Designing with DSP Builder Advanced Blockset

Fabio Sousa
DHW Engenharia e Representação
Objectives

Upon successful completion of this course, students will be able to:

- Implement DSP algorithms using Altera® DSP Builder Advanced Blockset
- Incorporate ModelIP cores in design
- Verify the hardware performance in Altera Quartus® II development tool
- Explore design architecture and performance tradeoffs using system level constraints
Agenda

- DSP Builder overview
- DSP Builder Advanced Blockset highlights
- Design rules and protocols
- Hardware integration
- Hardware resource trade-off
- Architecture exploration
Designing with DSP Builder Advanced Blockset

DSP Builder Overview
DSP Builder Design Flow

Development

System Level Simulation of Algorithm Model

Algorithm-level Modeling

MATLAB/Simulink

Implementation

RTL Implementation
RTL Simulation

Synthesis, Place ‘n Route, RTL Simulation

Precision, Synplify Quartus II, ModelSim

Single Simulink Representation

Verification

System Level Verification of Hardware Implementation

System-level Verification

Altera FPGA Altera Development Kits

© 2010 Altera Corporation—Confidential

ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.
DSP Builder- Standard and Advanced Blockset

Library: Altera DSP Builder Blockset
- All Blocks
- Arithmetic
- Complex Type
- IO & Bus
- MegaCore Functions
- Simulation Blocks Library
- Storage

Search Results: ( )
- AltLab
- Boards
- Gate & Control
- Interfaces
- Rate Change
- State Machine Functions

Library: Altera DSP Builder Advanced Blockset
- Base Blocks
- FFT Blockset
- Filters
- ModelBus
- ModelPrim
- ModelVectorPrim
- Waveform Synthesis

© 2010 Altera Corporation—Confidential
ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.
DSP Builder Standard Blockset

Algorithm Specification → RTL Implementation → Synthesis and Place-&-Route → Timing Analysis

1a. Create schematic and/or import HDL
1b. SignalCompiler converts model into VHDL code

2. SignalCompiler invokes Quartus II Software in the background and compiles design

3a. Specify timing constraints in Clock block
3b. Close timing with “Display Pipeline Depth” Block

4. “Test Bench” block performs RTL simulation via ModelSim and (optionally) compares against Simulink results

5a. Speed up simulation with Hardware-in-the-Loop
5b. Invoke SignalTap II Logic Analyzer from within Simulink

- WYSIWYG design tool with flexibility of MATLAB and Simulink
- Access to common DSP functions, IP blocks, device-specific features and hardware debug capability
- DSP Builder data type

DSP Builder Blockset
Demos Provided with DSP Builder Blockset

The Altera DSP Builder Blockset shortens DSP design cycles by helping you create the hardware representation of a digital signal processing design in an algorithm-friendly development environment and allows system, algorithm, and hardware designers to share a common development platform. You can combine existing MATLAB functions and Simulink blocks with Altera DSP Builder blocks and MegaCore functions, including it support the OpenCore Plus hardware evaluation feature, to link system-level design and implementation with DSP algorithm development.

Tutorial Designs
- Getting Started Tutorial
- Hardware in the loop - FreqSweep
- Switch Control
- Avalon-MM Interface
- Avalon-MM FIFO
DSP Builder Advanced Blockset

Algorithm Specification → RTL Implementation → Synthesis and Place-&-Route → Timing Analysis

1a. Design algorithm and architecture using Advanced Blockset
1b. Specify timing constraints in “Signals” block

2a. Specify HDL location in “Control” block
2b. Simulate model to generate VHDL

3a. Launch ModelSim with “Run ModelSim” block
3b. Run ModelSim in background via automatic scripts and compares against Simulink results

4. Close timing TimeQuest Timing Analyzer

5. Debug hardware with SignalTap II Logic Analyzer

- Intuitive text-book based algorithm design
- High-level constraint driven synthesis
- Rapid design exploration
- Simulink fixed point data type

Architectural first and implementation second!
User Benefits

- Hide the complexity of the FPGA

Key design questions:
- How much pipelining is required to hit the required system clock rate?
- How much time sharing can be exploited to save FPGA area/resources?
- How many design iterations are required to explore architectures?

HLS based Solution:
- Removes these complexities through automation
- Focuses on optimizing key DSP design elements
Core Technologies

- **IP generation**
  - Multi-rate, multi-channel filters
  - Waveform synthesis (NCO/DDS/Mixers)

- **Custom block creation using primitive library**
  - Vectorization, Folding, Zero Latency, Scheduled, Aligned RTL Generation

- **System Integration**
  - Memory synthesis
Quick IP Generation

Blocks
- Single Rate FIR
- Decimating / Interpolating FIR
- Fractional Rate FIR
- Decimating / Interpolating CIC
- NCO, Mixers
- FFT building blocks

Example: FIR
- Fixed point integration
- poly-phase symmetry
- half-band, L\textsuperscript{th}-band
- Memory mapped control

- Built-in vector support
- Built-in TDM (folding) support
- Parameterizable and scriptable
Custom IP Generation

- Vector support
- TDM (folding) support
- Parameterizable
- Zero latency block

**What to do and not when to do it**

Textbook based design entry

Schematic simplification using vector support
Timing Driven Synthesis

• Simply enter desired System Clock Frequency,
• No need to change model

**Small or fast RTL from the same model!**

---

### Simple 50-bit 4-input adder tree

<table>
<thead>
<tr>
<th># of Pipeline Stage</th>
<th># of LUT (LUT4s)</th>
<th>FMax (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>118</td>
<td>121</td>
</tr>
<tr>
<td>1</td>
<td>175</td>
<td>286</td>
</tr>
<tr>
<td>5</td>
<td>350</td>
<td>581</td>
</tr>
</tbody>
</table>
Library is Technology Independent

- Target a device using a device block (simple GUI)

*Same model generates optimized RTL for each FPGA family*
Memory Synthesis

- Automated processor ↔ hardware interface synthesis
- Builds pipelined memory mapped interface logic for:
  - IP (eg filter coefficients)
  - Primary Systems (eg registers, shared memories)
SOPC Builder Integration

- `<model>_hw.tcl` files generated for each system
- Slower bus clock connects to system
- Faster signal processing clock is external
- Hierarchical address spaces, so systems can be easily integrated

Build Complete, High Performance Programmable DSP systems
Documentation Generation

- Memory mapped information is collated into XML and processed to HTML for integrated Matlab Help

System level memory map is processed to datasheet representation
Example Applications

- High performance FIR and IIR Filters
- Digital Up Conversion
  - CDMA, WiMax, LTE wireless applications
  - SDR military applications
  - Head end broadcast applications
- Digital Down Conversion
  - CDMA, WiMax, LTE wireless applications
  - Radar and SDR military applications
  - Medical Imaging
- FFT/IFFT
  - Radix2, 4
  - Custom designs, non-standard lengths
Demos provided with Advanced Blockset
Standard versus Advanced Blockset

- **DSP Builder Standard Blockset**
  - Cycle accurate behavior
  - Multiple clock domain design
  - Control rich with state machine or backpressure
  - Access detailed device features
  - HDL import support
  - Hardware-in-the-loop support

- **DSP Builder Advanced Blockset**
  - Specification driven design with automatic pipeline and folding
  - Multichannel designs with automatic vectorized inputs
  - Automatic generation of memory mapped interface
  - Single system clock for the main datapath logic
  - Feedforward datapath with minimum control
  - Design exploration
When to Combine the two Blocksets

- Your design has portions which are typical of both design styles
  - Use both blocksets together to create your overall design.
  - Create Advanced Blockset module within Standard DSP Builder
  - You cannot mix and match old & new DSPB blocksets within the same hierarchy level

- For more information about the advanced blockset, refer to:
  - *DSP Builder Advanced Blockset Reference Manual*
DSP Builder Design Flow Advantage

■ Hardware Engineers
  - Extends RTL analysis and debug capabilities to system level Tool
    ● Access to MATLAB data formatting
    ● Access to a large library of Simulink models
  - Allows designing in an abstract format, eliminating the need for timing closure and generation of minimal control logic
  - Enables IP evaluation at system level

■ System Engineers
  - Allows rapid prototyping with minimal PLD expertise
  - Provides easy access to hardware evaluation
  - Create an *executable spec* to hardware engineers
Support Resource

- DSP Design Flow User Guide:
  - Chapter 3 – Interoperability
Designing with DSP Builder
Advanced Blockset

Simulink Refresher
Port/Signal Display

- Format -> Port/Signal Displays from Model Editor
  - Need to update diagram

- Displays various port/signal properties:
  - Sample Time Colors:
    - Change the color of blocks and wires in particular clock domain
    - Useful when creating multi-rate designs
  - Port Data Type:
    - Display the data type of the blocks
    - Can only connect ports of same data type
  - Signal Dimensions
    - Display the dimensions of particular signal wire
Simulink From/To Workspace Blocks

- Data can be imported from workspace, using a “From Workspace” block (under Simulink Source block).

