware tests layer. From the abstract model, a simulation model can be constructed in a tool such as Simulink [15]. Here, (relatively) high-fidelity simulations can be performed, i.e., the behavior of the system can be tested by injecting faults or failures. Because specifications may arise from legacy designs or other customers, in this

Received 15 August 2013; revision received 6 February 2014; accepted for publication 10 February 2014; published online 3 July 2014. Copyright © 2014 by Robert Rogersten, Huan Xu, Necmiye Ozay, Ufuk Topcu, and Richard M. Murray. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission. Copies of this paper may be made for personal or internal use, on condition that the copier pay the $10.00 per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923; include the code 2327-3097/14 and $10.00 in correspondence with the CCC.

*Master Student, School of Electrical Engineering (EES); rrog@kth.se.
†Research Assistant Professor, Aerospace Engineering and Institute for Systems Research; mumu@umd.edu. Member AIAA.
‡Assistant Professor, Electrical Engineering and Computer Science; necmiye@umich.edu.
§Research Assistant Professor, Electrical and Systems Engineering; utopcu@seas.upenn.edu.
¶Professor, Control and Dynamical Systems; murray@cds.caltech.edu.

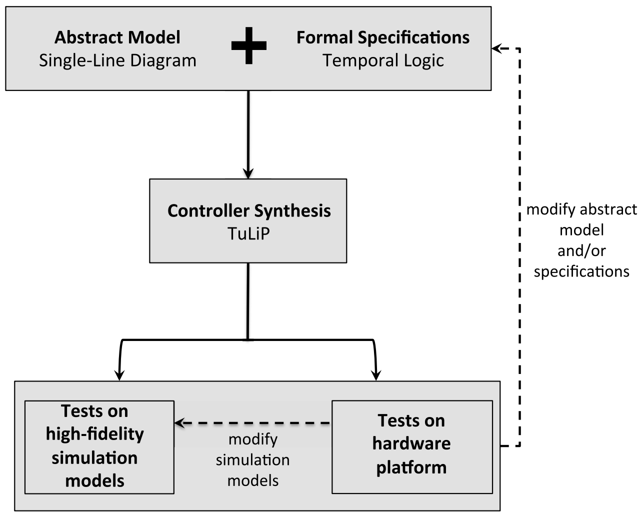


Fig. 1 Methodology of design flow for an aircraft electric power system. An abstract model and formal requirements are used to synthesize a controller. The resulting control protocols are tested in high-fidelity simulations and implemented on a hardware platform.

step of the methodology, we can adjust the types of components used in the model. For example, different batteries may have different voltage ratings that may not be able to satisfy all requirements given. Thus, the simulation layer provides information on how good the abstract model and specifications are and whether or not those need to be modified. These models can also be used for "testing" of design artifacts (e.g., controllers that are synthesized using the abstract models).

Previous work has discussed the implementation of software on hardware [16]. The hardware test step introduces the physical aspects of the system into the design stage. Thus, controllers can be tested for their correctness on a physical system. If any undesired behaviors arise that would not necessarily violate specifications but are not considered reasonable or "optimal," the hardware implementation provides information back to the specification and abstract model level.

Because of the growing complexity of electric systems in particular and embedded systems in general, the use of ad hoc techniques for design is becoming more difficult and time consuming. The advantages of a formal methodology, such as the one demonstrated in this paper, are the ability for systematic exploration of the design space as well as the ability to formally analyze and guarantee correctness. In this paper, we demonstrate this design flow for an electric power system and its academic-scale hardware implementation. This demonstration serves only the purpose of proof of concept. Extending the tools used in our study, modifying them to align with the needs of particular application areas, and transitioning them are among the important challenges that are faced. They are, however, beyond the scope of the current paper.

II. Background

An electric power system provides power to buses and subsystem loads. In more-electric aircraft, these loads include lighting, heating, and safety- and mission-critical subsytems (such as avionics, deicing, and flight actuation) [1]. Figure 2 shows a single-line diagram for an electric power system [17]. Each of the two engines power a high-voltage and low-voltage ac generator. Two additional generators are mounted on an auxiliary power unit and can be used to supply power in case of emergencies. The primary loads can be considered as safety or mission critical. Primary loads include avionics, communication systems, and window heating. The electric power system of an aircraft often contains transformers and rectifier units that broadly divide the system into four categories: namely, high-voltage ac, high-voltage dc, low-voltage ac, and low-voltage dc. The power is distributed from the generators to the loads in series connection through buses, transformers, rectifier units, and electronically controlled switches called contactors.

There are three levels of design challenges in an electric power system: the primary distribution problem, the secondary distribution problem, and the load-shedding problem. In primary distribution, the main concern is in providing power from generators to buses. Generators must be able to supply power to buses connected to safety-critical loads. In the secondary distribution problem, the design question is how much power should be allocated to system loads by buses. Finally, the load-shedding problem is, if a power failure or emergency situation were to arise, what loads should be shed and in what order, so that safety-critical loads can still be powered and the aircraft can land safely.

In this paper, we consider the primary distribution problem. The overall goal is to design a controller that can react to component failures by changing the topology so that new ways for power delivery are created. Moreover, the controller must ensure that that safety-critical buses and loads are always powered. The state of the system (i.e., the status of all contactors and health of components) is estimated by current and voltage sensor measurements. If a fault is detected, the controller reacts and reconfigures the contactors so that the system still satisfies all requirements. This controller can take actions depending on the system and environmental conditions during operation. The control logic not only accounts for a static configuration of contactors given a fault but also determines the correct sequence of contactor switches in order to guarantee all specifications are still satisfied.

Typical electric power system specifications are categorized in terms of safety, reliability, and performance. On aircraft with variable-frequency generators, a mismatch in frequency and voltage can lead to, for example, overcurrents and fires. A safety specification, therefore, would be to disallow any configuration of components in which more than one generator provides power to a bus. A typical reliability specification requires that the system must be able to account for a certain number or subset of failures. The total number of allowable, simultaneous failures is known as a reliability level. Every component has a probability of failure, determined from past operational data. Provided that failures on components are independent, the maximum number of components that can fail (i.e., the environmental conditions for which the controller must account) is determined by the reliability specification [18]. Finally, consider performance specifications that affect the overall quality of the flight. A standard performance specification would disallow buses connected to critical loads to be unpowered for a length of time greater than some predetermined time bound. This ensures proper operation of loads necessary for safe flight. Whereas this paper examines a limited subset of electric power system specifications, more general specifications have been used [19,20].

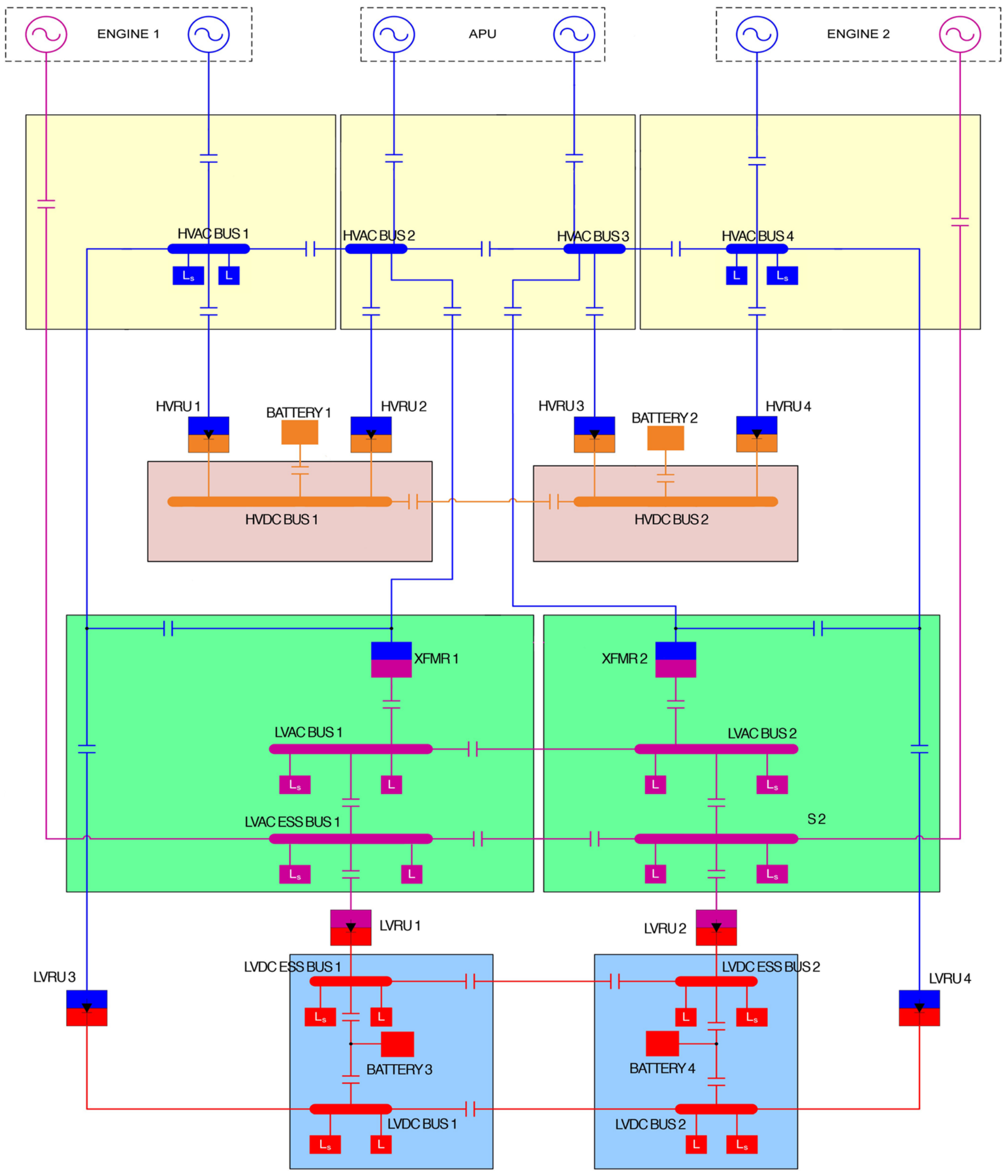


Fig. 2 Single-line diagram for an electric power system topology adapted by Richard Poisson of United Technologies Aerospace Systems from a Honeywell patent for a more-electric aircraft [17]. The differently colored panels represent the physical isolation between parts of the network or introduce logical decoupling between controllers responsible for different regions (HVAC, high-voltage alternating-current; HVDC, high-voltage direct-current; HVRU, high-voltage rectifier unit; LVAC, low-voltage alternating-current; XMFR, transformer; ES, essential; LVDC, low-voltage direct-current; LVRU, low-voltage rectifier unit).

