NASA/TM—2012-217444

# A Novel Technique for Running the NASA Legacy Code LAPIN Synchronously With Simulations Developed Using Simulink

*Daniel R. Vrnak, Thomas J. Stueber, Dzu K. Le*
*Glenn Research Center, Cleveland, Ohio*

June 2012

# NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include creating custom thesauri, building customized databases, organizing and publishing research results.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*

- E-mail your question via the Internet to *help@sti.nasa.gov*

- Fax your question to the NASA STI Help Desk at 443–757–5803

- Telephone the NASA STI Help Desk at 443–757–5802

- Write to:
  NASA Center for AeroSpace Information (CASI)
  7115 Standard Drive
  Hanover, MD 21076–1320

NASA/TM—2012-217444

# A Novel Technique for Running the NASA Legacy Code LAPIN Synchronously With Simulations Developed Using Simulink

*Daniel R. Vrnak, Thomas J. Stueber, Dzu K. Le*
*Glenn Research Center, Cleveland, Ohio*

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

June 2012

*Level of Review*: This material has been technically reviewed by technical management.

Available from

NASA Center for Aerospace Information
7115 Standard Drive
Hanover, MD 21076–1320

National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Available electronically at http://www.sti.nasa.gov

# A Novel Technique for Running the NASA Legacy Code LAPIN Synchronously With Simulations Developed Using Simulink

Daniel R. Vrnak, Thomas J. Stueber, and Dzu K. Le
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

## Abstract

This report presents a method for running a dynamic legacy inlet simulation in concert with another dynamic simulation that uses a graphical interface. The legacy code, NASA's LArge Perturbation INlet (LAPIN) model, was coded using the FORTRAN 77 (The Portland Group, Lake Oswego, OR) programming language to run in a command shell similar to other applications that used the Microsoft Disk Operating System (MS-DOS) (Microsoft Corporation, Redmond, WA). Simulink (MathWorks, Natick, MA) is a dynamic simulation that runs on a modern graphical operating system. The product of this work has both simulations, LAPIN and Simulink, running synchronously on the same computer with periodic data exchanges. Implementing the method described in this paper avoided extensive changes to the legacy code and preserved its basic operating procedure. This paper presents a novel method that promotes inter-task data communication between the synchronously running processes.

## Introduction

Advancing aeronautics research for societal benefit is one of NASA's Strategic Goals (Ref. 1). This goal involves developing innovative solutions and advanced technologies to improve future air transportation systems. To this end, the NASA Fundamental Aeronautics Program (FAP), a program within the NASA Aeronautics Research Mission Directorate (ARMD), is conducting long-term, cutting-edge research in all flight regimes—subsonic, supersonic, and hypersonic (Ref. 2). Mastering the science of air-breathing hypersonic flight in Earth's atmosphere is the challenge for the NASA FAP Hypersonic Project (Ref. 3). To make possible air-breathing hypersonic flight, research to develop Combined Cycle Engine (CCE) propulsion systems that integrate either high-speed turbine engines or rocket engines with scramjet engines is needed. Ongoing research at the NASA Glenn Research Center at Lewis Field (GRC), under the FAP hypersonic project, focuses on developing a propulsion system that combines the cycles of a turbine engine and a Dual-Mode ScramJet (DMSJ) combustor—a Turbine-Based Combined Cycle (TBCC) system. This propulsion system uses the turbine engine to boost a vehicle from takeoff through subsonic, transonic, and lower supersonic flight—less than Mach 4. Beyond Mach 4 conditions, the turbine system is ineffective. To continue further acceleration, a DMSJ is used and the turbine flowpath is closed off to protect the engine and to reduce drag (Ref. 4). The procedure to switch between the two engines at a transition Mach number is termed mode transition. To facilitate TBCC mode transition research, NASA FAP Hypersonic Project designed and developed a split flow design inlet with two flow-paths (Refs. 5 and 6). The flow-paths are aerodynamically designed to match engine airflow demands—a Low-Speed Flow-Path (LSFP) for the turbine engine and a High-Speed Flow-Path (HSFP) for the DMSJ. This inlet system, designed as a scaled-up version of a previously tested inlet (Ref. 4) was fabricated to serve as a large scale testbed with many parametric features. This testbed, identified as the CCE Large-scale Inlet for Mode transition eXperiments (CCE-LIMX) will enable investigations of mode transition procedures in the GRC 10- by 10-Foot Supersonic Wind Tunnel (SWT).

The research associated with the CCE inlet considers an air breathing hypersonic vehicle that employs a TBCC propulsion system. The propulsion system inlet is conducive to enabling mode transition and is designed to perform well prior, during, and after the inlet mode transition event. Prior to mode transition, flight speeds above Mach 2 and below the transition Mach number, the LSFP having a

mixture of internal and external compression allows optimal propulsion system performance by supplying airflow to the turbine at a required high pressure level with low distortion while minimizing drag. During mode transition the LSFP geometry is configured to optimize the propulsion system at the transition Mach number. This phase of operation is challenging because the LSFP must remain started and maintain distortion at the AIP below turbine engine tolerance levels while the inlet cycles through off-design points. Designing the LSFP to meet these aggressive goals requires the terminal shock to be maintained as close as possible to the inlet throat. This is a marginally stable operating condition because an external airflow transient, such as atmospheric turbulence, or an internal airflow transient, such as a reduction in engine airflow demand, can cause the terminal shock to move off of its design location. If the terminal shock moves forward of the throat, it will be expelled ahead of the inlet cowling. This shock expulsion, referred to as inlet unstart, causes a large rapid reduction in mass flow to and pressure at the engine, and thus a large thrust loss along with increased drag. The closer the terminal shock is to the throat, the smaller the disturbance necessary to unstart the inlet. It is customary to design turbine inlet systems to have a sufficiently large stability margin between nominal shock position and inlet throat to tolerate such transients without unstarting. Increasing the stability margin, operating super-critical, will lower pressure and increase distortion at the LSFP diffuser exit—decreased turbine airflow quality. Since any loss in inlet performance is reflected directly as a loss in propulsion-system thrust and efficiency, super-critical operation is avoided. An acceptable balance between system stability margin and performance can be provided through design and by employing active controls. Finally, inlet operating concerns after mode transition focus on the HSFP. The HSFP concerns include unstarting or an equally undesirable event where the DMSJ flame is extinguished from a large surge in airflow—blowout event. Additionally, a mission critical concern during mode transition is one where the LSFP unstart leads to a HSFP unstart or blowout. This paper focuses on a simulation technique for researching methods to prevent the LSFP from unstarting.

The CCE-LIMX LSFP is designed to integrate with a conventional turbine engine. Experience with supersonic turbine flow-paths (Ref. 7) reveal that flow field disturbances can be grouped into four windows—very high such as above 100 Hz, moderately high (approximately between 20 and 100 Hz), very low (lower than 1 Hz), and low (approximately between 1 and 20 Hz) frequency ranges. The LSFP includes a diffuser volume that will dampen the very high frequency disturbances. This flow-path also includes a passive stability bleed system to reject moderately high frequency disturbances (Ref. 7). Fuel flow control to the turbine will be sufficient for addressing flight condition changes or very low frequency disturbances. Finally, an active control system will be designed to reject the potential low frequency disturbances that are not addressed in the inlet design process.

To complement the CCE-LIMX hardware wind-tunnel tests and to support advanced propulsion system control research, analysis and control computational dynamic models are needed (Ref. 8). To meet these needs, this paper suggests uniting the NASA GRC LArge Perturbation INlet (LAPIN) legacy code (Refs. 9 to 12), designed to simulate the aerodynamics of a supersonic inlet, synchronously with another simulation, designed to run on a modern graphical operating system that can be programmed to simulate controllers (Ref. 13). A process that successfully partners LAPIN and the control simulation on the same graphical operating system will be a state-of-the-art computational control design modeling tool for supersonic inlet systems.

The methodology presented in this paper enables use of the LAPIN code to simulate the CCE-LIMX LSFP, provide periodic process signals to another simulation, and receive periodic signals from a controller process. The objective of this work is to enable this methodology on a modern graphical operating system. The following discussion section provides an overview of the LAPIN legacy code and its enhancements to meet this objective. Next, the approach to synchronously link the LAPIN process with a closed loop controller simulation using the Memory-Mapped File (MMF) structure that links the two processes together is presented. Finally, an example of synchronous simulation results is given.

## Acronyms

| | |
|---|---|
| AIP | Aerodynamic Interface Plane |
| ARMD | Aeronautics Research Mission Directorate |
| CCE | Combined Cycle Engine |
| CCE-LIMX | CCE- Large-scale Inlet for Mode transition eXperiments |
| CFD | Computational Fluid Dynamics |
| DMSJ | Dual-Mode ScramJet |
| FAP | Fundamental Aeronautics Program |
| GRC | Glenn Research Center |
| GUI | Graphical User Interface |
| HSFP | High-Speed Flow-Path |
| LAPIN | LArge Perturbation INlet |
| LSFP | Low-Speed Flow-Path |
| MMF | Memory-Mapped File |
| MS-DOS | Microsoft-based Disk Operating System |
| RAM | Random Access Memory |
| S-fcns | System Functions |
| SWT | Supersonic Wind Tunnel |
| TBCC | Turbine-Based Combined Cycle |

## Discussion

LAPIN is a one-dimension Computational Fluid Dynamics (CFD) legacy code primarily used for simulating the dynamics of supersonic propulsion inlets for control studies. It is a fast and complex stand-alone process that was coded using the FORTRAN 77 programming language to run on character-based operating systems such as the Microsoft Disk Operating System (MS-DOS). Currently, this code is run on modern graphical operating systems within a command shell. The LAPIN simulation is useful for flow field analysis of supersonic mixed-compression inlets susceptible to large flow field perturbations.

