

State machine modeling of the Space Launch System Solid Rocket Boosters

Joshua A. Harris* and Ann Patterson-Hine†

NASA Ames Research Center, Moffett Field, CA, 94035

The Space Launch System is a Shuttle-derived heavy-lift vehicle currently in development to serve as NASA’s premiere launch vehicle for space exploration. The Space Launch System is a multistage rocket with two Solid Rocket Boosters and multiple payloads, including the Multi-Purpose Crew Vehicle. Planned Space Launch System destinations include near-Earth asteroids, the Moon, Mars, and Lagrange points. The Space Launch System is a complex system with many subsystems, requiring considerable systems engineering and integration. To this end, state machine analysis offers a method to support engineering and operational efforts, identify and avert undesirable or potentially hazardous system states, and evaluate system requirements. Finite State Machines model a system as a finite number of states, with transitions between states controlled by state-based and event-based logic. State machines are a useful tool for understanding complex system behaviors and evaluating “what-if” scenarios. This work contributes to a state machine model of the Space Launch System developed at NASA Ames Research Center. The Space Launch System Solid Rocket Booster avionics and ignition subsystems are modeled using MATLAB®/Stateflow® software. This model is integrated into a larger model of Space Launch System avionics used for verification and validation of Space Launch System operating procedures and design requirements. This includes testing both nominal and off-nominal system states and command sequences.

Nomenclature

BEO	Beyond Earth Orbit
BS	Booster Stage
CS	Core Stage
FCOG	Flight Computer Operational Group
FSM	Finite State Machine
ICPV	Interim Cryogenic Propulsion Stage
MET	Mission Elapsed Time
MPCV	Multi-Purpose Crew Vehicle
MPS	Main Propulsion System
SLS	Space Launch System
SRB	Solid Rocket Booster
SRM	Solid Rocket Motor

I. Introduction

MANAGED spacecraft necessarily demand the most stringent levels of safety and reliability. These vehicles carry astronauts into the harsh and unforgiving environment of space and even the slightest failure has the potential to be catastrophic. The need for safety- and reliability-focused systems engineering has become

*NASA Aeronautics Scholarship Program Intern, Intelligent Systems Division, NASA Ames Research Center, Texas A&M University, AIAA Student Member.

†Branch Chief, Discovery and Systems Health, Intelligent Systems Division, NASA Ames Research Center.

even more important as space exploration missions become more ambitious (such as planned manned interplanetary missions) and as spacecraft themselves become increasingly complicated systems. This includes both hardware and software, as the ubiquity of computers has resulted in massive increases in software complexity and size.

Model-based systems engineering is a promising solution to this problem, especially for complex aerospace systems.¹ As systems become complex to the point where no single engineer can fully understand systems in their entirety it becomes critical to provide methods by which engineers can describe and understand the effects and interrelationships of multiple subsystems of a complex system. Tools such as the Unified Modeling Language (UML) derived Systems Modeling Language (SysML) provide the system engineer increasingly more powerful methods with which to represent the requirements and behaviors of complex systems, both in hardware and software.²

Another common tool for the systems engineer is the Finite State Machine (FSM), which can be used to mathematically model a system as a finite number of states, with interactions and transitions between states. The system behavior is thus abstracted to the possible states the system can be in. State machines, particularly those which can be modeled and executed in software, are especially useful for analyzing the system's behavior and for identifying and avoiding undesirable or potentially hazardous system behavior. State machines can also be used as a basis for the actual system Flight Software (FSW). Further system behavior can be captured by linking a state machine model to a physics-based model of the system dynamics. Particularly useful is the Statechart state machine model introduced by Harel.³

State machines form the basis for a systems engineering methodology known as *System State Analysis*. System state analysis uses state machines to model a system and identify undesirable or potentially hazardous states, evaluate requirements and objectives, examining the relationships between subsystems, evaluate operational procedures, and other systems engineering and fault management tasks. Examples of system state analysis in the open literature include Ingham et al.'s introduction and application of system state analysis to spacecraft systems engineering and control,⁴ the use of UML for state analysis,⁵ and, similarly, the use of SysML for state analysis.⁶

In this paper, a state machine model of the Solid Rocket Boosters (SRBs) of the Space Launch system is developed and integrated with an existing model of the SLS avionics and Main Propulsion System. The model of the SRBs focuses primarily on the booster avionics systems. System state analysis is applied to the booster model with a focus on both nominal and offnominal sequences for booster ignition and separation.