- Data can be exported to workspace, using a “To Workspace” block (under Simulink Sink block).

- Supported data format:
  - 2-D matrix (time column and row column):
    
    \[
    \begin{bmatrix}
    \text{TimeValues} & \text{DataValues}
    \end{bmatrix}
    \]
  
  - Matlab structure
    
    \begin{verbatim}
    var.time = [TimeValues]
    var.signals.value = [DataValues]
    var.signals.dimensions = [DimValues] %required if 2D data...
    \end{verbatim}
Designing with DSP Builder Advanced Blockset

Requirement, Installation and Licensing
Software Requirement

- Quartus II development tool, version 10.1
- MATLAB/Simulink R2009a, R2010a
  - Only 32-bit version of the MathWorks releases
  - Install with read only option unchecked
- DSP Builder, version 10.1
  - Install Quartus II development tool and MATLAB/Simulink first
- (Optional) ModelSim-Altera software
Licensing Requirement

- MATLAB/Simulink license
  - Licenses for Simulink fixed point blockset and Fixed-Point Toolbox
  - Licenses for Signal Processing Blockset and Communications Blockset
    - Used in the demonstration designs

- DSP Builder license
  - Required to simulate the model and generate HDL
Tool Download Locations

- **DSP Builder:**

- **MATLAB and Simulink evaluation:**

- **Quartus II development tool:**

- **ModelSim-Altera software:**
Support Resource

- DSP Builder Installation and Licensing:
    - Chapter 1 – System Requirement
    - Chapter 2 – DSP Builder Installation
Designing with DSP Builder Advanced Blockset

Design Flow Overview
Design Flow: Testbench Creation

- Top-level of a DSPB-AB design is a testbench
- Must include **Control** and **Signals** blocks
Design Flow: **Hardware Implementation**

- Enter the design in the subsystem
- **Device** block marks the top level of the FPGA device and specifies the target device used for the generated hardware
Design Flow: *Bus Creation*

- Define bus address and data widths in **Control** block
- Define base addresses for memory-mapped registers in the design
Design Flow: **Hardware Generation**

- Specify target Fmax
- Run the model in Simulink and optimize to meet Fmax
- Generate the hardware and build up the memory-mapped address at run time

© 2010 Altera Corporation—Confidential

ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.
Design Flow: *Hardware Verification*

- **RunModelSim** block loads the design into the ModelSim simulator.
- **RunQuartus** block loads the design into the Quartus II software.
- Memory-mapped interface provided for easy integration with SOPC Builder.
Tool Interoperability

- Advanced Blockset interoperates with other Altera & 3rd Party tools

MATLAB
Simulink
DSP Builder
Advanced Blockset
Model Technology
SOPC Builder
Support Resource

DSP Builder Advanced Blockset User Guide:
- Chapter 3 – ModelIP Tutorial
Designing with DSP Builder
Advanced Blockset

Advanced Blockset Library
Library Components

- Base Library
- ModelIP Library
  - Filters
  - Waveform Synthesis
- FFT Library
- ModelPrim (Primitive) Library
- ModelBus Library
Base Library

- Channel Viewer
- Control
- Device
- Run ModelSim
- Run Quartus
- Scale
- Signals
Base Library - Control Block

- Must be present at the top level of the model

Hardware generation

Memory mapped bus interface

Trade-off options
Base Library - Signals Block

- Must be present at the top level of the model
- Name the clock, reset and memory bus signals used in the RTL
- Provides information about the clock rate
  - Calculate the ratio of sample rate to clock rate to determine the amount of folding in the ModelIP filters
  - To determine the pipelining required at each stage of logic
System Clock vs. Bus Clock

Nios® II Embedded Processor

Bus Clock

System Clock

Block A → Block B → Block C → Block D
Base Library - Device Block

- Mark a particular Simulink subsystem as the top level of an FPGA device
- Allow you to specify the target device and speed grade for the device
- Supports the following Altera device families:
  - Stratix, Stratix GX, Stratix II, Stratix II GX, Stratix III, Stratix IV, Cyclone II, Cyclone III, Cyclone III LS, Cyclone IV GX, Arria GX and Arria II GX
ModelIP Library

- **Filters Library**
  - Single rate, multi-rate, and fractional rate FIR filters
  - Decimating and interpolating cascaded integrator comb (CIC) filters

- **Waveform Synthesis Library**
  - Numerically controlled oscillator (NCO)
  - Complex mixer
  - Real mixer
ModellIP Library – FIR filter

- Accepted system parameters
  - Input Rate per Channel
  - Rate Change Factor, when applicable
  - Number of Channels
- Coefficient generation through MATLAB function
- Memory mapped interface support to reload coefficient
What is a Super Sample Rate FIR?

- Symbol Rate > Clock Rate

Example:

```
256Msps 512Msps 1024Msps
```

\[ f_{\text{FPGA}} = 256\text{MHz} \]

- DSP Builder Advanced Blockset Assumes 2:1 Interpolation Half Band Filters
  - Automatically duplicates necessary hardware to generate parallel outputs
Super Scaler Systems

Current Design Methodology

- Complex control required
- User has to manage reordering of Odd / Even phases of half band filters

Advanced Blockset Methodology

- Significantly simplifies high speed design
- The tool automatically duplicates necessary hardware to generate parallel outputs
- Handles polyphase reordering
- Design is simple and clean
- Etc…….
Advanced Blockset Super Scaler Example

8 Channels of I&Q at 16 Msps Time Shared on the same bus at 256 MHz

8 NCOs with 6 MHz channel spacing = 48 MHz band

Up converted 256 MHz IF (Separate I/Q) Eight 6 MHz Signals in 48 MHz Band

Polyphase NCO: 16 Phases in parallel

Complex Mixer

Real Mixer

Note that at this point we have 16 phases of I, 16 phases of Q, 16 phases of sine and 16 phases of cosine all managed for us!
ModellIP Library – NCO

- **Accumulator bit width**
  - Determines the precision with which NCO frequency can be controlled
  - Limited to 15 – 30 (with a 32-bit memory map, e.g. Nios II) or up to 40 (constant Read/Write mode)

- **Phase Increment**
  - Vector that controls the generated frequencies
  - The length of vector = # of channel
  - \( \Phi_i(n) = \left[ \frac{f_0(n)}{f_s} \right] \cdot 2^{(\text{accumulator bit width})} \)

- **Phase Inversion (top two MSB):**
  - 00 -> no inversion for sine and cosine
  - 01 -> invert cosine and not sine
  - 10 -> invert sine and not cosine
  - 11 -> invert both sine and cosine

- **Memory mapped interface support**
- **Results seen in the Results tab**
- **Expected SFDR ~ 6.2 * # of output bits**
What is Frequency Hopping in NCO?

- Each channel can change frequency from a pool of multiple frequencies (a requirement in Multi-carrier GSM)
- Phase Increment and Inversion of NCO is a frequency matrix
- Each column corresponds to a channel, and each row to a “bank” of frequencies that would be used at a given point in time.
- The “bank” signal selects a row in the frequency matrix
- The frequency matrix can be updated real time using a processor
What is Frequency Hopping in NCO?

Phase transition is continuous

20.9MHz  24.5MHz
How to Update Frequency Matrix?

- NCO allocates phase increment memory registers automatically
- Access memory registers via Avalon-MM interface
- Calculate the address of memory registers

Example:

NCO Phase Increment Base Address = 1000
Accumulator bit width = 24
System Data Width (in Control block) = 32

Address chart:

<table>
<thead>
<tr>
<th>1000</th>
<th>1001</th>
<th>1002</th>
<th>1003</th>
</tr>
</thead>
<tbody>
<tr>
<td>1004</td>
<td>1005</td>
<td>1006</td>
<td>1007</td>
</tr>
<tr>
<td>1008</td>
<td>1009</td>
<td>1010</td>
<td>1011</td>
</tr>
<tr>
<td>1012</td>
<td>1013</td>
<td>1014</td>
<td>1015</td>
</tr>
</tbody>
</table>

Ch0  Ch1  Ch2  Ch3

bank0  bank1  bank2  bank3
FFT Library

- **Common folder**
  - Complex Multiplier
  - Complex Sample Delay
  - Dual Twiddle Memory
  - Negate
  - Negate Parameterizable

- **Radix2 folder**
  - Butterfly I (BFI)
  - Butterfly II (BFII)
  - Bit Reverse
  - Twiddle Generator
Example: 16-point FFT

Reference: A New Approach to Pipeline FFT Processor
Shousheng He & Mats Torkleson, Department of Applied Electronics, Lund University, Sweden

© 2010 Altera Corporation—Confidential
ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.
ModelPrim (Primitive) Library

- Arithmetic
- Delay
- Boolean
- ChannelIn, ChannelOut
- SynthesisInfo
ModelPrim Library– SynthesisInfo Block

