Matlab/Simulink

Using Simulink Models

Description

Simulink models may be loaded into a DSix solution and invoked to interact with the flight model. Any number of Simulink models may be configured for use with DSix, and may range in complexity from simple model enhancements to entire subsystems such as propulsion, hydraulics or flight control systems.

DSix provides a custom Simulink target, (Bihrle Generic Target, or BGT) that invokes Simulink code generation, and then organizes the output for use in DSix, auto-generating a DSix interface to the Simulink model.

In addition, Visual Studio build rules are provided that automate Simulink code generation using BGT, and then compiles the Simulink source , moving the resulting library into the DSix project's target directory.

Requirements

Building Simulink models in the context of a DSix project requires that DSix and Matlab/Simulink be installed on the same machine. Remote access to Matlab is not supported.

A Visual Studio C++ Compiler is required.

Installing a provided Visual Studio plugin that creates the DSix Simulink project build configuration is recommended.

Matlab (32-bit or 64-bit) versions from 2012b through 2018b are supported.

AltHold – A Simple Tutorial

In this tutorial, we will incorporate a simple altitude hold Simulink model into the BARJet flight model. The process will include creating a new Visual Studio project for the Simulink model, which will be added to the BARJet solution.

Once incorporated, rebuilding the solution will launch Simulink code generation, create a DSix interface into the Simulink model, and compile the Simulink model, moving the resulting .dll into the BARJet target directory.

We will then modify the flight model code to load and exercise the new library.

AltHold – A Simple Tutorial

What you will learn

Upon completion of this section, you will be able to:

- Configure your system for DSix/Simulink interaction
- Configure Simulink models to use the Bihrle Generic Target
- Create a Visual Studio project customized to build Simulink models for use with DSix flight model solutions.
- Modify DSix flight model source code to utilize compiled Simulink libraries

DSix/Simulink Initial System Setup

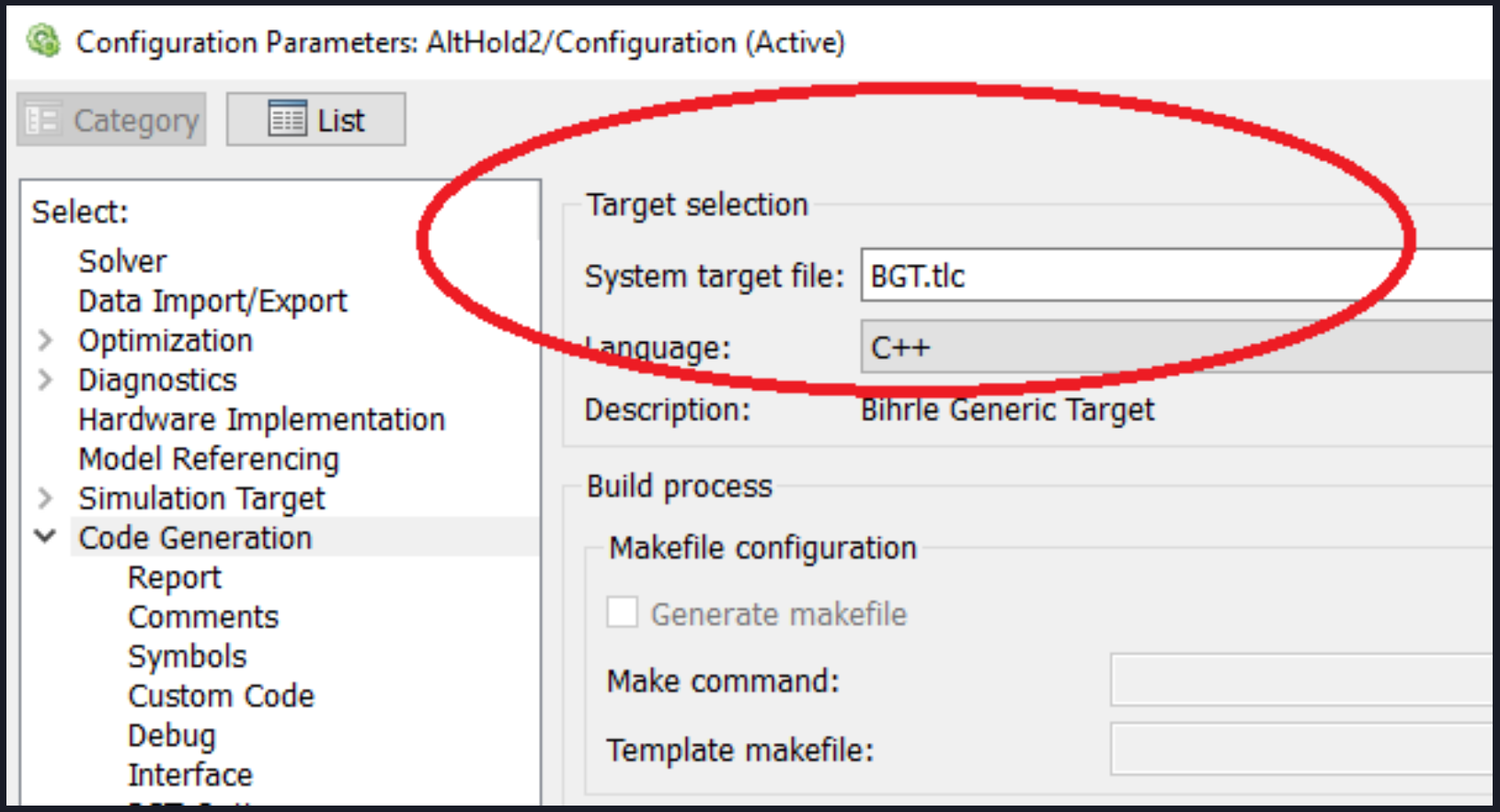
- Install Matlab/Simulink
- Install DSix
- User Environment Variable

  BGT_DEFAULT_MATLAB = <Matlabroot>\bin
- Install Visual Studio Extension

  <DSixRoot>\Matlab\BihrleGenericTarget\Packages
- Add BGT folder to Matlab Path

BGT.tlc

The Simulink model must select the BGT.tlc target



Simulink Visual Studio Project in DSix Workspace

A typical DSix workspace contains individual projects

- Data
- Flight Model
- Dictionary
- Simulink

Multiple Simulink “projects” may be included in a solution. This allows sub-sytems to be built independently of one another

DSix/Simulink Extension for Visual Studio

- DSix provides plugin extensions to Visual Studio that may be used to auto-generate a Simulink project in a DSix solution.

- Extensions are currently provided for Visual Studio versions 2010 – 2017. The extension installers may be found in <DSixRoot>\Matlab\BihrleGenericTarget\Packages.

- Installation requires Admin Rights.
- Close all Visual Studio instances before installing.

Why Create Separate Simulink VS Projects

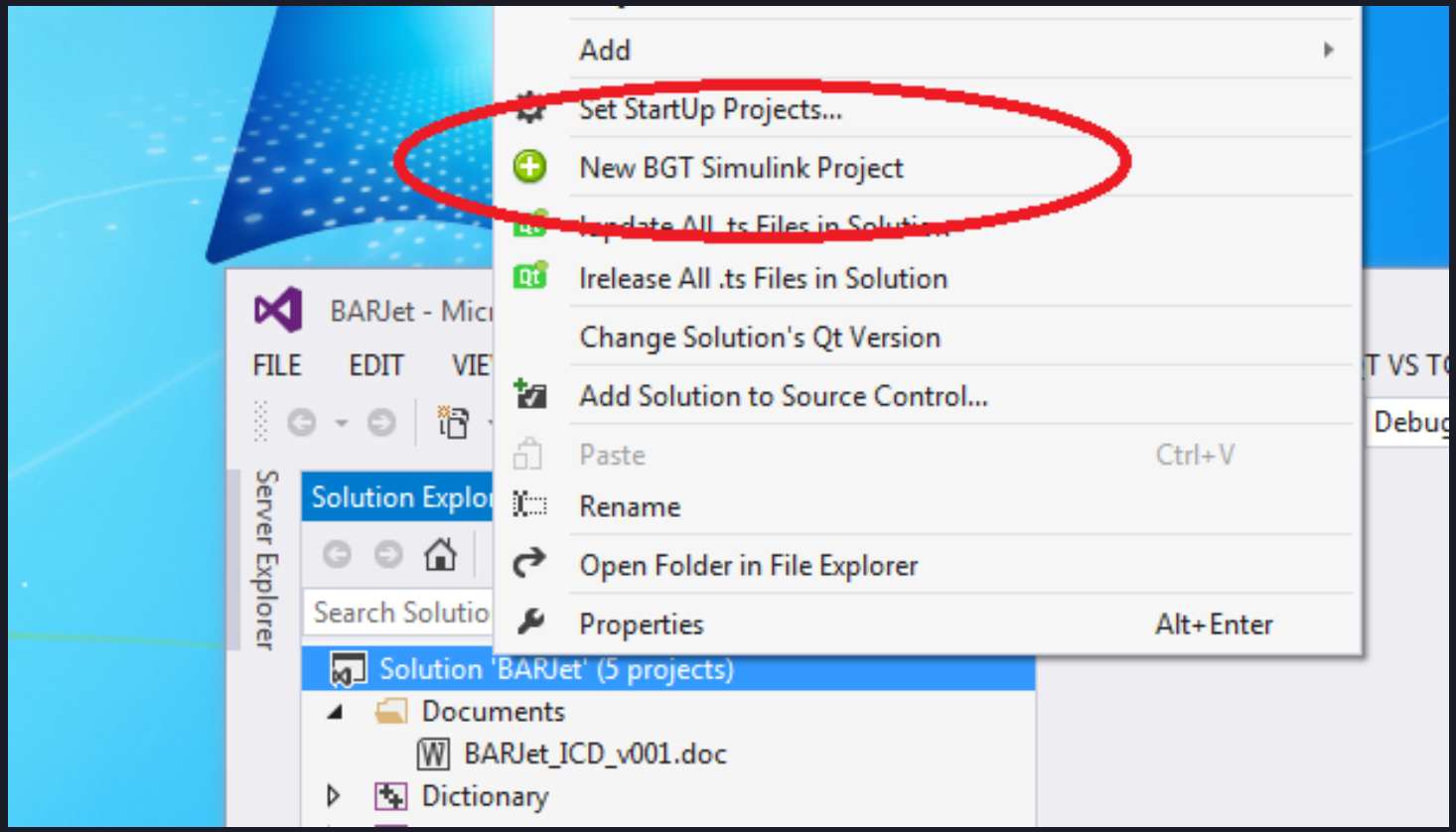
Some complicated simulation models may contain many Simulink models, representing several different subsystems. In these instances, it us usually desirable to group the models with others in the same subsystem. Keeping all propulsion models, for example, aids in project organization.

Keeping related models in a dedicated project also allows the user to rebuild all models of a particular subsystem without necessarily rebuilding other models stored in a different project.

In addition, older DSix models do not contain the current build rules that are used in building Simulink Models. Using the plugin will create a Visual Studio project in your model workspace that is equipped with these latest build rules.

Invoking the Plugin Extension

To add a Simulink project to a DSix flight model solution, right-click the solution. Select “New BGT Simulink Project”.



Project Naming Options

A name must be provided for the new Visual Studio Simulink project. Common default options are provided. If the default options are not desirable, a custom name may be provided. Custom names are subject to naming rules governing standard c identifiers.

- Letters, numbers and underscores only
- Must begin with a letter or underscore
- Name this project “AltHold”

Plugin Output

Running the plugin to add a project to the solution will result in the following being added to the <*projectRoot*>_*Simulink* directory:

- Visual Studio build rule files
  - RTW_SLX.props
  - RTW_SLX.targets
  - RTW_SLX.xml
- Visual Studio project file <*NewProject*>.*vcxprj*
- A folder, named for the <*NewProject*>. This folder will store all simulink model files (*.slx) that will be part of this project.