III. Modeling, Specifications, and Synthesis

At the core of the design methodology we advocate in this paper are models and specifications in mathematically based languages and the corresponding algorithms that automate the synthesis of software-based control protocols from these models and specifications. We now give an overview of these building blocks as tailored to the discussion in the subsequent sections.

A. Modeling and Specifications

The initial step in any model-based flow is determining the level of fidelity to be used in the design of the control protocols. Partly for aligning with the industrial practice and partly for leveraging the currently available synthesis tools, we use purely discrete (and finite) models for the evolution of the configuration of a power distribution network; for example, as shown in Fig. 2.

Roughly speaking, the variables of interest can be grouped as those under direct control (we will call these "control" variables) and those that can change without the control of the system (we will call those "environment" variables). As a modeling convention, we will consider that the environment variables evolve adversarially (specific meaning of "adversarial" will be concretized later in this section) against the system. Typically, controlled variables are the statuses of the contactors. They open and close with directives from the controller. Examples of environment variables include the health statuses of the generators and rectifier units that typically take binary values (i.e., healthy vs unhealthy).

Let x now be the set of variables (including both controlled and environment). Then, the evolution of the system can be described by sequences of valuations x_t (we will also call these valuations the "states" of the system) of x at the time steps $t = 0, 1, 2, \ldots$ Let M denote all sequences $x_0, x_1, x_2, \ldots$ that can be generated by the system. M can be considered as a model of the system. Note that, besides this abstract representation of the model, we can equivalently use a finite-state nondeterministic transition system in order to represent all possible behaviors of the system [21,22].

We will characterize the correctness of the system in terms of the properties satisfied by the sequences in M. To formalize this notion, we use temporal logic based languages [9,21]. Roughly speaking, temporal logic allows us to unambiguously specify and reason about infinite sequences of states. We specifically employ linear temporal logic (LTL) to describe system behavior. An LTL formula is built up from a set of atomic propositions and two kinds of operators: logical connectives and temporal modal operators. An atomic proposition is a statement over the system variables that has a unique truth value (true or false) for a given valuation of x. For example, let g and c denote the health status of a particular generator and the status of a particular contactor, respectively. Then, given a configuration of the system, the truth value of $g =$ healthy, $c =$ open, and $g =$ healthy and $c =$ closed can be determined, and all these statements are atomic propositions. In other words, atomic propositions are the lowest level of building blocks for specifying the system behavior and logical connectives, including negation , disjunction $\vee$, conjunction $\wedge$, and implication $\rightarrow$; and temporal operation, always $\Box$, eventually $\Diamond$, until $\mathcal{U}$, and next $\bigcirc$ connect these building blocks to create more sophisticated specifications of the system. For example, given atomic propositions p and q, we can write 1) invariance (a specific form of safety) properties as $\Box p$, 2) guarantee or reachability properties $\Diamond p$, 3) progress or recurrent properties as $\Box\Diamond p$, 4) response properties as $\Box(p \Rightarrow \Diamond q)$, and 5) next-step response properties as $\Box(p \Rightarrow \bigcirc\Diamond q)$.

In this paper, we use LTL as the specification language for convenience. Depending on the underlying model and properties of interest, one may consider other, potentially more suitable specification languages including timed temporal logic [23], probabilistic temporal logic [24], and branching-time logics [21]. For further details on the range of specification languages, we refer the reader to [9,21]; and for details of and a complete treatment for primary distribution in vehicular electric power networks, we refer to the reader to our earlier work [22].

B. Synthesis of Reactive Control Protocols

The overall goal of the design problem is synthesizing a control protocol that, when implemented on the electric power system, ensures that the controlled system satisfies its specifications. The correctness of the system is, though, not merely a function of the controlled variables. It needs to be interpreted in conjunction with the environment variables. For example, the generator from which each bus shall be powered is constrained by the health statuses of the generators, which cannot be controlled by the system. Hence, the control protocol needs to react to the changes in both the controlled variables and environment variables.

Furthermore, it is necessary to incorporate information on potential environment conditions under which the system is expected to operate. If the environment variables are not properly constrained, then the resulting control protocol may be overly conservative, and it and may not be possible to construct a protocol that ensures the satisfaction of the system requirements. For example, if all the generators simultaneously stay unhealthy for a long enough time, then it is not possible to satisfy the condition that the essential buses shall not be unpowered longer than some prespecified period. Hence, such behaviors of the environment shall be disregarded in the protocol design. An essential component of the protocol synthesis problem is the environment assumptions that specify what environment behaviors the controller shall correctly react to. Consequently, the overall goal is to design a protocol that determines how the controlled variables shall move at each point of the execution as a function of the behaviors of the controlled and environment variables so far in the execution as long as the environment assumptions are satisfied.

We now, equipped with LTL as a specification language, formally state the reactive synthesis problem. Let E and P be sets of environment and controlled variables, respectively. Let $s = (e, p) \in \mathrm{dom}(E) \times \mathrm{dom}(P)$ be a state of the system. Consider an LTL specification φ of assume-guarantee form

$$\varphi = \varphi_e \rightarrow \varphi_s \tag{1}$$

where, roughly speaking, φ_e is the conjunction of LTL specifications that characterizes the assumptions on the environment and φ_s is the conjunction of LTL specifications that characterizes the system requirements. The synthesis problem is then concerned with constructing a strategy, i.e., a partial function h: $(s_0 s_1 \ldots s_{t-1}, e_t) \mapsto p_t$, that chooses the move of the controlled variables based on the state sequence so far and the behavior of the environment so that the system satisfies φ_s as long as the environment satisfies φ_e. The synthesis problem can be viewed as a two-player game between the environment and the controlled plant: the environment attempts to falsify the specification in Eq. (1), and the controlled plant tries to satisfy it.

For general LTL, it is known that the synthesis problem has a doubly exponential complexity [25]. For a subset of LTL, namely, generalized reactivity (1) [GR (1)], Piterman et al. have shown that it can be solved in polynomial time (polynomial in the number of valuations of the variables in E and P) [26]. GR(1) specifications restrict φ_e and φ_s to take the following form for $\alpha \in \{e, s\}$:

$$\varphi_\alpha := \varphi^\alpha_{\text{init}} \wedge \bigwedge_{i \in I^\alpha_1} \Box \varphi^\alpha_{1,i} \wedge \bigwedge_{i \in I^\alpha_2} \Box\Diamond \varphi^\alpha_{2,i}$$

where $\varphi^\alpha_{\text{init}}$ is a propositional formula characterizing the initial conditions; $\varphi^\alpha_{1,i}$ are transition relations characterizing safe, allowable moves and propositional formulas characterizing invariants; and $\varphi^\alpha_{2,i}$ are propositional formulas characterizing states that should be attained infinitely often.

The details of the solution techniques for two-player temporal logic games [and GR(1) synthesis in particular] are beyond the scope of the current paper. We limit the discussion by noting that recent advances in computer science have resulted in a range of solvers focusing on various versions of the problems with different input–output structures. Some of these solvers are JTLV [27], gr1c,[**] Anzu [28], LiLY [29], Acacia [30], and Unbeast [31]. A detailed comparison of some of these tools can be found in [32]. It is also worth noting that almost all of these solvers are research oriented, with relatively poor user interfaces limiting their broader use in applications. Partly motivated by this gap between game solvers (and other software for formal control synthesis and verification), a number of higher-level packages that bridge libraries of low-level solvers with potential application domains have been developed. One such package is TuLiP [11], essentially a collection of Python-based routines for automatic synthesis of correct-by-construction controllers. When restricted to the problems studied in the current paper, TuLiP provides 1) a front end for modeling and syntax check for temporal logic-constraint problems and their translation into standard form accepted by the low-level solver (e.g., JTLV, gr1c, nuSMV, and others); 2) a link to these solvers; and 3) a back end to translate the outputs of the solvers to forms suitable for easier consumption by the users (e.g., graphical representations and simulation routines).

