Auto Layout: Software Design Description (Draft)

Harjot Nijjar

April 5, 2018

Table of Contents

Chapter 1

Introduction

This document is a Software Design Description (SDD) of the Auto Layout Tool. The purpose of the tool is to automatically format any Simulink model to improve its visual layout. This is accomplished by arranging the blocks and lines such that the readability of the model increases.

Chapter 2

GraphPlot Code

2.1 Introduction

With the introduction of MATLAB 2016b, Mathworks allowed users to create directed graphs (graphs with edges and nodes). This allowed for the creation of a graph that could represent the layout of a Simulink model by representing the model using a directed graph. Nodes would be used to represent blocks and edges would be used to represent connections between nodes (blocks). By using directed graphs, it became easier to create a basic layout for a Simulink model and then make changes to it.

2.2 Hierarchy

The function GraphPlotLayout is called by AutoLayout, and it calls other functions pertaining to the creation of the directed graph which represents the system. There are several family of functions that GraphPlotLayout calls which have similar functionality. The family of functions are: Arranging Blocks, Implicit Connections, Directed Graph. The Arranging Blocks family deals with the re-positioning of the blocks in the Simulink model. The Implicit Connections family deals with finding implicit connections in the Simulink model so that they can be added to the directed graph which represents the system. The last family of functions is Directed Graph which deals with the creation and editing of the directed graph.

2.3 Arranging Blocks

This family of functions is responsible for re-arranging and re-positioning the blocks in the model. The functions in this family are:

- isBranching
- arrangeSources
- arrangeSinks

2.3.1 isBranching

Prototype: b = isBranching(lh)

Functionality: Determine if a given line is a branching line

Code Description: The line is determined to be branching if the output of the line it is connected to more than 1 block. The number of outputs is determined by the DstPortHandle property.

2.3.2 arrangeSources

Prototype: [srcs, srcPositions, didMove] = arrangeSources(blk, doMove)

Functionality: Find the sources of a block and reposition its sources to vertically arrange them such that they are ordered with respect to the ports.

Code Description:

2.3.3 arrangeSinks

Prototype: [snks, snkPositions, didMove] = arrangeSinks(blk, doMove)

Functionality: Find the sinks of a block and reposition its sinks to vertically arrange them such that they are ordered with respect to the ports.

Code Description:

2.4 Implicit Connections

This family of functions is responsible for implicit connections throughout the model. The functions in this family are:

- findReadWritesInScope
- findDataStoreMemory
- findVisibilityTag
- findGotoFromsInScope
- findWritesInScope
- findReadsInScope
- findGotosInScope
- findFromsInScope

2.4.1 findReadWritesInScope

Prototype: blockList = findReadWritesInScope(block)

Functionality: Find all Data Store Read and Data Store Write blocks that are associated with a Data Store Memory block.

Code Description: The blocks are found by searching for blocks in the system that have the same DataStoreName property value as the Data Store Memory block. Also, a list of blocks that have the same DataStoreName property as the input Data Store Memory block are ignored by finding all Data Store Memory blocks with the same DataStoreName property and its associated Data Store Write and Data Store Read blocks. Then, all Data Store Write and Data Store Read blocks DataStoreName property value are found and put into an array. This array is compared to the previous array and blocks that are not in both arrays belong to the Data Store Memory block.

2.4.2 findDataStoreMemory

Prototype: mem = findDataStoreMemory(block)

Functionality: Find the assocaited Data Store Memory block for a given Data Store Write or Data Store Read block.

Code Description: The Data Store Memory block is found by finding all Data Story Memory blocks that have the same DataStoreName property as the input block. Then, the Data Store Memory block that corresponds to the input block is found by finding the DataStoreName Data Store Memory block that includes the input block in its scope and is the closest to the input block in terms of system level hierarchy. This is accomplished by comparing the Parent block property string value of the Data Store Memory blocks to the Parent block property string value of the input block.

2.4.3 findWritesInScope

Prototype: writes = findWritesInScope(block)

Functionality: Find all Data Store Write blocks that are associated with a Data Store Read block.

Code Description: The Data Store Write blocks are found by first finding the associated Data Store Memory block for the Data Store Read block, and this is accomplished by calling the function findDataStoreMemory. Then, all Data Store Read and Data Store Write blocks that are associated with the Data Store Memroy block are found and stored in an array by calling the function findReadWritesInScope. Next, an array is created which contains only Data Store Read blocks that share the same DataStoreName property value as the Data Store Memory block. Lastly, the blocks that are not in the two arrays are the Data Store Write blocks that are associated with the input Data Store Read block.

2.4.4 findReadsInScope

Prototype: reads = findReadsInScope(block)

Functionality: Find all Data Store Read blocks that are associated with a Data Store Write block.

Code Description:

2.4.5 findVisibilityTag

Prototype: visBlock = findVisibilityTag(block)

Functionality: Find the Goto Visibility Tag block that is associated with a scoped Goto or From block.