The paper is organized as follows. Section II reviews the theory of Finite State Machines (FSMs). Section III introduces the Space Launch System (SLS) and the reference configurations of the SLS used in this work. Section IV discusses the state machine model of the SLS created and its use for state analysis of the SLS. Finally, Section V presents conclusions and recommendations.

II. Finite State Machines

Finite state machines (FSMs) are a model of computation that represents logic in terms of a finite number of states, with transitions between states. These transitions can be based on event- or state-based logic. State machines can be used as a computational model for designing and implementing software, a technique commonly used for high-reliability software, especially Flight Software. From a systems engineering perspective, state machines are exceptionally useful for abstracting complex systems into a series of states that can be used to qualify the behavior of a system under specified circumstances.

State machines can be commonly represented as a state diagram or state transition table. A state diagram is a directed graph with the state represented as nodes and the transitions as edges. Figure 1 is a simple example of a finite state machine with two states, S_1 and S_2 , drawn as a state diagram. State S_1 is the default state which moves to State S_2 with transition T_1 . Here, T_1 is a event- or state-based condition that causes the state machine to move from S_1 to S_2 . When the machine enters S_2 , it remains in S_2 when T_3 occurs or returns to S_1 when transition T_2 occurs.

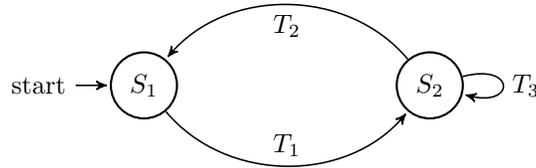


Figure 1: Example finite state machine.

Table 1 shows the same finite state machine represented as a state transition table. The left axis of the table lists the current state and the top axis lists the current transitions. The table is a mapping of state and transition pairs to the resulting state of the system. One such mapping is $(S_2, T_2) \mapsto S_1$. Invalid state/transition combinations are denoted by an en-dash.

State \ Transition	Transition		
	T_1	T_2	T_3
S_1	S_2	-	-
S_2	-	S_1	S_2

Table 1: State transition table for example state machine.

Both the state diagram and state transition table representations of the state machine are equivalent. The state diagram can be constructed from the information in the table and vice versa.

A. Executable state machines

Executable state machines can be evaluated in software and provide a framework that allows the system engineer to analyze a system as opposed to simply describing it. This brings several advantages, including the ability to analyze “what-if” scenarios by specifying combinations of states of interest and observing the resulting progression of states. Furthermore, executable state machines incorporate the time dimension. This provides the ability to examine the progression of a system from state to state, record system state time histories, and inject states and events at desired points in the system timeline. This latter point gives executable state machines the capability to model a system throughout its entire mission timeline.

The Stateflow package for the MATLAB/Simulink[®] environment provides a framework for creating executable state machines that can interface with the C and MATLAB languages.^{7,8,9} Stateflow denotes a state machine as a chart which consists of multiple states and sub-groupings of states. Each state executes actions at three possible times:

1. Upon entering the state
2. During the state
3. Upon exit of the state

These states enable the creation of state machines in Stateflow that can represent considerably complex systems with relative ease. Such mechanisms can also be used with internal variables to enable communications and complex internal state behavior. Stateflow additionally allows for states to be executed in series or in parallel (AND/OR state decompositions, respectively). The former is the general behavior of a state machine, where the system is in one state at a time, whereas the latter allows for a state machine/state chart to contain multiple subsystems simultaneously, each holding its own state. This adds greater flexibility at the cost of increased model complexity. Stateflow also allows for hierarchal states based on the Statecharts developed by Harel.

Figure 2 revisits the example state machine of Figure 1 as an executable state machine in Stateflow. Note that due to the syntax of Stateflow subscripts are replaced with underscore notation.