C. Closer Look at the Synthesized Controllers

Figure 3 shows different views of the resulting controller automaton for a toy example. This controller has four states. The top, left corner of Fig. 3 shows the output from TuLiP, which roughly lists each of the states; the corresponding configurations (i.e., the status of the two generators and three contactors in this example); and the states to which the system may transition (as a function of the environment move) from the current

```
State 0 <rgen:1, lgen:1 ,c1:1, c2:1, c3:0>
          With successors:1, 2, 3, 0

State 1 <rgen:0, lgen:0, c1:0, c2:0, c3:0>
          With no successors.

State 2 <rgen:0, lgen:1, c1:1, c2:0, c3:1>
          With successors: 1, 2, 3, 0

State 3 <rgen:1, lgen:0, c1:0, c2:1, c3:1>
          With successors: 1, 2, 3, 0
```

```
function [c3,c2,c1] = Controller(rgen,lgen)
global state;
switch state
case 0
if rgen == 1 && lgen == 1
state = 0; c3 = 0; c2 = 1; c1 = 1;
elseif rgen == 0 && lgen == 0
state = 1; c3 = 0; c2 = 0; c1 = 0;
elseif rgen == 0 && lgen == 1
state = 2; c3 = 1; c2 = 0; c1 = 1;
elseif rgen == 1 && lgen == 0
state = 3; c3 = 1; c2 = 1; c1 = 0;
else
disp('Cannot find a valid successor')
c3 = 0; c2 = 1; c1 = 1;
end
case 1
c3 = 0; c2 = 0; c1 = 0;
case 2
.
case 3
.
otherwise
.
end
```

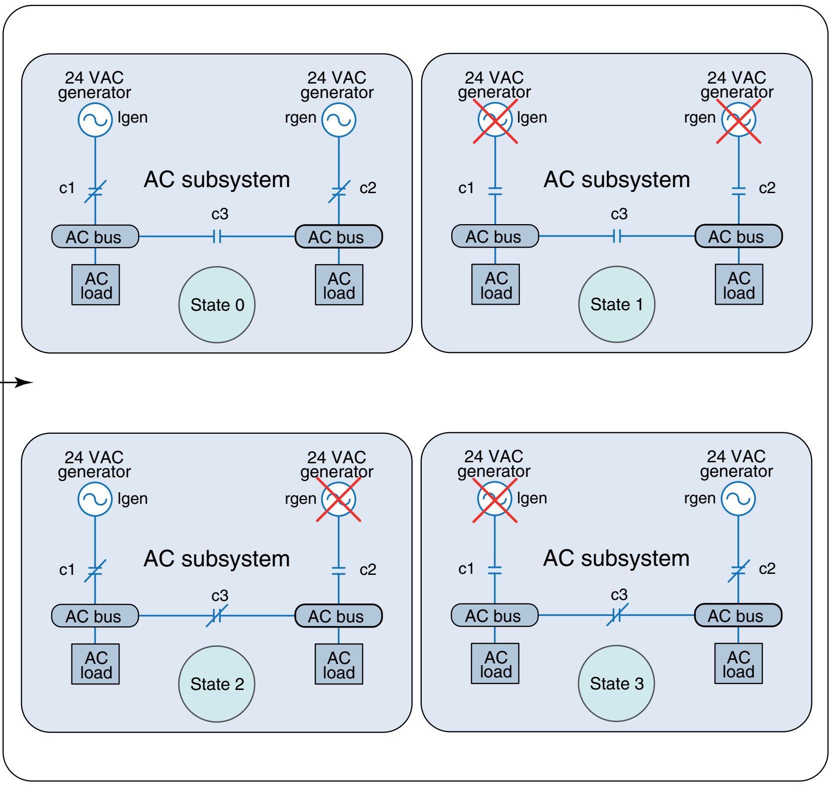


Fig. 3 Automatically generated controller from TuLiP (top, left) and its translation into a MATLAB function. The automaton is synthesized for a two-generator and three-contactor case. The generator status variables are rgen and lgen, and the contactor status variables are *c*1, *c*2, and *c*3. Each state has successors, which define to which state the controller can transition, depending on the current controlled and environment state. In addition, no-successor states exist.

one. The big box on the right-hand side pictures the configurations of the corresponding network in each of the four states. For example, state 0 (read from the text in the top, left box) corresponds to a configuration where both generators are healthy, two of the contactors are closed, and the one connecting the two buses is open. If one of the generators (rgen) becomes unhealthy, then the system transitions to state 2, contactor *c*2 opens, and contactor *c*3 closes in reaction to this change in the generator health. Finally, the left, bottom corner of Fig. 3 shows (part of) the control automaton written into a MATLAB function, which is used to drive the testbed, as discussed in the next section.

IV. Aircraft Electric Power Testbed

We now discuss an end-to-end implementation of the specify and synthesize design flow on an academic-scale electric power testbed we had developed in our recent work [16]. We begin with an overview of the testbed and its basic functionality. Figure 4 shows the physical layout of the testbed (left) and its single-line diagram (right). It was built to mimic some of the characteristics of the primary electric power distribution systems on aircraft. It contains transformers that supply power to ac systems and rectifier units that separate the dc part of the system from the ac part. We

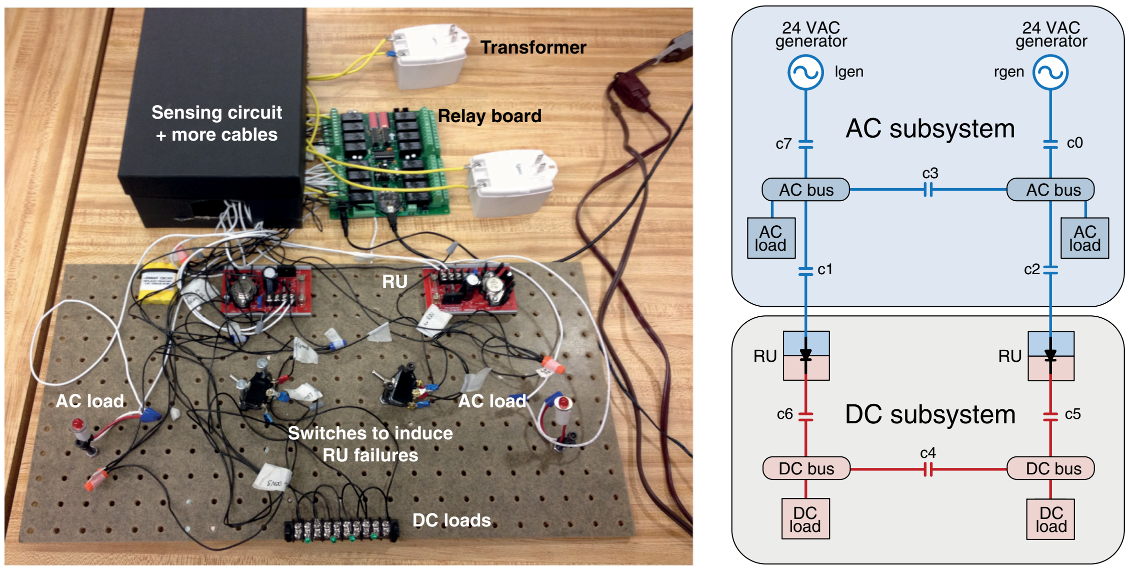


Fig. 4 Photograph of the physical layout of the electric power system testbed (left), and single-line diagram of the power system testbed (right). Contactors are represented by double bars. AC and dc sides of the system are separated by rectifier units (RUs).

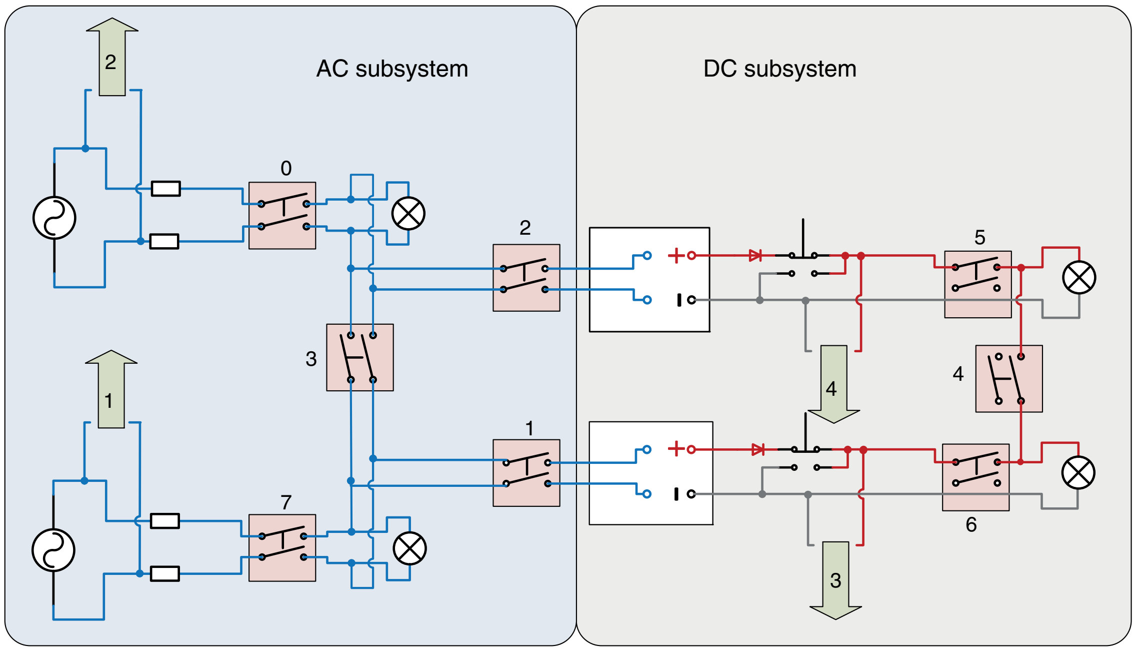


Fig. 5 Circuit schematic of the testbed. The topology is the same as in Fig. 4 (right). The relays are represented by the numbered boxes. The numbered arrows denote voltage-sensing connections from Fig. 8.

refer to these transformers as generators because we are only interested in the role as voltage sources. The generators and rectifier units are crucial for the safe operation of an aircraft. Therefore, the testbed focuses on the failure of these components.

A. Structure of the Hardware Testbed

The topology of the hardware testbed is shown in Fig. 4 (right); it contains two generator sources and two rectifier units. The generator sources are modeled by transformers with a secondary side voltage of 24 volts ac (VAC). The dc section is connected to the ac section through two rectifier units. The rectifier units contain a diode-rectifier bridge and dc bus capacitor to achieve a low ripple in the dc-side voltage. It also contains a variable dc voltage regulator that is tuned to 2.5 volts dc (VDC). Consequently, the testbed has two different voltage levels: 24 VAC and 2.5 VDC.