Code Description: The Goto Visibility Tag is found by finding all Goto Visibility Tag in the system that have the same GotoTag property value as the input block and by finding the Goto Visibility Tag that is closest to and above the input block with respect to the system level hierarchy. This is accomplished by comparing the string value of the parent property values for the Goto Visibility Tag blocks and the input block.

2.4.6 findGotoFromsInScope

Prototype: blockList = findGotoFromsInScope(block)

Functionality: Find all Goto and From blocks that are associated with a Goto Tag Visibility block.

Code Description:

2.4.7 findGotosInScope

Prototype: goto = findGotosInScope(block)

Functionality: Find the Goto blok associated with a From block.

Code Description:

2.4.8 findFromsInScope

Prototype: froms = findFromsInScope(block)

Functionality: Find all From blocks that are associated with a Goto block.

Code Description:

2.5 Directed Graph

This family of functions is responsible for the creating and editing of the directed graph which represents the connections of the various block at a system level. The functions in this family are:

- applyNamingConvention
- isdigraph
- plotSimulinkDigraph
- systemToDigraph
- addImplicitEdges

2.5.1 applyNamingConvention

Prototype: name = applyNamingConvention(handle)

Functionality: Modify a name by applying a naming convention to block and port names.

Code Description:

2.5.2 isdigraph

Prototype: tf = isdigraph(d)

Functionality: Returns a boolean value which is true if the input is a digraph, and false if it is not.

Code Description:

2.5.3 plotSimulinkDigraph

Prototype: h = plotSimulinkDigraph(sys, dg)

Functionality: Plot the directed graph that represents the connections between the various blocks in a system. The nodes are plotted such that the sources are to the left and sinks are to the right of a block.

Code Description:

2.5.4 systemToDigraph

Prototype: dg = systemToDigraph(sys)

Functionality: Create a directed graph for a subsystem. Blocks are represented as nodes and the connection between nodes are represented as edges.

Code Description:

2.5.5 addImplicitEdges

Prototype: dg = addImplicitEdges(sys, dg)

Functionality: Add edges to a directed graph to represent the implicit connections in a subsystem.

Code Description:

Chapter 3

Layout Code

3.1 Introduction

After creating a directed graph to represent the subsystem and moving the blocks based on the directed graph, the blocks are then repositioned and resized to improve the layout.

3.2 Hierarchy

The function AutoLayout is the top-level function for the AutoLayout tool. There are several family of functions that AutoLayout calls which have similar functionality. The family of functions are: GraphPlot, Moving Blocks, Block Positioning Calculations, Adjustments, and Text/String.

3.3 GraphPlot

Refer to Chapter 2.

3.4 Moving Blocks

This family of functions is responsible for the creating and editing of the directed graph which represents the connections of the various block at a system level. The functions in this family are:

- moveBlocks

3.4.1 moveBlocks

Prototype: moveBlocks(address, blocks, positions)

Functionality: Move blocks in a subsystem to new positions indicated by the input.

Code Description:

3.4.2 horzAdjustBlocks

Prototype: horzAdjustBlocks(layout, col, x)

Functionality: Hrozontally move the blocks in the layout to the right of a column by 'x' pixels.

Code Description:

3.4.3 vertMoveColumn

Prototype: layout = vertMoveColumn(layout, row, col, y)

Functionality: Vertically move the blocks in the layout downward by 'y' pixels.

Code Description:

3.4.4 vertAlign

Prototype: layout = vertAlign(layout)

Functionality: Align blocks to facilitate the use of straight lines in connections by repositioning blocks vertically. Currently only attempts to align blocks which connect to a single block through an inport/outport.

Code Description:

3.4.5 justifyBlocks

Prototype: layout = justifyBlocks(address, layout, blocks, justifyType)

Functionality: Align blocks to facilitate the use of straight lines in connections by repositioning blocks vertically. Currently only attempts to align blocks which connect to a single block through an in/outport.

Code Description:

3.4.6 updatePortless

Prototype: updatePortless(address, portlessInfo)

Functionality: Reposition the portless blocks in the layout.

Code Description:

3.4.7 handleAnnotations

Prototype: handleAnnotations(layout, portlessInfo, annotations, note_rule)

Functionality: Move annotations in the layout based on the note rule configuration setting.

Code Description:

3.5 Block

This family of functions is responsible for the calculating new block position values. The functions in this family are:

- hasPorts
- getPortlessBlocks
- getPortlessInfo
- getBlocksInfo

3.5.1 hasPorts

Prototype: hasPorts = hasPorts(block)

Functionality: Check if a block has any ports.

Code Description:

3.5.2 getPortlessBlocks

Prototype: portlessBlocks = getPortlessBlocks(blocks)

Functionality: Find blocks which have no ports from a list of blocks.