- Set the synthesis mode
  - Scheduled (Recommended)
  - WYSIWYG

- Labels a subsystem described by primitive blocks as the top level of a synthesizable subsystem tree
Data Path Vectorization

- **N** separate operations bundled into one operator
  - Example below shows 8 independent parallel multiplications

- All relevant arithmetic primitives supported
Data Path Vectorization: Supported Operations

- Generally supported operations on multiple data inputs
  - Scalar A (op) Scalar B
    - Currently supported
  - Scalar A (op) Vector B
    - Scalar value A is used for each element of B
  - Vector A (op) Vector B
    - must have equal vector width

- A or B can be independently real or complex
  - Real value treated as complex, with zero imaginary part

- Output type determined by input vector width and complexity
Data Path Vectorization: Restrictions

- Vector, Complex signals must be homogeneous
  - Elements must be of identical type and complexity
  - For multiple vector input, must have identical vector width
    - Some blocks support vector / scalar combinations

- Some blocks have additional constraints on inputs
  - Mux, Select: data signals must be of identical nature

- Control signals can not be complex
  - But some blocks will support vector types for these

- Some restrictions on available ‘mathematical’ operations
  - e.g. Abs is restricted to ‘element by element’ mode (no sqrt function)

- See Appendix for reference chart on current data path vectorization support status
ModelBus Library

- Bus Slave
- Bus Stimulus
- Bus Stimulus File Reader
- Register Bit (RegBit)
- Register Field (RegField)
- Register Out (RegOut)
- Shared Memory (ShareMem)
Example: demo_regs

Memory Interface

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>MODE[0]</td>
<td>ModeBit0</td>
</tr>
<tr>
<td>0x0000</td>
<td>MODE [1]</td>
<td>ModeBit1</td>
</tr>
<tr>
<td>0x0000</td>
<td>MODE [2]</td>
<td>ModeBit2</td>
</tr>
<tr>
<td>0x0000</td>
<td>MODE [3]</td>
<td>ModeBit3</td>
</tr>
<tr>
<td>0x0008</td>
<td>INDEX[15:0]</td>
<td>Index Register</td>
</tr>
<tr>
<td>0x0009</td>
<td>REGA[15:0]</td>
<td>InputA</td>
</tr>
<tr>
<td>0x000A</td>
<td>REGB[15:0]</td>
<td>InputB</td>
</tr>
<tr>
<td>0x000B</td>
<td>RESULT[0:0]</td>
<td>RegOut</td>
</tr>
<tr>
<td>0x0100</td>
<td>TABLE</td>
<td>LookupTable</td>
</tr>
</tbody>
</table>
Support Resource

DSP Builder Advanced Blockset User Guide:
- Chapter 1 – About DSP Builder Advanced Blockset
  - Base Blocks
  - ModelPrim Blocks
  - ModelIP Blocks
  - ModelBus Blocks
  - FFT Blockset
Designing with DSP Builder Advanced Blockset

Design Rules and Protocols
Cycle Accuracy and Latency

- ModelIP blocks
  - cycle-accurate

- Primitive blocks under subsystem
  - cycle-accurate at the boundaries

- Primitive blocks outside subsystem
  - The boolean logic and delay blocks are cycle-accurate
Protocol for Connecting Blocks

- Unified <data, valid, channel> protocol

Filters Library

- FIR
- CIC

ModelPrim (Primitive) Library

- Primitive Subsystem

Waveform Synthesis Library

© 2010 Altera Corporation—Confidential
ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off.
and Altera marks in and outside the U.S.
Time-Division Multiplexing (TDM) Technique

Clock Rate = Sample Rate

Clock Rate = 2*Sample Rate

TDM_CLK
Parameter Definition

- **ClockRate**
  - Clock Frequency (MHz)
- **SampleRate**
  - Sample Rate per Channel (MSPS)
- **ChanCount**
  - # of Channels
- **Period = floor(ClockRate/SampleRate)**
  - TDM factor, available timeslots
- **ChanWireCount = ceil(ChanCount/Period)**
  - # of channel wires required to carry all the channels
- **ChanCycleCount = ceil(ChanCount/ChanWireCount)**
  - # of carry channels per wire

2 channels with TDM factor = 3

=> ChanWireCount = 1, ChanCycleCount = 2
Vectorized Data I/O

- When ChanCount > Period
  - Multiple wires to accommodate all the channels
- Each ModelIP block will internally vectorize itself to build multiple blocks in parallel

4 channels with TDM factor = 3
ChanWireCount = 2, ChanCycleCount = 2
Vectorized Data I/O

- When ChanCount > Period
  - Multiple wires to accommodate all the channels

4 channels with TDM factor = 3
ChanWireCount = 2, ChanCycleCount = 2
More Complex Example

- **ChanCount > Period + SampleRate Change**

4 channels with TDM factor = 3, Upsample by 2

- Clock Rate = 180 MHz
- ChanCount = 4
- Input SampleRate = 60 MSPS
- Input Period = 3
- ChanWireCount = ceil(4/3) = 2
- ChanCycleCount = ceil(4/2) = 2

- Clock Rate = 180 MHz
- ChanCount = 4
- Output SampleRate = 120 MSPS
- Output Period = floor(180/120) = 1
- ChanWireCount = ceil(4/1) = 4
- ChanCycleCount = ceil(4/4) = 1
Multi-Channel Operation – ModelIP

- System parameterization of # of channels
Connecting ModelIP Blocks

- Connect ModelIP A directly to ModelIP B only if output channel and wire structure of ModelIP A match the input channel and wire structure of ModelIP B.

- Otherwise, some sort of adaptor needs to be implemented.
Displaying Latency for ModelIP Blocks

- Latency added by a ModelIP block can be displayed by adding the `<latency>` parameter as annotation.
- The latency will be displayed after simulation.
Connecting Filters to Complex Mixer

- Application: Digital Up Converter
  - FIR/CIC Output: i, q individual channel
  - Complex Mixer Input: Complex Channel
Connecting Real Mixer to Filter

- Application: Digital Down Converter
  - Real Mixer Output: Complex output
  - FIR/CIC Input: i, q individual channel
Connecting ModelIP Block to ModelPrim Subsystem

- The ModelPrim subsystem must be designed to accept and process the output channel and wire structure of the ModelIP block.
ModelPrim Subsystem: Multi-Channel Operation

- Assuming number of data wire remains one (interleaved data), then can simply increase the delay value by the increased channel count.

4 channels to 20 channels
Displaying Latency for ModelPrim Subsystems

- Latency of the subsystem is automatically displayed in the Chanout block
Support Resource

- DSP Builder Advanced Blockset User Guide:
  - Chapter 1 – About DSP Builder Advanced Blockset
    - Cycle Accuracy and Latency
    - Protocol for Connecting IP

- AN 544: Digital IF Modem Design with DSP Builder Advanced Blockset:
  - Chapter 2 – DSP Builder Advanced Blockset Tool Flow
    - Unified Interface
Exercise 1
Designing with DSP Builder Advanced Blockset

Hardware and Memory-Mapped Interface
Hardware Interface

- Typical ModelIP has four interfaces:
  - Control Signals:
    - Synchronous system clock
      - Also a synchronous bus clock (if different from system clock)
    - Two asynchronous resets
      - One returns internal state machines to initialized states
      - The second one resets bus registers
  - Input/Output Port: data, valid, channel
    - Input data is only written into the core when input valid is asserted
    - Output data is only valid when output valid is asserted
    - Data, valid and channel structure is determined by system parameters (sample rate and system clock rate)
  - Memory-mapped interface:
    - Input: address, write, data and write enable
    - Output: Read data and read valid
Typical FIR Filter Interface

- Single Rate FIR
- System Parameters: ChannelCount = 4, TDM factor = 4
Typical Memory-Mapped Interface

- Write Cycle: Cycle 2 - 6
- Read Cycle: Cycle 10 – 14
- h_rst used to reset memory-mapped registers
- If specified, a separate bus clock can be used
Support Resource

- DSP Builder Advanced Blockset User Guide:
  - Chapter 1 – About DSP Builder Advanced Blockset
    ● Hardware Interface section
Designing with DSP Builder Advanced Blockset

Hardware and Documentation Generation
Model Simulation

- At the start of each simulation run, each block (whether ModelIP block or ModelPrim subsystem) is re-synthesized into an internal representation of the hardware component and written out as VHDL RTL.

- The VHDL RTL are stored in the folder specified in the Control block (default folder: ../rtl).

- Help page for each block is also updated to show specific design documentations describing implementation of the block:
  - e.g. latency, port interface and resource utilization.
Generated Directory Structure and Files

my_project

matlab_design_files

rtl

my_design

Contains generated files for my_design model

ModelIP1

Contains generated files for ModelIP1

... 