The simulation starts after an operator enters the command *aamain* at the Command Shell prompt. Upon process initiation, a text document in FORTRAN 77 name-list format is read. This text document is organized into sections or name-lists such as Inlet Geometry, Bleed and Bypass Descriptions, Control and Boundary Conditions, and Output Settings among others. Within this document, inlet system design parameters, simulation environment, initial conditions and other operation conditions are defined. This input text document can be prepared using a text editor and a copy of this document must be saved in the same directory as the LAPIN program command-line executable—**aamain.exe**. After the LAPIN process reads the input text document and initializes all process variables with default values or with values based on data read from the text input file, it enters the main program loop. The main loop continuously cycles with the simulation time incrementing at the completion of each cycle. Within each main loop cycle, it uses a quasi-one-dimensional inviscid unsteady formulation, with time-dependent equations of motion, that includes engineering models of unstart and restart, bleed, bypass, and geometry effects to periodically calculate flow field analysis. The main loop is where the process variables impact the simulation calculations. Based on information from the input text document, process defining variables such as geometry can change as a function of simulation time. Periodically, as defined in the input text document, the LAPIN process will populate the output text document with simulation calculations. The main loop keeps cycling until a time limit or a cycle count limit is reached and then the process exits the main loop cycle. Upon exiting the main loop cycle, the output text document is completed and closed and then the

LAPIN process will terminate. After the simulation completes, the output text document may be inspected with a text editor or with LAPIN data reduction tools and procedures. This course of action for running the LAPIN code requires the operator to forecast the simulation activity with a well defined text input document and prevents the operator from inspecting simulation progress until the simulation completes.

What is desired is a capability for the LAPIN process to share variable values with other simulations, to read control signals from control processes, and to have its process modified based on the read control signals. Accomplishing these objectives required minor changes to the process flow within the main loop of the LAPIN FORTRAN 77 source code. The changes to synchronously link the LAPIN process with another controlling process enables simulation parameter variables to periodically be modified based on new information from the controlling process. Furthermore, the main loop enhancements enables select LAPIN process variables, which are refreshed within each cycle, to be made available to other simulations.

For this paper, the control simulation selected to work with the LAPIN process was developed using software that provides a Graphical User Interface (GUI) for building models as block diagrams. Specifically, this new simulation was developed using MathWorks Simulink software. Here to forth, simulations developed using Simulink software is referred to as Simulink models or Simulink simulations. This Simulink model will read and process signals from the LAPIN process and provide control signals for the LAPIN process to read. It is desired for the LAPIN process to work synchronously with and be responsive to the Simulink model control signals and the Simulink model to be responsive to the LAPIN process calculations for an inlet simulation. Since both simulations periodically require data from the other, it was desired to have the LAPIN process run on the same operating system as the Simulink process to promote complimentary data exchange.

The following four approaches were considered to enable the LAPIN process to exchange data with the Simulink simulation: One, translate the LAPIN FORTRAN 77 source code to a Simulink block diagram model. Two, translate the LAPIN FORTRAN 77 source code to another language suitable for embedding into the Simulink model. Three, insert the compiled FORTRAN 77 code into the Simulink model. Finally, develop a Memory-Mapped File (MMF) structure as a communication interface to enable synchronous data communication exchanges between the LAPIN process running in a command shell and the simulation running in Simulink. The first three approaches were considered and rejected in favor of the fourth approach that facilitates data exchange between two or more programs running simultaneously (Ref. 13). One drawback to using the selected method is an extra burden applied to the simulations to write and read data to and from the MMFs. However, the MMF method does not involve disk file writes and reads; instead, Random Access Memory (RAM) is used for fast operation. This approach that links the LAPIN process running in the Command Shell with the Simulink simulation using MMF is one that had not been considered before and was deemed to be a feasible challenge. The block diagram in Figure 1 is an overview illustration of this type of arrangement that is termed LAPIN-in-the-Loop. As illustrated in Figure 1, this method enables LAPIN to run in a Command Shell and initialize based on data read from a Text Input File, write simulation results to a Text Output File, use MMF to receive control information from a Simulink simulation Controller block, and send feedback data to a Simulink simulation Aero-Servo Simulation using MMF. Furthermore, this method enables the Simulink simulation to run synchronously with the LAPIN process running in the Command Shell on the same graphical operating system.

The MMF methodology is a means to facilitate periodic communication and data exchange between the LAPIN process and the Simulink model. To use MMF with these two processes, LAPIN code needed to be enhanced with additional FORTRAN coded subroutines, some C++ coding was needed to complement the LAPIN code and the Simulink model, and programming blocks called System Function or S-Functions (S-fcns) were needed to be added to the Simulink simulation. These MMF enabling methodologies are presented next.

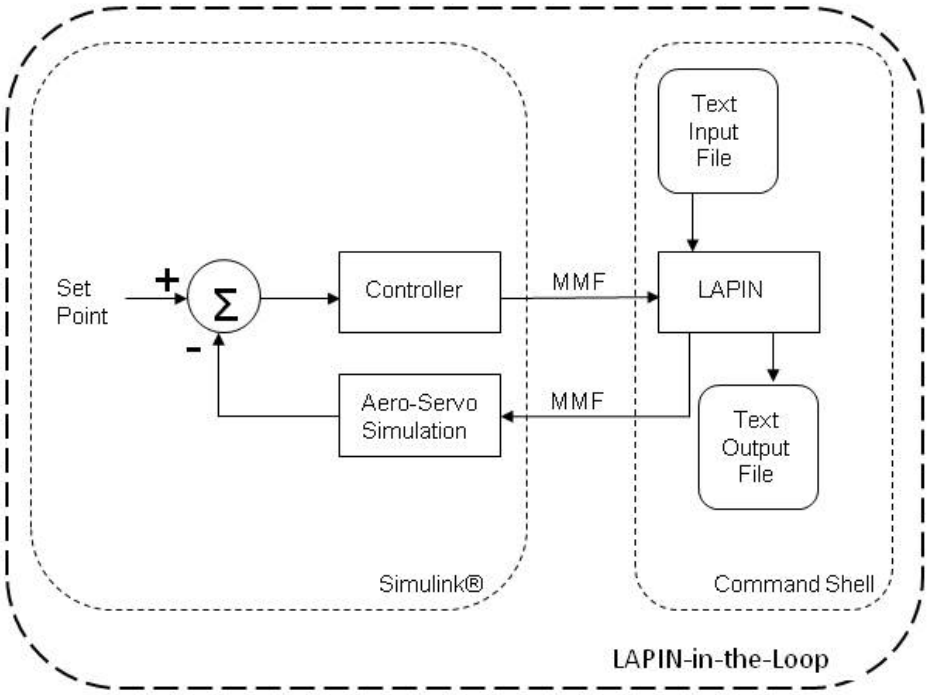


Figure 1.—Overview of the LAPIN-in-the-Loop method, where the LAPIN process running in a Command Shell, is controlled by another process that was designed using Simulink software. The Simulink model receives feedback signals from the LAPIN process and provides control signals for the LAPIN process.

## Memory-Mapped Files

Memory-Mapped files (MMFs) (Refs. 13 and 14) offer a unique memory management feature that allows applications to access files on disk in the same way they access dynamic memory: through pointers (Ref. 15). With this capability all or part of a file can be mapped to a specific range of addresses within the process's address space. Furthermore, a memory-mapped file can simultaneously be mapped by multiple processes. When two processes map the same memory, the data that one process writes to the memory is available to be read by the other process. This provides a mechanism for inter-process communication services while preserving the integrity of private address spaces for each process. For LAPIN-in-the-Loop, two MMFs are established. One is dedicated for the Simulink simulation to write commands for the LAPIN process to read. This MMF is also dedicated to the LAPIN process for reading Simulink commands. Similarly, the second MMF enables the LAPIN process to write data for the Simulink simulation to read. Having two distinct MMFs does not create a problem as long as they are assigned unique names.

The MMF tutorials, examples, and downloadable codes studied that coordinated MMF communication between Simulink simulations and Command Shell processes were suitable for use with C, C++, C#, and Visual Basic languages but not the FORTRAN language. Therefore, C++ coded subroutines were used as a glue between the LAPIN FORTRAN source code and the MMF enabling downloaded C++ code. The C++ subroutines were composed and compiled to create compatible object files for linking with the FORTRAN compiled object files. The Simulink software already includes S-fcn blocks that, when properly coded, enable MMF communication. These MMF communication enabling S-fcn blocks need to be compiled into executable files and their blocks placed in the simulation. The general architecture for LAPIN-in-the-Loop using MMFs is illustrated in Figure 2. The shaded elements with italic labels in Figure 2 represent the additional components beyond the LAPIN legacy code and

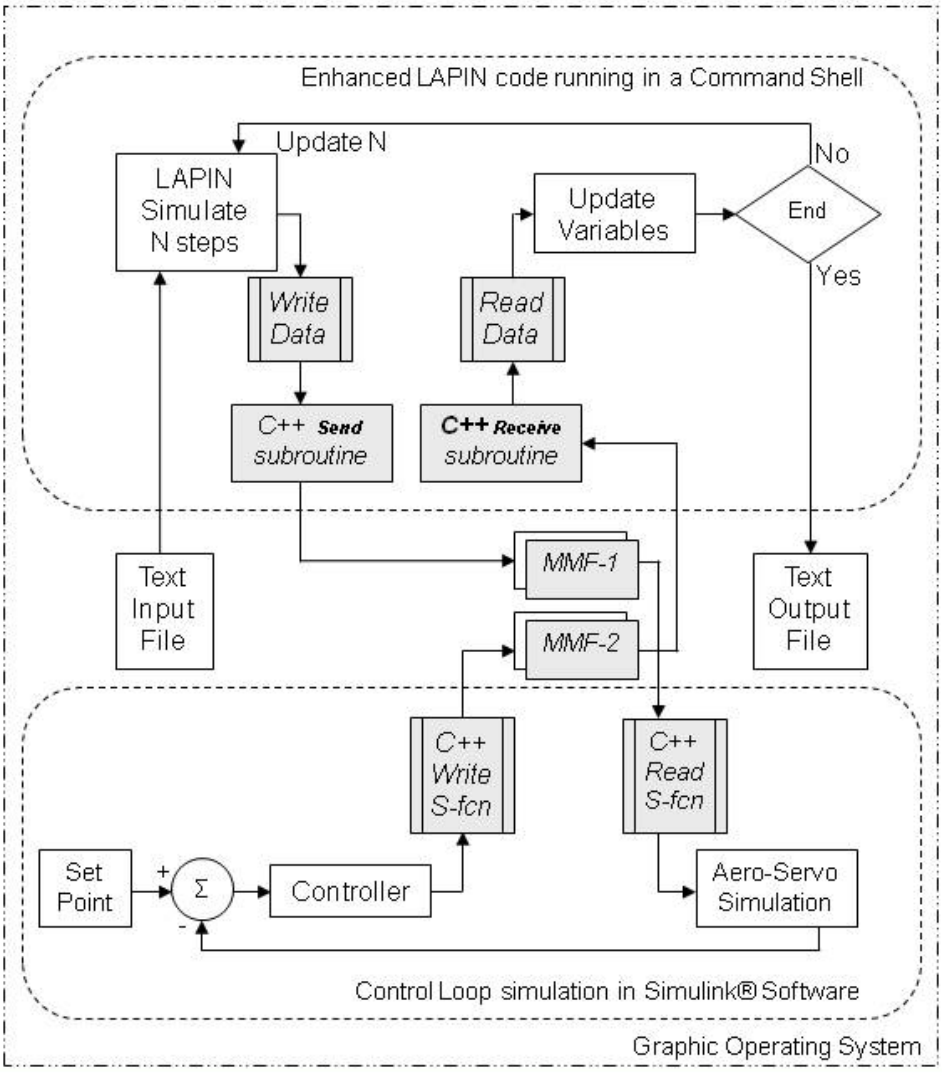


Figure 2.—Detailed diagram of the LAPIN-in-the-Loop method with included elements to promote cross communication using Memory Mapped Files (MMFs). The process running in the command shell requires code developed using C++ to communicate with the MMF files. Likewise, the Simulink S-fcns, coded using the C++ programming language, are necessary for the Simulink software simulations to communicate with the MMF files.