The single-line diagram also contains four buses: specifically, two ac buses and two dc buses. Multiple lamps attached to these buses are considered as the primary loads in the distribution network. The main design goal is to keep these loads powered even in the presence of failures in the generators or rectifier units. A detailed circuit schematic of the testbed hardware is shown in Fig. 5.

The contactors in electric power distribution networks on aircraft are designed to switch three-phase electric power. Their functionality is replicated by simpler relays in the testbed. In particular, a commercially available relay board[††] that provides a set of computer-controlled relays that can communicate with programming languages supporting serial communications (for example, MATLAB) is used. The relays on the board are numbered and range from 0 to 7. The same numbering convention is used in Fig. 4 (right), Fig. 5, and throughout the paper. We also remark that, for the results presented in this paper, the contactors $c1$ and $c2$ shown in Fig. 5 were not used and left closed. However they can be used to isolate the ac and dc subsystems and to test the ac subsystem separately.

The hardware testbed is also equipped with switches/plugs for injecting faults and with sensors that monitor the health status of components. These elements are discussed in detail in Sec. IV.D.

B. Specifications

The testbed mimics only a small fraction of the functionality that exists in the primary distribution networks on aircraft. Therefore, only a subset of the typical specifications is well defined for the testbed. We now discuss these specifications: both their descriptions in English and their translations into temporal logic statements.

As discussed in Sec. III, the formal specifications we consider have an assume-guarantee form, i.e., they contain both assumptions on the possible environment behavior and guarantees on the system behavior.

1) At least one of the generators is always healthy.
2) At least one of the rectifier units is always healthy.

The guarantees on the system behavior include the following:

1) No ac bus can be powered from two different ac sources simultaneous at any time.
2) AC and dc buses must be powered at all times.

The synthesis problem can then be considered as constructing a reactive control logic that ensures the realizability of the temporal logic specification $\varphi_e \rightarrow \varphi_s$. For the topology shown in Fig. 5, the specifications become

$$\varphi_e = \square\{((\text{gen}_1 = \text{healthy}) \vee (\text{gen}_2 = \text{healthy})) \wedge ((\text{ru}_1 = \text{healthy}) \vee (\text{ru}_2 = \text{healthy}))\} \tag{2}$$

$$\varphi_s = \square((c_0 = \text{closed}) \wedge (c_7 = \text{closed}) \wedge (c_3 = \text{closed})) \wedge \bigwedge_{i\in\{1,2,3,4\}} \square(\text{bus}_i = \text{powered}) \tag{3}$$

[††]The specific relay boards used in the testbed are supplied by RelayPROS (www.relaypros.com [retrieved 2014]).

where gen_1, gen_2, ru_1, and ru_2 are health statuses for the two generators and two rectifier units, respectively. The contactors c_0 and c_7 are next to the generators in the topology shown in Fig. 5, and c_3 is between the ac buses. Therefore, contactors c_0, c_7, and c_3 can never be closed at the same time, which otherwise would lead to paralleling two ac sources. The buses bus_1, bus_2, bus_3, and bus_4 can be considered to be in an electrical connection to the loads. The final part of φ_s ensures that a bus can never be unpowered, given that the environment assumptions hold. However, measurements are taken at discrete time intervals. A continuous implementation has to allow a certain unpowered time. For a more detailed discussion of electric power system requirements and their conversion to LTL specifications, we refer the reader to [6].

In addition to the centralized control protocol that realizes the global specification $\varphi_e \rightarrow \varphi_s$, we synthesize a distributed reactive control protocol following the theory in [14]. More specifically, we decompose the global specification into local specifications for the ac and dc parts of the system in such a way that, if the local specifications are realizable separately, then they can be implemented together and ensure the correctness of the global specification (under additional mild technical assumptions discussed in [14]). For example, the relatively simple global specification in Eqs. (2) and (3) are decomposed into $\varphi_{e,\mathrm{AC}} \rightarrow \varphi_{s,\mathrm{AC}}$ and $\varphi_{e,\mathrm{DC}} \rightarrow \varphi_{s,\mathrm{DC}}$ for the ac and dc parts respectively, where

$$\begin{aligned}
\varphi_{e,\mathrm{AC}} &= \square(((\mathrm{gen}_1 = \mathrm{healthy}) \vee (\mathrm{gen}_2 = \mathrm{healthy})),\\
\varphi_{s,\mathrm{AC}} &= \square((c_0 = \mathrm{closed}) \wedge (c_7 = \mathrm{closed}) \wedge (c_3 = \mathrm{closed})) \wedge\wedge_{i\in\{1,2\}} \square(\mathrm{bus}_i = \mathrm{powered}),\\
\varphi_{e,\mathrm{DC}} &= \square(((\mathrm{ru}_1 = \mathrm{healthy}) \vee (\mathrm{ru}_2 = \mathrm{healthy})), \quad \text{and}\\
\varphi_{s,\mathrm{DC}} &= \square((c_5 = \mathrm{closed}) \wedge (c_4 = \mathrm{closed}) \wedge (c_6 = \mathrm{closed})) \wedge\wedge_{i\in\{3,4\}} \square(\mathrm{bus}_i = \mathrm{powered}) \qquad (4)
\end{aligned}$$

C. Synthesized Controller Automata

We use TuLiP to synthesize the control protocols for the global and the distributed specifications for the ac and dc parts. The centralized controller realizing the global specifications has 16 states (i.e., one state for each of the possible environment configurations in this case). On the other hand, each of the automata for the ac and dc parts contains four states.

Figure 6 shows part of the output from TuLiP for the centralized controller. Each entry in the list has two lines that correspond to one state in the automaton. The valuations of the environment variables (i.e., the health statuses of the generators and rectifier units) and the controlled variables (i.e., the statuses of the contactors) are in the first line. The second line lists the possible transitions from the current state. Out of these possible transitions, the one that is implemented as the transition in the controlled variables is picked based on the transition in the environment variables. For example, if from state 0, in which all generators and rectifier units are healthy, the environment transitions to a configuration in which rgen $= 0$ and rru $= 0$, then the controller transitions to state 4 and the contactor statuses switch to the values listed under state 4.

By its construction, as long as the environment satisfies its assumptions, then the controller can execute indefinitely and the contactors take actions such that the system requirements are satisfied. However, if the environment assumptions are violated, then the controller may end up in a state with no outgoing transition, referred to as the “no-successor” state in Fig. 6. For example, if both generators or both rectifier units are unhealthy (as in state 2), the controller will enter a no-successor state.

D. Determining the Statuses of the Generators, Rectifier Units, and Buses

Note that the execution of the reactive protocols synthesized in Sec. IV.C requires the knowledge of the valuations of the environment variables at each execution step. The lines of code in Fig. 7 illustrate the control cycle workflow, which consists of three steps: read environment variables, run control logic, and assign values to controlled variables. The four environment variables are represented in Fig. 7 and are labeled as lgen, rgen, lru, and rru. The controlled variables are $c0$, $c7$, $c3$, $c5$, $c6$, and $c4$. The health statuses of the generators and rectifier units are not directly monitored. Their values are deduced from certain voltage measurements at appropriate locations in the power distribution network. Let V_0 be a

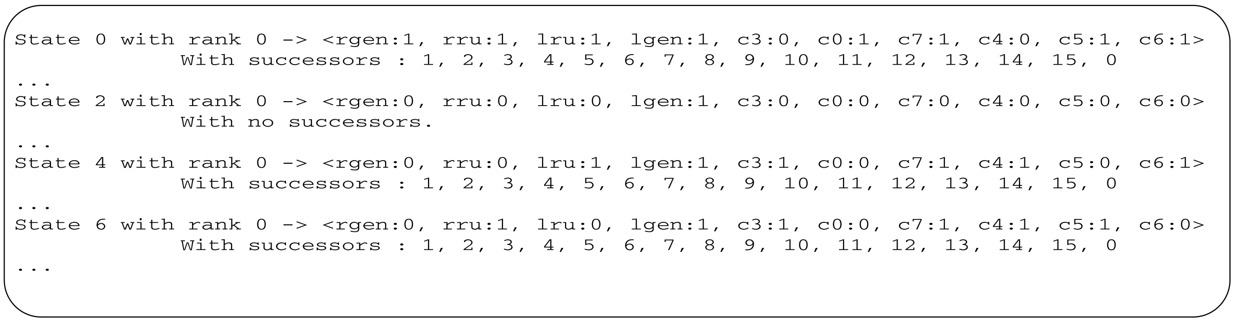

```
State 0 with rank 0 -> <rgen:1, rru:1, lru:1, lgen:1, c3:0, c0:1, c7:1, c4:0, c5:1, c6:1>
            With successors : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0
...
State 2 with rank 0 -> <rgen:0, rru:0, lru:0, lgen:1, c3:0, c0:0, c7:0, c4:0, c5:0, c6:0>
            With no successors.
...
State 4 with rank 0 -> <rgen:0, rru:0, lru:1, lgen:1, c3:1, c0:0, c7:1, c4:1, c5:0, c6:1>
            With successors : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0
...
State 6 with rank 0 -> <rgen:0, rru:1, lru:0, lgen:1, c3:1, c0:0, c7:1, c4:1, c5:1, c6:0>
            With successors : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0
...
```

Fig. 6 TuLiP autogenerated automaton. All states are not represented in the figure. The environment variables are lgen, rgen, lru, and rru. The controlled variables are *c*0, *c*7, *c*3, *c*5, *c*6, and *c*4.