Code Description:

3.5.3 getPortlessInfo

Prototype: [portlessInfo, smallOrLargeHalf] = getPortlessInfo(portless_rule, systemBlocks, portlessBlocks)

Functionality: Get the names of the blocks which are portless and information about their final positioning in the layout.

Code Description:

3.5.4 getBlocksInfo

Prototype: blocksInfo = getBlocksInfo(sys)

Functionality: Find the relative layout of the blocks in a subsystem.

Code Description:

3.5.5 getRelativeLayout

Prototype: [layout] = getRelativeLayout(blocksInfo)

Functionality: Create an array of struct which contains a block's name and its current position.

Code Description:

3.5.6 rectCenter

Prototype: [x,y] = rectCenter(positions)

Functionality: Find the coordinates of the center of a rectangle.

Code Description:

3.5.7 getBlockSidePositions

Prototype: sidePositions = getBlockSidePositions(blocks, side)

Functionality: Find the positions all blocks in the layout.

Code Description:

3.5.8 sortRelativeLayout

Prototype: grid = sortRelativeLayout(grid, colLengths)

Functionality: Sort the blocks in the columns of the grid by descending top positions (top block to lowest block).

Code Description:

3.5.9 repositionPortlessBlocks

Prototype: portlessInfo = repositionPortlessBlocks(portlessInfo, layout, portlessRule, smallOrLargeHalf, sortPortless)

Functionality: Determine the new positions for the portless blocks in the layout to the side in the layout specified by the portless rule configuration setting.

Code Description:

3.5.10 sideExtremes

Prototype: [leftBound, topBound, rightBound, botBound] = sideExtremes(layout, portlessInfo, ignorePortlessBlocks)

Functionality: Find the bounding positions of the layout.

Code Description:

3.6 Adjustments

This family of functions is responsible for making adjustments to the layout. The functions in this family are:

- makeSumsRectangular
- getAutoLayoutConfig
- updateLayout
- redraw lines

3.6.1 getAutoLayoutConfig

Prototype: getAutoLayoutConfig(parameter, default)

Functionality: Get parameters from the tool configuration file to change the behaviour of the Auto Layout tool.

Code Description:

3.6.2 makeSumsRectangular

Prototype: makeSumsRectangular(blocks)

Functionality: Change the shape of a sum block to rectangular.

Code Description:

3.6.3 updateLayout

Prototype: updateLayout(address, layout)

Functionality: Updates the layout by moving blocks to their new calculated positions.

Code Description:

3.6.4 redraw_lines

Prototype: redraw_lines(name, varargin)

Functionality: Updates the layout by moving blocks to their new calculated positions.

Code Description:

3.6.5 adjustForPorts

Prototype: layout = adjustForPorts(layout)

Functionality: Adjust the top and bottom positions to resize blocks to accomodate their ports without disturbing the relative layout.

Code Description:

3.6.6 dimIncreaseForPorts

Prototype: [pos, yIncrease] = dimIncreaseForPorts(block, pos, varargin)

Functionality: Determine the amount to increase the top and bottom positions of a block to accomodate its ports within it.

Code Description:

3.6.7 desiredHeightForPorts

Prototype: desiredHeight = desiredHeightForPorts(block, varargin)

Functionality: Determine a new height for the block to accomodate for the ports within it.

Code Description:

3.6.8 fixSizeOfBlocks

Prototype: layout = fixSizeOfBlocks(layout)

Functionality: Set the sizes of inports, outports, bus creators, bus selectors, mux and demux blocks to their default sizes.

Code Description:

3.7 Text/String

This family of functions is responsible for calculating the required space for blocks based on the text inside the block. The functions in this family are:

- resizeBlocks
- adjustForText
- dimIncreaseForText

3.7.1 resizeBlocks

Prototype: [layout, portlessInfo] = resizeBlocks(layout, portlessInfo)

Functionality: Calculate new block sizes based on the text inside the blocks.

Code Description:

3.7.2 adjustForText

Prototype: layout = adjustForText(layout)

Functionality: Adjust blocks in a subsystem by increasing the width of the blocks to fit the text inside the blocks, while calculating new positions to avoid blocks overlapping each other.

Code Description:

3.7.3 dimIncreaseForText

Prototype: [pos, xIncrease] = dimIncreaseForText(block, pos, varargin)

Functionality: Determine the amount to increase the right and left positions of a block in order to fit the entire text within the block.

Code Description:

3.7.4 getBlockTextWidth

Prototype: neededWidth = getBlockTextWidth(block)

Functionality: Determine an appropriate block width in order to fit the text within it based on the type of block

Code Description:

3.7.5 blockStringDims

Prototype: [height, width] = blockStringDims(block, string)

Functionality: Determine the height and width that a string would have within a block based on the font style and size.

Code Description:

3.7.6 getTextDims

Prototype: dims = getTextDims(string, fontName, fontSize, varargin)

Functionality: Determine the height and width that a string would have within a block based on the font style and size.

Code Description: