# 

T.C.

CUMHURİYET ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

# PROJE ADI

JavaScript ile Logic Devre Simülasyonu

## ENES YILDIZ

## 2014123070

### *Özet:*

*Bu projede JavaScript ve HTML kullanarak 1 ve 0 çıkışı üreten elemanlarımız 2 ve 3 girişi bulunan AND ve OR kapıları ve NOT kapısı oluşturacak tasarımları yaptım. 0 çıkışı zaten varsayılan olarak inputlarda bulunuyor, görüntüde varlığını gösterebilmek için 0 çıkışı veren elemanı da ekledim. Mouse eventları ile de sürükle bırak yapıp sonrasında yine input ve outputları Mouse yardımı ile birleştiriyorum. Mantık devresinin sonucunu da bir led yardımı ile simüle ediyorum. Sonuç 1 çıktığında ledin rengi siyahtan yeşile dönüyor. Sıfır olursa siyahta kalıyor.*

***Anahtar Kelimeler: Mantık devre tasarımı, JavaScript, HTML***

*Abstract*

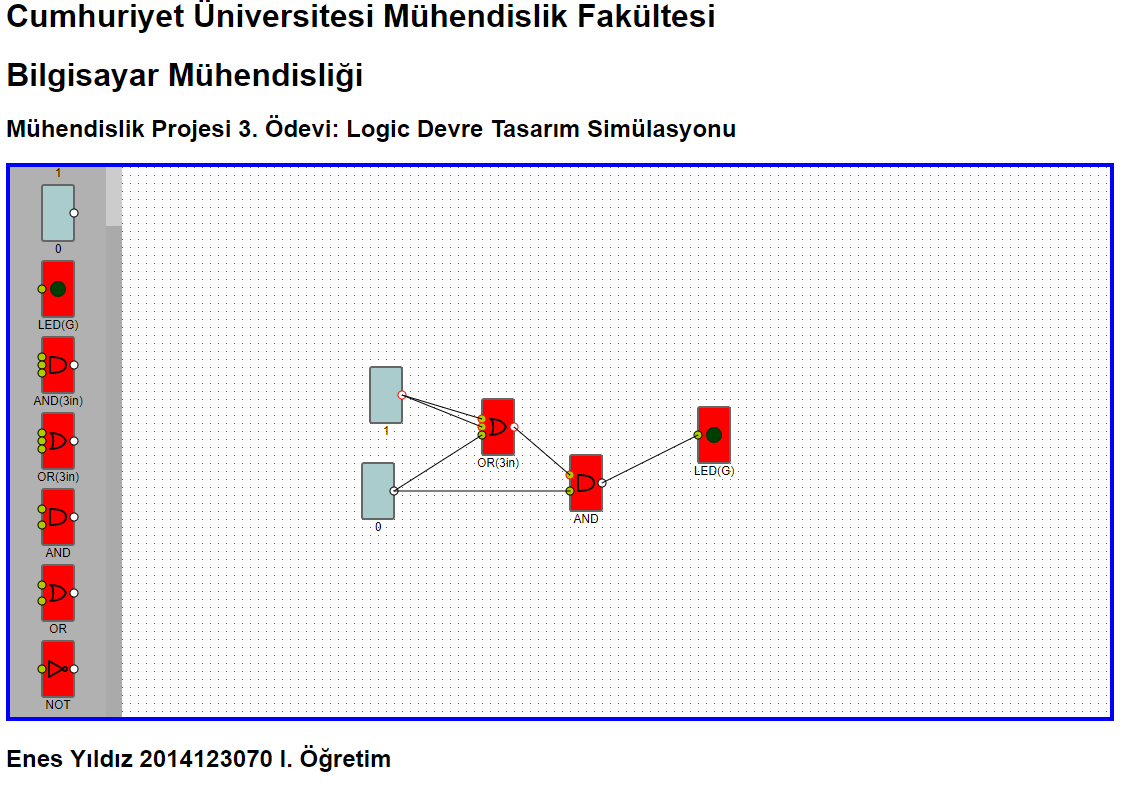
In this Project I have used JavaScript and HTML to create elements which have 1 and 0 outputs, 2 and 3 inputs AND and OR Gates and 1 input 1 output NOT gate. 0 output is default for Gate’s inputs but I wanted to show it so I created an element for it. I used Mouse events to put Gates elements to workboard. I use Mouse events to connect inputs and outputs. I simulate result of logic circuits with a led. When the result is 1 the led will be green . If the result is 0 the led stay as black.