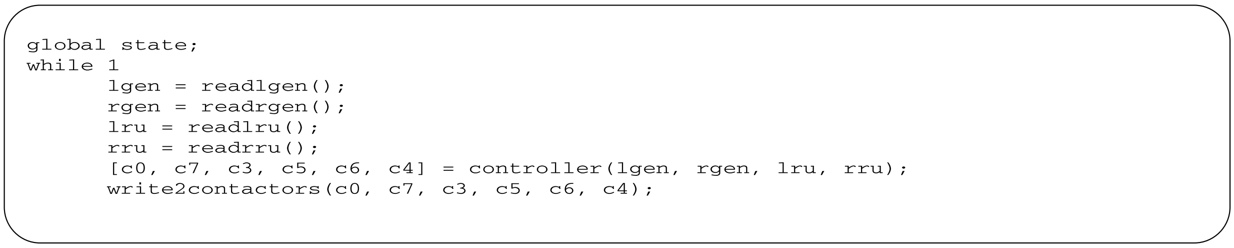

```
global state;
while 1
      lgen = readlgen();
      rgen = readrgen();
      lru = readlru();
      rru = readrru();
      [c0, c7, c3, c5, c6, c4] = controller(lgen, rgen, lru, rru);
      write2contactors(c0, c7, c3, c5, c6, c4);
```

Fig. 7 Code that implements a control cycle on the hardware testbed. In each control cycle, the sensors are first read for the generators and then the sensors for the rectifier units. After that, the script generated by TuLiP is used to decide on the contactor statuses. Finally, the states of the contactors are set.

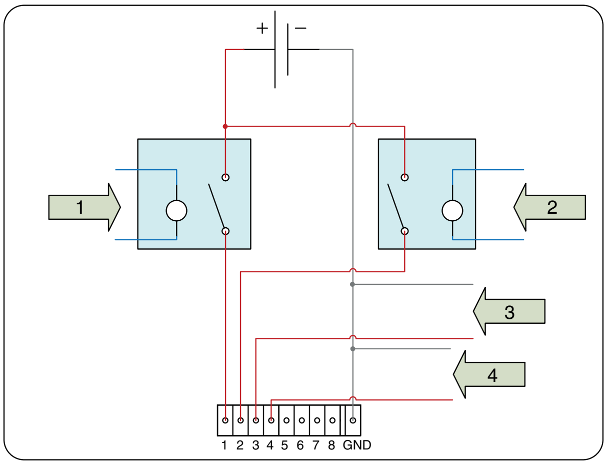


Fig. 8 Sensing configuration for the testbed. The numbered arrows denote voltage-sensing connections from Fig. 5.

prespecified positive constant and T_r be the time that elapses while the corresponding sensor reading takes place. Then, a generator or a rectifier unit will be registered unhealthy if the magnitude of its voltage reading stays below V_0 for a time T_{sense} that elapses so it includes T_r. Time T_{sense} therefore has to be greater than or equal to T_r. For example, the function readrgen( ) in Fig. 7 checks if the right generator is above or below the threshold V_0.

Each of the rectifier units in the testbed consists of a single-phase diode rectifier followed by a capacitor and a voltage regulator. The capacitor is connected to the dc bus in order to reduce the voltage ripple at the input of the voltage regulator. The specification used in controller synthesis requires that all buses are powered at all times. However, there exists time T during which the bus is unpowered that needs to elapse when the controller take actions. See Sec. IV.E for the details on how the constant T is chosen based on the characteristics of the testbed estimated empirically. This situation does not necessarily mean that relevant components are influenced by that the voltage is below V_0 during time T. For example, consider a rectifier unit connected to an ac bus. It contains a capacitor that charges to the peak voltage each half-cycle of the ac voltage sine curve and then discharges at a slower rate through the load while the rectified voltage drops before the beginning of next half-cycle. Therefore, duration, called T_{RU}, of time that it takes for the capacitor voltage to drop below an acceptable value depends on the capacitance of the capacitor and the amount of current drawn by the load. If T_{RU} is strictly larger than T, then it can be guaranteed that the dc voltage stays above a prespecified threshold provided that the corresponding rectifier unit is healthy. Furthermore, the time over which a generator has to remain healthy during each control cycle is not arbitrarily small because it needs to be healthy for at least a time, called T_r, during which the sensor is read. Otherwise, we would violate the environment assumptions. The time T_r is enough to charge the capacitor to the peak voltage. Therefore, the capacitances and the current drawn by the dc loads in the testbed are arranged so that T_r is large enough to charge the capacitors in the rectifier units to their peak voltage.

For proper operation of the controller, the sensors shall provide complete and consistent information. To this end, their placement, functionality, and accuracy play crucial roles in design. Analog-to-digital (A/D) inputs on the relay board are used to monitor the system conditions; the input connections range from 1 to 8, as shown in Fig. 8. The system can have four threshold values because it has four sensors. The A/D inputs on the relay board can read 0 to 5 VDC. The first two sensors will have a threshold value of $V_0 = 5V_{\mathrm{AC}}/256$, and the other two sensors will have a threshold value of $V_0 = 5V_{\mathrm{DC}}/256$. We check the voltage on an A/D input with an accuracy of 8 bits; therefore, V_{AC} and V_{DC} are scaled in the range of 0–256. The first lines of the code in Fig. 7 read the voltages from each sensor and check if the voltage measurement is above or below the threshold values, V_{AC} and V_{DC}. The status of each environment variable can then be assigned as healthy or unhealthy accordingly.

The voltage-sensing connections are represented by the numbered arrows in Fig. 8, which correspond to the numbered arrows in Fig. 5. The transformers that act as generator sources can be unplugged in order to simulate a generator failure. The A/D inputs cannot handle 24 VAC; therefore, voltage sensing for generator failures on 24 VAC is handled using additional relays. The relays connect a 3.6 V circuit to a battery when triggered by the voltage from the transformers. Therefore, the threshold value V'_{AC} for the system is set whenever the additional relays are not triggered anymore. The threshold value V_{AC}, which is read from the A/D inputs, is set to 100, approximately 2 V using an 8 bit resolution. The voltage V'_{AC} is set by the relay manufacturer but is usually a low value compared to when they are triggered. The relays used in the testbed have a minimum turn-off voltage of 3.6 VAC and a maximum turn-on voltage of 18 VAC.

The rectifier units are connected to a switch that can be used to generate a fault in the dc subsystem. The voltage sensors of the rectifier units are directly connected to the A/D inputs of the relay board because the voltage is tuned to 2.5 VDC using the variable dc voltage regulator on the rectifier units. When the status of a switch that injects a fault on the dc subsystem is changed, there will be no potential difference between ground and the wire connected to the sensor; therefore, V_{DC} can be chosen anywhere between 0 and 128 and is usually set to 100 (i.e., equal to V_{AC}).

E. Testbed Characteristics

We now describe the characteristics of the hardware testbed. The characteristics depend on the relay delay time T_d and control cycle times T_c and T'_c. The relay delay time is the time delay between the time a command to actuate the relay is written on the relay board and the time the action (i.e., relay opening or closing) is completed. The control cycle times are defined as

$$T_c = 4T_r + T_I + T_w \quad T'_c = 4T_r + T_I \tag{5}$$

where T_r is the time taken to read the health status from one environment variable, T_I is the time taken to run the MATLAB script generated from TuLiP, and T_w is the time taken to write information to the board. We also have to consider the control cycle time T'_c because, if the system remains the same, writing information to the board is not necessary in that iteration.

As discussed in Sec. IV.D, we reason the definition of when a bus becomes unpowered based on these timing characteristics. Consider the code listed in Fig. 7, which shows that the controller reads the health status from each environment variable in a specified order. Therefore, we have to first include T_c, and then part of T'_c, from the previous control cycle in the limit T. We approximate the time to read the health status from

Table 1 Summary of the variables that characterize the testbed[a]

Variable	Maximum value, ms	Description
T	587.9	Time limit for the bus to stay unpowered
T_d	20	The relay delay time
T_c	333.3	Control cycle time ($T_r + T_I + T_w$) The mean and minimum values are 303.7 and 282.5 ms, respectively.
T_c'	234.1	Control cycle time without relay changes ($T_r + T_I$) The mean and minimum values are 187.5 and 166.6 ms, respectively.
T_I	1	Time it takes to run the control logic generated from TuLiP.
T_r	58.5	Time it takes to read information from one sensor.
T_w	166.7	Time it takes to write information to all relays that need to take actions

[a]Values for T_c, T_c', and T_I were calculated from 20, 250, and 400 measurements, respectively. The times were calculated on a Macbook Pro with a 2.3 Ghz Intel Core i7 Processor.

generators and rectifier units as $T_r \approx T'_c/4$ because the time T_I is negligible compared to T_r. Therefore, a reasonable estimate of time T can be calculated with

$$T \approx \max(T_d) + \max(T_c) + \frac{4-n}{4}\max(T_c') \tag{6}$$

where $n \in \{1,2,3,4\}$ is a number that denotes the order of when the faulty environment variable is read in the code. Table 1 summarizes the variables that characterize the testbed. The relay delay time T_d can be found from the board specifications and should be less than 20 ms. We also get a relay delay time from the additional relays that measure the ac voltage. However, we assume that both the delay time from the board relays and the additional relays never exceed 20 ms. The control cycle times and the time it takes to run the code are estimated empirically. We calculated $T_r \approx T_c'/4 = 58.5$ ms and used Eq. (5) to calculate $\max(T_w) = \max(T_c) - \min(T_c') = 166.7$ ms and $\text{mean}(T_w) = \text{mean}(T_c) - \text{mean}(T_c') = 116.2$ ms.

For an example calculation, consider a configuration of two generators and two rectifier units, such as shown in the topology of Fig. 4, where one generator is read first in the code ($n = 1$) and the other generator is read second ($n = 2$). The rectifier units are read third and fourth, respectively, in the code. The maximum unpowered time for the left ac bus on the hardware testbed can be calculated with Eq. (6). Thus, $T \approx \max(T_d) + \max(T_c) + \frac{3}{4}\max(T_c') = 587.9$ ms. The unpowered times for the right ac bus, left dc bus, and right dc bus are calculated in the same way as 470.4, 411.8, and 353.3 ms, respectively.

Thus, it can be concluded that the unpowered time depends on where the fault is injected. The components connected to the right dc bus are least affected in the case of a fault, whereas the components connected to the left ac bus are most affected in the case of a fault.

V. Simulation Models for the Testbed

As discussed earlier, the correctness of an automatically synthesized control software should be interpreted with respect to the abstract models and specifications used in synthesis. Therefore, before control software is implemented and tested on actual hardware, it is useful to develop high-fidelity simulation models to explore potential shortcomings of the abstract models used in synthesis as well as to test continuous time properties not precisely captured by LTL specifications. This section details the simulation models for the hardware testbed including potential control architectures.

In this work, we used MATLAB Simulink [15], a graphical tool with a wide variety of built-in functions that can be assembled into complete systems, and, in particular, the SimPowerSystems toolbox [33], which is a physical modeling tool for electric power systems. With SimPowerSystems, models for an entire electric power system can be built just as they would be assembled from physical components. The constituent blocks are linked together with ideal conductors and may be linear, nonlinear, continuous, or discrete. It is also easy to integrate TuLiP controllers into Simulink models, as TuLiP has the ability to export controllers in the form of a MATLAB script that can be used as a Simulink block.

The SimPowerSystems models used in this study are built in accordance with the hardware. The generator units are connected to be 180 deg out of phase in order to create a shortcut when paralleled. The rectifier units in the Simulink model are built from a transformer, diode bridge, and capacitor to smooth out the ripple from the ac-to-dc conversion. Generators and rectifier units are equipped with fault injection inputs and fault sensors. The delays in the relay opening and closing times are modeled using saturated integrators to capture the formation of the electromagnetic field when the relays are actuated. Figure 9 shows the topology when a centralized control architecture is used. In the this model, the embedded MATLAB function block, bus power control unit (BPCU), runs the control logic script generated from TuLiP. There are several adjustable parameters in the model that are initialized with a configuration script. The relay delay time T_d is set to 20 ms. The time T_r it takes to read a sensor value is modeled with a delay between the sensing and the control command times. Because sensors are sequentially read, as indicated in Sec. IV.E, we set the delays from the fault sensors to kT_r, where $k \in \{0,1,2,3\}$ is the order in which the sensor is read ($k = 0$ is the first sensor, and $k = 3$ is the fourth). The time T_I for running the control logic, the mean time of $4T_r$, and the mean time of T_w to write the information to the relays are lumped into a sampling time T_s of the BPCU block; therefore, we let $T_s = 4T_r + T_I + T_w = T_c$. The mean value of T_c is chosen according to Table 1. To reflect the variability in timing, a uniform random value is added to the sensor reading delays $T_r \approx T_c'/4$ so that the overall control cycle time T_c' ranged between its maximum and minimum value given in Table 1. The configuration script is also used to define different scenarios that involved different combinations of fault conditions.

When we implement the distributed logic on the hardware testbed, it is still centralized in that only one relay board is connected to one computer; that is, all sensors and relays are connected to the same computer, which leads to the same timing characteristics, regardless of whether a centralized or distributed logic is used. With Simulink, it is possible to mimic the behavior of distributed control architecture with two relay boards controlled by different automatons running on two different computers. The distributed Simulink model is shown in Fig. 10, where ac and dc subsystems are sensed and controlled by different embedded MATLAB function blocks. There are several advantages of this distributed architecture. First, it increases the robustness of the system. For instance, even if the computer running the control logic for the dc subsystem fails or both rectifier units fail, the ac subsystem will continue operating and providing power to the ac buses. Second, it reduces the control cycle time by reducing the number of sensors each controller reads from. This effect would be particularly noticeable for the hardware used in the testbed, as the largest contribution to the control cycle times is the total time it takes to read data from the board, i.e., $4T_r$ for the centralized case and $2T_r$ for the distributed case. Finally, in the distributed architecture, it is possible to introduce and study the effects of asynchrony by choosing different sampling times for the two different controllers in the Simulink model.

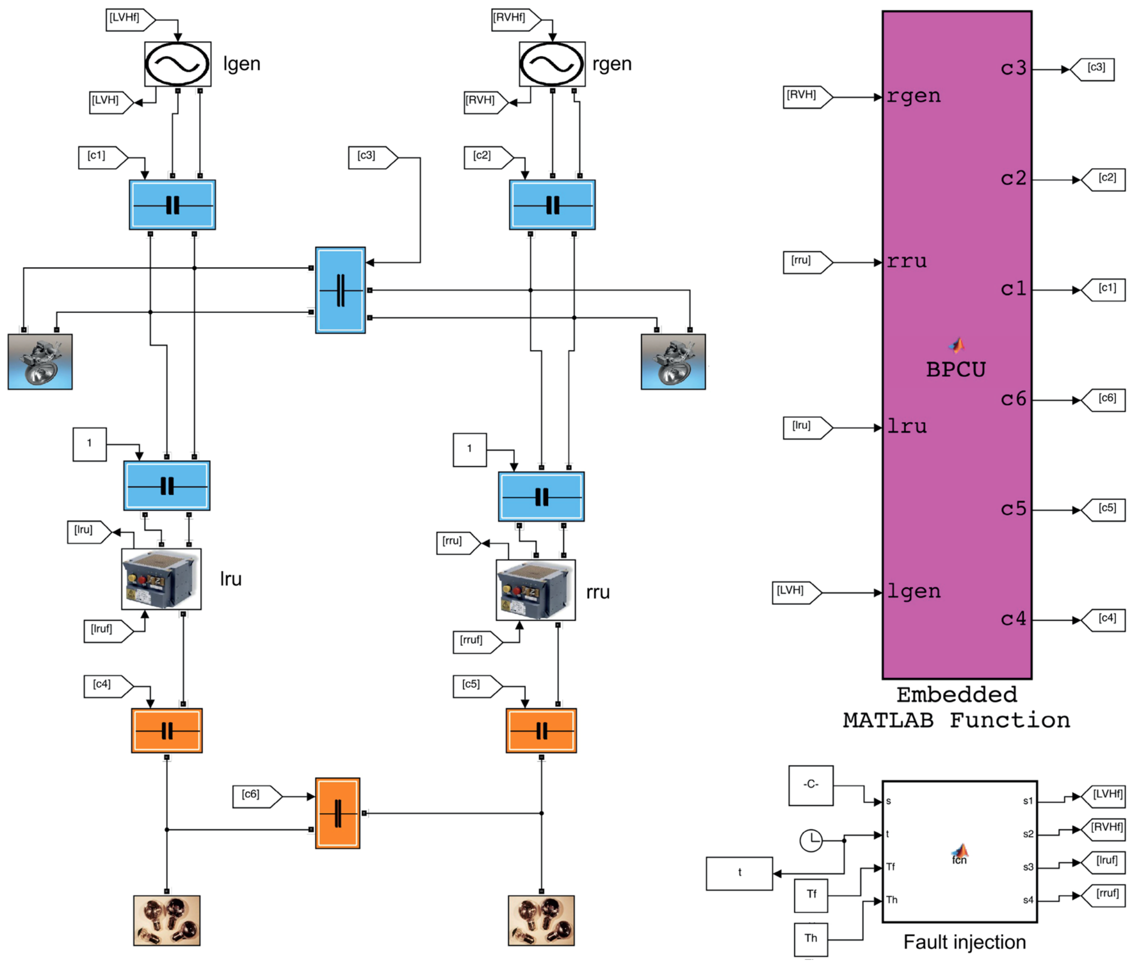


Fig. 9 Example SimPowerSystems model that corresponds to the single-line diagram in Fig. 4. The embedded MATLAB function called BPCU controls the system with a 16-state TuLiP automaton. There are two ac loads connected to the ac subsystem and two dc loads, located at the bottom of the figure, connected to the dc subsystem. In addition, a MATLAB function can be used for fault injection at a specific or random time.

VI. Controller Tests

We now discuss implementation of the controllers by running tests on both Simulink and hardware. The following examples also illustrate the differences between the high-fidelity simulations and testbed characteristics.

A. Example Control Test on Hardware

Figure 11 shows the voltage measurement for a centralized 16-state controller. The measurement was taken on the ac bus when the power cord to the transformer, which is read by the sensor for $n = 2$ in Eq. (6), was unplugged. The power cord was unplugged at $t = 2.83$ s, which is denoted by the first vertical line. The second vertical line from the left indicates when the controller reacted and powered up the bus through another path, which occurred at $t = 3.1$ s. Afterward, we plugged the power cord back in. At time $t = 3.73$ s, the controller reacted to the power cord, which can be seen by the discernible change in the sine curve. Once the transformer was plugged in again after a fault, the time during which the bus had been without power is not noticeable because the controller sends simultaneous commands to the two relays.

The measured bus unpowered times are listed in Table 2 for $n = 2$, with a maximum value of $T_{\max} = 414.9$ ms. As calculated in Sec. IV.E, time $T = 470.35$ ms; therefore, $T_{\max} < T$. We used a digital storage oscilloscope (Rigol DS1052E 50 MHz) for the measurements. The measurement data were imported into MATLAB to plot sinusoidal curves (e.g., Fig. 11) and analyze the signal to estimate the unpowered times.

B. Example Control Test on Simulink

Figure 12 illustrates the bus voltage measurements of the Simulink model when a fault was injected on a generator, which is read by the sensor for $n = 2$. Note the similarities with the hardware measurements based on the unpowered time and change in the sine curve when the faulty generator was switched on again.

The measured unpowered bus times are listed in Table 3; the maximum value is $T_{\max} = 333.0$ ms. Thus, we can verify with Eq. (6) that $T_{\max} < T$.

C. Comparison Between Simulation Results

Figures 11 and 12 show the similarities in the ac voltages measured in the Simulink-based simulations and in the hardware tests. Figure 13 illustrates the measured voltage on the dc bus when a rectifier fault was injected. The same unpowered behavior is seen in both figures of Fig. 13. However, no change could be detected in the voltage when the rectifier unit became healthy in the Simulink-based simulation. This is partly due to the ideal behavior of the components; e.g., contactor delays.