typical Simulink simulations need for LAPIN-in-the-Loop. The top half of the Figure 2 diagram illustrates the LAPIN running in the Command Shell as is the case with the LAPIN legacy code. For the LAPIN legacy code, the arrow from the LAPIN Simulation steps block would bypass the read and write blocks and connect to the Update Variables block. For LAPIN-in-the-Loop, Write Data and Read Data FORTRAN elements are added to the LAPIN legacy main program loop. Also, as illustrated in Figure 2, these added elements use C++ Send and Receive subroutines to glue the FORTRAN elements to the MMF enabling code. The lower half of Figure 2 illustrates a control loop simulation using Simulink software. For a typical control loop Simulink simulation, the arrow from the Controller block would bypass the read and write blocks and directly connect to the Aero-Servo Simulation block. Here, additional C++ Write S-fcn and C++ Read S-fcn blocks are added to the Simulink simulation to enable MMF data exchanges. The LAPIN-in-the-Loop Controller block, running in the Simulink software, includes an element that sends signals to MMF-2 via the C++ Write S-fcns. The Aero-Servo Simulation block, also running in the Simulink software, uses signals read from MMF-1 via the C++ Read S-fcn. The

LAPIN process writes data for the Simulink Aero-Servo Simulation to MMF-1 via the C++ Send Subroutine and reads control signals from MMF-2 via the C++ Receive Subroutine. The MMF files, as well as the text input and output files, are external to the Command Shell and Simulink software. The LAPIN-in-the-Loop product, as illustrated in Figure 2, has both the LAPIN process and the Simulink simulation running on the same graphical operating system.

Having established two MMF files and the processes to populate and read these files, data flow scheduling and handshaking protocols were developed to insure quality data exchanges—reads from the MMF occur only once and after the MMF has been populated with new data. Furthermore, writes to the MMF occur only after the previously written data has been read. Finally, the processes were synchronized by employing the operating system sleep command to allow sharing processor time—when one process is waiting for the other, it is put into a sleep state to allow the other process to have full use of the processor. The appendix contains the details on how this process operates and was coded. Since it is desired to have all simulations keep in time step and finish at the same moment, time step counts between data exchanges were made to be individually settable. Properly setting the LAPIN process time step size is crucial for a successful and timely simulation. The Simulink model has a settable length of run time and a time step size that best suits its operation. Since the time step size for the LAPIN process is typically required to be much smaller than the Simulink time step size, operating the Simulink simulation at the smaller step size will unnecessarily slowdown the simulation. Since data exchanges are not required at every time step for both simulations, the step period for the Simulink system can be set to a whole multiple of the LAPIN process step period. These approaches prevent one simulation from racing ahead of the other and results in both processes advancing towards simulation completion synchronously.

## Simulation Results

The Simulink model illustrated in Figure 3 is a closed loop control system block diagram that was designed to demonstrate LAPIN-in-the-Loop, using the MMF method presented in this paper, maintaining synchronous operation between the Simulink simulation and the LAPIN process while facilitating two way communications between them. The Figure 3 illustration can be correlated to the Figure 2 control loop simulation in Simulink software by considering the sfun_send block to be the Figure 2 C++ Write S-fcn blocks and the sfun_rec block to be the Figure 2 C++ Read S-fcn blocks. LAPIN is a supersonic inlet simulation compiled using the legacy FORTRAN 77 code with the enhancements discussed in this paper. The Simulink simulation was also designed to read and display the normal shock position calculations based on data from the LAPIN process and apply signals to the MMF that will affect the LAPIN process. The Simulink control signals are expected to impact the LAPIN process such that the feedback signals from LAPIN will indicate a change in the calculated normal shock location. The Simulink process uses S-Functions to support linking to the MMFs. Simulink S-Function blocks are required to have at least one input port; however, the S-Function block used to read from the MMF does not have use for an input port. Therefore, a constant value block is connected to the input port of the sfun_rec S-Function block. The sfun_rec S-Function block will read the MMF and apply normal shock position x-coordinate values on an output port that is connected to a scope and to an output port that is connected to a block that records normal shock position data in the **xshk.mat** file in MATLAB MAT-File format. The scope can illustrate the x-coordinate values with respect to time while the simulation is running. The data from **xshk.mat** can be used for post processing tasks. The other S-Function illustrated in Figure 3, sfun_send block, applies values to the MMF for the LAPIN process to read. For this example, the following four signal values are applied after each iterative step of the Simulink simulation: A constant throat height of 0.87 inches, a constant Cowl angle of 0.1°, a constant normalized bleed plenum exit area of 0.0005, and a sinusoidal varying Exit Mach number. The Perturb Exit Mach signals, representing an inlet perturbation, are sent to the MMF within the sfun_send S-Function. These signal values are also recorded in the **ExitMach.mat** file in MATLAB MAT-File format for later report generation. Fluid dynamic analysis of the supersonic inlet considered in this

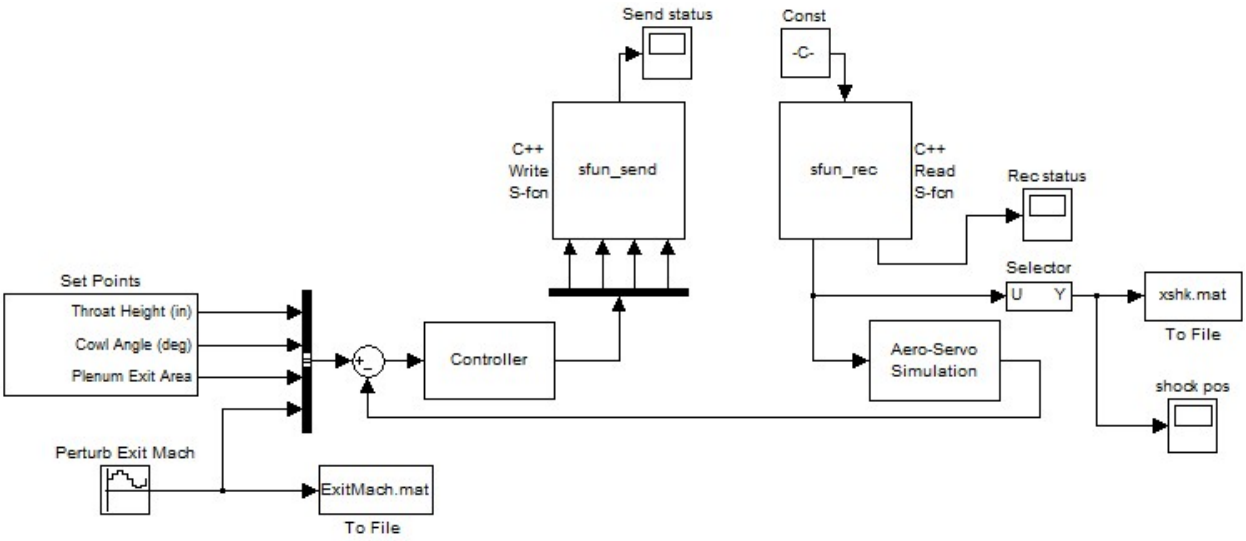


Figure 3.—Simulink model used to demonstrate cross communication with LAPIN using the LAPIN-in-the-Loop MMF method.

example reveals that when the throat height, cowl position, and bleed exit area are held constant, the normal shock position calculation is dependent on the aft Mach number value. Therefore, the sinusoidal Perturb Exit Mach signal will drive the LAPIN process by effectively changing the aft Mach number boundary condition in the inlet simulation. Furthermore, reading changes in calculated normal shock position because of the applied aft Mach number perturbations is an indication of effective cross communication and synchronization between the processes.

The LAPIN process and the Simulink simulation were initiated to run simultaneously. Verification that the LAPIN process Exit Mach number value changed was captured in the calculated shock position as illustrated in Figure 4. The traces in Figure 4 illustrate the normal shock position calculations, described as a distance (inches) from the inlet aerodynamic throat, and the sinusoidal changing Exit Mach number perturbation values generated by the Simulink simulation. The Exit Mach number perturbation control values are plotted against time in the upper plot of Figure 4. The LAPIN process periodically reads these commands from the MMF and incorporates the new Exit Mach number values into its process parameters. The new Exit Mach numbers affect the calculated location of the normal shock position. The LAPIN process periodically writes its calculated normal shock positions to the MMF. In turn, the Simulink simulation timely reads Normal Shock Position data from the MMF using the sfun_rec block. The lower plot in Figure 4 shows the Normal Shock Position values, with respect to simulation time, obtained from the LAPIN process. Inspection of the plots in Figure 4 indicates that the Simulink simulation influenced the LAPIN process calculations for shock position.

This demonstration also revealed an additional benefit of LAPIN-in-the-Loop. Simulink software can be used to iteratively illustrate LAPIN process parameter calculations as the simulation proceeds–legacy LAPIN required waiting until the simulation terminated before the output file can be examined. Furthermore, the Simulink software can be written to timely act on the information read from the MMF for determining control signals for the other MMF.