Adding Simulink Model(s) to the Project

- Copy the file AltHold.slx to your BARJet_Simulink\AltHold\ folder
- In the Visual Studio solution, right-click the "*Simulink-AltHold*" project and select *..Add..Existing Item*.
- Browse to, and select AltHold2.slx
- Select the BARJet_FlightModel project. Then select ..Project..Project Dependencies from the Visual Studio menu.
- Ensure that the _FlightModel is dependent upon the AltHold project.
- Rebuild the solution

Editing Flight Model Source Code

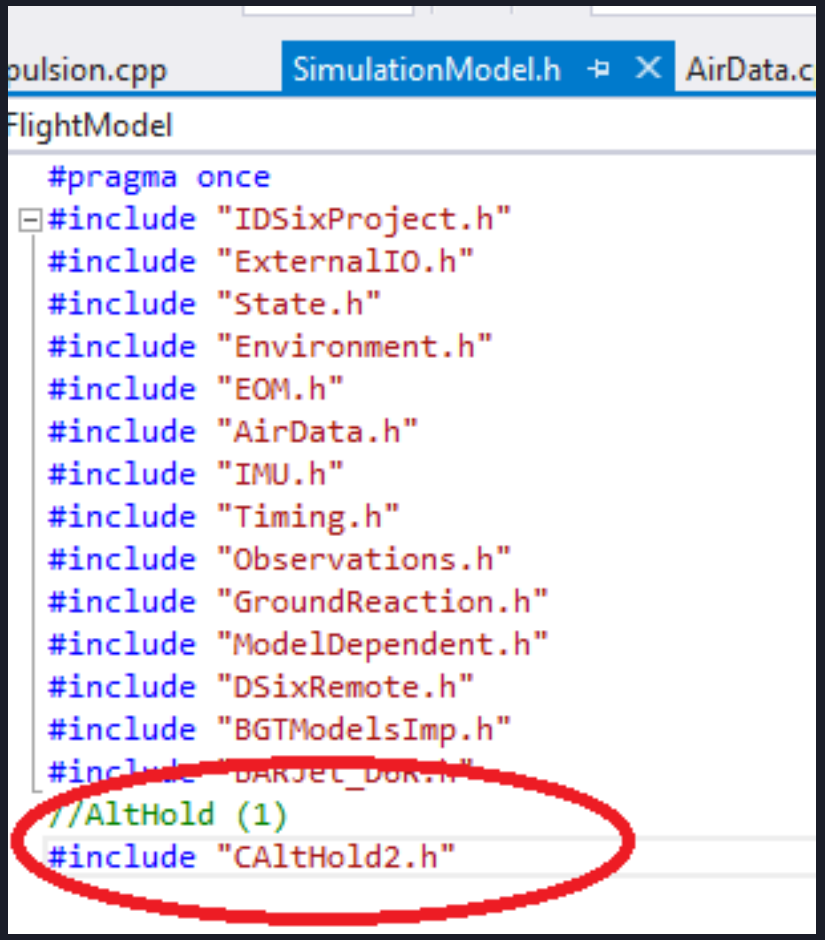
Now that the AltHold model has been compiled inside the DSix solution, the project's flight model code must be modified to load and exercise the model.

In this simplified demonstration, code will be edited in the following files:

- SimulationModel.h
- SimulationModelInit.cpp
- ComponentMapping.cpp
- SimulationModel.cpp

SimulationModel.h

Include the header for Simulink CAltHold



SimulationModel.h

Declare a member pointer to CAltHold

```
class CSimulationModel :
    public IDSixProjectEx
{
protected:
    CExternalIO          *m_pExternalIO;
    CState               *m_pState;//
    CEnvironment         *m_pEnvironment;//
    CEOM                 *m_pEOM;
    CAirData             *m_pAirData;//
    CIMU                 *m_pIMU; //
    CTiming              *m_pTiming;//
    CObservations        *m_pObservations;//
    CGroundReaction      *m_pGroundReaction; //
    // Model Dependent Components
    CFlightControl       *m_pFlightControl;
    CAerodynamics        *m_pAerodynamics;
    CPropulsion          *m_pPropulsion;
    CWeightAndBalance    *m_pWeightAndBalance;//
    CBGTModels           *m_pBGTModels;

    //AltHold (2)
    CALTHOLD2            *m_pAltHold;

    // Hold Data
    CModelDependent      *m_pDep;
```

SimulationModel.h

Declare a “SetInput” function for CAltHold

```
// ------------------------------------
// ComponentMapping
// ------------------------------------

    void SetStateInput();
    void SetEOMInput();
    void SetEnvInput();
    void SetAirDataInput();
    void SetIMUInput();
    void SetTimingInput();
    void SetObsInput();
    void SetGRInput();
    void SetBGTInput();
    //AltHold (3)
    void SetAltHoldInput();

    void SetStateSnapshotInput();
```

SimulationModel.h

Declare a “GetOutput” function for CAltHold

```
    void GetEOMOutput();
    void GetStateOutput();
    void GetEnvOutput();
    void GetAirDataOutput();
    void GetIMUOutput();
    void GetTimingOutput();
    void GetObsOutput();
    void GetGROutput();
    void GetRCTOutput();
    //AltHold (4)
    void GetAltHoldOutput();

// ------------------------------------------
// DSixRemote support
// ------------------------------------------
    void ConnectToDSixRemote();
```

SimulationModel.cpp

Initialize the AltHold class in OnInitInterface

```
    GetGROutput();

    m_pEOM->Initialize();
    GetEOMOutput();

    m_pObservations->Initialize();
    GetObsOutput();

    DSixRemoteSendData();

    //AltHold (5)
    m_pAltHold->Initialize();
    ////////////////////////
}

void CSimulationModel::OnShutdown()
{
    ReleasePointers();
```

SimulationModel.cpp

Assign AltHold member pointer to NULL in ClearPointers()

```
void CSimulationModel::ClearPointers()
{
    m_pExternalIO = NULL;
    m_pState = NULL;
    m_pEnvironment = NULL;
    m_pEOM = NULL;
    m_pAirData = NULL;
    m_pIMU = NULL;
    m_pTiming = NULL;
    m_pObservations = NULL;
    m_pGroundReaction  = NULL;
    m_pDep = NULL;
    m_pAerodynamics     =   NULL;
    m_pBGTModels        =   NULL;
    m_pPropulsion       =   NULL;
    m_pFlightControl    =   NULL;
    m_pWeightAndBalance =   NULL;
    m_pInternals    = NULL;
    //AltHold (6)
    m_pAltHold = NULL;

    m_bInitialized = false;
}
```

SimulationModel.cpp

Instantiate the AltHold pointer and call Initialize in InitializePointers()

```
void CSimulationModel::InitializePointers()
{
    ReleasePointers();
    m_pExternalIO = new CExternalIO;
    m_pState = new CState;
    m_pEnvironment = new CEnvironment;
    m_pEOM = new CEOM;
    m_pAirData = new CAirData;
    m_pIMU = new CIMU;
    m_pTiming = new CTiming;
    m_pObservations = new CObservations;
    m_pGroundReaction = new CGroundReaction;
    m_pInternals = new INTERNALS;

    //AltHold (7)
    m_pAltHold = new CALTHOLD2;
    m_pAltHold->LoadModel();
    ////////////////////////////////////
```

SimulationModel.cpp

Cleanup in ReleasePointers()

```
void CSimulationModel::ReleasePointers()
{
    if (!m_bInitialized)
        return;

    m_bInitialized = false;

    delete m_pExternalIO;
    delete m_pState;
    delete m_pEnvironment;
    delete m_pEOM;
    delete m_pAirData;
    delete m_pIMU;
    delete m_pTiming;
    delete m_pObservations;
    delete m_pGroundReaction;
    delete m_pInternals;

    //AltHold (8)
    m_pAltHold->CloseModel();
    delete m_pAltHold;
    ////////////////////////////////////
```

ComponentMapping.cpp

Map DSix variables to the AltHold inputs in ComponentMapping.cpp

Note that some variables (flgAltHold) have not yet been created in DSix.

