Contents

1 Expression to BEAST converter

Entry point is method «convert».

1.1 Variables:

- idCounter – used to generate new id for devices by incrementing each time;
- devices – stores all created devices;
- connectors – stores all created connectors;
- joints – stores current lowest input or toggle that goes from inputs, each time new lower joint appears overwrites old value;
- maxDepth – stores max depth of tree;
- elementsHeightCounter – used to calculate y of current device, increments when creating new device on same level;
- inputsEndpointX – amount of space taken by inputs;

1.2 Constants:

1) INPUTS_START_POINT_X
2) INPUTS _START_POINT_Y
3) DISTANCE_BETWEEN_LOGIC_ELEMENTS_Y
4) DISTANCE_BETWEEN_LOGIC_ELEMENTS_X
5) DISTANCE_BETWEEN_ROOT_AND_LED
6) DEVICES_START_POINT_Y
7) DISTANCE_BETWEEN_ INPUTS _X
8) DISTANCE_BETWEEN_ INPUTS _AND_LOGIC_ELEMENTS_X

This constants are described in figure 1.1, 1.2.

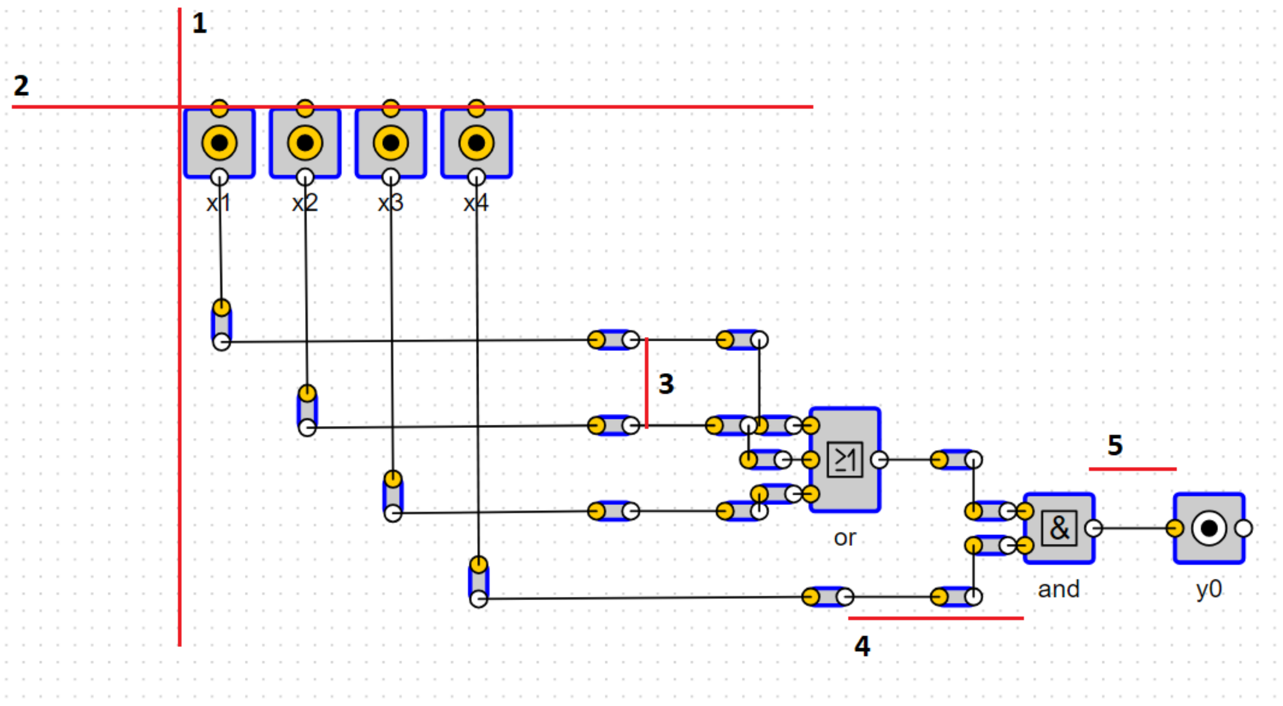


Figure 1.1 – Constants

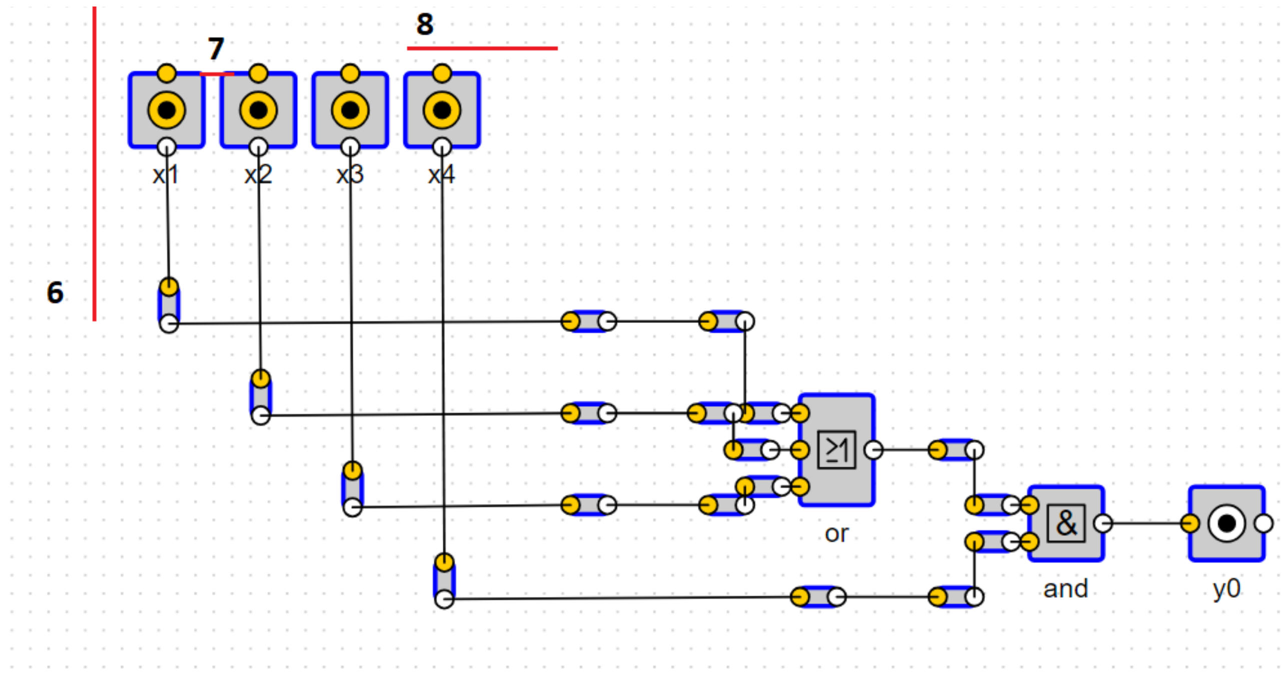


Figure 1.2 – Constants

1.3 Methods:

1.3.1 convert(projectName, expression)

Arguments:

“projectName”	is a name of a project.
«expression»	is a string of type «x1 + x2x3/x4».

Functionality:

Converts expression to json tree. Draws all devices

Return:

«Circuit» for BEAST.

1.3.2 findAllVariables(tree)

Arguments:

«tree»	json tree that came from expression

Functionality:

Get all unique variables in tree.

Example: x1+x2x1 will return (1,2);

Return:

Set of unique variables.

1.3.3 createInputs(variables)

Arguments:

“Variables”	unique variables from tree

Functionality:

For each variable there will be created and drawn an «input» device.

Note: all input devices store in global «joints» variable;

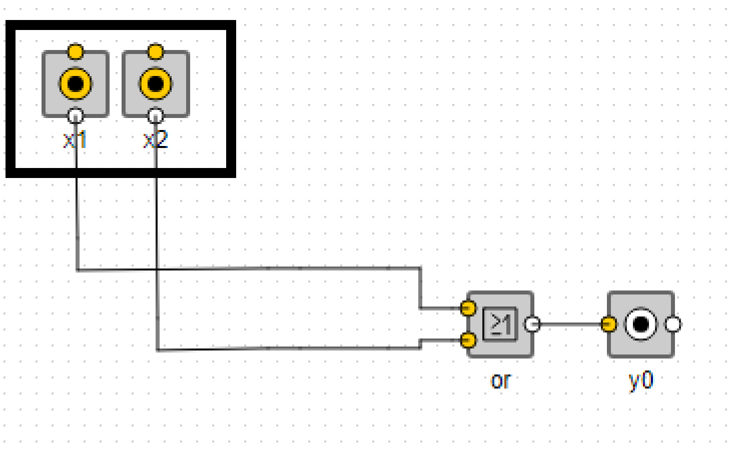


Figure 1.3 – Inputs

1.3.4 getMaxDepth(tree)

Arguments:

“Tree”	json tree that came from expression

Functionality:

Method goes recursively inside tree to get max depth.

Return: distance from root to deepest leave.

1.3.5 buildCircuitObject(projectName)

Arguments:

“projectName”	string for project name

Functionality:

Builds an output object later to be returned to BEAST

1.3.6 createDevices(tree)

Arguments:

“Tree”	json tree that came from expression

Functionality:

Method is called to draw all devices and joints except inputs and outputs.

Calls two helpers methods processVariable and processDevice;

Return: root of tree

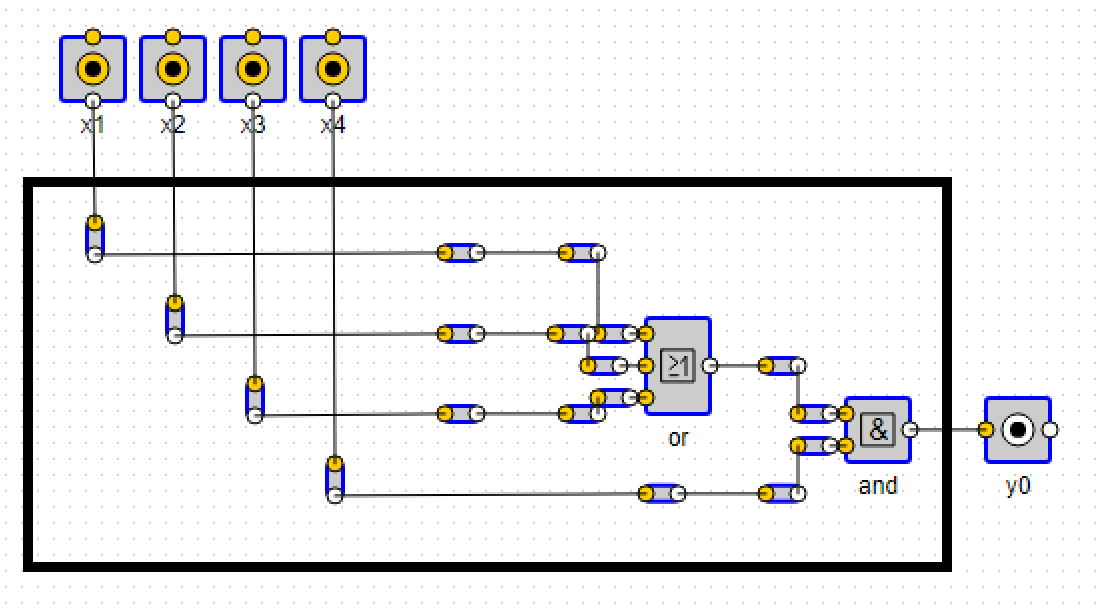


Figure 1.4 – Devices

1.3.7 processVariable(tree, depth)

Arguments:

“tree”	current node of tree
“depth”	current depth of tree

Functionality:

1. Creates nearest joint to input that later will be connected with actual devices. Their «x» position related to depth of node.

2. Extends line from input to bottom with calling method extendBottomLineFromInput.

3. Connects joints from step 1 and 2.

Figure 1.5 shows what method draws. Joints in black box are created in step 1 and returned by method. Joints in red box generated by method extendBottomLineFromInput – step2.