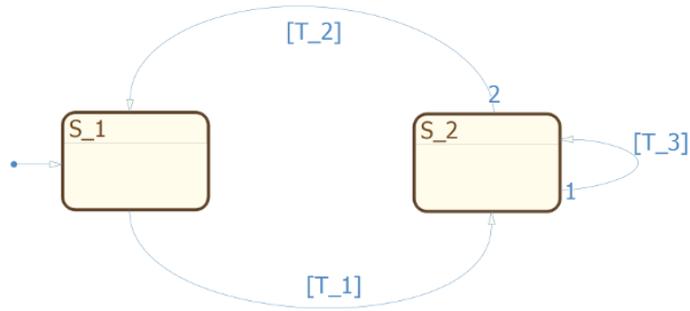


Figure 2: Executable state machine example in Stateflow.

Figure 3 shows the state machine in Figure 2 being evaluated for the case where T_1 and T_3 are true and T_2 is false. State S_1 is active in Figure 3a and S_2 is active in Figure 3b. Entry actions are defined for the start state to set the values of the transition conditions T_i but are not shown in the figures.

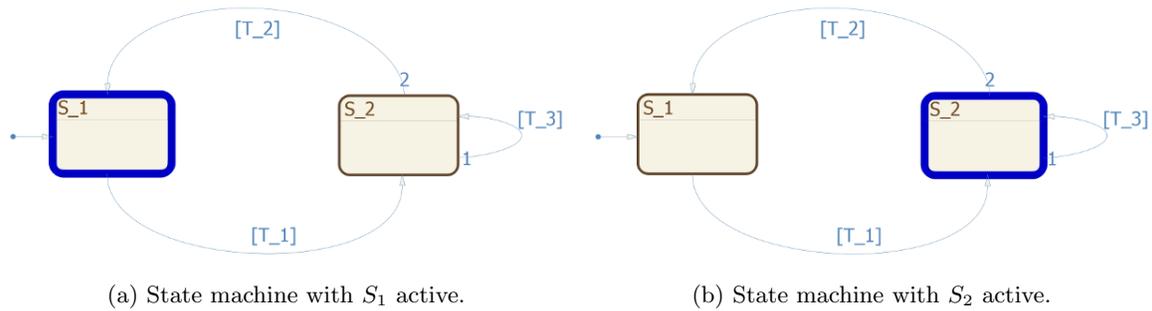


Figure 3: Evaluated state machine example.

The state highlighting in Figure 3 is particularly useful for visualizing the system states interactively, and has proven useful for state analysis. The use of executable state machines for system state analysis is discussed further in Section IV.

III. Space Launch System Overview

The Space Launch System (SLS) is NASA's newest launch vehicle. The SLS is a Shuttle-derived heavy-lift vehicle currently under development with first launch scheduled for 2017. Potential destinations for the SLS include near-Earth asteroids, the Moon, Mars, Lagrange points, and other Beyond Earth Orbit (BEO) destinations. The SLS is a modular rocket with multiple configurations and payloads.¹⁰

The SLS will initially fly in the Block I configuration, shown in Figure 4. This configuration consists of the Core Stage (CS), two Solid Rocket Boosters (SRBs), the Interim Cryogenic Propulsion Stage (ICPS), and the Orion/Multi-Purpose Crew Vehicle (MPCV) spacecraft. The ICPS for Block I of SLS is the same second stage as used on the Delta III and Delta IV expendable launch vehicles. The Block I configuration of the SLS is the reference configuration for the state machine model presented in this paper.