We will created these variables before rebuilding the solution.

```
void CSimulationModel::SetAltHoldInput()
{
    //EXTU_ALTHOLD2_T
    //inputs defined in ALTHOLD2_ModelInterface.h
    EXTU_ALTHOLD2_T AltHoldInputs;

    AltHoldInputs.flgAltHoldEngage = m_pDep->flgAltHold;
    AltHoldInputs.Alt = m_pDep->AirDataOut_Hp;
    AltHoldInputs.Hdot = m_pDep->IMUOut_Vv;
    m_pAltHold->SetInputs(&AltHoldInputs);
}
```

ComponentMapping.cpp

Map DSix variables to the AltHold outputs in ComponentMapping.cpp

Note that some variables have not yet been created in DSix.

We will created these variables before rebuilding the solution.

```
void CSimulationModel::GetAltHoldOutput()
{
    //EXTU_ALTHOLD2_T
    //outputs defined in ALTHOLD2_ModelInterface.h
    EXTY_ALTHOLD2_T AltHoldOutputs;
    m_pAltHold->GetOutputs(&AltHoldOutputs);
    if (AltHoldOutputs.EngageFlag) //repeater of input flag.
    {
        //setting elevator deflection.
        m_pDep->Dh_Ave = AltHoldOutputs.Dh_AP;
        //test variable
        m_pDep->AltHoldDhAve = AltHoldOutputs.Dh_AP;
        m_pDep->AltHoldTarget = AltHoldOutputs.AltTarget;
    }
}
```

SimulationModel.cpp

Reset the AltHold model when DSix resets.

```
void CSimulationModel::OnReset(int flgResetType)
{
    ZeroAll();
    bool bResetData = DSixRemoteGetData(true);
    if (m_pDep->IC_icflgGroundInit) {
        m_pDep->Pilot_GearLever = 1.0;
    }
    switch (flgResetType){
        case IOS_RESET:
            SetTimingInput();
            m_pTiming->OnReset();
            GetTimingOutput();

            //AltHold (10)
            pD6Data->VARSetDefaultVal("flgAltHold", 0.f);
            SetAltHoldInput();
            m_pAltHold->Reset();
            GetAltHoldOutput();
            ////////////////////////////
```

SimulationModel.cpp

Step the AltHold model when DSix steps. Note that in many cases, the order in which Simulink models are called is important. In this case, AltHold is called after the flight control component and prior to the Aerodynamics component.

```
        m_pPropulsion->OnStep();
    }
    else {
        m_pBGTModels->OnStep();
        m_pFlightControl->OnStep();

        //AltHold (11) In this case, the model should be
        //called after flightControl (overridding flightcontrol output),
        //and prior to Aerodynamics.
        SetAltHoldInput();
        m_pAltHold->Step(-1);
        GetAltHoldOutput();
        ////////////////////////////////////////
        m_pAerodynamics->OnStep();
        m_pPropulsion->OnStep();
    }
```

Testing the AltHold Model in the BARJet

At this point, the AltHold library is available, and code has been added to utilize the library. After some quick setup in the BARJet, we will be ready to see the AltHold library in action.

- Add AltHold variables to the BARJet project
- Set up a variable display to show relevant data
- Set up initial flight conditions for our test run
- Configure I/O devices, setting a joystick button to toggle AltHold on/off
- Open DSixGraphics_II for a visual representation of the results