Input number is obtained from tree node.

Return: joint that later will be connected with actual devices.

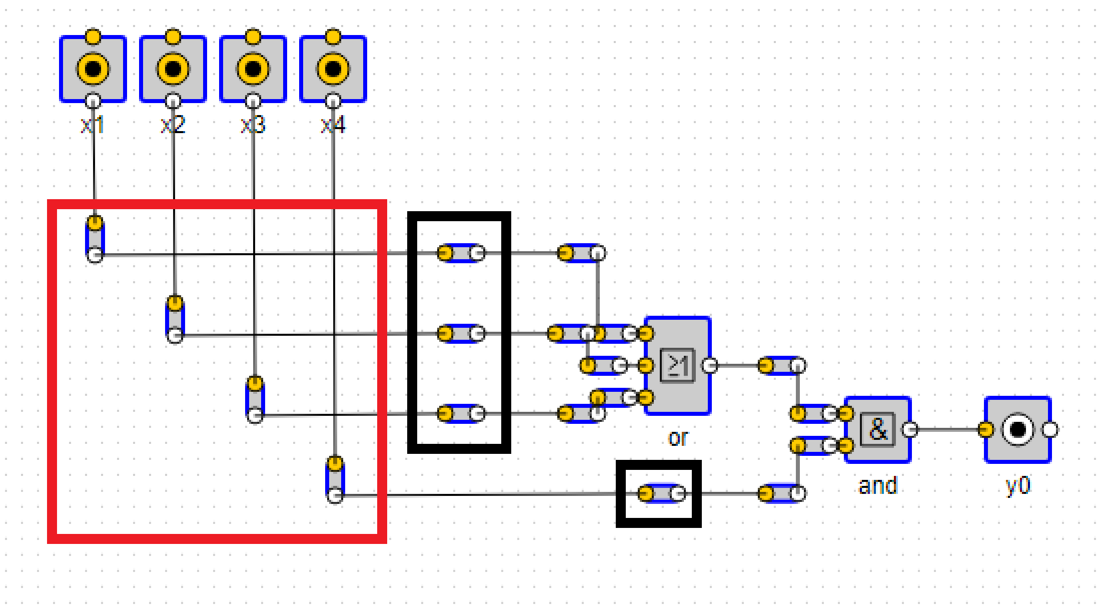


Figure 1.5 – processVariables functionaliy

1.3.8 extendBottomLineFromInput(variable, id, y)

Arguments:

“variable”	key to get input of this variable.
“id”	id of right joint (black box in image above)
“y”	y of right joint.

Functionality:

Extends inputs with help of joints to bottom. Red box in image above.

And globally remembers the lowest joint in «joints» array.

1.3.9 processDevice(tree, depth)

Arguments:

“tree“	current node of tree
“depth“	current depth of tree

Functionality:

1. Creates all actual devices («or», «and», etc)

2. Connects them with their children (connectChildrenWithParent method).

Notes: parent «y» calculated as a middle between «y» of his highest and lowest child.

Return: device created with current node.

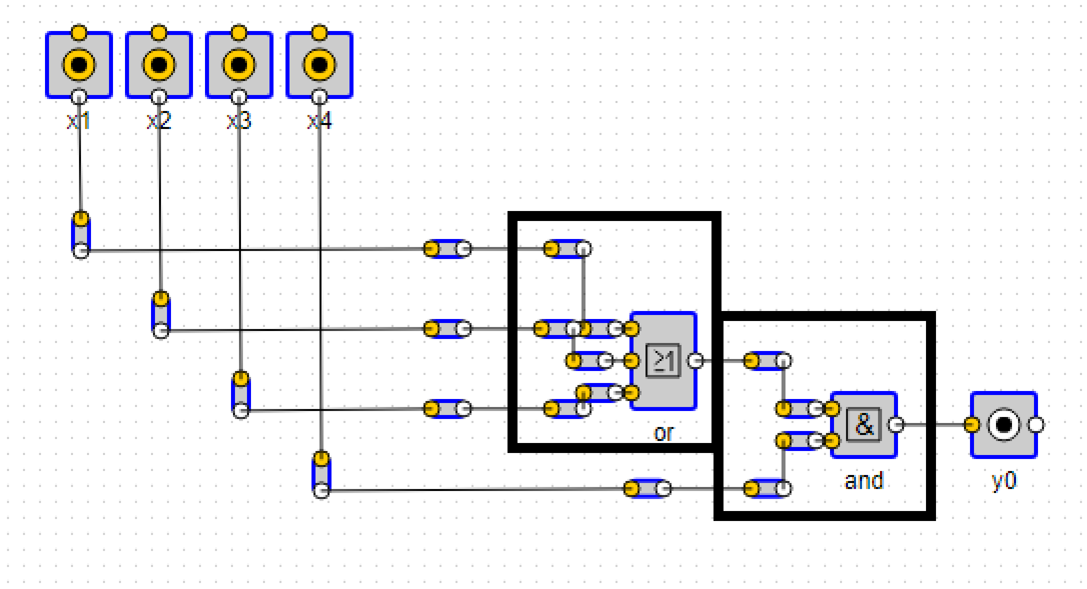


Figure 1.6 – Process single device

1.3.10 calculateNumberOfChildrenHigherThanPort(children, device, distanceBetweenPorts)

Arguments:

”children”	children devices of current device
”device”	parent devise itself
”distanceBetweenPorts”	distance between two ports of parent device

Functionality:

For each parent port calculates if child output port is higher than corresponding parent input port.

Explanation is on figure 1.7. In red box are children. In black boxes are parents input ports. Method calculates amount of such children.

Return: number of children higher than port.

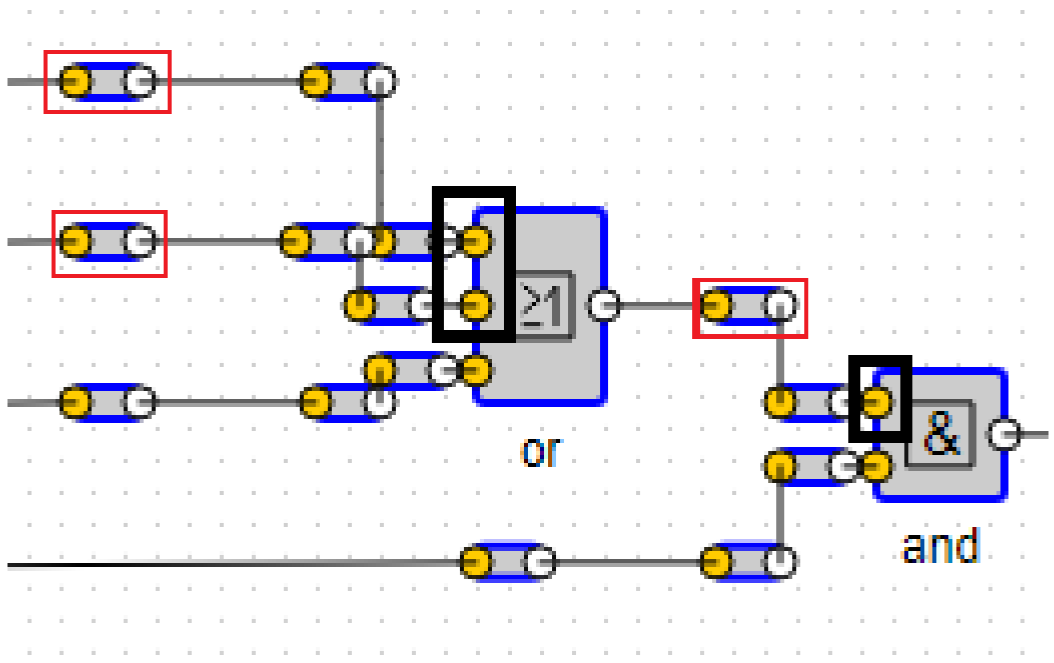


Figure 1.7 – Devices and it»s childrens

1.3.11 connectChildrenWithParent(parent, children, indexFrom, indexTo, isReversed)

Arguments:

”parent”	parent device
”children”	all children devices of parent
”indexFrom”	starting index
”indexTo”	end index
”isReversed”	order of connecting

Functionality:

Conecting children from «indexFrom» in to «indexTo» in normal or reversed order.

If parent has only one input, then we directly connect child to parent.

If there more than one input, next steps are:

1. Creates joint in some distance to right from child.
2. Create joint below first joint.
3. Connects to parent.

How it looks can be seen on figure 1.8. In red box joint from step 1, in black box joint from step 2.

Note: currently method called two times, first for children that have output port higher than corresponding parent input port, second for left children.

“isReversed” used to calculate «x» position of joint right.

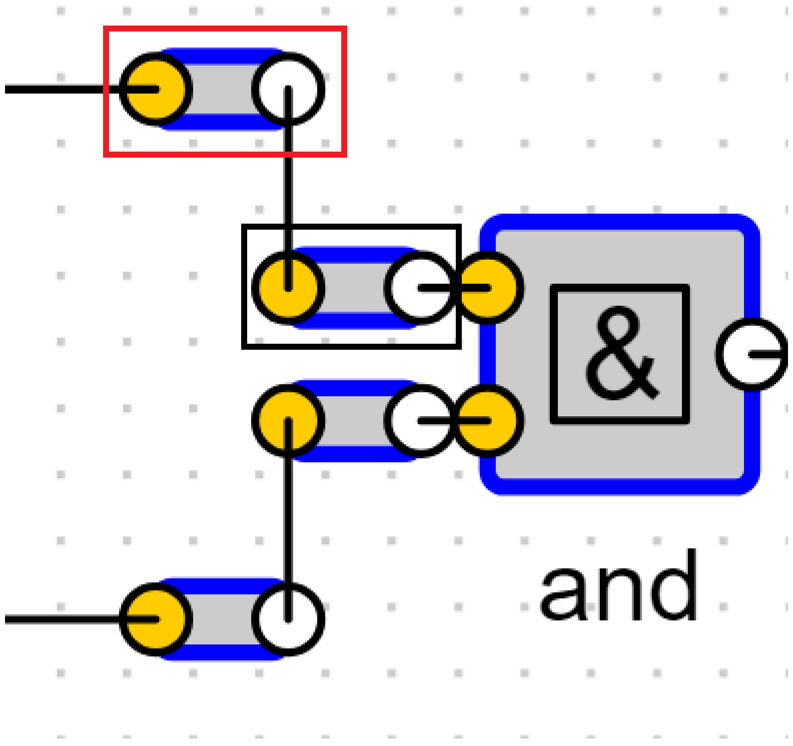


Figure 1.8 – Connections from children to parent

1.3.12 addOutputToRoot(root)

Arguments:

”root” - root element of json tree.

Functionality:

Draws output to right of root element.

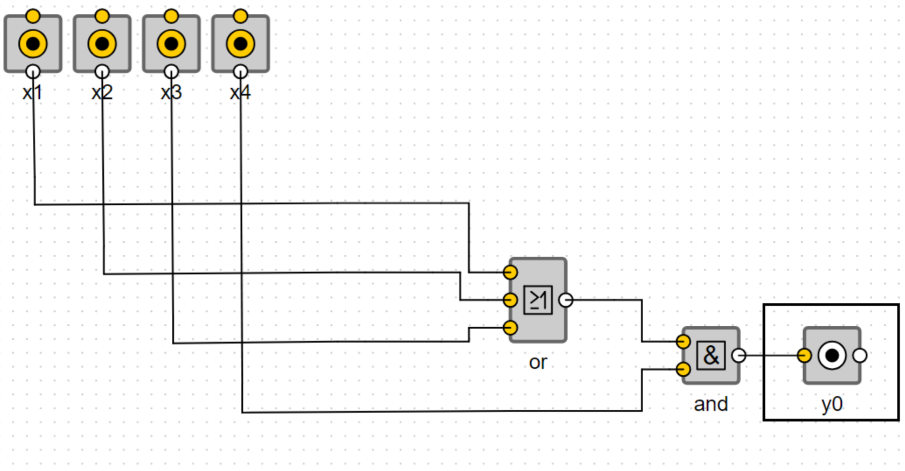


Figure 1.9 - Output

2 Finite state machine sequence to BEAST converter

Entry point is method «convert».