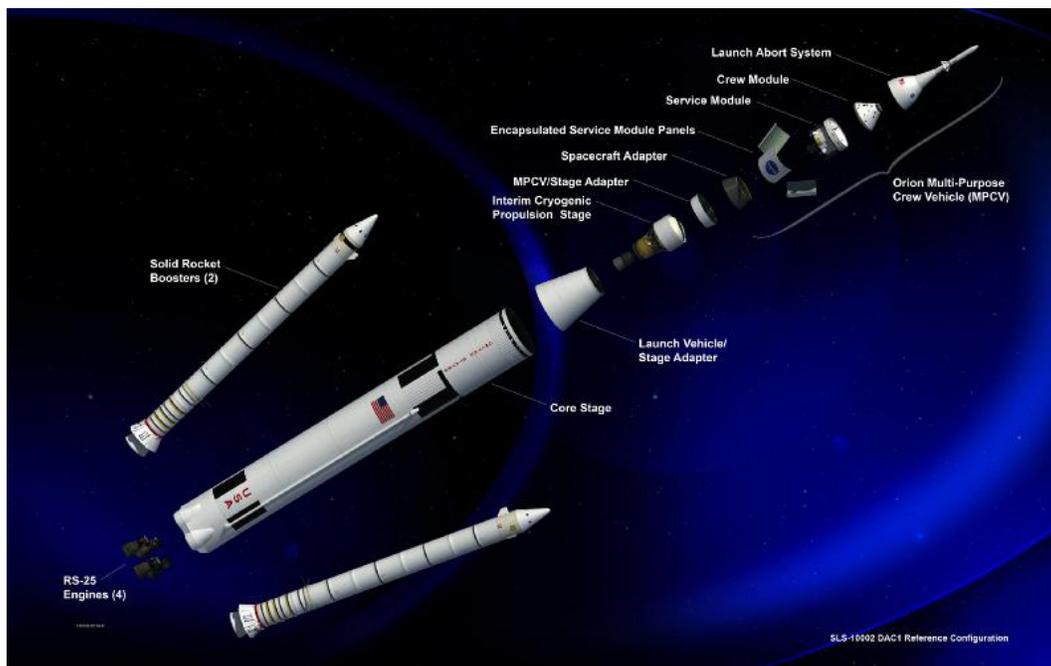


Figure 4: Space Launch System Block I configuration (NASA image).

The Core Stage of the Block I vehicle is propelled by four Shuttle-heritage RS-25 rocket engines fueled by LOX/LH₂, and contains the Main Propulsion System and most of the avionics. The avionics on the Core Stage contain the Flight Computer Operational Group (FCOG), three triple-redundant flight computers that control the SLS vehicle. The FCOG additionally monitors and sends commands to the SRBs and communicates with MPCV/Orion.

The Solid Rocket Boosters (SRBs) used in the Block I configuration are based on the boosters used in the Space Transportation System (Shuttle) and the solid rocket Core Stage of the Ares I rocket. The SRBs feature a five-segment Solid Rocket Motor (SRM) and the associated electronics for command and monitoring of the rocket engine. The SRBs receive commands from and send telemetry to the FCOG on the SLS Core Stage. These commands are passed from the FCOG to the SRB control electronics over a data bus between the CS and SRBs and from there through Line Replaceable Units to the appropriate destination, e.g. the SRM or the Thrust Vector Control subsystem.

Future configurations of the SLS include the 130 ton Block II, which replaces the SRBs with as-of-yet undetermined Advanced Boosters and a new Earth Departure Stage upper stage of the rocket based on the J-2X rocket engine. The Block II configuration is planned for introduction in the 2030s and is consequently not a concern for the current state machine model.

IV. SLS State Analysis

The SLS State Analysis project is an effort by the Discovery and Systems Health group at NASA Ames Research Center in collaboration with SLS engineering elements at other NASA centers to create an accurate executable state machine model of the Flight Software and Flight Hardware of the SLS Block I vehicle for Mission and Fault Management analysis. The SLS state analysis project builds on previous work done on the Ares I rocket for the Constellation project and work done during investigation of unintended acceleration of Toyota vehicles.¹¹ This previous work demonstrates the applicability of system state analysis for both aerospace and non-aerospace systems.

The deliverable of the SLS state analysis project is an executable state machine model of the Space Launch System Block I configuration created in Stateflow and analysis of the SLS using the created model. Potential analyses of the SLS using the model include:

- Identification of undesirable or hazardous states

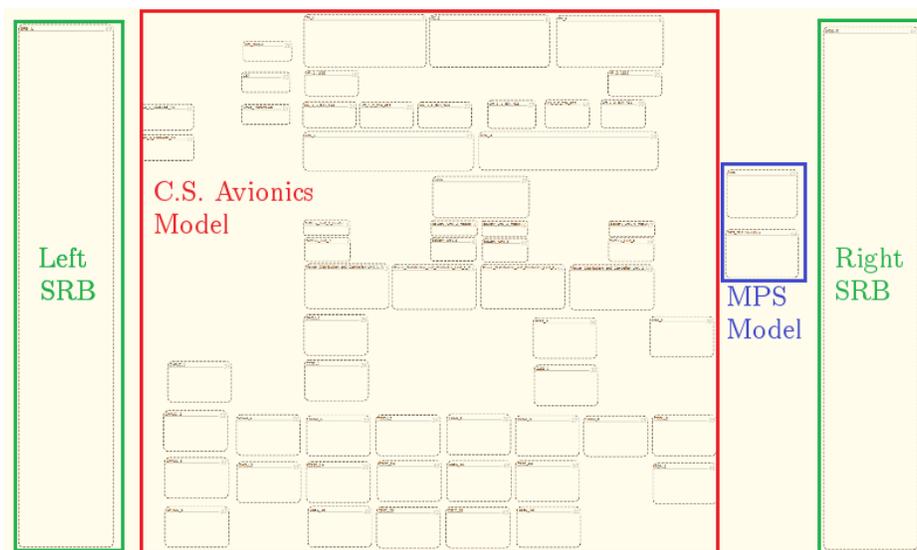


Figure 6: High-level overview of main model with labeled subsystems.

States which correspond to the Core Stage avionics are outlined in red, while the Main Propulsion System is a group of subcharts outlined in blue. Finally, the subcharts corresponding to the two Solid Rocket Boosters are outlined in green. These models are described in the following section.

A. Solid Rocket Booster state machine model

The current iteration of the model features Core Stage avionics and the Main Propulsion System. This section describes work done to model the Solid Rocket Boosters and integrate them with the main model. The left and right SRBs are modeled as states in the main model, with their subsystems modeled as substates. The SRB model includes the following systems:

- Avionics
- Flight Safety System
- Separation
- Solid Rocket Motor
- Thrust Vector Control

Each of these subsystems is described further in this section. Note that the SRB model focuses on the system states and abstracts the system dynamics and physics into a small number of appropriate states. Certain states are abstracted at a very high level into two states: nominal and offnominal. Figure 7 shows the above subsystems in the SRB model.

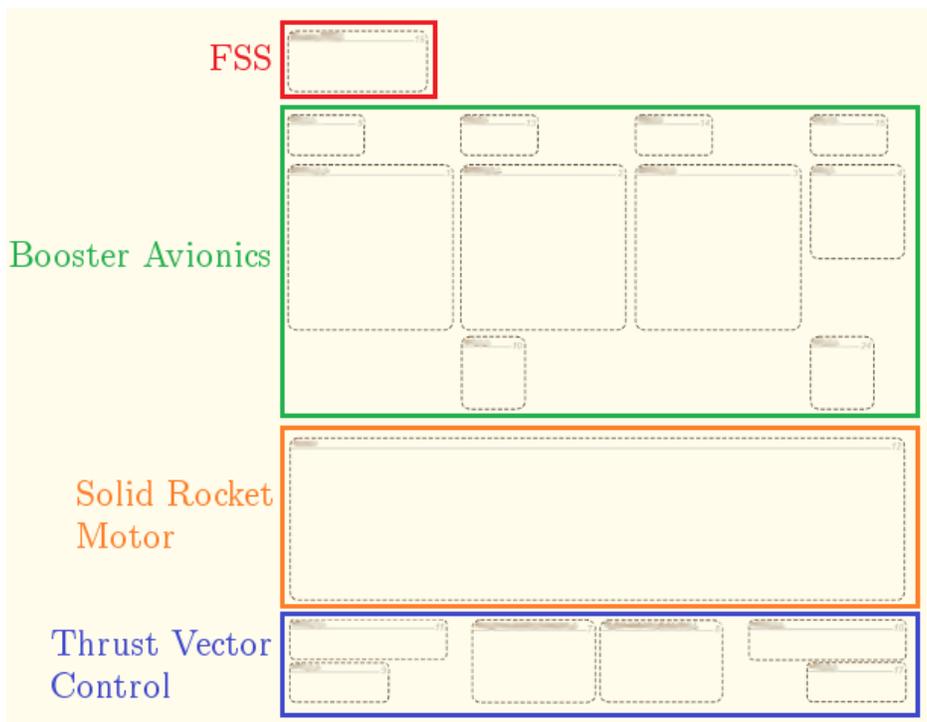


Figure 7: Solid Rocket Booster Stateflow model.

1. Avionics

The SRB features triple-redundant avionics systems that are responsible for control of the SRB subsystems and power distribution in the booster. These systems receive commands from and relay telemetry to the FCOG. Communications data buses are implemented using Stateflow buses with fields corresponding to commands and telemetry. This approach makes it easy to simulate the passage of commands and telemetry, but does not model communications delays, bus buffer sizes, or the order in which messages are sent or received.

2. Flight Safety System

The Flight Safety System (FSS) monitors for inadvertent separation of the SRBs from the CS and for destruct commands from the ground. The FSS is modeled as a subsystem of the SRB and connects to the booster avionics to receive safe and arm commands and to the CS to receive commands from the Range Safety Officer. The FSS is mostly independent from the other booster systems. Figure 8 is a redacted example of logic from the FSS. Note the pseudo-states used to implement command and telemetry.

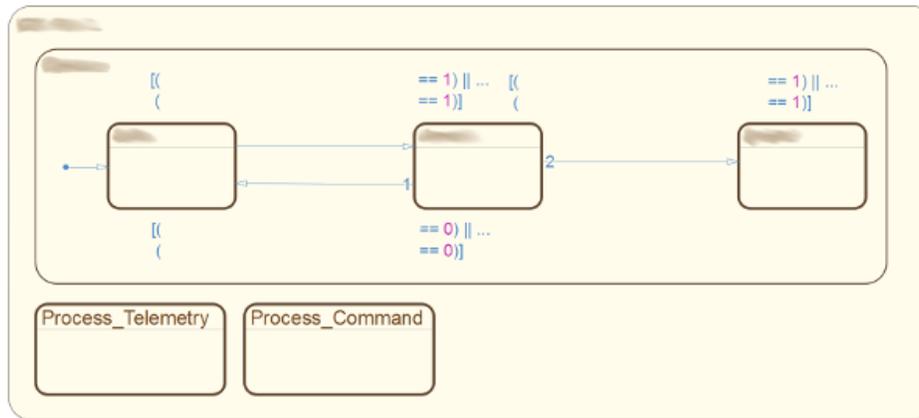


Figure 8: Example of logic from an FSS subsystem.

3. Separation Systems

There is not a “separation system” in the booster; in this paper the term is used to describe the various components and subsystems involved with the separation of the SRBs from the Core Stage. All of the components involved are modeled, but they are not individually modeled in detail. The use case for booster separation is further discussed in Section IV.B.2.

4. Solid Rocket Motor

The SRM itself is abstracted into three states (off, ignited, and burnout) in order to avoid the need to create a full Simulink model of a solid rocket engine. The main interest in the SRM is ignition. Thus, the ignition controllers and safe and arm devices are fully modeled in the state machine. This allows analysis of the booster ignition procedures, discussed further in Section IV.B.1.

5. Thrust Vector Control

The TVC system is another system that requires a physical model for a high-fidelity simulation. The servoactuators used in the TVC are some of the most complex mechanisms in the vehicle. Controlling the actuators requires feedback of differential pressure sensors. The TVC system is abstracted in the state machine model since this feedback is for multiple continuous variables and the TVC system would arguably need its feedback control logic modeled. The pressure transducers are modeled at a high level, and the actual position of the TVC nozzle is not included as a state. The servovalves that drive the TVC nozzle are modeled as being in a nominal state and three off-nominal states corresponding to the failure modes of these devices.

B. Solid Rocket Booster use cases

The two main areas of interest for state analysis of the booster at the time of the paper are SRB ignition during the pre-launch phase and SRB separation during the launch phase. The following subsections describe each of these cases.

1. Ignition

Prior to launch the flight computers issue a command sequence to ignite the SRMs. At a high level, this involves arming the firing devices and then sending the commands to ignite the SRMs. Aside from nominal ignition, there are two important off-nominal ignition cases: failure to ignite one or both SRMs, and the SRMs igniting with too much time separation between the ignition of each individual SRM.

2. Separation

The Solid Rocket Boosters nominally separate from the Core Stage once two conditions have been met:

$$t \geq \hat{t} \quad (1)$$

$$p \leq \hat{p}. \quad (2)$$

In Eq. (1), t is the Mission Elapsed Time (MET) in seconds and \hat{t} is the minimum time before booster separation is permitted. The MET is represented using a clock object in Simulink. The current implementation issues the booster separation commands at the appropriate time using the main command sequence file. Similarly, in Eq. (2), p is the pressure in the Solid Rocket Motor in psi and \hat{p} is a minimum pressure value before the SRB is considered to have burned out. In Stateflow, these conditions can be combined to form a transition condition as $[(t \geq \hat{t}) \ \&\& \ (p \leq \hat{p})]$. A simplified linear model of the pressure is implemented in Simulink and connected to Stateflow as the continuous pressure data does not fit well into the state machine formulation. This model will be replaced with actual pressure data from the SRMs when it becomes available.