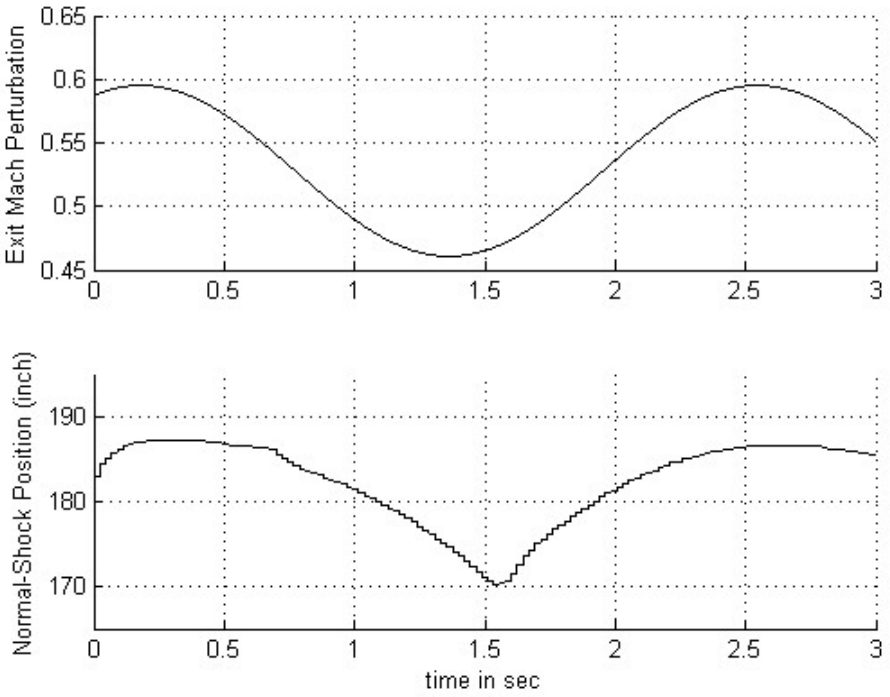


Figure 4.—Upper plot illustrates a control signal delivered from the Simulink simulation to the LAPIN process that describes a changing Exit Mach number boundary condition. The lower plot illustrates a changing normal shock axial position calculation from the LAPIN process due to the changing exit boundary condition.

## Summary

This report focused on integrating the NASA legacy LArge Perturbation INlet (LAPIN) model with a MathWorks Simulink simulation. A brief overview of the legacy LAPIN process was given along with a description of the project objectives to enable the LAPIN process to be controlled by a Simulink controller simulation. Furthermore, both the LAPIN process and the Simulink simulation run on the same modern graphical operating system. To meet these objectives, the Memory Mapped File (MMF) data sharing technique was employed. The main loop of the LAPIN simulation, written and compiled with FORTRAN 77 software, was enhanced with custom C++ compiled subroutine code to facilitate communication with an MMF. The Simulink simulation employed S-functions to support its communication with the MMFs. The product of this work makes available the LAPIN simulation to timely and synchronously interact with Simulink simulations. The capability of LAPIN to simulate inlets was preserved. However, Simulink simulations can be developed to enhance the operation of LAPIN with additional blocks to illustrate and analyze LAPIN process data prior to simulation completion and generate control signals to be applied to the LAPIN process. Simulation results from a LAPIN-in-the-Loop demonstration was given. This approach is a novel technique for running legacy FORTRAN 77 software in a command shell synchronously with a model designed in Simulink and with both running on the same modern graphical operating system. The product of this work is useful to the Fundamental Aeronautics Program Hypersonic Project by making available legacy dynamic inlet analysis software for control research pertaining to the Combined Cycle Engine Large-scale Inlet for Mode transition eXperiments (CCE-LIMX).

# Appendix—Details of LAPIN Source Code Revisions and Simulink Functions

## Introduction

This appendix describes the steps taken to modify the NASA GRC LArge Perturbation INlet (LAPIN) legacy simulation, a character mode executable that was written using FORTRAN 77 language, to run independently, synchronously, and cooperatively with a MATLAB Simulink simulation using Memory Mapped Files (MMF). The compiled FORTRAN code can be run on a modern operating system within a command shell. The Simulink simulation is run on the same operating system as the command shell. To make way for cross communication between the two processes, two MMFs were needed. One MMF is written to by the LAPIN process and read from by the Simulink simulation. The other MMF is written to by the Simulink simulation and read from by the LAPIN process. The simulation process defined by the LAPIN FORTRAN code shall here forth be referred to as LAPIN and the code that defines the LAPIN process shall here forth be referred to as the LAPIN code. The objectives are for LAPIN to simultaneously run with a Simulink simulation, both simulations to be time synchronized, have the capability to periodically exchange data, and have the capability to be responsive to periodically received data.

This appendix is a complement to the legacy LAPIN documentation, **LAPIO.PDF** (Ref. A1). Although some explanation of the LAPIN code and its operation is given, this appendix focuses on the modifications that were necessary to promote the LAPIN-in-the-Loop process. A reader interested in running the legacy LAPIN or LAPIN-in-the-Loop is encouraged to read the LAPIN documentation to understand how to populate the LAPIN input text document, **lapin.dat**, and how to read the LAPIN output text document, **0lapin.out**. The approach identified in this document can also be used with other FORTRAN coded simulations that are run within the command shell of a graphical operating system and for applications with available source code. Furthermore, the second simulation is not limited to being a Simulink simulation.

This document describes the legacy LAPIN code modifications to enable communication with the MMFs, required S-Function blocks for the Simulink simulations to communicate with the MMFs, methodology employed to share data between the two simulations, and how time synchronization is maintained. This appendix will first introduce the legacy LAPIN code with a brief overview and process description. Next, the enhancements made to the legacy LAPIN code to enable data exchanges with a Simulink process using MMFs are presented. The following section will present how to interface the MMFs with Simulink code. Next, process synchronization is addressed followed by descriptions for controlling the model center-body, cowl geometry, bleeds, and downstream boundary conditions. Finally, an example running the LAPIN-in-the-Loop simulation is given.

## Legacy Lapin

LAPIN is structured to repeatedly and iteratively solve the flow path variables of a supersonic inlet simulation. A typical supersonic inlet that is modeled using LAPIN is illustrated in Figure A1. The major features of the inlet are a variable throat height by way of an adjustable center-body, a variable bleed plenum exit area, variable upstream free-stream air flow conditions, and variable downstream boundary conditions. The cowl is a major component of the inlet that is not adjustable with the legacy LAPIN simulation.

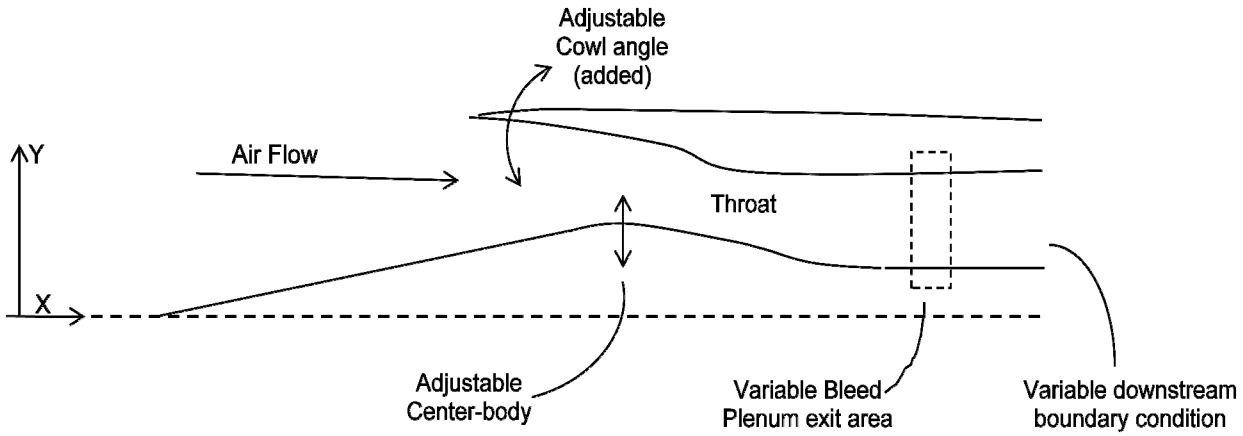


Figure A1.—A typical supersonic inlet model for LAPIN simulation study.

The inlet flow path is segmented into a grid with 379 locations and flow path variables are calculated at each grid point. The 11 primary flow path variables calculated are density, velocity, static pressure, static temperature, Mach number, the Courant, Freidricks, Lewy (CFL) parameter, mass flow rate, cross-sectional area, total pressure, total temperature, and corrected mass flow. Simulated mechanical changes in the flow path geometry will cause time based changes in the calculated flow path variables.
To aid explanation of the LAPIN code modifications, a simplified flow diagram is shown in Figure A2 that focuses only on the main elements pertaining to this work. As illustrated in Figure A2, the LAPIN process starts by reading the Process Input File. LAPIN, when initiated, will read the **lapin.dat** input text document. This input file includes information that fully describes the inlet geometry, the variability of the inlet geometry, simulation initial conditions, time step control, and other process regulations. After reading the input file, the process will initialize the process variables, establish a common data area, and load some process variables into the common data area. The process variables that are loaded in the common data area are accessible to all LAPIN code subroutines. After initialization, LAPIN enters the main program loop. The main loop is revealed in a flow-chart diagram that illustrates the top level operation of the LAPIN code. This information is documented in the Portable Document Format (PDF) file, **MAIN-FLO.PDF** (Ref. A1) and it can be viewed using Adobe (Adobe Systems, Inc., San Jose, CA) Acrobat Reader software. The LAPIN main-loop is represented in Figure A2 by the "Update Time", "LAPIN Calcs", "Output Step?", "OUTPUT (to file)", and "Max step or time?" blocks. The LAPIN Calcs block represents various subroutines that are called within the main program loop. The main loop continuously cycles with the simulation time incrementing at the completion of each cycle. Within the main loop, simulated movement of or changes to major components can only occur on a time schedule. The time schedule of events is defined in the **lapin.dat** input file. Also within the main loop, the output subroutine is periodically called to populate a text document named **0Lapin.out** with main duct flow field simulation data. The main loop keeps cycling until a time limit or a cycle count limit is reached and then the process exits the main loop cycle. Upon exiting the main loop cycle, **0Lapin.out** will be completed and then LAPIN will terminate.

The LAPIN input text document is segmented into multiple sections that are identified with specific labels. Refer to the LAPIN documentation file **LAPIO.PDF** for more information on how to populate the **lapin.dat** input document and read the **0Lapin.out** output text document. The following is a listing of the LAPIN input text document headings that are addressed in this appendix:

- BC defines the boundary condition to be perturbed and when it will be perturbed.
- BYBL defines the inlet bleed and bypass characteristics and schedules.
- CONTRL defines simulation time step increment and downstream perturbations.
- GEOM defines the inlet geometric characteristics and schedules.

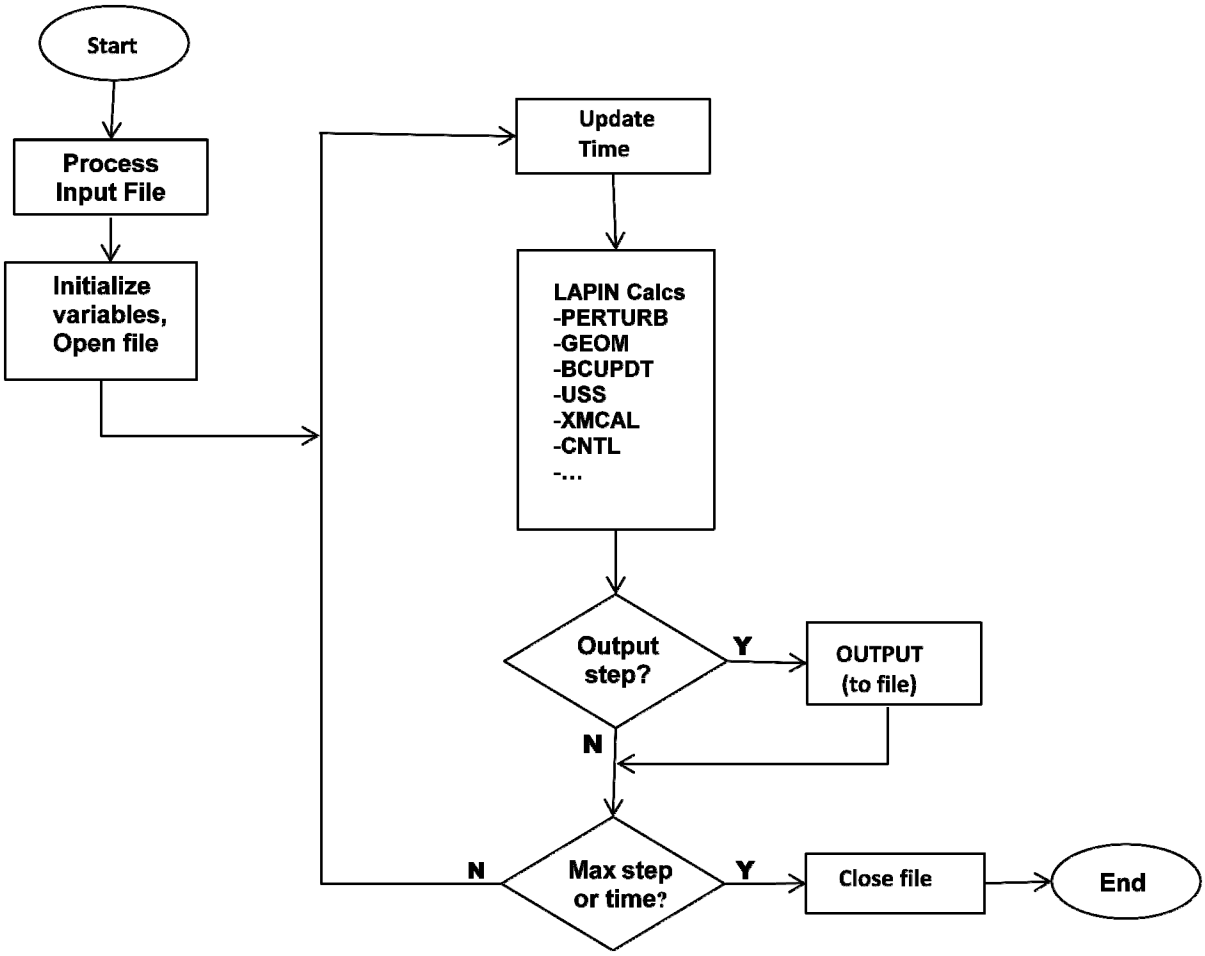


Figure A2.—This simplified flow diagram of the LAPIN process reveals the following three steps: Step one to read process input files and initialize all process variables. Step two is the main loop that includes the Update Time, LAPIN Calcs, Output step, Output (to file), and Max step or time blocks. Finally, step three is the programmed termination process.

The perturbations to the boundary conditions are defined under the CONTRL and BC sections. In these sections the process variables IPERTBC and IDSBCTYPE and the process arrays TIMDS and DSBCALS are defined. The IPERTBC variable is defined in the CONTRL section to identify the disturbance source—free-stream, exit boundary condition, center-body translation, or bleed exit area. The exit boundary condition perturbations are further defined under the BC label in the LAPIN input text document. In this section, the IDSBCTYPE variable identifies the downstream (exit) parameter that varies with respect to time. The downstream conditions are specified as functions of time in the TIMDS array and their characteristics are defined in the DSBCALS array. The process defined in the perturb subroutine uses IDSBCTYPE and TIMDS values to apply downstream boundary condition perturbations to the simulation.

LAPIN includes the capability to allow mass extraction from the core flow through scheduled variations of bleed and bypass. This schedule is defined with a table under the BYBL section of the **lapin.dat** file (PEARBL). For example, an inlet model may have 13 bleeds (NBL = 13) and no bypasses (NBY = 0) with twelve of the bleeds located in the throat area and the 13th bleed located well aft of the throat. This 13th bleed can be modeled as an aft bypass with a plenum. While in the main loop, the LAPIN code found in the uss subroutine uses a linear interpolation subroutine on the PEARBL table to set the simulated plenum exit area variable (AREX) with respect to time.

LAPIN has another means to control simulated bleed air flow. This other approach uses a bleed mass extraction table (MASSBL) that is defined with respect to simulation time in the **lapin.dat** input data file. A linear interpolation subroutine is used to select values from the MASSBL table to determine a bleed mass extraction value for the process (BLMAST) at each time step.

Under the GEOM label in the LAPIN input text document is a flag to select one of four inlet types. For this work, only inlet type 3, that makes available a variable center-body contour, is of interest. For inlet type 3, the input file includes up to ten center-body coordinate tables (XCBTAB and YCBTAB). The input file also requires a table of throat height values (THHT) that correspond with the center-body coordinate tables. This section includes a table that defines process arrays TRTIME and THTRAN for scheduling when the throat height will change and its next simulated throat height value (TH). The LAPIN code that defines the process for changing the center-body coordinates is located in the geom subroutine. As illustrated in Figure A2, the geom subroutine within the LAPIN Calcs block is repeatedly called within the LAPIN process main loop. When LAPIN time step meets a TRTIME point, linear interpolation about the THHT table with the corresponding value of THTRAN is used to determine, also through linear interpolation, new center-body coordinates.

The LAPIN output subroutine writes flow field data to the **0Lapin.out** text file as the simulation runs. The data in the output text document includes a time-step number, the simulation time point, normal shock location, and a listing of the primary variables at each grid point. Except for Mach and CFL, the values are normalized by either initial free-stream conditions or sonic conditions—the choice is defined with an input text document variable (NOPTPRT). The frequency for invoking this routine is set with another input text document variable (NC). The value of NC is the number of time steps between calls to print a solution to the output file.

## Lapin Enhancements

### Overview

To enable a LAPIN-in-the-Loop configuration with a MATLAB Simulink simulation, a few modifications to the legacy LAPIN code were needed to make LAPIN responsive to non-periodic external control signals and to make LAPIN data available to other processes. Specifically, the LAPIN process needed to be enhanced to enable cross-communication using MMFs with a Simulink simulation. To achieve cross-communication capability, additional FORTRAN subroutines were added to the LAPIN code and some C++ functions were necessary to complement the LAPIN code. Also, some modifications to the LAPIN code was required to use the information received through the MMF data exchange process to appropriately modify the LAPIN simulation. Table A1 is a listing of subroutines that were added or modified, as indicated in column 4 with an A or an M. The second column in Table A1 identifies the type of file as being either a C++ code, FORTRAN (F) code, or a character based file (txt).

TABLE A1.—LIST OF FILES MODIFIED OR ADDED TO LAPIN

| Element name | File Type C++, FORTRAN (F), or Text (txt) | Description | Modified (M) or Added (A) |
|---|---|---|---|
| aamain.f | F | Top level code for LAPIN | M |
| geom.f | F | Geometry change process | M |
| uss.f | F | Bleed change process | M |
| perturb.f | F | Downstream perturbation process | M |
| input.f | F | Reads the LAPIN input text file | M |
| lapin.dat | txt | LAPIN input text document. | M |
| movecowl.f | F | Simulated cowl movement process | A |
| mmfout.f | F | Enables LAPIN writing to MMF | A |
| mmfdata.cmn | F | Common block file that holds data for MMF operation | A |
| MMFile.h | C++ | Header file holding MMF class definition | A |
| cpfunc.cpp | C++ | MMF interaction functions used by LAPIN that includes sendcsub and recvcsub subroutines. | A |

Since the FORTRAN language does not have the capability to define Memory Mapped Files (MMF), a language compatible with LAPIN FORTRAN source code and MMF techniques needed to be used. The approach requires the object files generated by both compilers to be compatible so they could be linked together into one executable. For this development The Portland Group (Lake Oswego, OR) Visual FORTRAN compiler was used with the Microsoft Visual C compiler. Object files generated by both compilers are compatible so they can be linked together such that subroutines written in Visual C++ can be called by LAPIN code. Also, Visual C++ is a recommended compiler for Simulink compatibility.

The C++ class documentation found in a MathWorks MATLAB Digest newsletter article (Ref. A2) presented functions and data structures needed for MMF interfacing. The following two changes were made to the C++ class described in the newsletter: First the class constructor was changed, requiring the MMF name to be assigned at object creation. Since two different MMFs were needed, this change was necessary so that the same class definition can be used for creating multiple, independent MMFs. Second, the class was expanded with one new function. The new function was added to facilitate data synchronization between two processes—one functioning as a data producer (writer) and the other functioning as a data consumer (reader).

The file named **cpfunc.cpp** coded in C++ contains the functions needed for MMF interaction for both the write-to (sendcsub) and read-from (recvcsub) operations. For either operation, the functions may perform any of the following three procedures based on the value of the passed argument N: MMF initialization for N = 0, MMF termination for N = –1, and MMF read and write for N > 0. The initialization function will create MMFs by calling the MMF class constructor and passing unique names to use. For both operations, a pointer is returned for invoking functions in the class object. The termination function invokes the class destructor which unmaps and closes the MMF.

Table A2 is a nomenclature listing of LAPIN code variables that are addressed in this appendix. Some of these variables have not changed from their original use in the legacy LAPIN code. These variables are included to help define other variables that are either new or have expanded dimensions. All variable names are in CAPS and the only variable added to the code is the cowl angle variable (CW). Five of the variables in Table A1 are assigned values received from Simulink and are highlighted with a boldface font.

The variables AREX, BLMAST, TH, CW, and EXITBC are populated with values based on Simulink process calculations. The others are initialized with values from the LAPIN input text document. To acquire information for setting AREX, BLMAST, TH, CW, and EXITBC values in the LAPIN process, the recvcsub subroutine call was added to the main loop for reading the MMF just before the call to the subroutine geom. The recvcsub subroutine enables the process to read the MMF and determine values for the throat height, cowl angle, bleed or bypass plenum exit area, bleed or bypass mass extraction, and the core flow downstream boundary condition and save them in the common data area that was defined in **mmfdata.cmn**. Similarly, another new subroutine, called mmfout, was created to write LAPIN main duct flow field data to the MMF for use by other processes. The mmfout call is also in the main loop just before the output subroutine call and it uses the sendcsub subroutine to communicate with the MMF. Figure A3 shows a simplified flow diagram for LAPIN operation after the MMF enhancements were added. The elements with shaded background and dash outline indicate new code added to the LAPIN process. The elements with shaded background and solid outline indicate subroutines that already existed in legacy LAPIN that were modified to support LAPIN-in-the-Loop.

TABLE A2.—LIST OF VARIABLES ADDRESSED IN THIS APPENDIX
[All variable names are identified with caps. The variables with boldface font are determined based on simulink simulation calculations.]

| |
|---|
| • Bleed and Bypass Variables |
| ○ **AREX**, plenum exit area |
| ○ BLL, bleed section lengths, |
| ○ **BLMAST**, bleed mass extraction |
| ○ BLPERP, fraction of inlet perimeter occupied by each bleed section, |
| ○ BLPOR, fraction of bleed section surface that is open, |
| ○ BLPVOL, bleed plenum volumes, |
| ○ BLX, bleed section axial locations |
| ○ IBLLOC, inlet surface with bleed pattern |
| ○ MASSBL, table of mass flow rate values through bleeds. |
| ○ NBL, number of bleeds, |
| ○ NBY, number of bypasses, |
| ○ PEARBL, a two-dimensional table to correlate bleed extraction mass with simulation time. |
| • Center-body Variables |
| ○ **TH** is the current throat height being used by the simulation. |
| ○ THHT is a table of up to 10 predefined throat height calculations that correspond to the center-body positions defined by XCBTAB and YCBTAB. |
| ○ THTRAN is a table of center-body position parameters, desired throat height values, at each TRTIME point that will be used to define the next center-body contour. |
| ○ TRTIME is a table of time points when the center-body is scheduled to transition from one THTRAN position to another. |
| ○ XCBTAB is a table of X coordinates for each of 10 center-body contours |
| ○ YCBTAB is a table of Y coordinates for each of 10 center-body contours |
| • Cowl Variables |
| ○ **CW** is the desired cowl angle |
| ○ XCOWLTAB is a table of X coordinates for one cowl contour |
| ○ YCOWLTAB |
| ▪ was a table of Y coordinates for one cowl contour |
| ▪ is a multiple column table of Y coordinates for multiple cowl contours. |
| • General Variables |
| ○ NC is the number of time steps between calls to print a solution to the 0LAPIN.out output text file |
| ○ NOPTPRT defines if flow variables are normalized by initial free-stream conditions or by sonic conditions. |
| ○ DTIM is the time step increment |
| • Perturbations |
| ○ **EXITBC** is the value applied to the downstream boundary condition |
| ○ IDSBCTYPE indicates type of varying downstream boundary condition |
| ○ IPERTBC indicates which boundary to perturb |

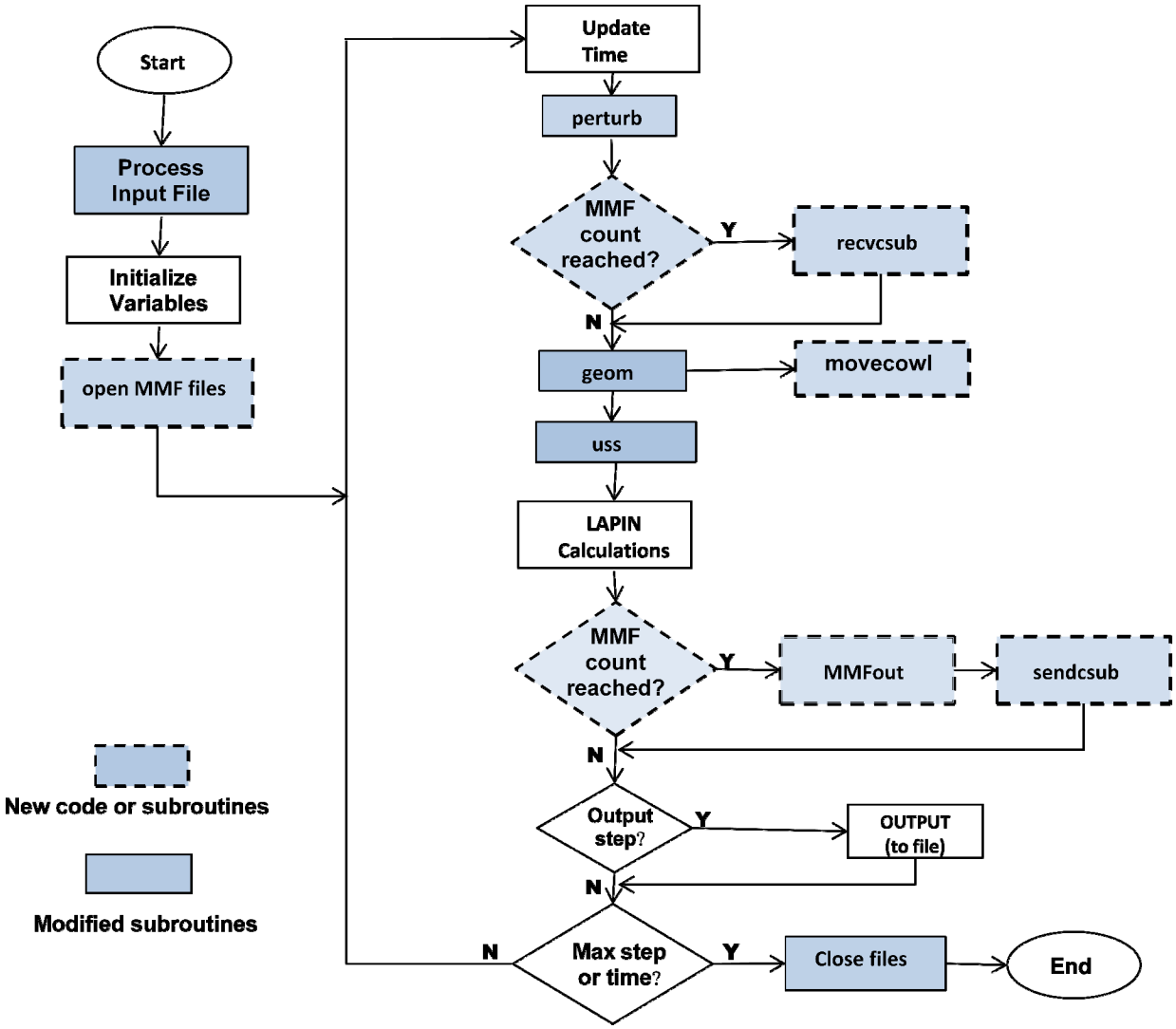


Figure A3.—Simplified LAPIN flow diagram showing enhancements for MMF communications.

The following two subsections present modifications to the LAPIN code to enable writing to and reading from the MMF.

### LAPIN Writing Data to the MMF

LAPIN simulation information is saved to the MMF in packages referred to as data sets. A data set assembled for sharing with the Simulink model consists of two data arrays, a process time stamp, a calculated normal shock position, and fixed separator values between each of these elements. The first data array is organized into records consisting of 11 parameters. Each record includes the grid point location or X value, followed by 10 LAPIN calculated parameters. Table A3 is an example record that shows the content of the first data array.

TABLE A3.—AN EXAMPLE RECORD OF CALCULATED DATA THAT WILL BE WRITTEN TO THE MMF AS PART OF THE LAPIN PROCESS AND USED BY THE SIMULINK SIMULATION

| Variable | Definition |
|---|---|
| X | Axial grid point location |
| RHO | Density |
| U | Velocity |
| T | Static temperature |
| M | Mach number |
| CFL | Courant, Freidricks, Lewy parameter |
| MASS | Mass flow |
| AREA | Area |
| PT | Total pressure |
| TT | Total temperature |
| CMASS | Corrected mass flow |

The X locations begin at the cowl lip and monotonically increase downstream for 225 grid points. To save memory and process time, instead of including data with respect to all 225 LAPIN grid points in the first data array, only data from every 40th grid point is included. The second data array captures static pressure data over the same 225 grid points. The static pressure values are important for control purposes; therefore, those values are gathered at every grid point. The X-location and static pressure data pairs, at every grid point starting at the cowl lip, are gathered into an array that gets appended to the first data array. These two data arrays are separated by a fixed value marker. The second array is followed by another fixed value marker to separate it from a LAPIN process clock time stamp and the calculated grid location of the normal shock.

This data set is written to the MMF and made available to the Simulink Model with a call to the newly added C++ subroutine, sendcsub. The process defined by the sendcsub subroutine populates a segment of memory defined in the common block that is also dedicated to the MMF. The sendcsub uses the third MMF interaction operation that the C++ function was designed to perform—writing and reading MMF data. Therefore, with a value for the variable N passed to the sendcsub being greater than zero, the LAPIN sendcsub subroutine will write N number of double precision real values to the MMF. This data will become available for use by the Simulink model. However, before writing new data to the MMF, this subroutine will first read the MMF process flag that indicates if the previous data was read by the Simulink process. If all previous data was not read yet, the LAPIN process will enter a time-limited Sleep function—Sleep function is an operating system function. This approach briefly releases the processor from the LAPIN process so that the Simulink model will get needed processor time to complete reading the last data put into the MMF. After all data has been read by the Simulink process, the sendcsub subroutine passes the data-set array pointer to the MMF class for writing to the MMF. The MMF class handles the rest which includes updating the MMF process flag from 0 to N. For diagnostic purposes, the subroutine also displays the LAPIN waiting time for the Simulink model to catch up. Finally, if after an extended period the Simulink simulation still has not read the data, the sendcsub subroutine process will display, in the command shell window, a brief message indicating the function timed out without sending the current data to the MMF. In this case the previous data will be lost as the new array data over-writes the previous data. The sendcsub subroutine then exits and LAPIN resumes simulation.