2.1 Variables

- inputsEndpointX = 0;
- maxDepth – show max depth of all trees;
- deviceId – used to generate id;
- devices – all created devices are stored here, later this goes to circuit;
- connectors - all created connectors are stored here, later this goes to circuit;
- elementsHeightCounterZ - used to calculate y of current device, increments when creating new device on same level for states;
- elementsHeightCounterY - used to calculate y of current device, increments when creating new device on same level for outputs;
- yDistanceFromInputs - y distance from inputs to top z joints;
- dffOutputCounter – used to increase x positon for each new dff output;
- topJointCounter – used to increase y position for each new top joint from dff outputs;
- jointsFromInputs – arrays that stores current joints that extends from inputs;
- firstBackZ0PositionX – stores x position of the first output of dff, it»s the position of joint from output of «z0»;
- topToBottomRightZJoints – map, for each z variable, stores position of most top, least top, most bottom and least bottom joints for dff output joint at the moment;
- zVariablesCount – stores number of unique z variables;
- topToBottomRightXJoints – array used to extend for each right x variable line;
- prevOscJoint – stores previous oscillator joint, overrides when next one created;
- isMealy – Boolean, it become true if it’s a Mealy machine;

- showXOnRightWhenMoore – flag, can be changed. If true will show x event if it»s a Moore machine;

2.2 Constants

1) INPUTS_START_POINT_X;

2) INPUTS_START_POINT_Y;

3) INPUTS_NOT_MARGIN_X;

4) INPUTS_NOT_MARGIN_Y;

5) DISTANCE_BETWEEN_INPUTS_X;

6) JOINT_Y_INDENT – depending on angle sometimes we need some more indent;

7) BASE_DISTANCE_BETWEEN_INPUTS_AND_LOGIC_ELEMENTS_X;

8) DISTANCE_BETWEEN_LOGIC_ELEMENTS_Y;

9) BASE_DISTANCE_BETWEEN_LOGIC_ELEMENTS_X;

10) DISTANCE_BETWEEN_LINES_Y;

11) DISTANCE_BETWEEN_LINES_X;

Constants are shown on figure 2.1, 2.2.

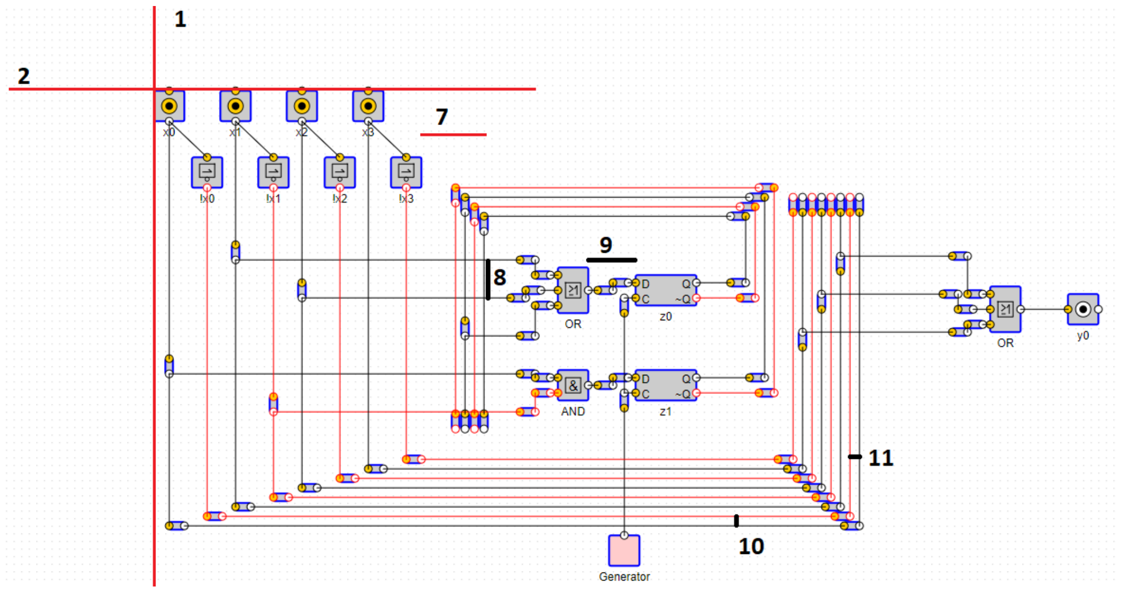


Figure 2.1 - Constants

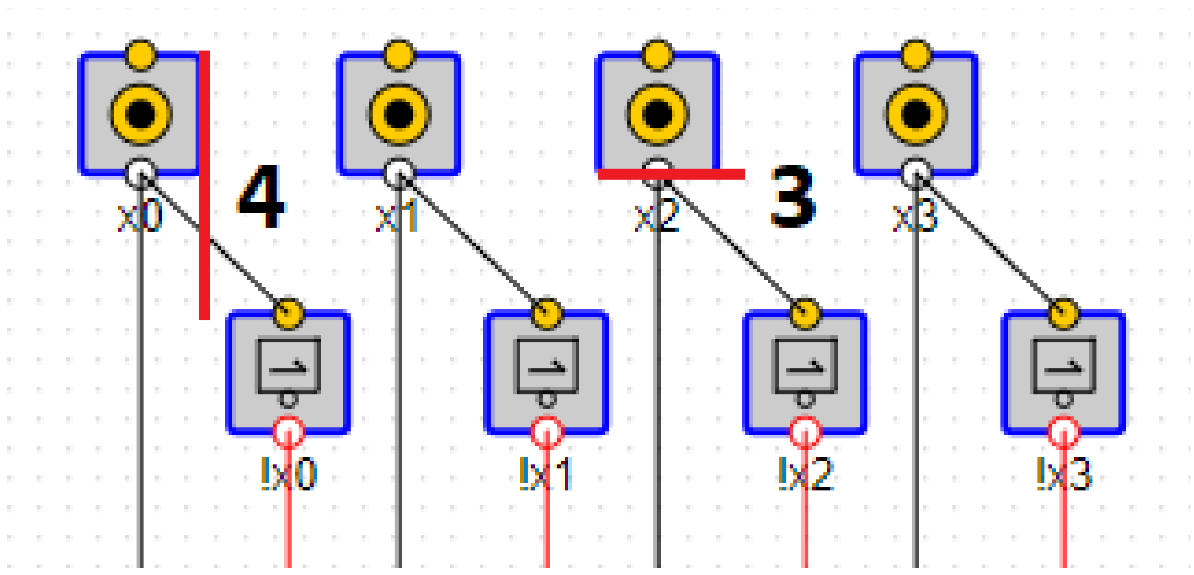


Figure 2.2 - Constants

2.3 Methods

2.3.1 convert(projectName, expressions)

Arguments:

“projectName”	is a name of a project.
“expressions”	is an object of type {y: {«y0»:»x1+x2»}, z: {«z0»: «x1+x2»}}.

Functionality:

Converts expressions to BEAST json. Draws all devices

Return:

”Circuit” for BEAST.

2.3.2 checkExpressions(expressions)

Arguments:

«expressions»	is an object of type {y: {“y0”:”x1+x2”}, z: {“z0”: “x1+x2”}}.

Functionality:

Checks if expressions are using correct variables. So if there exists only «z» expression and some other expression trying to use «z1» error will occurred.

2.3.3 areAllOperandsCorrect(possibleZ, item)

Arguments:

«possibleZ»	all used «z» variables in all expressions.
«item»	expression to check.

Functionaliy:

Checks if expression is using correct variables.

2.3.4 createZTrees(expressions, result)

Arguments:

«expressions»	is an object of type {y: {«y0»:»x1+x2»}, z: {«z0»: «x1+x2»}}.
«result»	variables that stores all global results.

Functionality:

Builds all the left side of application, shown on figure 2.3.

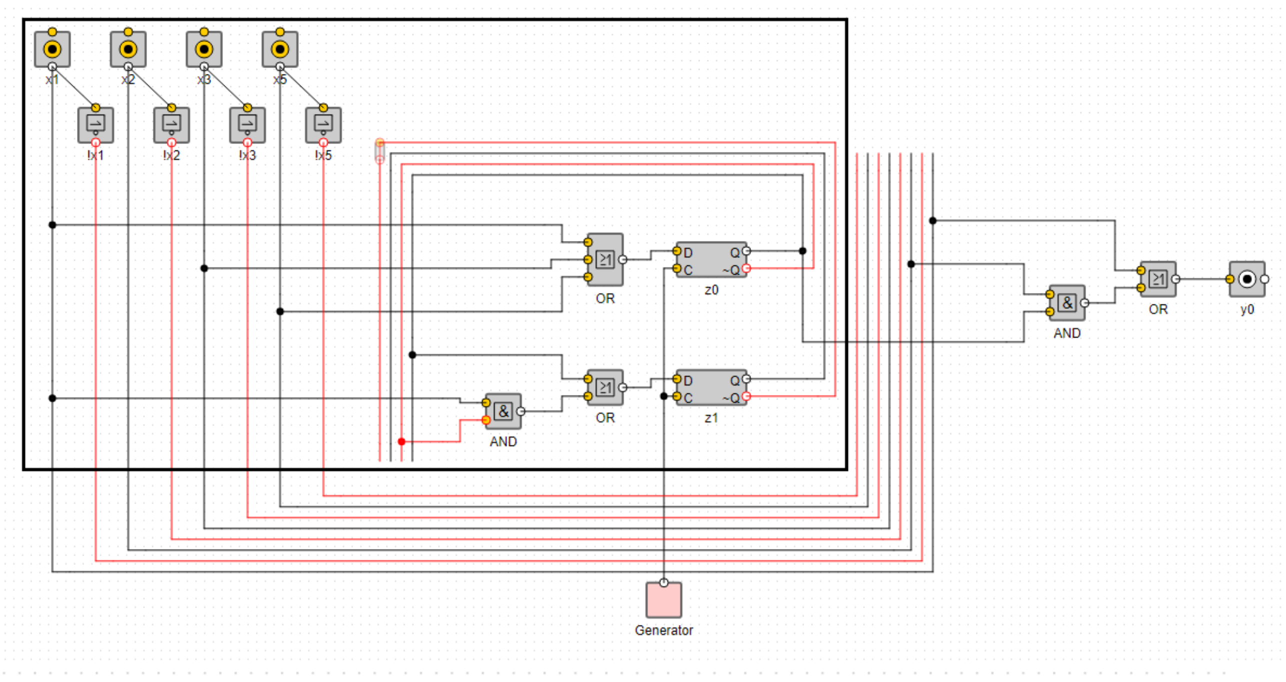


Figure 2.3 – Responsibility of createZTress function

2.3.5 buildCircuitObject(projectName)

Arguments:

«projectName»	string for project name.

Functionality:

Builds an output object later to be returned to BEAST

2.3.6 buildTree(expression)

Arguments:

«expression»	single expression to build tree

Functionality:

Converts expression to json tree with a help of «sane-expression-parser». Because «sane-expression-parser» understands only «x» variables.

1. Function maps all variables to «x» (for example «z0» -> «x0»)
2. Converts to tree
3. Maps variables back («x0» -> «z0»). Function backVariablesToOriginalNames().

Return: json tree of expression.

2.3.7 backVariablesToOriginalNames(node, reverseMap)

Arguments:

«node»	current node in a tree.
«reverseMap»	map with original names.

Functionality:

Goes recursively through tree and change variable names to old original ones.

2.3.8 removeVariablesNots(node)

Arguments:

«node»	current node of tree.

Functionality:

By default when converting to tree «NOT» looks like this. «/x2» -> {not:{variable:x2}}.

This method changes «{not:{variable:x2}}» to «{variable:x2, not: true}.

2.3.9 getVariables(variableLetter, node, variables)

Arguments:

«variableLetter»	letter of variable: «x», «z».
«node»	current tree node.

«variables» - array to push variable that were found

Functionality:

Get all unique variables in tree and push them in array.

Example: x1+x2x1 will return (1,2);

Return:

Set of unique variables.

2.3.10 createInputs(variables)

Arguments:

«variables»	unique variables from tree

Functionality:

For each variable there will be created and drawn an «input» device and «not» device. Method creates part in black box on figure 2.4.

Note: all input devices store in global «jointsFrominputs» variable;

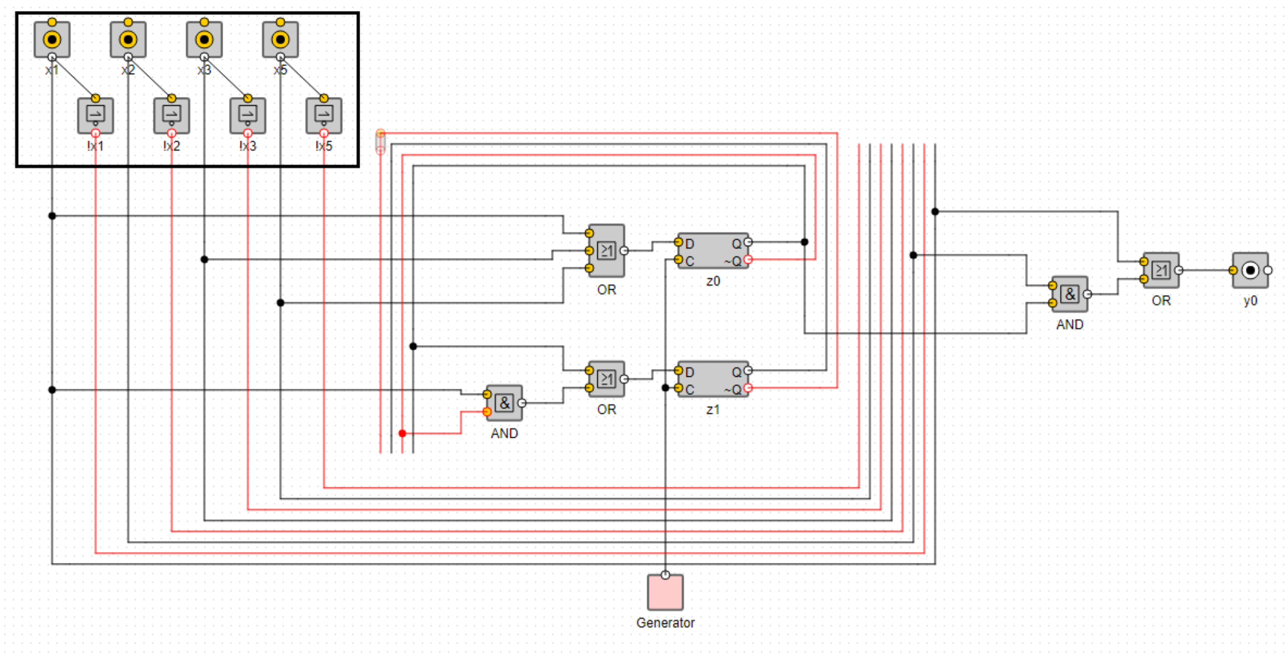


Figure 2.4 – Inputs

2.3.11 createLeftTopZJoints(zVariables)

Arguments:

«zVariables»	all «z» variables

Functionality:

Create left top z joints. It’s in black box shown on figure 2.5.

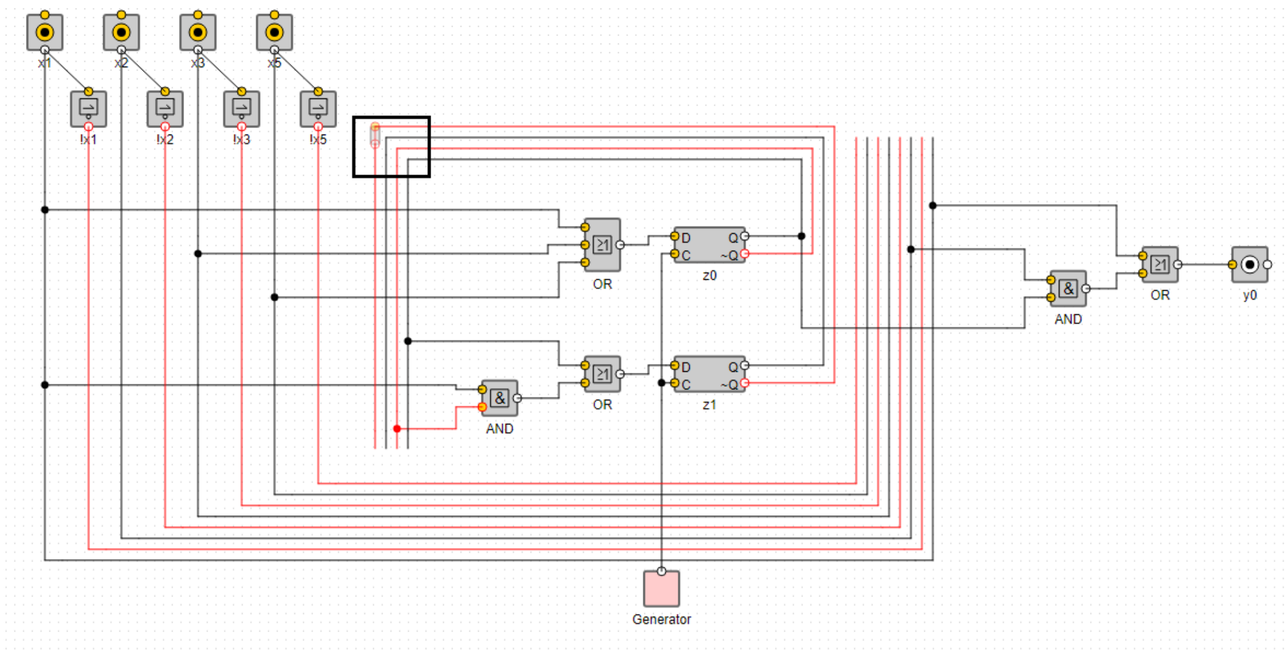


Figure 2.5 – Left top Z joints

2.3.12 calculateYDistanceFromInputs(count)

Arguments:

«count»	number of z variables.

Functionality:

Calculates y distance from inputs to top z joints.

2.3.13 calculateMaxChildrenForEachLevel(tree, maxChildrenForEachLevel)

Arguments:

«tree»	json tree.
«maxChildrenForEachLevel»	array to push into.

Functionality:

On each level depth function finds one parent that has most children and push it to array.

2.3.14 calculateXWidthForEachLevel(maxChildrenForEachLevel, maxDepth)

Arguments:

«maxChildrenForEachLevel»	array of max amount of children for single parent
«maxDepth»	max depth of all left trees

Functionality:

Depending on «maxChildrenForEachLevel» function calculates the width for each level between parent and children

2.3.15 getMaxDepth(tree)

Arguments:

«tree»	json tree that came from expression

Functionality:

Method goes recursively inside tree to get max depth.

Return: distance from root to deepest leave.

2.3.16 drawTrees(tree, xWidthForEachLevel, maxDepth)

Arguments:

«tree»	tree to draw
«xWidthForEachLevel»	width between parent and child for each level
«maxDepth»	max depth of tree

Functionality:

Calls inner method to draw a left side devices. Shown on figure 2.6.

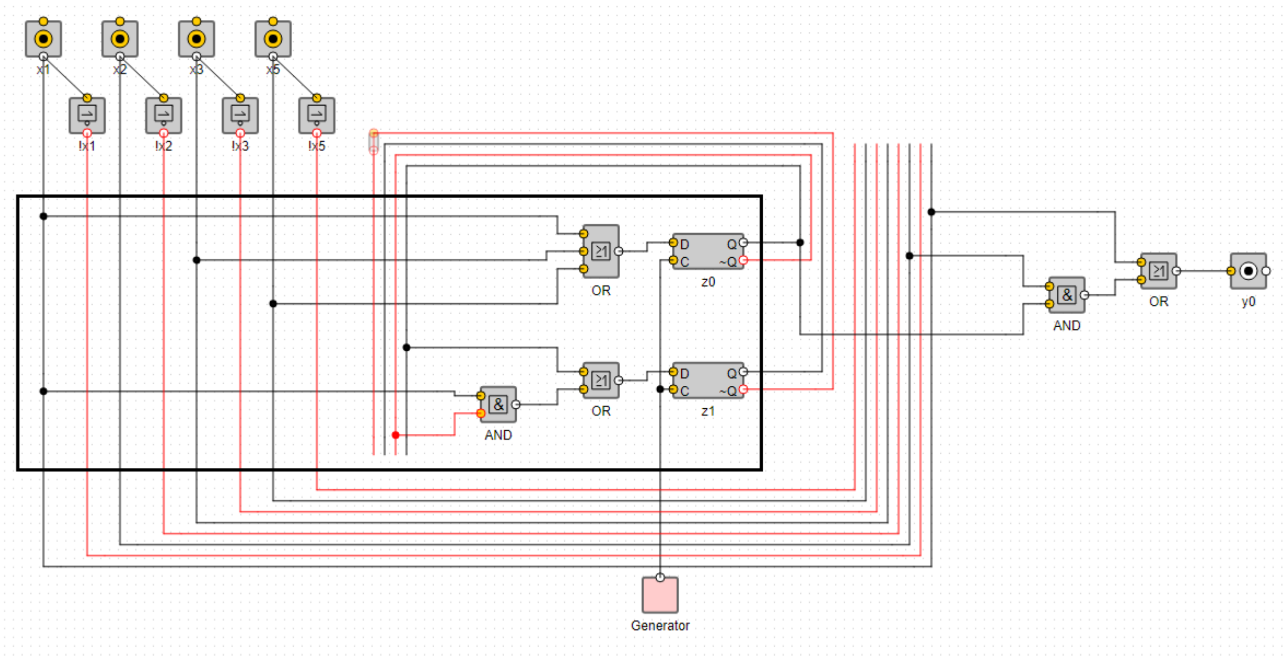


Figure 2.6 – Function drawTrees

2.3.17 extendOscJoints(dff)

Arguments:

«dff»	dff device

Functionality: Extends line from all «dff» to bottom to oscillator.

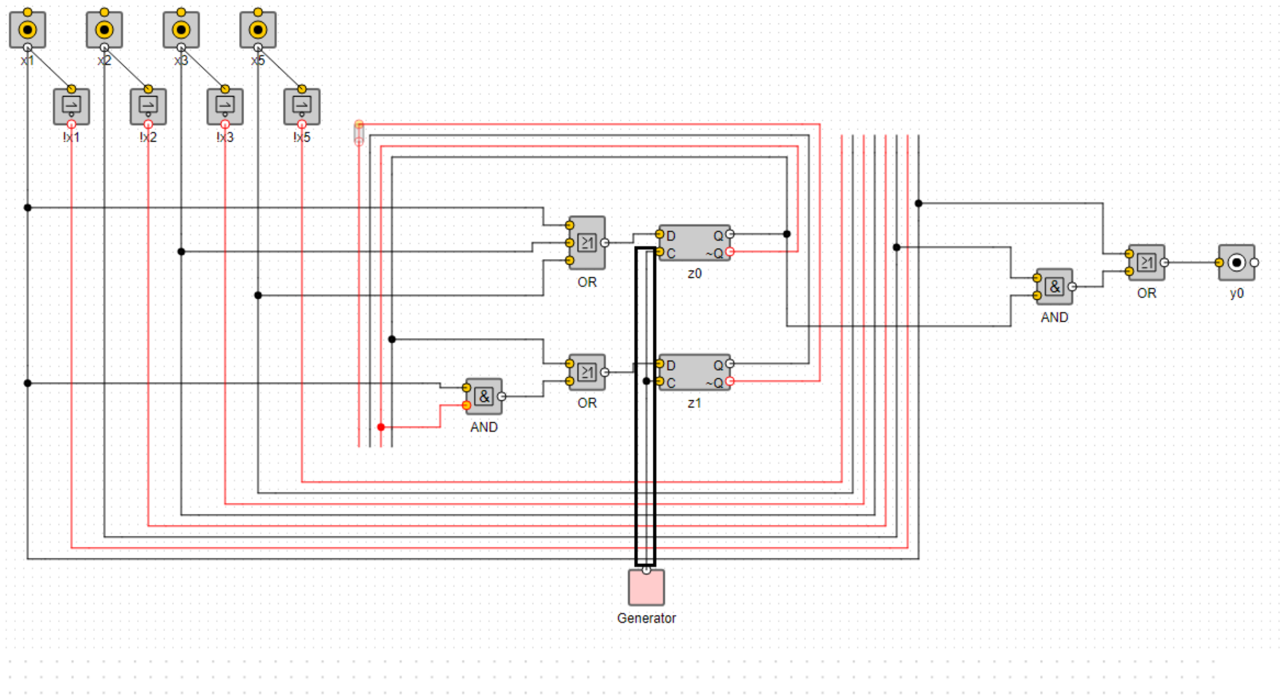


Figure 2.7 – Oscillators connectors

2.3.18 createDevices(tree, xWidthForEachLevel, depth)

Arguments:

«tree»	tree to draw
«xWidthForEachLevel»	width between parent and child for each level
«maxDepth»	max depth of tree

Functionality: Calls inner methods to draw devices.