Interesting separation cases include inadvertent separation and failure to separate after the boosters have burned out.

V. Summary and Recommendations

This paper describes the creation of a finite state machine model of the Solid Rocket Boosters for the Space Launch System in the Stateflow software package. The booster ignition and separation sequences were modeled and state analysis was run for both nominal and offnominal scenarios. This model was integrated with a larger model of the Space Launch System Core Stage avionics and Main Propulsion System.

The Space Launch System state machine model is found to be a useful tool for systems engineering and fault management. The state machine model can easily be used to test different launch scenarios and detect off-nominal behavior. Finally, the following recommendations are made for future work to improve the model:

1. The command and telemetry bus models should be reworked to more accurately simulate the actual data buses. What happens if multiple faults occur and overload the data buses, and how do you recover from this? Implementing this change would allow the state machine to demonstrate this situation and similar cases involving faults with the buses.
2. Accurate physics of Space Launch System systems should be implemented in Simulink and integrated with the state machine to increase the fidelity of the model once the Space Launch System has been fully modeled. Systems that would benefit from this include the Thrust Vector Control systems on both the Booster Stage and Core Stage, the Solid Rocket Motors, and the RS-25 engines on the Core Stage. Models of these systems should be available from the engineering teams working on them.
3. Mutually exclusive states in the Solid Rocket Boosters should be identified and added to the main model's existing framework for disallowed states.

Future work will include further analysis on the Solid Rocket Booster model for additional use cases.

Acknowledgments

J.A. Harris thanks the NASA Aeronautics Scholarship Program for funding and the opportunity to conduct this work at NASA Ames Research Center.

References

¹Ingham, M. et al., "A Model-Based Approach to Engineering Behavior of Complex Aerospace Systems," *Proceedings of the Infotech@Aerospace Conference*, American Institute of Aeronautics and Astronautics, Garden Grove, CA, June 2012.

²Friedenthal, S., Moore, A., and Steiner, R., *A Practical Guide to SysML: The Systems Modeling Language*, Kaufmann OMG Press, 2nd ed., 2012.

³Harel, D., “Statecharts: A visual formalism for complex systems,” *Science of computer programming*, Vol. 8, No. 3, 1987, pp. 231–274.

⁴Ingham, M. D., Rasmussen, R. D., Bennett, M. B., and Moncada, A. C., “Engineering complex embedded systems with state analysis and the mission data system,” *Journal of Aerospace Computing, Information, and Communication*, Vol. 2, No. 12, 2005, pp. 507–536.

⁵Murray, A. and Rasmussen, R., “A UML profile for State Analysis,” *Aerospace Conference, 2010 IEEE*, IEEE, 2010, pp. 1–13.

⁶Wagner, D. et al., “An Ontology for State Analysis: Formalizing the Mapping to SysML,” Institute of Electrical and Electronics, Big Sky, MT, Mar 2012.

⁷The MathWorks Inc., “MATLAB[®] Release 2012b,” <http://www.mathworks.com/products/matlab/>, September 2012, Computer software.

⁸The MathWorks Inc., “Simulink[®] 8.0,” <http://www.mathworks.com/products/simulink/>, September 2012, Computer software.

⁹The MathWorks Inc., “Stateflow[®] Release 2012b,” <http://www.mathworks.com/products/stateflow/>, September 2012, Computer software.

¹⁰Anon., “Preliminary Report Regarding NASA’s Space Launch System and Multi-Purpose Crew Vehicle,” Tech. rep., NASA, January 2011.

¹¹Kirsch, M.T. (ed.), “Technical Support to the National Highway Traffic Safety Administration (NHTSA) on the Reported Toyota Motor Corporation (TMC) Unintended Acceleration (UA) Investigation,” Tech. Rep. TI-10-00618, NASA Engineering and Safety Center, January 2011.