ModelIP1

Contains generated files for ModelIP1

ModelPrim subsystem1

Contains generated files for ModelPrim subsystem1

... 

ModelPrim subsystemN

Contains generated files for ModelPrim subsystemN

rtl directory

- XML describing attributes of the model
- XML describing boundaries of the model (when used with standard blockset)

rtl/<model> subdirectory

- ModelSim .do file (to compile, load and simulate design)
- Top level testbench .vhdl file
- Tcl scrip to add the files to existing Quartus II project
- Qip file that contains all necessary files for Quartus II software
- VHDL file for each component in the model
- XML files describing boundaries of subsystems
- XML files describing attributes of subsystems
- XML files containing information about Advanced DSP Builder blocks that are translated to HTML and displayed in MATLAB help viewer

rtl/<model>/<subsystem> subdirectory

- ModelSim .do files for the subsystem (used for automatic testbench flow)
- Intel .hex files used to initiate memories used by subsystem
- .sdc TimeQuest constraint file
- .tcl (only exists for the subsystem containing device) to set up Quartus II project
- <model>_hw.tcl needed for SOPC Builder integration
Generated Documentation: Overview

- Online Help page is automatically generated
- For each design, Help pages are generated for the following:
  - Control block
  - Signal block
  - ModelIP block
  - Scheduler within ModelPrim subsystem
- To see the generated Help page:
  - Right click and select Help
Generated Documentation: *Control Block*

- The generated Help page consists of two sections:
  - All memory-mapped register addresses map
  - Resource usage
Generated Documentation: **Signal Block**

- The generated Help page consists of clock and reset information.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Name</td>
<td>clk</td>
<td>Name of system clock signal</td>
</tr>
<tr>
<td>Clock Rate</td>
<td>245.760000</td>
<td>System Clock Frequency</td>
</tr>
<tr>
<td>Clock Goal</td>
<td>245.760000</td>
<td>Fitter Target Frequency</td>
</tr>
<tr>
<td>Reset Name</td>
<td>areset</td>
<td>Name of system reset signal</td>
</tr>
<tr>
<td>Reset Active</td>
<td>1</td>
<td>Active state of system reset</td>
</tr>
</tbody>
</table>
Generated Documentation: **ModelIP Core**

- The generated help page consists of:
  - Basic information
  - ModelIP parameter settings
  - Port Interface
  - Input/output Data Format (Filters only)
  - Resource Utilization

### ModelIP Core Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpolation Rate</td>
<td>25</td>
<td>The interpolation rate of the filter</td>
</tr>
<tr>
<td>Number of stages</td>
<td>6</td>
<td>Number of integrator and comb sections</td>
</tr>
<tr>
<td>Differential Delay</td>
<td>1</td>
<td>Differential Delay</td>
</tr>
<tr>
<td>Number of channels</td>
<td>16</td>
<td>Number of channels</td>
</tr>
<tr>
<td>Final Decimation</td>
<td>2</td>
<td>Optional decimation on output</td>
</tr>
<tr>
<td>Clock Rate</td>
<td>245.7676 MHz</td>
<td>The frequency of the clock when running in hardware</td>
</tr>
</tbody>
</table>

### Port Interface

- **xln_v**: in, width 1, Qualifying signal to indicate validity of data signals
- **xln_c**: in, width 8, Qualifying signal to indicate channel of data signals
- **xln_0**: in, width 16, Data signal
- **bypass_i**: in, width 1, General Purpose Input

### Input Data Format (Repeats every 50 clock cycles)

```
<c00><c01><c02><c03><c04><c05><c06><c07><c08><c09><c10><c11>
```

### Output Data Format (Repeats every 4 clock cycles)

```
<c00><c01><c02><c03>
<c04><c05><c06><c07>
<c08><c09><c10><c11>
```

### Resource Utilization

<table>
<thead>
<tr>
<th>Component</th>
<th>Type</th>
<th>LUT4s</th>
<th>Mults</th>
<th>Memory bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td></td>
<td>3553</td>
<td>0</td>
<td>2310</td>
</tr>
</tbody>
</table>
Generated Documentation: *ModelPrim Subsystem*

- The generated help page consists of:
  - Basic information
  - Port Interface
  - Resource Utilization
Support Resource

- DSP Builder Advanced Blockset User Guide:
  - Chapter 1 – About DSP Builder Advanced Blockset
    - Model Simulation
    - Generated Files
Designing with DSP Builder Advanced Blockset

Hardware Integration
**System Integration: Assigning Base Addresses**

**IP Cores:**
- How many registers does each IP core need?
- How many words does each register require?
  - Depends on data width of the bus and the width of the parameterizable variables (i.e. FIR coefficient)
  - For example: a register for a 32-bit FIR coefficient would require two words on a 16-bit bus
- Each IP core should be assigned an unique base address
  - Base address (current IP) + registers widths (current IP) < Base address (next IP)

**Registers:**
- Each register should have a unique base address
System Integration: Integrating to a SOPC Builder System

- `<model>_hw.tcl` files generated for each system
  - Include `<model>_hw.tcl` at IP search path
- One slave port and one base address per each DSPB-AB module
- Data ports are exported to the top level as conduit signals
System Integration:
Programming Nios II Processor

- Requirement:
  - SOPC Builder base address (from system.h)
  - Base address for each IP core in Advanced Blockset (from Help documentation)
  - Register offsets for the register-of-interest (from Help documentation)

- Use IOWR/IORD to update registers

To update register for coefficient x:

IOWR(base_addr_SOPC + base_addr_FIR, coef_x_offset, data)
IORD(base_addr_SOPC + base_addr_FIR, coef_x_offset)
Quartus II Project Integration

Create a new Quartus II project:
- Click on Run Quartus II block to generate Quartus II Project
- Generated in the same folder as the .mdl file
- .qpf (project) .qsf (settings) and .qip files have the same name as the subsystem containing Device block

Add to existing Quartus II project:
- Source the .add.tcl file in the subsystem containing Device block
- Alternatively add reference to .qip file in the subsystem
Compilation Report

- Graphical window containing all compilation processing information
  - Resource usage
  - Device pin-out
  - Settings and constraints applied
  - Messages

- Opens automatically when processing begins

**Recommendation:** Go through report for a design to get sense of information being provided

- Information also available as text files in project directory
  - Ex. `<project_name>.fit.rpt` & `<project_name>.map.rpt`
Resource/Timing Verification: Useful Reports

- Analysis & Synthesis Summary
- Fitter Summary
- Fitter Resource Usage Summary
- Fitter Resource Usage by Entity
- Multi-Corner Timing Analysis Summary
- Fmax Summary
Support Resource

- DSP Builder Advanced Blockset User Guide:
  - Chapter 1 – About DSP Builder Advanced Blockset
    - Creating a Quartus Project
    - Comparison with RTL Simulation

- DSP Builder Advanced Blockset Reference Manual:
  - Chapter 2 – Base Library
    - Run ModelSim
Exercise 2
Designing with DSP Builder Advanced Blockset

Design Optimization –
Memory and Multiplier Trade Offs
Memory and Multiplier Thresholds

- DSP Builder tries to balance the implementation between logic elements (LE) and block memories/embedded multipliers.
- The memory and multiplier thresholds can be used to influence these trade-offs.
- The trade-offs can be verified in Quartus II software compilation reports.
Hard Multiplier Threshold

- Determines how multiplications are implemented:
  - Soft multipliers are built out of logic elements
  - Hard multipliers are built using embedded multipliers for Cyclone devices or DSP blocks for Stratix devices

- The threshold is the number of logic elements you are willing to use to save multipliers.
  - For example: If threshold is set to 100, then:
    - 18x18 (~350 LEs) would be built using hard multiplier
    - 16x8 (64 LEs) would be built out of soft LEs

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Implementation</th>
</tr>
</thead>
</table>
| -1        | Default behavior: Use all hard multipliers  
           24x18 -> 18x18 + 18x18 (two hard multipliers) |
| 300       | Keep 18x18 multipliers (~350 LEs) hard  
           24x18 -> 18x18 + 6x18 (hybrid multipliers) |
| 1000      | Keep multipliers soft |
| 10        | 24x18 -> 36x36  
           Not enough to allow adder to combine results of two 18x18 multipliers |
Memory Threshold

- Determines if memories are implemented with registers or one of the embedded memories:
  - Cyclone and Cyclone II Devices: M4K
  - Cyclone III Devices: M9K
  - Stratix and Stratix II and Arria Devices: M512, M4K or M-RAM (512K)
  - Stratix III and IV Devices: MLAB (320-bit or 640-bit in ROM mode), M9K, M144K

- Indicates the maximum memory bits that can be built using registers
  - Threshold of -1 uses default trade-off option

- May not be applicable to all devices and can be overwritten other requirements
  - MLAB gets pushed to M9K with a 300MHz Fmax requirement
  - Delays of length < 3 cycles must be implemented with registers (nature of block RAM)

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDelay RAM Block Threshold</td>
<td>Determines whether registers or M-LABs</td>
</tr>
<tr>
<td></td>
<td>Applicable to delays only</td>
</tr>
<tr>
<td></td>
<td>Default trade-off: 20-bit</td>
</tr>
<tr>
<td>CDual Port RAM Threshold</td>
<td>Determines whether registers or M-RAMs (e.g. M512, M4K)</td>
</tr>
<tr>
<td></td>
<td>Applicable to dual-port memories only</td>
</tr>
<tr>
<td></td>
<td>Default trade-off: 1280-bit</td>
</tr>
<tr>
<td>M-RAM Threshold</td>
<td>Determines whether registers or M-RAMs (e.g. M512, M4K)</td>
</tr>
<tr>
<td></td>
<td>Default trade-off: 1,000,000-bit (essentially never used)</td>
</tr>
</tbody>
</table>
Advanced Scripting:  
Change Parameters on the Control Block

```
% Load the Simulink model
load_system('model_name');

% Get the Control block
control = find_system('model_name','type','block', 'MaskType', ...
    'DSP Builder Advanced Blockset Control Block');

% Get the Control block
if(isempty(control))
    error('The design must contain a Control Block.');
end;

% Example: set the RTL destination directory
dest_dir = ['../rtl', num2str(freq)];
set_param(control{1}, 'destination', dest_dir);
```

Other changeable thresholds:
- distRamThresholdBits
- hardMultiplierThresholdLuts
- mlabThresholdBits
- ramThresholdBits
Advanced Scripting:
*Change Parameters on the Signals Block*

```matlab
% Load the Simulink model
load_system('model_name');

% Get the Signals block
signals = find_system('model_name','type','block', 'MaskType', ..
    'DSP Builder Advanced Blockset Signals Block');

% Get the Signals block
if(isempty(signals))
    error('The design must contain a Signals Block.');
end;

% Example change target clock frequency
fmax_freq = 300.0;
set_param(signals{1},'freq', fmax_freq);
```
Advanced Scripting: Run Hardware Compile

% Run Simulink simulation; which also generates RTL
t = sim('model_name');

% Run Quartus II compilation flow
[success, details] = run_hw_compilation(<model>, '<relative H/W path>');
% where details is a struct containing resource and timing information:
% details.Logics, details.Comb_Aluts, details.Mem_Aluts, details.Reg,
% details.ALM, details.DSP_18bit, details.Mem_Bits, details.M9K,
% details.M144K, details.IO, details.FMax, details.Slack, details.Required,
% details.FMax_unres, details.timingpath, details.dir, details.command,
% details.dir

% view all H/W compilation reports
disp(details)

% view individual compilation reports
disp(details.timingpath)

% Note: unused resources may appear as -1, rather than 0.
Support Resource

- **DSP Builder Advanced Blockset User Guide:**
  - Chapter 1 – About DSP Builder Advanced Blockset
    - Memory and Multiplier Trade-Off Options subsection
      - Base Blocks -> Control Blocks

- **Device handbooks:**
  - Specifically the DSP and memory sections
  - [http://www.altera.com/literature/lit-index.html](http://www.altera.com/literature/lit-index.html)
    - Device Documentation section
Designing with DSP Builder Advanced Blockset

Design Architecture Exploration – TDM Hardware Reuses
TDM Revisited

Clock Rate = Sample Rate

Clock Rate = 2*Sample Rate

© 2010 Altera Corporation—Confidential
ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS & STRATIX are Reg. U.S. Pat. & Tm. Off. and Altera marks in and outside the U.S.
TDM Design Consideration

- **TDM factor = Clock Rate/Sample Rate**
  - TDM factor should be integer

- **ModelIP Block:**
  - TDM hardware re-use is already built-into ModelIP blocks
  - Optimized ModelIP blocks (including TDM hardware re-use) are generated automatically during hardware generation
  - Note: Still keep track of channel/wire structure

- **ModelPrim Subsystem:**
  - Depending on the channel/wire structure, the ModelPrim might be reused via primitive folding or data vectorization
  - Otherwise, the primitive subsystem needs to be redesigned for the new channel/wire structure
## TDM Design: ModelIP Trade-Off

**Example: 49-tap Symmetric Single Rate FIR Filter**

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Resource</th>
<th>LUT4s</th>
<th>Mults</th>
<th>Memory bits</th>
<th>TDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Rate = 72 (MHz) Sample Rate = 72 (MSPS)</td>
<td>2230</td>
<td>25</td>
<td>0</td>
<td></td>
<td>TDM = 1</td>
</tr>
<tr>
<td>Clock Rate = 144 (MHz) Sample Rate = 72 (MSPS)</td>
<td>1701</td>
<td>13</td>
<td>468</td>
<td></td>
<td>TDM = 2</td>
</tr>
<tr>
<td>Clock Rate = 288 (MHz) Sample Rate = 72 (MSPS)</td>
<td>1145</td>
<td>7</td>
<td>504</td>
<td></td>
<td>TDM = 4</td>
</tr>
<tr>
<td>Clock Rate = 72 (MHz) Sample Rate = 36 (MSPS)</td>
<td>1701</td>
<td>13</td>
<td>468</td>
<td></td>
<td>TDM = 2</td>
</tr>
</tbody>
</table>
Primitive TDM Design: Primitive Folding

- Automatic time-sharing of hardware
  - Global optimization based on user specified system constraints (sample rate < clock rate)
  - Naturally related to multi-channel parallel-serial (interleaved) conversion.
Primitive Folding: Usage

- Primarily used when the channels are interleaved or when data sample rate << clock rate
- Enable primitive folding at both ChannelIn and ChannelOut blocks
- Define sample rate and # of used TDM slots
  - Total available slots (folding factor) = Clock rate / sample rate
- Between the ChannelIn and ChannelOut blocks, the tool automatically changes one interleaved data wire into a parallel bus (one wire / channel)
- The parallel bus can be demuxed as different operations can be performed on each channel
  - See CRC design example
Folding Example FIR, 8 reloadable coefficients

\[ \text{Folding Factor} = \frac{\text{Clock Rate}}{\text{Input Sample Rate}} \]
Primitive Folding: Design Consideration

- With folding:
  - Design the system for single channel
  - Tool takes care of resource-reuse details (e.g. data muxing and signal routing etc).

- Without folding:
  - Must design the system for the specific # of channel
  - Need to design data muxing and addressing scheme
**Primitive Folding: Implementation Consideration**

- **With folding:**
  - Enable folding at both ChannelIn and ChannelOut blocks (w/ matching parameters)
  - Disable constrain latency (discussed later)
  - Discrete variable-step solver

- **Without folding:**
  - Make sure folding is turned off for both ChannelIn and ChannelOut blocks
  - Discrete fixed-step solver
**Primitive TDM Design: Data Vectorization**

*Example: Channel Alignment Block*

Parameters: 2 channel data

Desired TDM scenario: Go from TDM factor of 1 to TDM factor of 2

Problem: With primitive folding enabled, the NCO channel and valid are passed into channel alignment block through data ports, thus will be duplicated automatically into a parallel bus!
Primitive TDM Design: Data Vectorization

Example: Channel Alignment Block

User parameter:
TDM factor = 1

User parameter:
TDM factor = 2

- Tool generates dual port RAM that stores parallel (two) data in one slot (2x data width)
- Address line stays constant

- Tool generates dual port RAM that stores one data in one slot (1x data width)
- Address line toggles between two channels
Support Resource

- **DSP Builder Advanced Blockset User Guide:**
  - Chapter 1 – About DSP Builder Advanced Blockset
    - Time Division Multiplexing
  - Appendix B – Primitive Folding

- **AN 544: Digital IF Modem Design with DSP Builder Advanced Blockset:**
  - Chapter 2 – DSP Builder Advanced Blockset Tool Flow
    - Time-Division Multiplexing (TDM) Support
    - Multichannel TDM Operation
Cycle Accuracy and Latency Revisited

- **ModelIP blocks**
  - Cycle-accurate
  - Latency added by a ModelIP block can be displayed by adding the `<latency>` parameter as annotation.
  - The latency will be displayed after simulation.

- **Primitive blocks under subsystem**
  - Cycle-accurate at the boundaries
  - Latency of the subsystem is automatically displayed in the Chanout block after simulation.
Distributed Delays

- ModelPrim blocks within ChannelIn and ChannelOut blocks are NOT cycle-accurate
- Advanced blockset automatically balances delays in valid and channel path to meet fmax requirement
- User inserted simple delays in the datapath are assumed to be part of algorithm, rather than pipelining
  - Non-critical delays might be optimized out by compiler
Latency Constraint

- Set latency constraint for subsystem using Constraint Latency parameter within SynthesisInfo block
  - Can use MATLAB workspace variable or expression
  - Restricted to positive integer
  - The subsystem can contain both ModelIP and ModelPrim blocks
  - Any constraint set by fmax has higher priority than latency constraint
  - Does not work with folding
How to Constrain Latency of ModelIP Blocks?

- Add a primitive subsystem following the ModelIP block
- The primitive subsystem has only ChannelIn, ChannelOut and SynthesisInfo blocks
Latency Parameter

- Command to determine latency of ModelIP blocks or ModelPrim subsystems graphically:
  - `eval(get_param(<full path to block>,'latency'))`

- Latency is calculated in the initialization of the model, at the start of simulation
  - All latencies are calculated at the same stage and thus cannot use latency parameter of one block to constraint another at the same time
How to Use Latency Parameter?

- Command to determine latency of ModelIP blocks or ModelPrim subsystems is:
  - `eval(get_param(<full path to block>,'latency'))`

- Use latency parameter to constrain latency of a ModelIP block:
  - Set the latency constraint in the dummy primitive subsystem latency to (desired ModelIP latency – current ModelIP latency)
Latency and fmax Constraint Conflict

- Default latency constraint implied by specified fmax requirement
- Simulation error if requested latency constraint conflicts with default latency constraint
  - Include difference between requested and default latency constraint
Support Resource

- DSP Builder Advanced Blockset User Guide:
  - Appendix – Latency Management
For more information...
Reference Material

- DSP technology center
  - http://www.altera.com/technology/dsp

- DSP user guide and reference manual

- DSP Advanced Blockset user guide and reference manual

- Application note:
## Instructor-Led Training

With Altera's instructor-led training courses, you can:

- Listen to a lecture from an Altera technical training engineer (instructor)
- Complete hands-on exercises with guidance from an Altera instructor
- Ask questions & receive real-time answers from an Altera instructor
- Each instructor-led class is one or two days in length (8 working hours per day).

## Online Training

With Altera's online training courses, you can:

- Take a course at any time that is convenient for you
- Take a course from the comfort of your home or office (no need to travel as with instructor-led courses)

Each online course will take approximate one to three hours to complete.

http://www.altera.com/training

View training class schedule & register for a class
Altera Technical Support

- Altera DSP literature page:  [http://www.altera.com/literature/megafUNCTIONS/lit-ipdsp.jsp](http://www.altera.com/literature/megafUNCTIONS/lit-ipdsp.jsp)

- Consult Altera applications (factory applications engineers)
  - Hotline:  (800) 800-EPLD (7:00 a.m. - 5:00 p.m. PST)

- Field applications engineers: contact your local Altera sales office

- Receive literature by mail:  (888) 3-ALTERA

- FTP:  [ftp.altera.com](ftp.altera.com)

- World-wide web:  [http://www.altera.com](http://www.altera.com)
  - Use solutions to search for answers to technical problems
  - View design examples
Appendix

MATLAB Reference
MATLAB Overview

- High-level technical computing language
  - Simple C like language
  - Highly efficient for dealings with vectors and matrices
  - Built-in mathematical functions in different disciplines
    - E.g. digital communications, DSP, image and video processing

- Interactive environment for algorithm development
  - 2D and 3D graphing tool for data visualization
# MATLAB Workspace

- Consists of the set of variables built up and stored during a MATLAB session
- Can perform workspace operations using the workspace browser

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clc</code></td>
<td>Clears the command window</td>
</tr>
<tr>
<td><code>clear</code></td>
<td>Removes all variables from memory</td>
</tr>
<tr>
<td><code>clear var1 var2</code></td>
<td>Removes the variables <code>var1</code> and <code>var2</code> from memory</td>
</tr>
<tr>
<td><code>who</code></td>
<td>Lists the variables currently in memory</td>
</tr>
<tr>
<td><code>whos</code></td>
<td>Lists the current variables and sizes</td>
</tr>
<tr>
<td><code>;</code></td>
<td>Suppresses screen printing</td>
</tr>
<tr>
<td><code>...</code></td>
<td>Continues a line</td>
</tr>
<tr>
<td><code>load FILENAME var1 var2*</code></td>
<td>Load all variables stored in <code>FILENAME.mat</code> into current workspace</td>
</tr>
<tr>
<td></td>
<td>Optionally, can choose to load selected variables only</td>
</tr>
<tr>
<td><code>save FILENAME var1* var2*</code></td>
<td>Save all variables in workspace to an external <code>FILENAME.mat</code></td>
</tr>
<tr>
<td></td>
<td>Optionally, can save selected variables only</td>
</tr>
</tbody>
</table>
MATLAB Variables

- No need to declare; simply created by assignment
  - myVar = 1.05; % type double
  - myVar2 = [3 + 5*I, 2+6*i]; % double 1D array
  - myVar3 = sqrt(2).*ones(2,5); % double 2D array
  - myVar4 = 'This is a data string'; % character array

- Numbers are by default type double precision floating point
  - Other supported number datatype:
    - single: single precision floating point
    - uint8, uint16, uint32, uint64: unsigned integer – 8, 16, 32 and 64 bits
    - int8, int16, int32, int64: signed integer – 8, 16, 32 and 64 bits
    - logical: one bit logic type
  - Built-in functions to convert numbers to other datatype:
    - single(x)
    - uint8(x), uint16(x), uint32(x), uint64(x)
    - int8(x), int16(x), int32(x), int64(x)
    - logical(x)
    - fi(x,s,w,f) %convert x to fixed point representation
      - s - signedness (0 for unsigned and 1 for signed)
      - w - word length (bits)
      - f - fractional precision (bits)
MATLAB Arrays

- \( g = [m:q:n] \)
  - One dimension row array/vector
  - \( m \) - starting point
  - \( n \) - ending point
  - \( q \) - incrementing step (optional); by default \( q = 1 \)
  - Examples:
    - \( g = [1:1:10] \)
    - \( g = [100:-10:10]' \)

- Array addressing:
  - Array indices are the row and column numbers in an array
  - Example:
    - \( g = [1 \ 2 \ 3 \ 4; \ 5 \ 6 \ 7 \ 8] \)
    - \( g(1,3) = 3, \ g(2,3) = 7 \)
    - \( g(1,:) = [1 \ 2 \ 3 \ 4], \ g(:,1) = [1; \ 5] \)
# Useful Array Functions (1/2)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| `[u,v,w] = find(A)` | - A – array (1D or 2D)  
- u – row indices when A>0  
- v – col indices when A>0  
- w (optional) – resulting array | find(A)  
[u,v,w] = find(A>5)  
B(A>5) = A(A>5)+5 |
| `length(A)`       | - returns number of element when A is an 1D array  
- returns the larger value of m or n when A is a 2D mxn array | length(A)  
i = 1:length(A)  
A(length(A)-1) |
| `[x,k] = max(A)`  | - k = index (indices) of the largest element in row vector  
- x = largest element when A is an 1 D array  
- x = a row vector containing largest element of each column when A is 2D | max(A)  
A_norm = A./max(A)  
A_norm = A./max(max(A)) |
## Useful Array Functions (2/2)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>([x,k] = \text{min}(A))</td>
<td>(k = \text{index (indices)}) of the min element in row vector (x = \text{min element when } A \text{ is an 1D array} ) (x = \text{a row vector containing min element of each column when } A \text{ is 2D} )</td>
<td>\text{min}(A) (\text{A}<em>\text{norm} = A ./ \text{min}(A)) (\text{A}</em>\text{norm} = A ./ \text{min}(\text{min}(A)))</td>
</tr>
<tr>
<td>(\text{size}(A))</td>
<td>Returns a row vector ([m n]) containing the sizes of the (mxn) array, (A)</td>
<td>\text{size}(A) (B = \text{ones}(\text{size}(A)))</td>
</tr>
<tr>
<td>(\text{sum}(A))</td>
<td>Sums the elements in each column of array (A) and returns a row vector containing sum</td>
<td>\text{sum}(A) (\text{A}_\text{norm} = A ./ \text{sum}(\text{sum}(A)))</td>
</tr>
<tr>
<td>(\text{linespace}(a,b,n))</td>
<td>Creates a row vector of (n) regularly spaced values between (a) and (b)</td>
<td>\text{linespace}(0,1,100)</td>
</tr>
<tr>
<td>(\text{logspace}(a,b,n))</td>
<td>Same as (\text{linespace}), but logarithmically spaced</td>
<td>\text{logspace}(0,1000,3)</td>
</tr>
</tbody>
</table>
# Array Operations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>Form</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Scalar-array addition</td>
<td>$A + b$</td>
<td>$[6,3] + 2 = [8,5]$</td>
</tr>
<tr>
<td>-</td>
<td>Scalar-array subtraction</td>
<td>$A - b$</td>
<td>$[8,5] - 2 = [6,3]$</td>
</tr>
<tr>
<td>+</td>
<td>Array addition</td>
<td>$A + B$</td>
<td>$[6,5] + [4,8] = [10,13]$</td>
</tr>
<tr>
<td>-</td>
<td>Array subtraction</td>
<td>$A - B$</td>
<td>$[6,5] - [4,8] = [2,-3]$</td>
</tr>
<tr>
<td>.*</td>
<td>Array multiplication</td>
<td>$A.*B$</td>
<td>$[3,5] .* [4,8] = [12,40]$</td>
</tr>
<tr>
<td>./</td>
<td>Array right division</td>
<td>$A./B$</td>
<td>$[4,4] ./ [2,8] = [4/2, 4/8]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$= [2, 0.5]$</td>
</tr>
<tr>
<td>.\</td>
<td>Array left division</td>
<td>$A./B$</td>
<td>$[4,4] .\ [2,8] = [4\2, 4\8]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$= [0.5, 2]$</td>
</tr>
<tr>
<td>.^</td>
<td>Array exponentiation</td>
<td>$A.^b$</td>
<td>$[2,4] .^ 2 = [2^2, 4^2]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$= [4, 16]$</td>
</tr>
</tbody>
</table>
## Vectors

<table>
<thead>
<tr>
<th>Command</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>g = [1, 2, 3]</code></td>
<td><code>g = [1 2 3]</code></td>
</tr>
<tr>
<td><code>g = [1 2 3]</code></td>
<td><code>g = [1 2 3]'</code></td>
</tr>
<tr>
<td><code>g = [1; 2; 3]'</code></td>
<td><code>g = [1; 2; 3]</code></td>
</tr>
<tr>
<td><code>g = [1; 2; 3]</code></td>
<td><code>g = [1 2 3]'</code></td>
</tr>
<tr>
<td><code>g = [1, 2, 3]'</code></td>
<td><code>g = [1 2 3]'</code></td>
</tr>
<tr>
<td><code>g = [1 2 3]'</code></td>
<td><code>g = [1 2 3]'</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a = [1, 2, 3]</code></td>
<td><code>g = [1 2 3 4 5 6]</code></td>
</tr>
<tr>
<td><code>b = [4, 5, 6]</code></td>
<td></td>
</tr>
<tr>
<td><code>g = [a, b]</code> or <code>g = [a' b']</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a = 1</code></td>
<td><code>g = [1 2 3]</code></td>
</tr>
<tr>
<td><code>b = [2; 3; 4]</code></td>
<td><code>g = [1 2 3 4]</code></td>
</tr>
<tr>
<td><code>g = [a; b]</code></td>
<td></td>
</tr>
</tbody>
</table>
## Matrices

<table>
<thead>
<tr>
<th>Command</th>
<th>Output</th>
</tr>
</thead>
</table>
| `g = zeros(2,3)`                             | `g = \[
            0 0 0 \\
            0 0 0
          \]`                           |
| `g = ones(2,3)`                              | `g = \[
            1 1 1 \\
            1 1 1
          \]`                           |
| `g = [1, 2, 3; 4, 5, 6]` or `g = [1 2 3; 4 5 6]` | `g = \[
            1 2 3 \\
            4 5 6
          \]`                           |
| `g = [zeros(2,3), ones(2,3)]`                 | `g = \[
            0 0 0 1 1 1 \\
            0 0 0 1 1 1
          \]`                           |
| `g = [zeros(2,3); ones(2,3)]`                 | `g = \[
            0 0 0 \\
            0 0 0 \\
            1 1 1 \\
            1 1 1
          \]`                           |
Plotting Multiple Graphs in MATLAB

- Vectors:
  \[ t = 1:2:99; \quad x1 = 1:50; \quad x2 = 1:4:200; \]

- Plotting multiple graphs at the same time:
  ```matlab
  figure; % create a new figure
  plot(t, x1, 'b+-', t, x2, 'go-.' );
  ```

- Plotting both vectors, one at a time:
  ```matlab
  figure; % create a new figure
  plot(t, x1, 'b+-'); hold on; % hold the graph
  plot(t, x2, 'go-.' );
  hold off; % release the hold
  ```

- Plotting the two vectors, separately, but on the same graph:
  ```matlab
  figure(3); % create a new figure and label it figure 3
  subplot(211); % divide figure window into two sub-windows
  plot(t, x1, 'b+-'); % and activate first sub-window
  subplot(212);
  plot(t, x2, 'go-.' );
  ```
**Plotting in MATLAB**

- **Syntax:** `plot(x1,y1,'s1')`
  - Plotting `y1` vs. `x1` (`x1` and `y1` are vectors of equal dimension)
  - `s1` – optional character string

  Composed from one or all elements from following (default `'b.-'`):  

<table>
<thead>
<tr>
<th>Color</th>
<th>Code</th>
<th>Description</th>
<th>Code</th>
<th>Description</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>b</td>
<td>Point</td>
<td>.</td>
<td>Solid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Green</td>
<td>g</td>
<td>Circle</td>
<td>o</td>
<td>Dotted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>r</td>
<td>X-mark</td>
<td>x</td>
<td>Dash-dotted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cyan</td>
<td>c</td>
<td>Plus</td>
<td>+</td>
<td>Dashed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Magenta</td>
<td>m</td>
<td>Star</td>
<td>*</td>
<td>No line</td>
<td>(none)</td>
<td></td>
</tr>
<tr>
<td>Yellow</td>
<td>y</td>
<td>Circle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Black</td>
<td>k</td>
<td>Square</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>White</td>
<td>w</td>
<td>Diamond</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Triangle (down)</td>
<td>v</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Triangle (left)</td>
<td>&lt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Triangle (right)</td>
<td>&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pentagram</td>
<td>p</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hexagram</td>
<td>h</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Useful MATLAB Plot Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| `figure(n)`      | - n – an integer (optional)  
                  | - Forces the figure(n) to be visible  
                  | - If N/A, then creates new figure          | figure;                               |
|                  |                                                                              |                                              | figure(2);                            |
| `subplot(m,n,p)` | - Splits the active figure window into an array of subwindows (m rows and n columns)  
                  | - Directs subsequent plotting commands to the pth subwindow                  | subplot(211);                          |
|                  |                                                                              |                                              | subplot(325);                         |
| `close`          | Closes the current active figure                                             | close all;                                  |
| `xlabel('text')` | Places text string below x-axis                                             | xlabel('time');                             |
| `ylabel('text')` | Places text string next to y-axis                                           | ylabel('data');                             |
| `title('text')`  | Places text string at top of plot                                           | title('my plot');                           |
| `legend('l1','l2',...)` | Creates a legend using strings l1, l2, ...                                 | legend('data1','data2','data3');            |
| `axis([xmin xmax ymin ymax])` | Sets the minimum and maximum limits of the x- and y-axes                      | axis([0 1000 -60 100]);                     |
| `grid`           | Displays gridlines                                                          | grid on;                                    |
|                  |                                                                              | grid off;                                    |
| `hold`           | Freezes current plot for subsequent plot commands                            | hold on;                                     |
|                  |                                                                              | hold off;                                     |
Structures

- MATLAB arrays with data container, called field
- Field can contain any type of data (e.g., string, double array, logical value)
- Can be used for pseudo OO programming
- Also can be used to import/export data from Simulink
- To create structures:
  - Use `struct` command:
    
    \[ S = \text{struct}('field1',\text{VALUES1},'field2',\text{VALUES2},...) \]
  - Create structures directly:
    
    \[
    \begin{align*}
    S.field1 &= \text{VALUES1} \\
    S.field2 &= \text{VALUES2}
    \end{align*}
    \]
- To create array of structures:
  
  \[
  \begin{align*}
  S(1).field1 &= \text{VALUES1a}; \quad S(1).field2 &= \text{VALUES2a} \\
  S(2).field2 &= \text{VALUES1b}; \quad S(2).field2 &= \text{VALUES2b}
  \end{align*}
  \]
### Helpful Built-In MATLAB Functions

MATLAB comes with a variety of built-in commonly used functions:

- For more information, type `help <function_name>` at MATLAB command prompt.