Tables 2 and 3 show that the unpowered time is slightly lower in the Simulink-based simulation compared to that in the hardware testbed. Table 4 lists the unpowered times of the distributed logic in Simulink. Note the decrease in the unpowered times compared to the values shown in

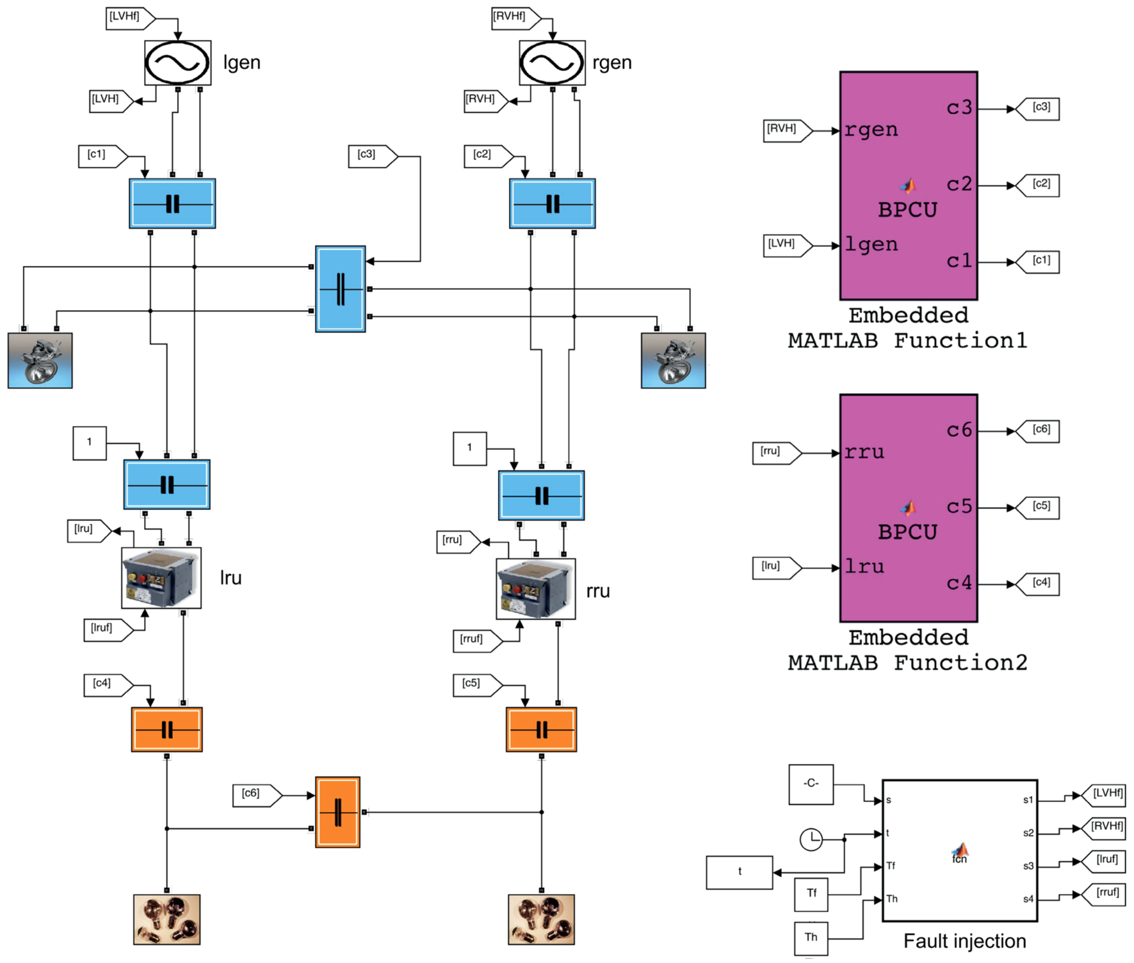


Fig. 10 Example SimPowerSystems model that corresponds to the single-line diagram in Fig. 4. The model has two embedded MATLAB functions called BPCU; each of them runs on a four-state TuLiP automaton. There are two ac loads connected to the ac subsystem and two dc loads in the bottom connected to the dc subsystem. In addition, there is a MATLAB function that can be used for fault injection at a specific or random time.

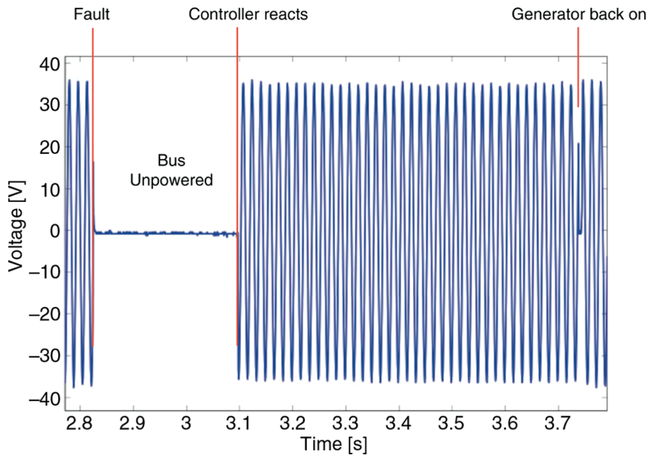


Fig. 11 Bus voltage measurement when a generator was switched off and then turned back on. The first vertical line indicates the fault, the second vertical line is when the controller reacts, and the third line is when the generator was turned back on.

Table 2 Time for which the bus was unpowered after a fault had been injected on the hardware testbed[a]

	Bus-unpowered time, ms
Mean	333.9
Maximum	414.9
Minimum	232.7

[a]These values were calculated using measurements from 10 fault injections.

Table 3 Time for which bus was unpowered after a fault had been injected in the Simulink modela[a]

	Bus unpowered time, ms
Mean	269.7
Maximum	379.0
Minimum	146.0

[a]These values were calculated using measurements from 10 fault injections.

Tables 2 and 3. An interesting observation from the executions of the centralized and distributed controllers (synthesized to realize the local specifications discussed in Sec. IV.B) is that, if the centralized controller senses that both rectifier units are unhealthy (i.e., the environment assumption on the dc side is violated), the entire controller stops working because a no-successor state has been reached. On the other hand, in the case of the distributed controllers, the ac subsystem continues executing and its own requirements are still fulfilled, whereas the dc subsystem stalls at a no-successor state with no guarantees on the satisfaction of its requirements.

VII. Conclusions

A formalized workflow was demonstrated for the design of control protocols for primary distribution in electric power systems on more-electric aircraft. The steps of the workflow include 1) establishing formal specifications that capture safety and performance requirements and abstract models of the allowable evolution of the underlying system; 2) automatically synthesizing control protocols from these specifications and models; and 3) validating/testing these protocols on high-fidelity simulations models and on hardware. For the hardware tests, we employed an academic-scale testbed developed in recent work [16] to initiate some of the salient features of power networks on aircraft.

TuLiP was used, which was developed for temporal logic planning [11], for synthesizing the control protocols and equipped with a MATLAB function translator to convert the TuLiP-generated control automata to a Simulink-compatible format. One of the challenges in the synthesis of reactive controllers is scalability as the size of the system and the number of the requirements increase. Compositional synthesis of distributed

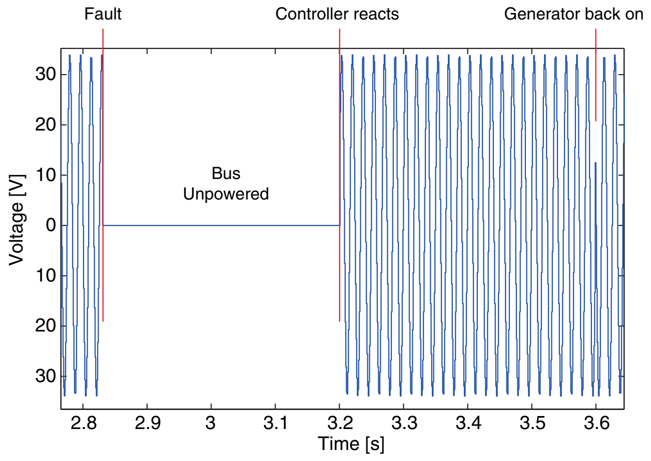
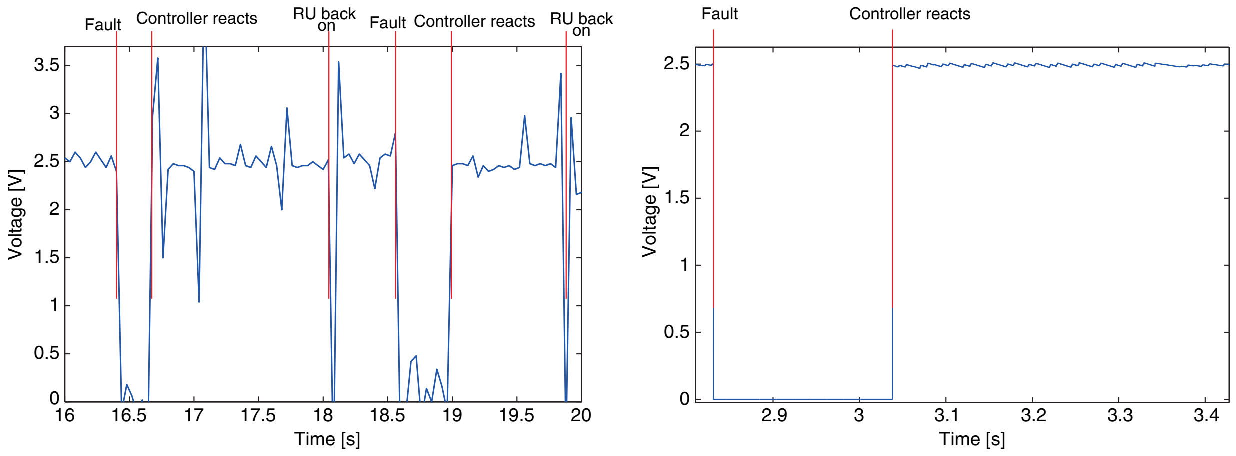


Fig. 12 Bus voltage measurement in Simulink when a generator was switched off and then turned back on.



a) Fault injected in the hardware

b) Fault injected in the simulink model

Fig. 13 Bus voltage measurement on the testbed when a rectifier unit was switched off and then turned back on a) when the rectifier was turned off and on twice, and b) when a fault was injected at 2.83 ms in the Simulink model.