2.3.19 processVariable(tree, depth)

Arguments:

«tree»	current node of tree
«depth»	current depth of tree

Functionality:

1. Creates nearest joint to input that later will be connected with actual devices. Their «x» position related to depth of node.

2. Extends line from input to bottom with calling method extendBottomLineFromInput. Method do it for both «x» and «z» variables.

3. Connects joints from step 1 and 2.

Input number is obtained from tree node.

Return: joint that later will be connected with actual devices

.

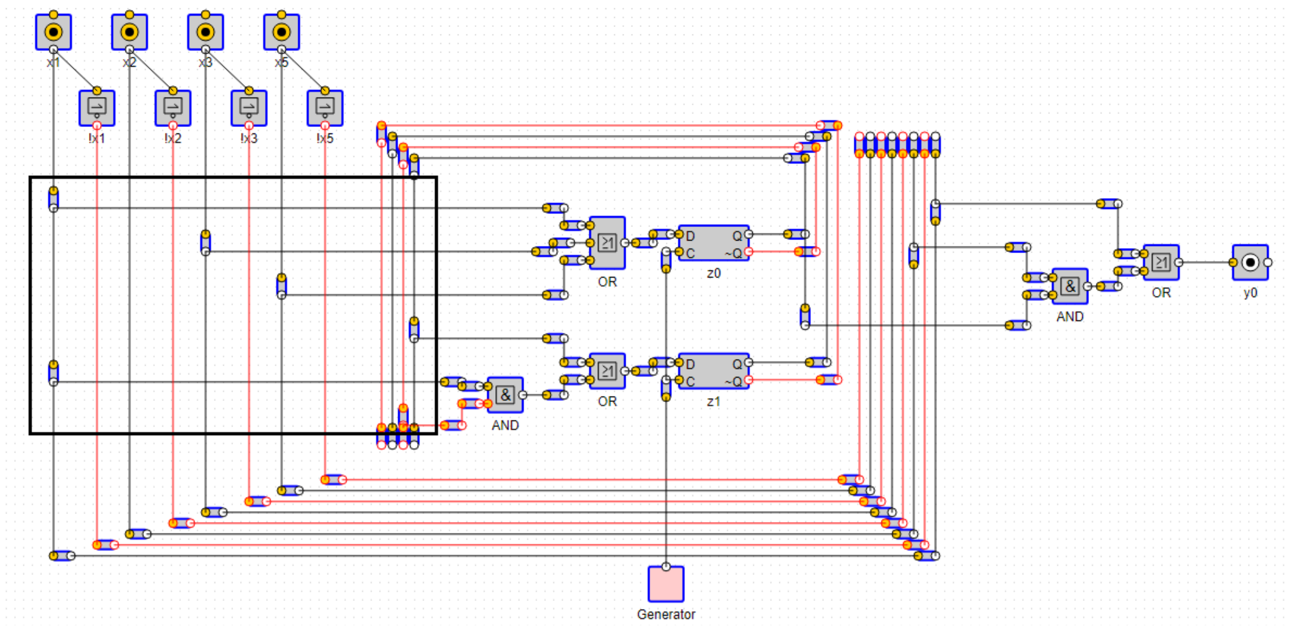


Figure 2.8 – Processing variables

2.3.20 extendBottomLineFromInput(variable, id, y)

Arguments:

«variable»	key to get input of this variable.
«id»	id of right joint (black box in image above)
«y»	y of right joint.

Functionality:

Extends inputs with help of joints to bottom. Red box in image above.

And globally remembers the lowest joint in «joints» array.

2.3.21 prcoessDevicesFactory(param)

Arguments:

«param»	x or z letter

Functionality: Decides what method of «createDevices» to call.

2.3.22 processDevice(node, distanceBetweenChildAndParent, param, optimizeJoints, depth)

Arguments:

«node» current node of tree

«distanceBetweenChildAndParent» distance between child and parent for each level

«optimizeJoints» flag to show if connection between parent and child should be optimized. Use 2 joints instead of 3.

«depth» - current depth of node

Functionality: Draws current device and his connection to children

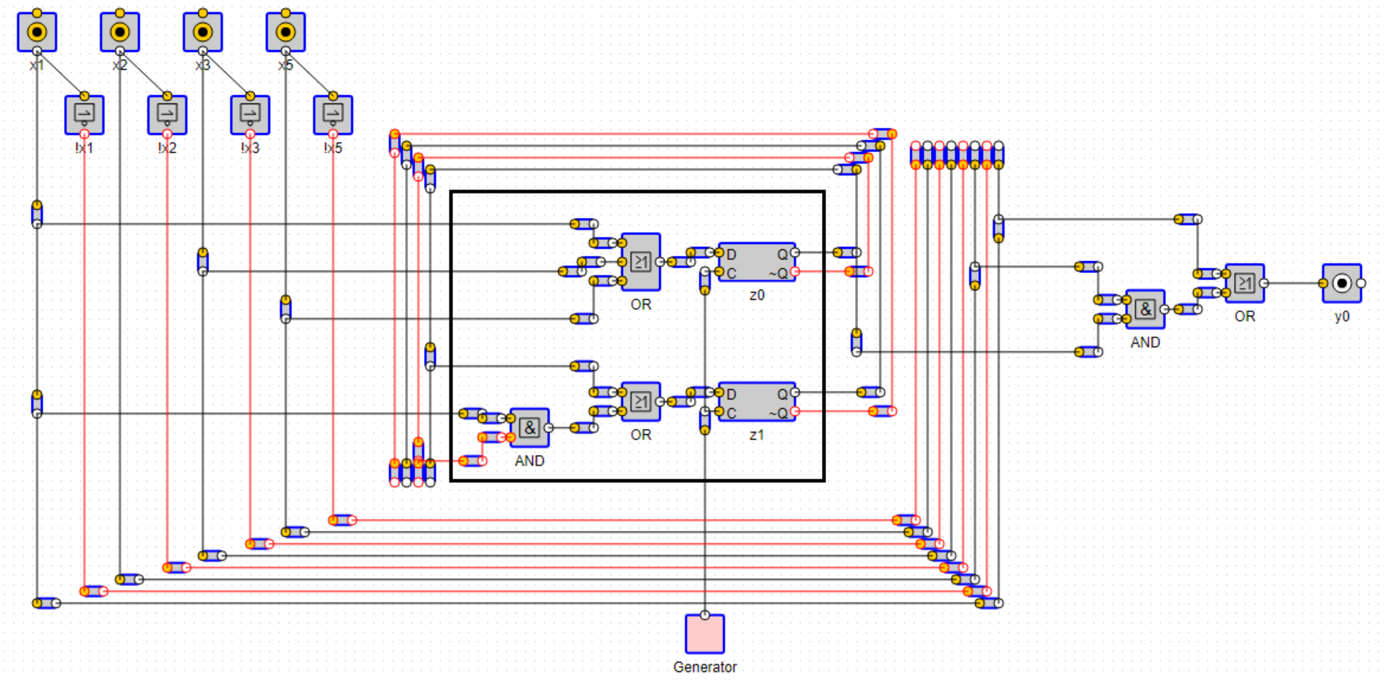


Figure 2.9 – Processing devices

2.3.23 calculateNumberOfChildrenHigherThanPort(children, device, distanceBetweenParents)

Arguments:

«children»	children devices of current device
«device»	parent devise itself
«distanceBetweenPorts»	distance between two ports of parent device

Functionality:

For each parent port calculates if child output port is higher than corresponding parent input port.

Return: number of children higher than port.

2.3.24 connectChildrenWithParent(parent, children, indexFrom, indexTo, options)

Arguments:

«parent»	parent device
«children»	all children devices of parent
«indexFrom»	starting index
«indexTo»	end index
«options»	possible options: isReversed, optimizeJoints

Functionality:

Conecting children from «indexFrom» in to «indexTo» in normal or reversed order.

If parent has only one input, then we directly connect child to parent.

If there more than one input, next steps are:

1. Creates joint in some distance to right from child.
2. Create joint below first joint.
3. Connects to parent.

If optimizeJoints is true and child is joint

1. Instead of creating a new one child moves to right
2. Create joint below first joint.
3. Connects to parent.

Note: currently method called two times, first for children that have output port higher than corresponding parent input port, second for left children.

isReversed used to calculate «x» position of joint right.

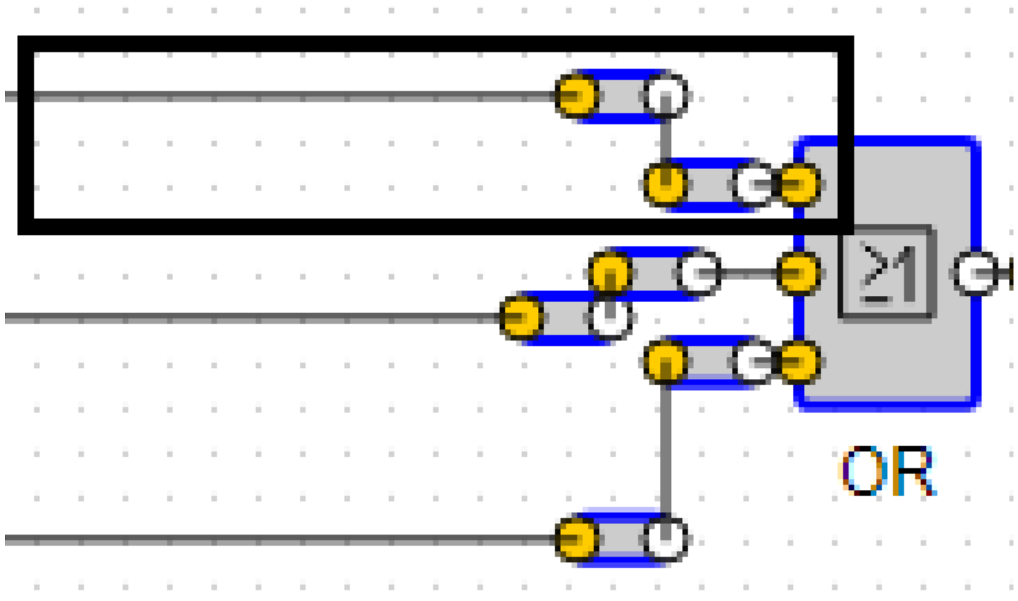


Figure 2.10 – Children with optimizing joints

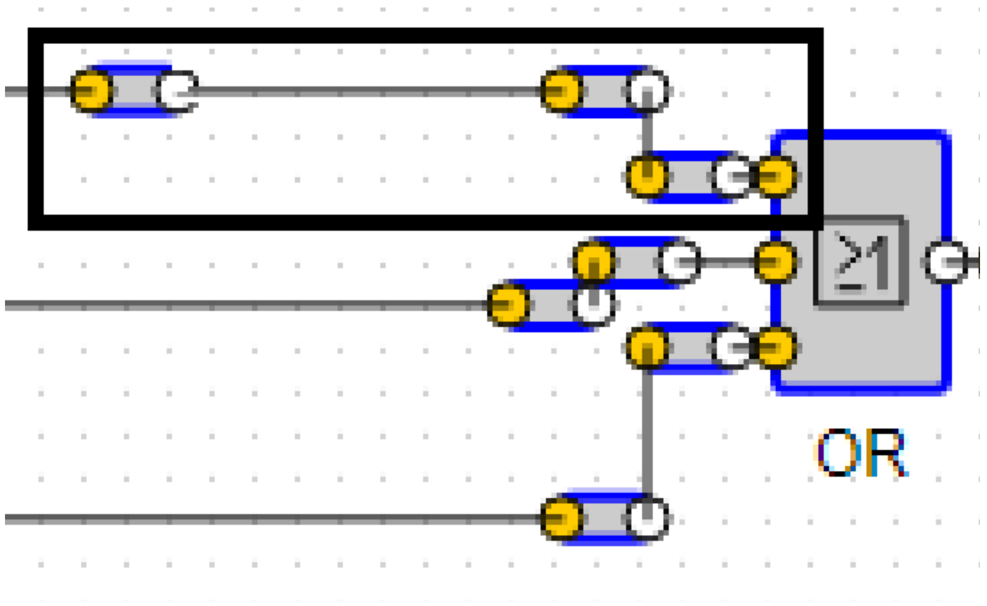


Figure 2.11 – Children without optimizing joints

2.3.25 extendLeftZVariablesToBottom()

Functionality: Extends left «z» variables to bottom point.

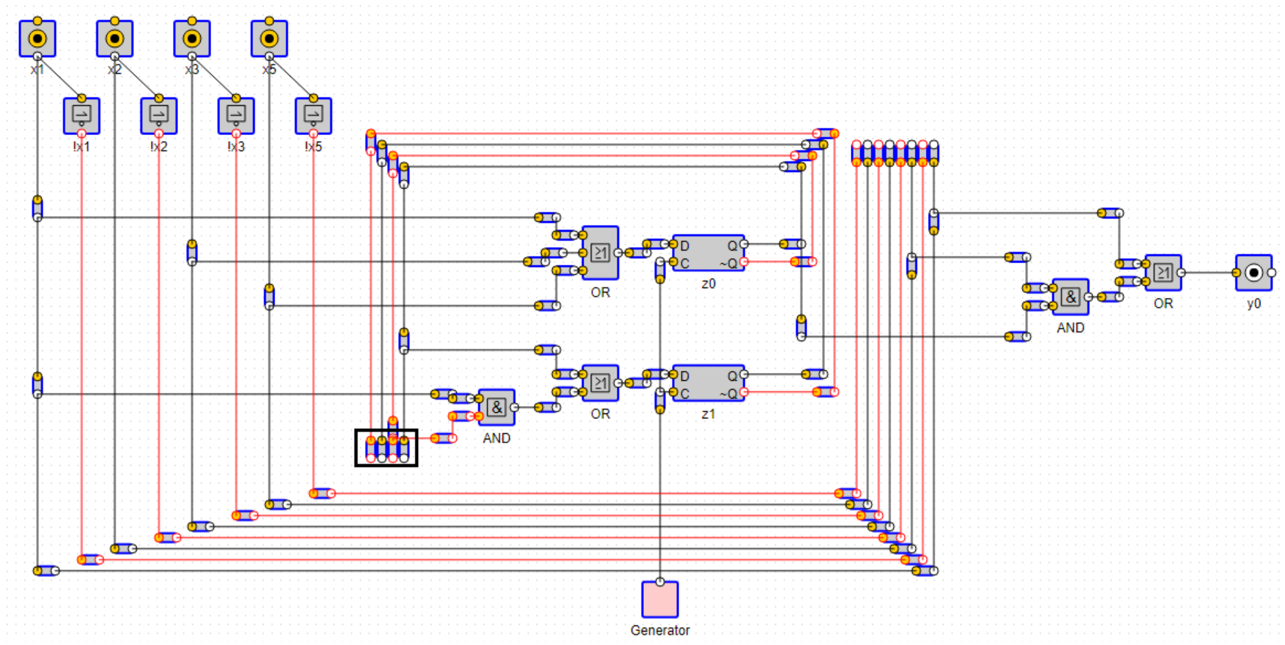


Figure 2.12 – Bottom point of left Z variables

2.3.26 createOutputJointsForDffs()

Functionality: Finds all «dff» and calls createOutputJointsForSingleDff() for each of them.

2.3.27 createOutputJointsForSingleDff(dffDevice, dffOutputName)

«dffDevice»	dff device
«dffOutputNam»	dff device name, usually it»s variable name like «z0»

Funcnionality: Create joints for both of dff outputs and their top joints.

Outputs joints and top joints are not connected at this time.

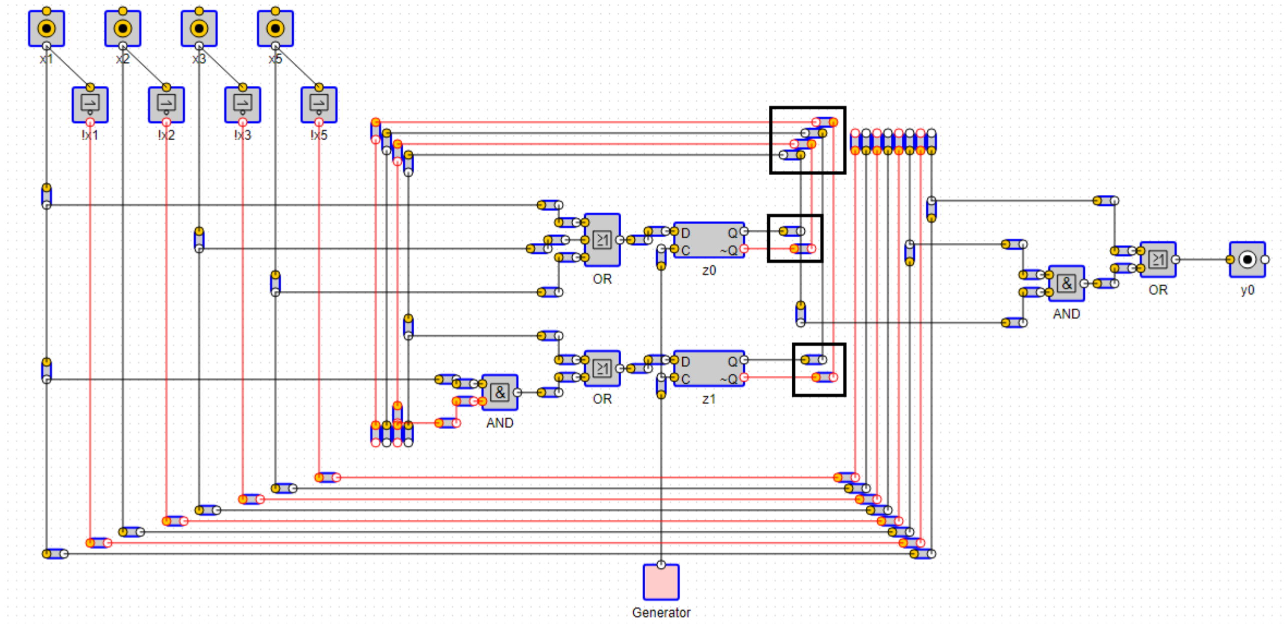


Figure 2.13 – Output joints of dff devices

2.3.28 calculateYForTopJoints(topJointCounter)

Arguments:

«topJointCounter»	current index of variable from top to bottom

Functionality: calculate y for top «z» joints.

2.3.29 extendTopRightZJointsToLeft(rightTopJoints, leftTopJoints)

Arguments:

«rightTopJoints»	right top z joints
«leftTopJoints»	left top z joints

Functionality: Connects right top z joints with left z joints.

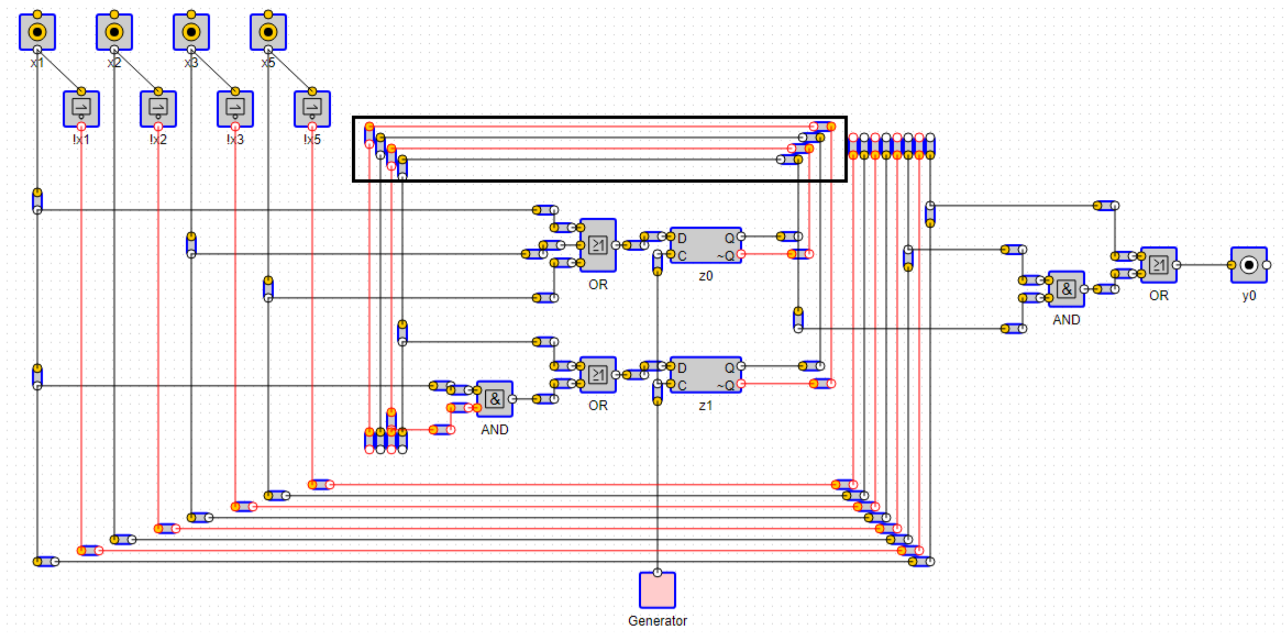


Figure 2.14 – Connections between right and left top Z joints

2.3.30 drawXVariablesForMealy()

Functionality:

1. Extends left x joints to bottom
2. Create joints on same height on right side
3. Connects them.

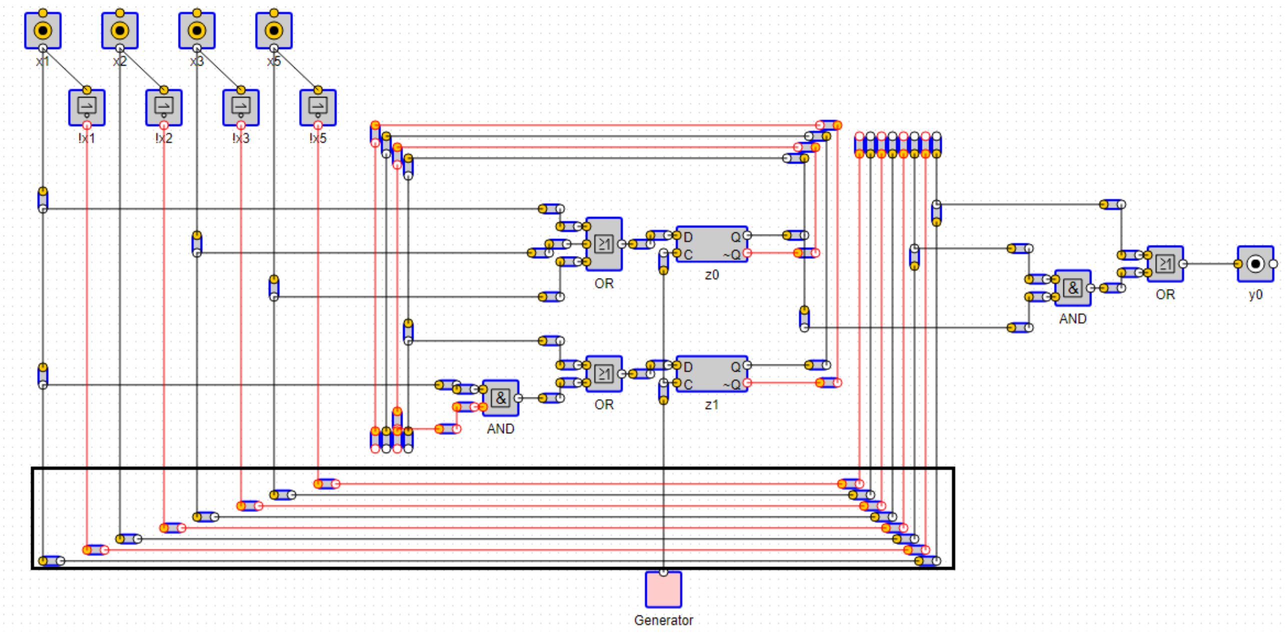


Figure 2.15 – Bottom connectors for Mealy machine

2.3.31 drawOsc()

Functionality: Draws oscillator at bottom

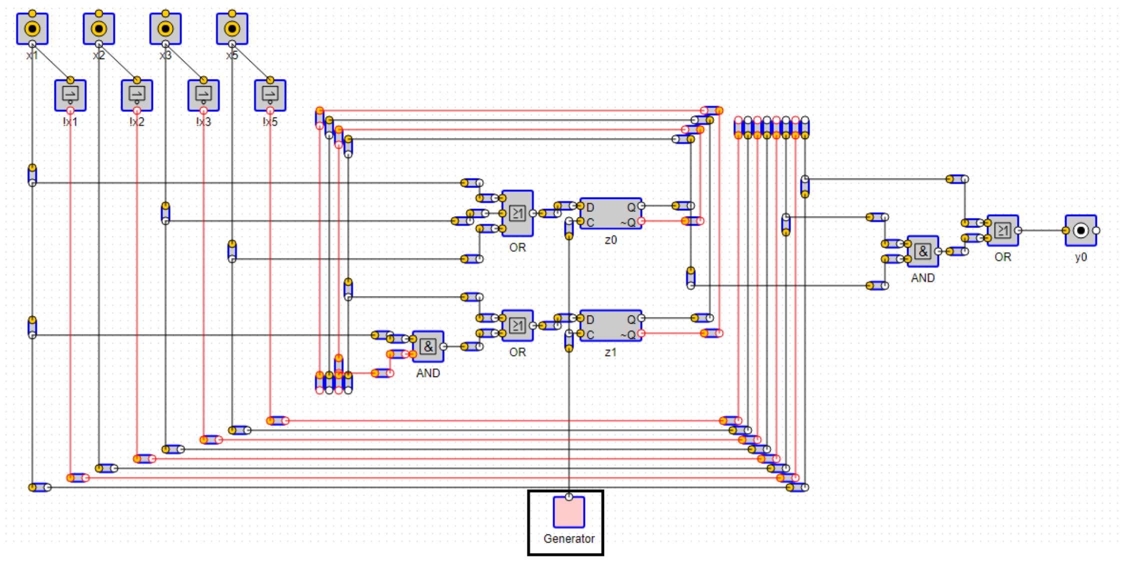


Figure 2.16 – Oscillator

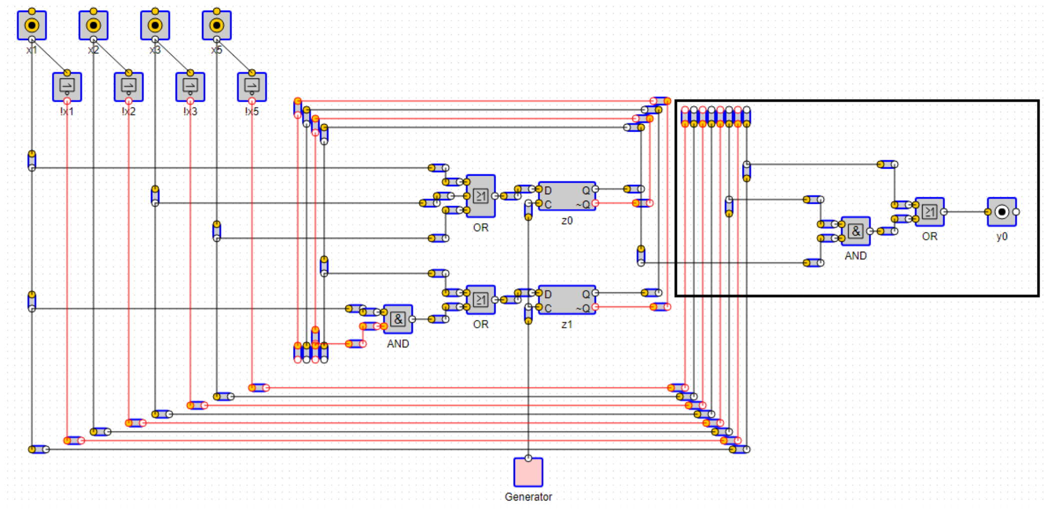
2.3.32 createYTrees(expressions, result)

Arguments:

«expressions»	is an object of type {y: {«y0»:»x1+x2»}, z: {«z0»: «x1+x2»}}.
«result»	variables that stores all global results.

Functionality:

Builds the right side of application, shown on figure 2.17



.

Figure 2.17 – Right side of application

2.3.33 drawYTrees(trees, distanceBetweenChildAndParent)

Arguments:

«tree»	tree to draw
«distanceBetweenChildAndParent»	width between parent and child for each level

Functionality:

Calls inner method to draw a right side devices.

2.3.34 createYDevices(tree, distanceBetweenChildAndParent, depth)

Arguments:

«tree»	tree to draw
«distanceBetweenChildAndParent»	width between parent and child for each level
«maxDepth»	max depth of tree

Functionality: Calls inner methods to draw devices.

2.3.35 processYVariable(tree)

Arguments:

«tree»	current node of a tree

Functionality: Decides what function to call depending on node variables «x» or «z»

2.3.36 proceessYVariableX(tree)

Arguments:

«tree»	current node of a tree

Functionality: Extends from x variables from top right to bottom

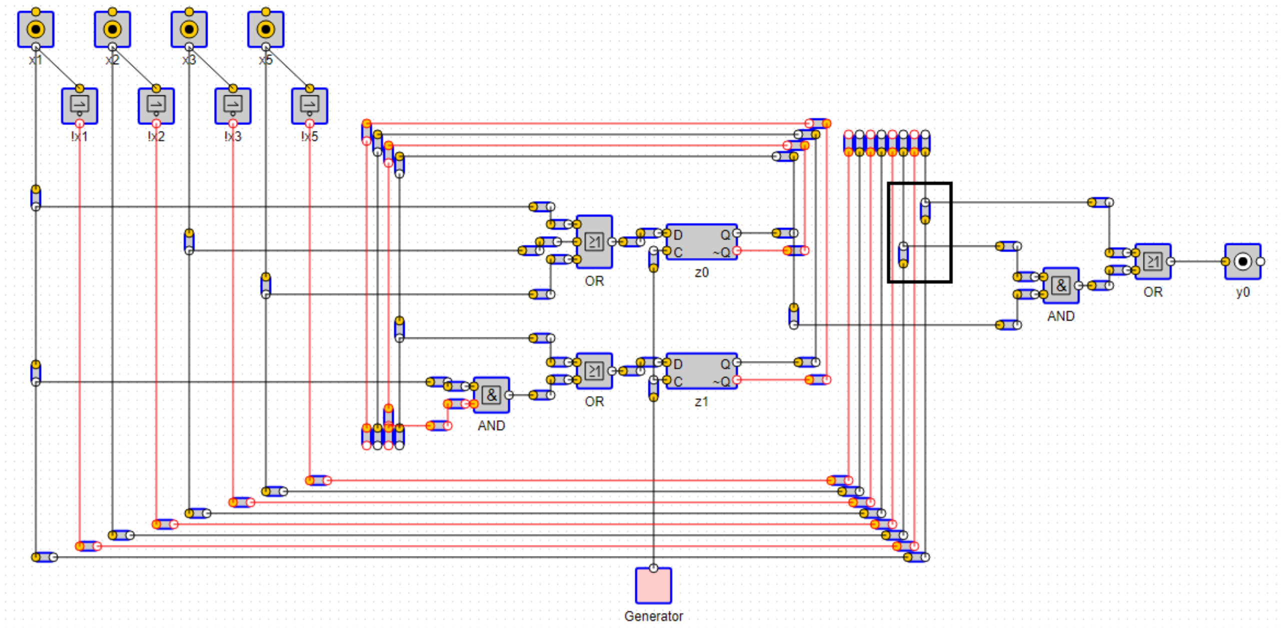


Figure 2.18 – Processing of right x variables

2.3.37 processYVariableZ(tree)

Arguments:

«tree» current node of a tree

Functionality: Extends top right z joints to bottom to be used with devices

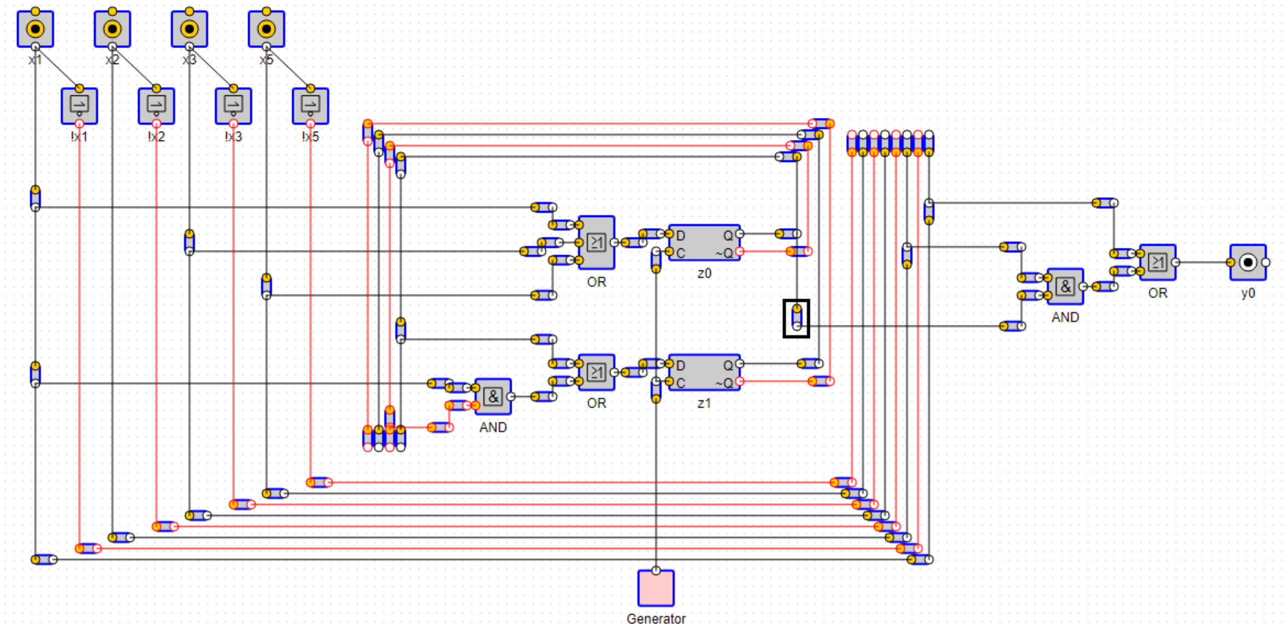


Figure 2.19 – Processing of z variables

2.3.38 createRightTopXVariablesJoints()

Functionality: Create right top x joints

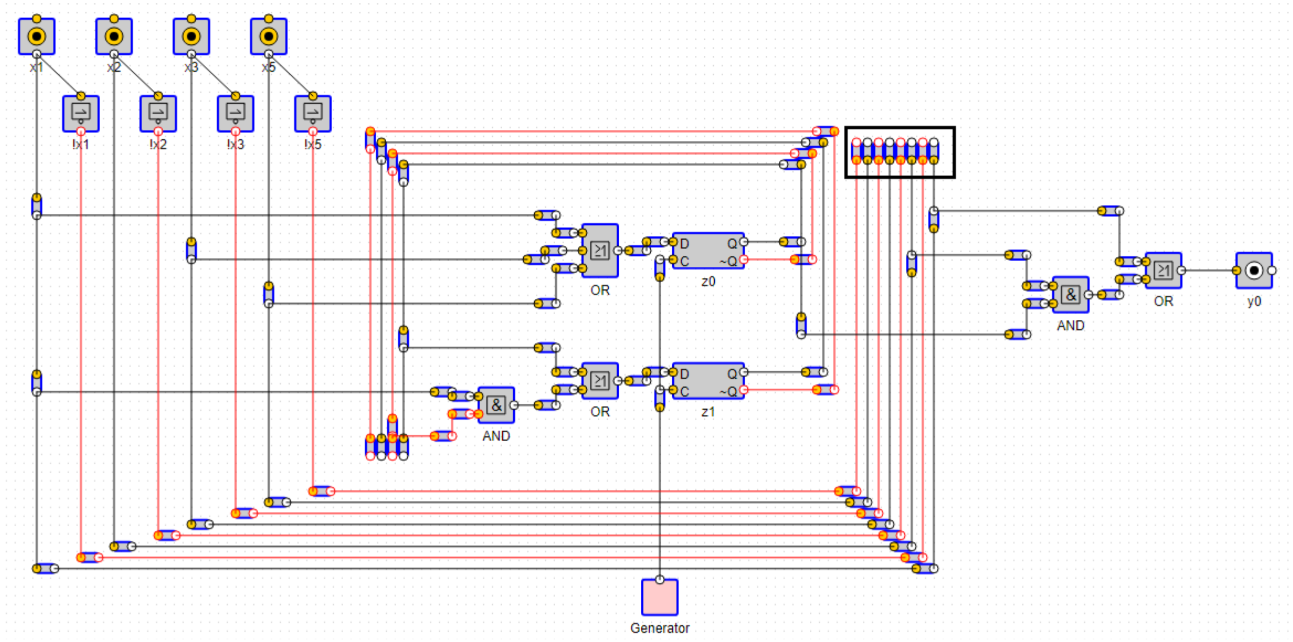


Figure 2.20 – Top right x joints

2.3.39 connectRightXTopToBottomJoints()

Functionality: Connects top x joints with bottom

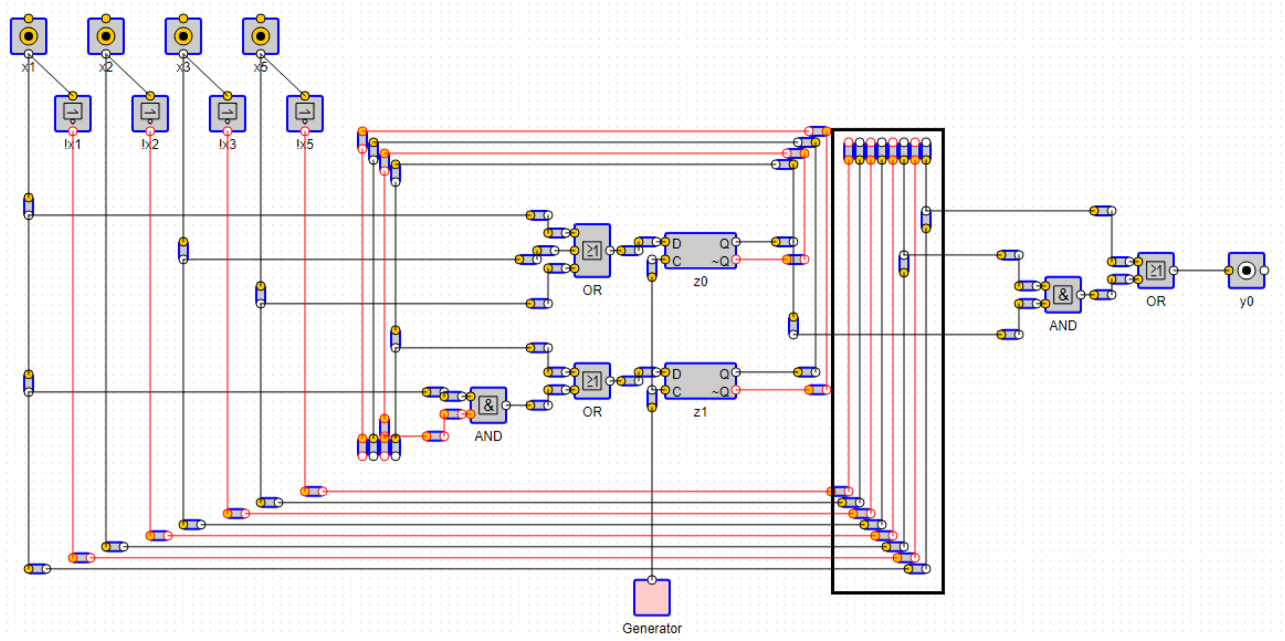


Figure 2.20 – Connections of right x lines

2.3.40 connectRightZTopToBottomJoints()

Functionality: connect right z joints.

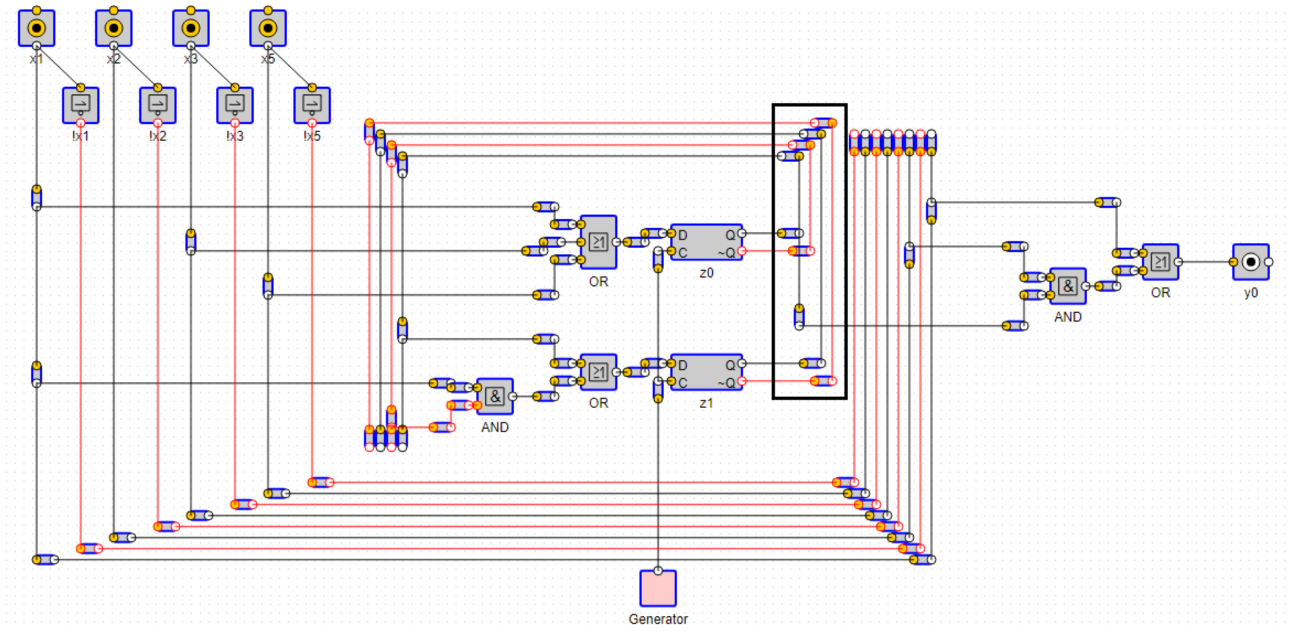


Figure 2.21 – Connections of right z lines

3 BEAST to expression

Entry point is method «extractExpressions».

3.1 Methods

3.1.1 extractExpressions(circuit, getComponent)

Arguments:

«circuit» BEAST json circuit,

«getComponent» function from BEAST controller, to retrieve component by it»s id

Functionality:

Main function that returns expressions

Return: expressions out of circuit

3.1.2 buildTrees(circuit, getComponent)

Arguments:

«circuit» BEAST json circuit,

«getComponent» function from BEAST controller, to retrieve component by it»s id

Functionality:

1. Checks for cycles
2. Build trees out of json

Example: x1 + x2 => {type: «OR», children: [{type:»VAR», value:»x1», id:»dev1»}, { type:»VAR», value:»x2», id:»dev2»}], id: «dev0»}

Return: tree of elements

3.1.3 findAllOutputDevices(devices)

Arguments:

«devices» all devices from json BEAST

Functionality: Finds all devices that are either «LED» or «Out».

Return: array of all outputs

3.1.4 transformConnectorsToMap(connectors)

Arguments:

«connectors»	all connectors from json BEAST

Functionality: Creates map where key is current connector and value is array of his children. So later we can go with Breadth first seach on this array.

Return: map of connectors where key is current connector and value is array of his children

3.1.5 getIdAndPort(connector)

Arguments:

«connector»	single connector side from json BEAST, «to» or «from» (to: «dev1.in0»).

Functionality: Gets id and port from connectors «to» or «from».

Example: (to: «dev1.in0») -> {id: dev1, port: in0}

3.1.6 transformDevicesToMap(devices)

Arguments:

«devices»	all devices from json BEAST

Functionality: Creates map of devices where key is id of devices and value is device.

Return: map of devices where key is id of devices and value is device

3.1.7 isCycled(device, connectors, devicesMap)

Arguments:

«device»	root device of tree

«connectors»	map of all connectors
«devicesMap»	map of all devices

Functionality: Recursively checks if current tree has cycle

Return: true if tree is cycled, false otherwise

3.1.8 buildTree(device, connectors, devicesMap)

Arguments:

«device»	root device of tree
«connectors»	map of all connectors
«devicesMap»	map of all devices

Functionality: Recursively goes into devices and add devices to tree.

Function goes recursively until finds «In» port, all devices after port are ignored

If device is «Component» then function «replaceComponentWithSubtree» is called it will build subtree from this component and replace with previous device.

3.1.9 replaceComponentWithSubtree(device, children)

Arguments:

«device»	current device that is component
«children»	all children devices of this device

Functionality:

1. With a help of outer method «getComponent» retrieves BEAST json of component.

2. Calling method «buildTrees» of this class with recursively build tree of component.

3. Then «In» ports of component tree will become parents of «children» devices.

Return: Root of a subtree

3.1.10 takeLeavesFromTree(tree)

Arguments:

«tree» current node of tree

Functionality: Recursively finds all leaves of tree and return them as array.

Return: array of leaves of tree.

3.1.11 buildExpression(node)

Arguments:

«node» current node of a tree

Functionality: Recursively goes down through tree and build an expression.

Return: expression

3.1.12 typeIsOk(type)

Arguments:

«type» type of device

Functionality: Decides whether this type can be in expression

Return: true if type can be in expression

13.1.3 shouldHaveParenthesis(node)

Arguments:

«node» current node of a tree

Return: Whether this node should be wrapped by parenthesis

3.1.14 isNotAtFront(node)

Arguments:

«node» current node of a tree

Functionality: Checks if node is some kind of NOT like NAND, NOR etc.

Return: true if node is a derived from NOT type

3.1.15 typeToSign(type)

Arguments:

«type»	type of device

Functionality: converts type to mathematical sign. «OR» -> «+», «AND» -> «*».

Return: mathematical sign corresponding to type

4 Table of figures