<table>
<thead>
<tr>
<th>Category</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File I/O</td>
<td><code>fopen()</code></td>
<td>Open file in mode specified</td>
</tr>
<tr>
<td>- Used to read/write from/to file</td>
<td><code>fprintf()</code></td>
<td>Print data to a file using designated format</td>
</tr>
<tr>
<td></td>
<td><code>fclose()</code></td>
<td>Close file</td>
</tr>
<tr>
<td>String/number manipulation</td>
<td><code>num2str()</code></td>
<td>Convert number to string</td>
</tr>
<tr>
<td></td>
<td><code>str2num()</code></td>
<td>Convert string to number</td>
</tr>
<tr>
<td>Random input stimulus generation</td>
<td><code>randn()</code></td>
<td>Generate random sequence (normal distribution 0 mean, standard deviation 1)</td>
</tr>
<tr>
<td></td>
<td><code>rand()</code></td>
<td>Generate random sequence (uniform distribution)</td>
</tr>
<tr>
<td>FIR filter design</td>
<td><code>firpm()</code></td>
<td>FIR equal ripple filter design</td>
</tr>
<tr>
<td></td>
<td><code>firl()</code></td>
<td>FIR filter design using windows method</td>
</tr>
<tr>
<td></td>
<td><code>freqz()</code></td>
<td>Plot frequency response of filter</td>
</tr>
<tr>
<td>Coding</td>
<td><code>convenc()</code></td>
<td>Convolutionally encode binary data</td>
</tr>
<tr>
<td></td>
<td><code>vitdec()</code></td>
<td>Convolutionally decode binary data using Viterbi algorithm</td>
</tr>
<tr>
<td></td>
<td><code>rsenc()</code></td>
<td>Reed-Solomon encoder</td>
</tr>
<tr>
<td></td>
<td><code>rsdec()</code></td>
<td>Reed-Solomon decoder</td>
</tr>
<tr>
<td>Fourier Transform</td>
<td><code>fft()</code></td>
<td>Discrete Fourier Transform</td>
</tr>
<tr>
<td></td>
<td><code>ifft()</code></td>
<td>Inverse Discrete Fourier Transform</td>
</tr>
<tr>
<td></td>
<td><code>fftsift()</code></td>
<td>Shift zero-frequency component to middle of spectrum</td>
</tr>
</tbody>
</table>
MATLAB Script .m File

- External file that contains a series of MATLAB statements
- Commands are executed in order
- Commands within script have access to all variables within workspace
- Variables generated by script become part of workspace
- Any statement following percent sign (%) is a comment
- Can call other scripts and functions
- Invoke the script in Matlab command prompt, other script files or in the .m file editor.
Programming Flow Control (1/2)

- if-else statement

  ```
  if expression
  % statement
  else if expression 2
  % more statement
  else
  % more statement
  end
  ```

- switch statement

  ```
  switch <variable>
  case <value 1>
  % statement 1
  case <value 2>
  % statement 2
  case <value 3>
  % statement 3
  otherwise
  % statement
  end
  ```
Programming Flow Control (2/2)

- **for loop**

```plaintext
for iterator = start_value:increment:end_value
    % increment is optional;
    % default increment by 1
    % more statements
end
```

- **while loop**

```plaintext
while condition
    % statements
    % more statements
end
```
MATLAB .m Files

- MATLAB commands can be entered at command prompt
- Not efficient when commands get complicated
- Solution: use .m files!!
- Two types of .m file:
  - Script .m file:
  - Function .m file:
Function .m files

- Functions:
  - True subprogram that takes input argument(s) and/or output parameter(s)
  - Variables defined and used inside function (not input/output argument) are **local**
    to the function (i.e. NOT seen from workspace)

- General Syntax:
  - Output arguments are:
    - enclosed in “[ ]” and separate by “,” e.g. [data_out, time_out]
    - optional and only needed when function does return something
  - Input arguments are:
    - enclosed in “( )” and separated by “,” e.g. (data_in, time_in)
    - optional; functions without input arguments are legal
  - Function begins with `function` and ends with `end`

```matlab
function [outputArgs] = myFunction(inputArgs)
% myFunction
% this is my first MATLAB function

% more comments
% MATLAB statements
% MATLAB statements
outputArgs = % some more statement
end
```
Function Structure

First line must be function definition

Function name; also the file should have the same name (i.e. average.m)

Output argument

Input argument

Comments

Function body

A blank line within comments; comments before the blank line will show up when asked via `help average`

Note: Comments that follow immediately after the function definition will show up at Matlab with the `help <function name>` command.

```
function avr = average (x)
    %AVERAGE computes the average value of a vector x and returns it in avr
    % Notes: an example of a function
    n = length(x);
    avr = sum(x)/n;
end
```
Support Resource

MATLAB Programming:
- Chapter 2 – Basic MATLAB Concepts
- Chapter 3 – Data Storage and Manipulation
- Chapter 4 – Graphics
- Chapter 5 – M-file Programming
Appendix

Simulink Reference
Simulink

- Hierarchical block diagram design & simulation tool
- Digital, analog/mixed signal & event driven
- Visualize signals
- Integrated with MATLAB
Simulink Library Browser

- Navigate through standard, add-on, custom block libraries
- Create models by dragging blocks from library browser to model file
- To bring up Simulink Library Browser:
  - Type `simulink` at MATLAB command prompt
  - Click on Simulink icon at MATLAB toolbar
  - From MATLAB desktop:
    - Start -> Simulink -> Library Browser
# Model Viewing Shortcuts

<table>
<thead>
<tr>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoom in</td>
<td>r</td>
</tr>
<tr>
<td>Zoom out</td>
<td>v</td>
</tr>
<tr>
<td>Zoom to normal (100%)</td>
<td>1</td>
</tr>
<tr>
<td>Pan left</td>
<td>d or Ctrl + Left Arrow</td>
</tr>
<tr>
<td>Pan right</td>
<td>g or Ctrl + Right Arrow</td>
</tr>
<tr>
<td>Pan up</td>
<td>e or Ctrl + Up Arrow</td>
</tr>
<tr>
<td>Pan down</td>
<td>c or Ctrl + Down Arrow</td>
</tr>
<tr>
<td>Fit selection to screen</td>
<td>f</td>
</tr>
<tr>
<td>Fit diagram to screen</td>
<td>space</td>
</tr>
<tr>
<td>Pan with mouse</td>
<td>Hold down p or q and drag mouse</td>
</tr>
<tr>
<td>Delete selection</td>
<td>Delete or Backspace</td>
</tr>
<tr>
<td>Move selection</td>
<td>Use arrow keys</td>
</tr>
</tbody>
</table>
# Block Editing Shortcuts

<table>
<thead>
<tr>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select one block</td>
<td>Left Mouse Button (LMB)</td>
</tr>
<tr>
<td>Select multiple blocks</td>
<td>Shift + LMB</td>
</tr>
<tr>
<td>Copy block from another window</td>
<td>Drag block</td>
</tr>
<tr>
<td>Move block</td>
<td>Drag block</td>
</tr>
<tr>
<td>Duplicate block</td>
<td>Ctrl + LMB and drag; Or RMB and drag</td>
</tr>
<tr>
<td>Connect blocks</td>
<td>LMB one block and Ctrl + LMB another block</td>
</tr>
<tr>
<td>Caution: Only use when ports match up</td>
<td></td>
</tr>
<tr>
<td>Disconnect block</td>
<td>Shift + drag block</td>
</tr>
<tr>
<td>Open selected subsystem</td>
<td>Enter</td>
</tr>
<tr>
<td>Go to parent of selected subsystem</td>
<td>Esc</td>
</tr>
</tbody>
</table>
## Line Editing Shortcuts

<table>
<thead>
<tr>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select one line</td>
<td>LMB</td>
</tr>
<tr>
<td>Select multiple lines</td>
<td><strong>Shift</strong> + LMB</td>
</tr>
<tr>
<td>Draw branch line</td>
<td><strong>Ctrl</strong> + drag line; Or RMB + drag line;</td>
</tr>
<tr>
<td>Route lines around blocks</td>
<td><strong>Shift</strong> + draw line segments</td>
</tr>
<tr>
<td>Move line segment</td>
<td>Drag segment</td>
</tr>
<tr>
<td>Move vertex</td>
<td>Drag vertex</td>
</tr>
<tr>
<td>Create line segments</td>
<td><strong>Shift</strong> + drag line</td>
</tr>
</tbody>
</table>
## Signal Labeling Shortcuts

<table>
<thead>
<tr>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create signal label</td>
<td>Double-click line and enter text</td>
</tr>
<tr>
<td>Copy signal label</td>
<td><strong>Ctrl</strong> + drag label</td>
</tr>
<tr>
<td>Move signal label</td>
<td>Drag label</td>
</tr>
<tr>
<td>Edit signal label</td>
<td>Click in label and edit</td>
</tr>
<tr>
<td>Delete signal label</td>
<td><strong>Shift</strong> + select label, then press <strong>Delete</strong></td>
</tr>
</tbody>
</table>
# Annotation Editing Shortcuts

<table>
<thead>
<tr>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create annotation</td>
<td>Double-click empty space in diagram and enter text</td>
</tr>
<tr>
<td>Copy annotation</td>
<td>Ctrl + drag annotation</td>
</tr>
<tr>
<td>Move annotation</td>
<td>Drag annotation</td>
</tr>
<tr>
<td>Edit annotation</td>
<td>Click in text and edit</td>
</tr>
<tr>
<td>Delete annotation</td>
<td>Shift + select annotation, then press <strong>Delete</strong></td>
</tr>
</tbody>
</table>
Support Resource

Simulink basics:
- Section 2 – Simulink Software Basics
- Section 3 – Creating a Simulink Model
- Section 4 – Modeling a Dynamic Control System