### LAPIN Reading Data From the MMF

The recvcsub subroutine is designed to read data for the LAPIN process from the MMF. When the LAPIN process enters this subroutine, it will not proceed to read from the MMF until the Simulink simulation has completed its process writing data to the MMF. First, the recvcsub process will read the MMF process flag to determine if data is available for reading. If this value is zero, then the data is not ready to be read. Each time the check for MMF data returns a zero, a Sleep function is called such that the Simulink process gets needed processor time to complete its task, which includes writing data to the MMF. When data is ready to be read, the MMF class process will read data from the MMF and populate a LAPIN held memory array. After reading all data, the recvcsub subroutine will reset the MMF process flag to zero; thus, allowing the Simulink model to write more data. For diagnostic purposes, this subroutine will also display in the command shell window the length of time the LAPIN process waited for data. This is useful for determining how long LAPIN waits for the Simulink process to catch up with LAPIN. Finally, if no data is available after an extended period, the recvcsub process will display in the command shell a brief message indicating the function timed out without reading new data from the MMF. Then LAPIN will continue using the data read from the last successful read.

## MMF Interface on the Simulink Side

### Overview

The following two custom coded Simulink System Function (S-Function) blocks were added to the Simulink simulation to enable reading data from and writing data for the LAPIN simulation: sfun_send writes data to the MMF and sfun_rec reads data from the MMF. These blocks can be coded in C, C++, Ada or M code—C++ was chosen for these S-Functions. To facilitate MMF communication, these S-Functions use persistent memory objects, known as work vectors, that are defined in the Simulink model. These work vectors hold information that streamlines communication with the MMFs. An example of one such work vector is a pointer work vector that is used to hold the pointer to the MMF object. To coordinate data exchange, hand-shaking protocols, an S-Function internal counter variable is declared and initialized to zero in the heading area of the S-Function. While the Simulink process is running, this internal S-Function variable will retain its value.

The following two subsections present some detail regarding the use of S-Functions to write to and read from the MMF.

### SIMULINK Writing Data to the MMF

The S-Function block, named sfun_send, has four input ports, one for each of the variables it will write to the MMF. The block also has one output port, which may be connected to a Simulink scope block, that is used only for diagnostic purposes. The block is also coded to have one settable S-Function block dialog parameter. This S-Function block parameter is the synchronization parameter and it defines the number of Simulink time steps to occur between interactions with LAPIN by means of the MMF.

MATLAB Simulink provides a template to aid incorporating S-Functions coded in C or C++. There are numerous callback routines in the template, many of which are optional depending on what the S-Function needs to accomplish. Table A4 identifies the following four S-Function callback routines employed by the sfun_send program: mdlInitializeSizes, mdlStart, mdlOutputs, and mdlTerminate.

TABLE A4.—S-FUNCTION CALLBACK ROUTINES DEFINED IN THE SFUN_SEND S-FUNCTION SIMULINK BLOCK

| sfun_send S-Function callback routines | Callback description |
|---|---|
| mdlInitializeSizes | • Defines four input and one output ports for the S-Function block. The four input ports receive signals that represent Simulink model commands for the LAPIN simulation. The one output is used only for diagnostic information.<br>• Defines the number and types of work vectors to be used by the Simulink simulation. |
| mdlStart | • Called once when the Simulink model is started to create the MMF object and saves the MMF pointer into a work vector.<br>• The MMF name passed in this function call is the same name used by the LAPIN C++ subroutine that receives data. This practice assures that both use the same MMF. |
| mdlOutputs | • Called after every time step. This code defines the MMF interaction process for sending Simulink command signals to the LAPIN process. |
| mdlTerminate | • Prior to simulation termination, the process defined by this callback will cleanup and close the MMF by calling the destructor method in the MMF class. |

When the Simulink simulation runs the process defined in the sfun_send S-Function, the process defined by the mdlOutputs code will compare the S-Function block synchronization parameter with an internal Simulink counter variable. If the synchronization parameter is greater than the internal counter, the internal counter will be incremented and the function returns with no MMF interaction. When the value of the synchronization parameter is equal to the internal counter value, MMF interaction begins. Similar to the LAPIN MMF communication scheme, this subroutine will first read the MMF process flag that indicates whether the previous data was read by the LAPIN process. A process flag greater than zero indicates the LAPIN process has not read the previous data written to the MMF by the Simulink simulation. If all previous data was not read yet, the Simulink process runs a Sleep function for 10 milliseconds. The Sleep function will allow LAPIN to get processor time to complete tasks that includes reading previous data written into the MMF by the Simulink process. After 10 milliseconds, the MMF process flag is checked again. When the process flag returns a zero, the sfun_send S-Function will populate the MMF with values suitable for determining the LAPIN process variables highlighted in boldface in Table A2. The values the Simulink simulation writes to the MMF are from data read on the S-Function input ports. If the MMF process flag repeatedly returns a value greater than zero for 150 consecutive repetitions (1.5 seconds), the S-Function output port value will be set to –2.5 indicating the function timed out without sending the current data to the MMF. Otherwise the output value is set to the number of repetitions necessary before sending data to the MMF. A Simulink Scope block can be connected to this output port for diagnostic purposes. In either case, the internal counter is reset to one and the Simulink model resumes its simulation. At the end of the simulation, the mdlTerminate callback is invoked to unmap and close the MMF.

### SIMULINK Reading Data From the MMF

The S-Function block, named sfun_rec, is used to enable the Simulink simulation to receive data from LAPIN by way of the MMF. This S-Function has one input port and two output ports. The input port is used to set the data sample time. One output port is used as part of the data exchange hand-shaking protocol for reading LAPIN data by means of the MMF. The second output port is connected to a scope for display of diagnostic information. This block also has a settable S-Function block dialog variable that is used as a synchronization parameter to compare against a counter for the number of Simulink time steps between interactions with LAPIN. Usually, this block parameter value is set to the same value as the synchronization parameter in the sfun_send block.

The receive S-Function, sfun_rec, is similar in many respects to the Send S-Function. It is based on the Simulink supplied template for C++ coded S-Functions. This S-Function also uses the same callback routines as the sfun_send S-Function listed in Table A4. A slight difference is that its mdlInitializeSizes callback only defines one input and two output ports. The input port is not used and is connected to a constant value block in the Simulink simulation. One of the output ports transmits data read from the LAPIN populated MMF to the Simulink model. The other output port is for diagnostic purposes. The sfun_rec S-Function employs only one pointer work vector to hold the pointer to the MMF object. The mdlStart callback, which is called once when the Simulink model is initiated, creates the MMF object, passes an MMF name to the object, and saves the pointer into the pointer work vector. The MMF name passed is the same name used by the LAPIN send C++ subroutine, which assures that both are using the same MMF.

When the Simulink simulation runs the process defined in the sfun_rec S-Function, the mdlOutputs callback compares the S-Function block synchronization parameter with a Simulink internal counter variable. If the synchronization parameter is greater than the internal counter, the internal counter will increment and the function returns with no MMF interaction. When the value of the synchronization parameter is equal to the internal counter value, MMF interaction begins. Similar to the LAPIN MMF communication C++ subroutine, the process first checks the MMF process flag to see if there is LAPIN data available for reading. If the flag indicates data is not available with a value of zero the Sleep function is run for 10 milliseconds. The sleep function will allow LAPIN to get processor time to complete tasks that includes writing the data into the MMF for use by the Simulink process. If the MMF process flag repeatedly returns a value of zero for 150 consecutive iterations, the Simulink simulation will move forward using previous read values and apply a value of –2.5 to its diagnostic output port. When the process flag indicates data is available with a value greater than zero, the sfun_rec S-Function reads data from the MMF that was populated by LAPIN and copies the data to its output port. Then, to complete the read transaction, the MMF process flag is reset to zero enabling LAPIN to write more information into the MMF. Finally, the S-Function diagnostic output port is set to the repetition count value. A Scope block can be connected to this diagnostic output port for process monitoring and diagnostic purposes. In either case the internal counter is reset to one. The function then returns and the Simulink model resumes its simulation. At the end of the simulation the mdlTerminate callback is invoked to unmap and close the MMF.

## Time Synchronization of the Simulation Processes

One of the variables under the GEOM section in the LAPIN **lapin.dat** input file is an integer multiplier that defines how many base time steps occur between calls to interact with the MMFs (JSPLIT). Therefore, with a LAPIN base time step (DTIM) set to 0.00002 seconds (20 microseconds), setting the JSPLIT parameter to 1000 will result in LAPIN interactions with the MMF every 0.02 seconds. This interval needs to match the Simulink process base time step size, Ts, and S-Function block dialog parameter (synchronization) values, Ksync. For example, if the Simulink model time step is set for 0.0005 seconds, then the S-Function block dialog parameters should both be set to 40. This will have the Simulink model interact with the MMFs every 0.02 seconds, which matches the LAPIN interval. Note that any Simulink model to be used with LAPIN this way needs to be configured with a fixed time step size. For clarity this arrangement can be expressed by the equation: JSPLIT × DTIM = Ts × Ksync.

## Controlling Model Geometry

The addition of the recvcsub subroutine allows the LAPIN process to read values for setting the AREX, BLMAST, TH, and CW variables based on signals read from the MMF common data area as opposed to being on a time schedule. Also, the LAPIN perturb subroutine code was modified so values defining the perturbation characteristics, EXITBC, can be read from the MMF common data area. A new value for TH can be applied to the THHT table to determine new center-body coordinates using linear

interpolation on the XCBTAB and YCBTAB tables rather than relying on the TRTIME schedule. A similar approach was used to accommodate a changing cowl position. However, the legacy LAPIN input file only accommodated a single X and Y coordinate table for the cowl (XCOWLTAB, YCOWLTAB)—LAPIN was not coded to allow a movable cowl. To enable a variable cowl, a method to define additional cowl contours was coded into a new FORTRAN subroutine in a file called **movecowl.f**. If a change in the CW value is detected, the LAPIN process calls the movecowl subroutine that has access to ten tables of cowl contour Y coordinates, defined with FORTRAN DATA statements. The X coordinates are assumed to stay the same for each contour as defined by XCOWLTAB in the LAPIN input text document. The CW value is used to select one of the 10 tables of Y coordinates. Then a linear interpolation routine is used to establish the new cowl Y coordinates at each X grid point. The linearly interpolated Y coordinates are then made available to the geom subroutine. The revised values for AREX and BLMAST are also used by the LAPIN process to update the bleed plenum conditions by means of code changes made in the uss subroutine.