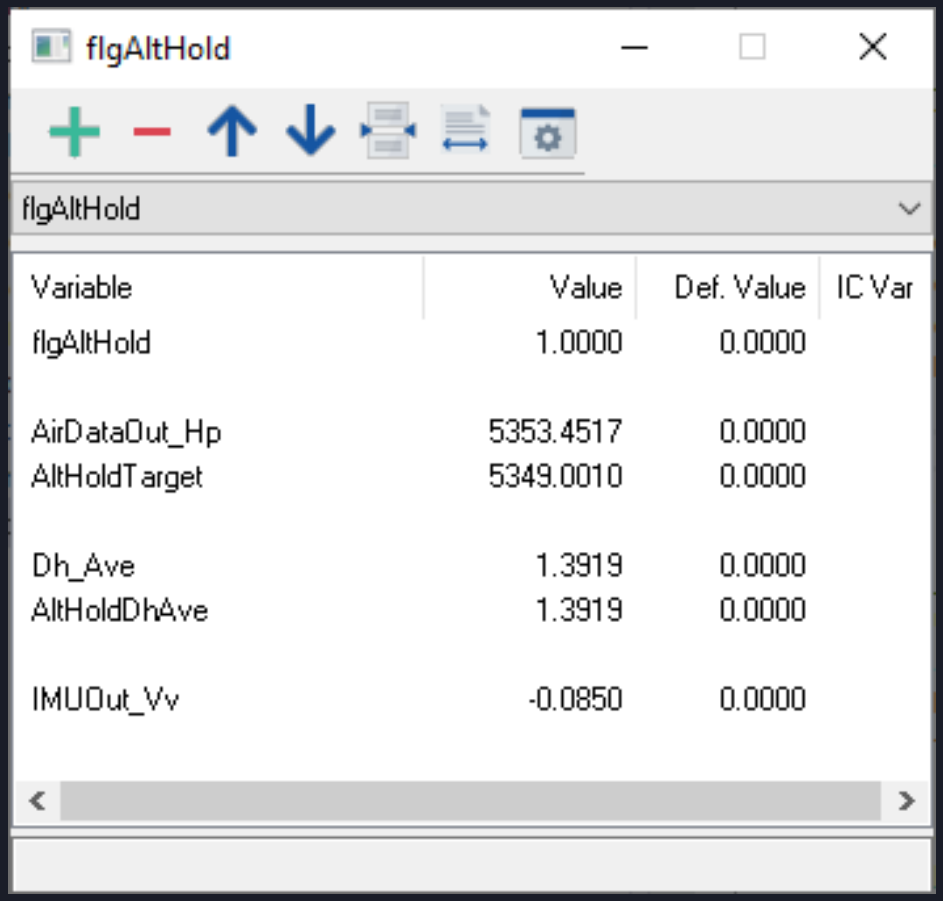
Adding AltHold Variables to the Project

- Start DSix, and load the BARJet project
- From the DSix menu, select ..Edit..Variables
- Click “Add”. Enter the variable name “flgAltHold”. This will represent an input into the AltHold library
- Add the variable “AltHoldTarget”. This will be the Simulink model’s target altitude, established at the time the model is engaged.
- Add the variable AltHoldDhAve. This is just a test variable, mapped to the model’s output and used to confirm that the actual DSix variable, Dh_Ave is not being overwritten
- Close the Edit Variables box with OK, and save the project.

Set up a DSix Variable Display

From the DSix menu, select ..View..Variable Display..New

Add variables to match the following display, then save the DSix project.



Initial Conditions

From the DSix menu, select ..Edit..Initial Conditions.

Select the “In Flight” condition set, and set initial altitude to 5000’.

Close with OK, and save the DSix project.

I/O Devices

We need to configure I/O devices to utilize the current I/O hardware. This task will depend on the specific hardware. See the IOD slide deck for some general examples.

Once the stick is properly set up for flight, we will map a button on the stick to enable/disable our AltHold model.

- ..Tools..IO Devices, “Events” tab
- Set DSix Variable Event mapped to flgAltHold

  On value = 1, Off value = 0, type = “Toggle”
- Trigger = “Stick Buttons”. Select a button on the stick.

  Edge Triggered, Rising Edge
- OK, Save DSix Project

Graphics Display

If a DSixGraphics_II window is not already open, select

..View..Image Generator.. Open New Graphics II Window

If this entry is not available, select ..Tools Module Manager, and mark the graphics II module to load. Restart DSix, and reload the project. Now open a graphics II window.

AltHold in Action

We should now be ready to test our AltHold model inside the BARJet.

Start a simulation run, using your joystick to manipulate pitch. Take note of the results in the graphics and the variable display.

Try to get the aircraft reasonably level to reduce porpoising, and click the trigger that was mapped earlier.

Note that the flight model no longer responds to manual pitch inputs.

Observe the results in the graphics and variable displays.

Summary

This completes the AltHold Simulink tutorial.

It should be noted that this was a simplistic example for introductory purposes. More complex examples are beyond the scope of this training session.

In general, Simulink models are implemented into a DSix flight model in the form of DSix model components, rather than through standalone model libraries.


DSIX
Flight Simulation Environment