Table 4 Time for which bus was unpowered after a fault had been injected in the distributed logic Simulink model[a]

	Bus-unpowered time, ms
Mean	204.4
Maximum	274.0
Minimum	121.0

[a]These values were calculated using measurements from 10 fault injections.

control protocols can be used to partly alleviate this limitation by recasting the problem into multiple smaller synthesis problems, for example, one for each of the panels in the network shown in Fig. 2. Current work focuses on partly automating the search for suitable interface specifications to be used in compositional synthesis (see, e.g., [34]). Another possibility for circumventing the scalability limitation is developing efficient synthesis algorithms and corresponding tools for aircraft electric power networks tailored to this problem domain.

On the hardware testbed, faults were injected by unplugging the power cords and changing the switches. With this method of fault injection, it is relatively difficult (if not impractical) to switch off a generator and a rectifier unit within the same control cycle. A more accurate approach to generate faults would be using an additional relay board that would enable systematical study of synchronous, correlated, and cascaded failures and their influence on controller performance; with the current method of fault injection, it could be difficult to switch off a generator and a rectifier unit within the same control cycle.

Through the high-fidelity simulations, it was shown that the bus unpowered time significantly decreases when distributed controllers running with different automata are used on two different computers and on two relay boards. Therefore, it would be more suitable on the hardware testbed to use a distributed control architecture more like that on an aircraft. On an aircraft, the controllers are embedded systems designated for specific tasks. To increase its reliability and performance, the hardware model could be adapted to run the relay boards through microcontrollers. Embedded code for these microcontrollers can be readily generated using MATLAB.

Acknowledgments

The authors wish to acknowledge the funding from the Industrial Cyber-Physical Systems Center and the U.S. Air Force Office of Scientific Research (award FA9550-12-1-0302), and they thank Rich Poisson from United Technologies Aerospace Systems for helpful discussions about the development of the hardware testbed.

References

[1] Rosero, J., Ortega, J., Aldabas, E., and Romeral, L., "Moving Towards a More Electric Aircraft," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 22, No. 3, 2007, pp. 3–9.
doi:10.1109/MAES.2007.340500

[2] Moir, I., and Seabridge, A., *Aircraft Systems: Mechanical, Electrical and Avionics Subsystems Integration*, 3rd ed., Wiley, New York, 2008.

[3] Feldman, A., Kurtoglu, T., Narasimhan, S., Poll, S., Garcia, D., de Kleer, J., Kuhn, L., and van Gemund, A., "Empirical Evaluation of Diagnostic Algorithm Performance Using a Generic Framework," *International Journal of Prognostics and Health Management*, Vol. 1, No. 2, 2010, pp. 1–28.

[4] Poll, S., et al., "Advanced Diagnostics and Prognostics Testbed," *In Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)*, 2007, pp. 178–185.

[5] Elston, J., Argrow, B., Frew, E., Houston, A., and Straka, J., "Evaluation of Unmanned Aircraft Systems for Severe Storm Sampling Using Hardware-In-The-Loop Simulations," *Journal of Aerospace Computing, Information, and Communication*, Vol. 8, No. 9, 2011, pp. 269–294.
doi:10.2514/1.53737

[6] Nuzzo, P., Xu, H., Ozay, N., Finn, J., Sangiovanni-Vincentelli, A., Murray, R., Donze, A., and Seshia, S., "A Contract-Based Methodology for Aircraft Electric Power System Design," *IEEE Access*, Vol. 2, 2014, pp. 1–25.
doi:10.1109/ACCESS.2013.2295764

[7] Clarke, E. M., and Wing, J. M., "Formal Methods: State of the Art and Future Directions," *ACM Computing Surveys*, Vol. 28, No. 4, 1996, pp. 626–643.
doi:10.1145/242223.242257

[8] Woodcock, J., Larsen, P. G., Bicarregui, J., and Fitzgerald, J., "Formal Methods: Practice and Experience," *ACM Computing Surveys*, Vol. 41, No. 4, 2009, pp. 19:1–19:36.
doi:10.1145/1592434

[9] Manna, Z., and Pnueli, A., *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer, New York, 1992.

[10] Emerson, E. A., "Temporal and Modal Logic," *Handbook of Theoretical Computer Science*, Vol. 2, 1990, pp. 995–1072.

[11] Wongpiromsarn, T., Topcu, U., Ozay, N., Xu, H., and Murray, R., "TuLiP: A Software Toolbox for Receding Horizon Temporal Logic Planning," *International Conference on Hybrid Systems: Computation and Control*, 2011, pp. 313–314.

[12] Xu, H., Topcu, U., and Murray, R., "A Case Study on Reactive Protocols for Aircraft Electric Power Distribution," *IEEE Conference on Decision and Control*, IEEE, Piscataway, NJ, 2012, pp. 1124–1129.

[13] Wongpiromsarn, T., Topcu, U., and Murray, R. M., "Formal Synthesis of Embedded Control Software: Application to Vehicle Management Systems," AIAA Infotech@Aerospace, AIAA Paper 2011-1506, 2011.

[14] Ozay, N., Topcu, U., and Murray, R. M., "Distributed Power Allocation for Vehicle Management Systems," *IEEE Conference on Decision and Control*, IEEE, Piscataway, NJ, 2011, pp. 4841–4848.

[15] Simulink, Ver. 8.0 (R2012b), MathWorks, Natick, MA, 2012.

[16] Rogersten, R., Xu, H., Ozay, N., Topcu, U., and Murray, R. M., "An Aircraft Electric Power Testbed for Validating Automatically Synthesized Reactive Control Protocols," *International Conference on Hybrid Systems: Computation and Control*, 2013, pp. 89–94.

[17] Michalko, R., "Electrical Starting, Generation, Conversion and Distribution System Architecture for a More Electric Vehicle," U.S. Patent No. 7439634, 2008.

[18] Lyu, M. R., (ed.), *Handbook of Software Reliability Engineering*, Vol. 3, McGraw–Hill, New York, 1996.

[19] Wood, A. J., and Wollenberg, B. F., *Power Generation, Operation, and Control*, Wiley, New York, 2012.

[20] "Aircraft Electric Power Characteristics," U.S. Dept. of Defense MIL-STD-704F, 2004, http://www.wbdg.org/ccb/FEDMIL/std704f.pdf [retrieved 30 Jan. 2014].

[21] Baier, C., and Katoen, J., *Principles of Model Checking*, MIT Press, Cambridge, MA, 1999.

[22] Xu, H., Topcu, U., and Murray, R. M., "Specification and Synthesis for Aircraft Electric Power Distribution," *IEEE Transactions on Control of Networked Systems*, 2013.

[23] Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., and Yi, W., *UPPAAL: A Tool Suite for Automatic Verification of Real-Time Systems*, Springer, New York, 1996, pp. 232–243.

[24] Hinton, A., Kwiatkowska, M., Norman, G., and Parker, D., "PRISM: A Tool for Automatic Verification of Probabilistic Systems," *Tools and Algorithms for the Construction and Analysis of Systems*, Springer, New York, 2006, pp. 441–444.

[25] Pnueli, A., "Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends," *Current Trends in Concurrency. Overviews and Tutorials*, edited by de Bakker, J. W., de Roever, W. P., and Rozenberg, G., Springer–Verlag, New York, 1986, pp. 510–584.

[26] Piterman, N., Pneuli, A., and Sa'ar, Y., "Synthesis of Reactive (1) Designs," *Verification, Model Checking and Abstract Interpretation*, Vol. 3855, 2006, pp. 364–380.
doi:10.1007/11609773_24

[27] Pnueli, A., Sa'ar, Y., and Zuck, L., "JTLV a Framework for Developing Verification Algorithms," *International Conference on Computer Aided Verification*, 2010, pp. 171–174.

[28] Jobstmann, B., Galler, S., Weiglhofer, M., and Bloem, R., "Anzu: A Tool for Property Synthesis," *Computer Aided Verification*, Springer, New York, 2007, pp. 258–262.

[29] Jobstmann, B., and Bloem, R., "Lily—a LInear Logic sYnthesizer," 2006, data available online at http://www.iaik.tugraz.at/content/research/design_verification/lily/ [retrieved 4 Sept. 14].

[30] Filiot, E., Jin, N., and Raskin, J.-F., "Antichains and Compositional Algorithms for LTL Synthesis," *Formal Methods in System Design*, Vol. 39, No. 3, 2011, pp. 261–296.
doi:10.1007/s10703-011-0115-3

[31] Ehlers, R., "Unbeast: Symbolic Bounded Synthesis," *Tools and Algorithms for the Construction and Analysis of Systems*, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 272–275.

[32] Ehlers, R., "Experimental Aspects of Synthesis," *Electronic Proceedings in Theoretical Computer Science*, Vol. 50, 2011, pp. 1–16.
doi:10.4204/EPTCS.50.1

[33] SimPowerSystems, Ver. 5.7 (R2012b), MathWorks, Inc., Natick, MA, 2012.

[34] Alur, R., Moarref, S., and Topcu, U., "Counter-Strategy Guided Refinement of GR(1) Temporal Logic Specifications," *Formal Methods in Computer-Aided Design*, 2013, pp. 26–33.
doi:10.1109/FMCAD.2013.6679387

L. Long
Associate Editor