## Running the LAPIN-in-the-Loop Simulation

The diagram in Figure A4 illustrates the block diagram arrangement of a Simulink model with enabling MMF S-Function blocks. The constant block named "for time step control" is used by sfun_rec to set the block's time step, not by the value of the constant but by the block's Sample Time setting. The Simulation Subsystem block in Figure A4 is where the Simulink simulation calculates new data for the LAPIN process. A scope block is applied to one of the read S-Function output ports for MMF reading diagnostic purposes. The other read S-Function output port transmits signals read from the MMF to the Simulation Subsystem. A second scope is applied to the only write S-Function output port for MMF write diagnostic purposes. The data the Simulink simulation writes to the MMF is transmitted to the four input ports of the write S-Function block from the four Simulation Subsystem exit ports.

The following sequential steps define a procedure for starting the LAPIN-in-the-Loop processes: First start MATLAB and open the Simulink model. Prepare the LAPIN input text document and the Simulink models to simulate the same length of time. Next, open the command shell window and navigate to the directory where the following LAPIN files reside: **aamain.exe** and **lapin.dat**. Then in the command shell, enter '**aamain.exe**' to start the LAPIN process. LAPIN will initialize quickly and perform an MMF interaction. Without delay, start the Simulink model. If all is going well, the command shell window will display messages from LAPIN's sendcsub and recvcsub C++ subroutines showing the iteration counts in the range of 300 or less. The LAPIN and Simulink models should end at about the same time. If all went well, the LAPIN process will terminate after sending a message to the command shell that the MMFs were deleted, followed by FORTRAN STOP.

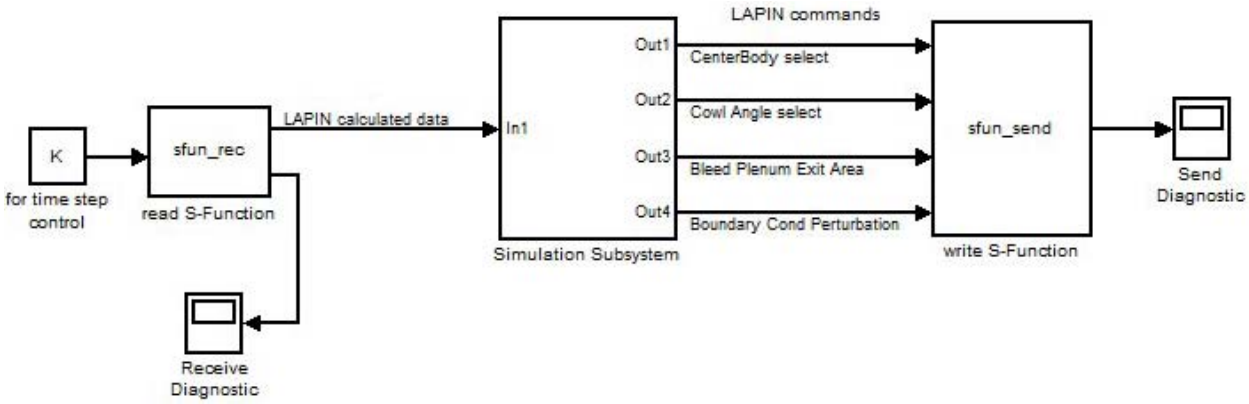


Figure A4.—A sample Simulink model with send and receive S-Functions for interaction with the LAPIN process using MMF communication.

A two-line MATLAB script file can be written to coordinate starting the simulation. The first line should be '!**aamain.exe**&' and the second 'sim(model_name);'. This script can be run from the MATLAB window after opening the Simulink model. Running this script will open the command shell window and the base directory will be the current MATLAB folder where the Simulink model is located. The LAPIN executable will be started and run as a background task. This allows the second line to be executed which is to start the Simulink model running. If the LAPIN files are not located in the same directory as the Simulink model, then the full directory path will need to be supplied as part of the first line.

After the LAPIN and the Simulink simulations are completed, the LAPIN output file, **0lapin.out**, may be opened with a text editor to examine data generated by the LAPIN process. The LAPIN I/O Guide provides information on output variable definitions and value normalization. If bleeds are used, the **0lapin.bld** file will be generated by LAPIN with bleed specific data. Scope plots can be employed in the Simulink model for examination of that process.

LAPIN checks the input file at startup and performs more checks as it runs. A possible error could be an out-of-range command value sent by the Simulink model. If an error is detected, LAPIN will halt execution and display 'FORTRAN STOP' in the command shell window without the MMFs deleted message. In the event of a shortened simulation due to an error, the LAPIN output file, **0lapin.out**, could be examined with a text editor. The output text document should have a message on what caused the early termination at the end of the file. Meanwhile, the Simulink model will still be running and trying to exchange data using the MMF. Therefore, the Simulink process will need to be stopped. The Simulink interruption may be delayed because the model S-Function may need time to complete MMF communication attempts before the model can be stopped. In the event that the Simulink model ends early, the LAPIN simulation should be stopped. This can be done by activating the Command Shell window the then pressing the key combination Control C.

## References


1. National Aeronautics and Space Administration, "2011 NASA Strategic Plan," www.nasa.gov/news/budget, February 14, 2011.
2. NASA facts, "Fundamental Aeronautics Program," www.aeronautics.nasa.gov/pdf/fap_factsheet_02_22_11.pdf, February 22, 2011.
3. NASA,"Hypersonic Research" www.aeronautics.nasa.gov/fap/NASA_FA_Hypersonics_Final.pdf, March 2009.
4. Saunders, J.D., Slater, J.W., Dippold, V., Lee, J., Sanders, B.W., and Weir, L.J., "Inlet Mode Transition Screening Test for a Turbine-Based Combined-Cycle Propulsion System," JANNAF-1024, May 13, 2008.
5. Sanders, B.W. and Weir, L.J., Aerodynamic Design of a Dual-Flow Mach 7 Hypersonic Inlet System for a Turbine-Based Combined-Cycle Hypersonic Propulsion System, NASA/CR—2008-215214, June 2008.
6. TechLand Research, Inc., Test Requirements for Combined-Cycle Engine (CCE) Large-Scale Inlet Mode Transition (LIMX) Test, NASA NRA Contract No. NNC08CA60C, September, 2009.
7. Sanders, B.W. and Mitchel, G.A., Throat-Bypass Bleed Systems for Increasing the Stable Airflow Range of a Mach 2.5 Axisymmetric Inlet with 40-Percent Internal Contraction," NASA/TM—X-2779, May 1973.
8. Stueber, T.J., Le, D.K., and Vrnak, D.R., "Hypersonic Vehicle Propulsion System Control Model Development Roadmap/Activities," NASA/TM—2009-215483.
9. Varner, M.O., Martindale, W.R., Phares, W.J., Kneile, K.R., and Adams, Jr., J.C., "Large Perturbation Flow Field Analysis and Simulation for Supersonic Inlets," NASA CR-174676, 1984.
10. Martindale, W.R., et. al., "Large Perturbation Flow Field Analysis and Simulation for Supersonic Inlets—Program Modifications," NAS3-24105 Task 2608, 1987
11. Sverdrup Technology, Inc., "Lapin Code Modifications," Task Order 4217, Work Elements No. 3 and 5, 1991.
12. Sverdrup Technology, Inc., "Lapin Code Modifications," Task Order 4217, Work Element No. 4, 1991.
13. Stueber, T.J., Vrnak, D.R., Le, D.K., and Ouzts, P.J., "Control Activity in Support of NASA Turbine Based Combined Cycle (TBCC) Research," NASA/TM—2010-216109, March 2010.
14. Gaudette, T., "Using Memory Mapped Files for Fast Data Transfer," MATLAB Digest—March 2004, The MathWorks, Inc.
15. Kath, R., "Managing Memory-Mapped Files," msdn.microsoft.com, February 1993.


## References for Appendix


A1. MAIN-FLO.PDF and LAPIO.PDF are LAPIN documentation files included as part of the LAPIN distribution package.

A2. Gaudette, T., "Using Memory Mapped Files for Fast Data Transfer," MATLAB Digest, March 2004, The MathWorks, Inc.

| REPORT DOCUMENTATION PAGE | Form Approved OMB No. 0704-0188 |
|---|---|



| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED *(From - To)* |
|---|---|---|
| 01-06-2012 | Technical Memorandum | |

**4. TITLE AND SUBTITLE**
A Novel Technique for Running the NASA Legacy Code LAPIN Synchronously With Simulations Developed Using Simulink

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**
Vrnak, Daniel, R.; Stueber, Thomas, J.; Le, Dzu, K.

**5d. PROJECT NUMBER**

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**
WBS 599489.02.07.03.11.01

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
National Aeronautics and Space Administration
John H. Glenn Research Center at Lewis Field
Cleveland, Ohio 44135-3191

**8. PERFORMING ORGANIZATION REPORT NUMBER**
E-18175

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
National Aeronautics and Space Administration
Washington, DC 20546-0001

**10. SPONSORING/MONITOR'S ACRONYM(S)**
NASA

**11. SPONSORING/MONITORING REPORT NUMBER**
NASA/TM-2012-217444

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
Unclassified-Unlimited
Subject Categories: 01 and 59
Available electronically at http://www.sti.nasa.gov
This publication is available from the NASA Center for AeroSpace Information, 443-757-5802

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**
This report presents a method for running a dynamic legacy inlet simulation in concert with another dynamic simulation that uses a graphical interface. The legacy code, NASA's LArge Perturbation INlet (LAPIN) model, was coded using the FORTRAN 77 (The Portland Group, Lake Oswego, OR) programming language to run in a command shell similar to other applications that used the Microsoft Disk Operating System (MS-DOS) (Microsoft Corporation, Redmond, WA). Simulink (MathWorks, Natick, MA) is a dynamic simulation that runs on a modern graphical operating system. The product of this work has both simulations, LAPIN and Simulink, running synchronously on the same computer with periodic data exchanges. Implementing the method described in this paper avoided extensive changes to the legacy code and preserved its basic operating procedure. This paper presents a novel method that promotes inter-task data communication between the synchronously running processes.

**15. SUBJECT TERMS**
Hypersonic inlets; Control simulation; Concurrent processing

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | STI Help Desk (email:help@sti.nasa.gov) |
| U | U | U | UU | 31 | 19b. TELEPHONE NUMBER *(include area code)* 443-757-5802 |