***Keywords: Logic Circuit Design, JavaScript, HTML***

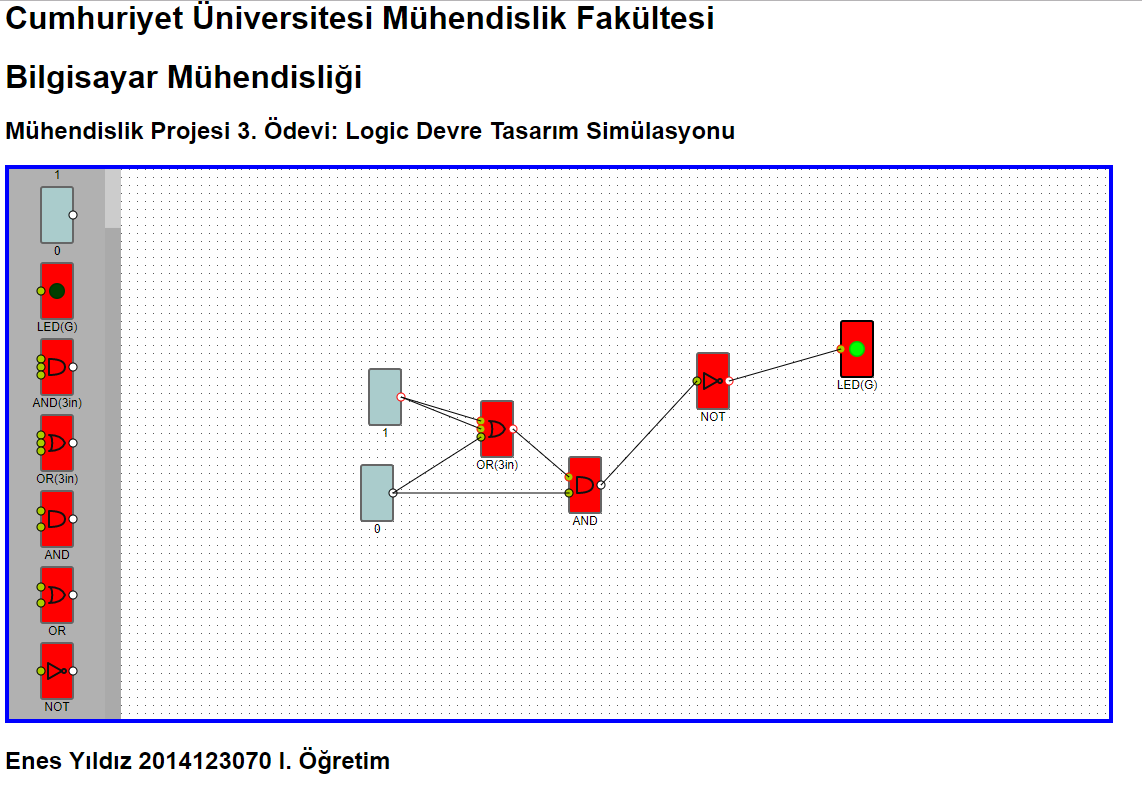
## Giriş

Bu projede bizden istenilen 2 ve 3 girişi bulunan AND ve OR kapıları ve bunlara ek olarak NOT kapısının simüle edilmesi. Bunun için back-end kısmında JavaScript ve front-end kısmında HTML kullanıyorum. Kapıları gösterdiğim HTML sayfasında sadece JavaScript dosyalarına göndereceğim veriler JSON olarak bulunuyor ayrıca kurduğum devre de bu sayfada JSON verisi olarak tarayıcının cache belleğine kayıt ediliyor böylece sayfa yenilenene kadar bu verilerin kaybolması engelleniyor ve gösterilebiliyor. Yapılan bağlantılar boarda eklenen elemanlar vs. tarayıcının cache’inden gelen veriler ile anlık olarak işleniyor ve sonuçları da gösteriliyor. JavaScript dosyalarında ise bu front-end kısmında kaydettiğim verilerin işlenmesi, ekrana sonuçların yazılması vs. işlemleri ve Mouse eventlarını kontrol ediyorum. CSS dosyaları yardımıyla da çizimlerimizin renklerini ayarladık

## UYGULAMA



Soldaki toolboxdan aldığım mantık devre elemanlarını Mouse yardımıyla sağdaki boarda sürüklüyüp bırakıyoruz. Ve yine Mouse yardımıyla input ve outputları birleştiriyoruz. Şekildeki devrede devrenin sonucu 0 olduğu için ledimiz siyah bir sonraki resimde 1 üreten bir devre ile ledin yandığını göstereceğim.



Bir önceki 0 sonucunu üreten devre ile led arasına bir NOT kapısı ekledik ve sonucu otomatik olarak 1’e çevirdik. Sonuç 1 olunca da led yandı ve yeşil rengi aldı.

## SONUÇLAR

Sonuç olarak bu projede çeşitli Mouse eventlarını kullanmayı, tarayıcının cache belleğini kullanmayı, HTML ile çeşitli şekiller çizmeyi kullanarak öğrenmiş olduk. Projede istenilen tüm mantık elemanlarını ve bunların sonuçlarını doğru olarak göstererek projeyi başarıyla tamamladım.

## KAYNAK KODLAR

simcir.js dosyası, Mouse eventları temel fonksiyonlar vs. bu dosyada bulunmaktadır.

'use strict';

var simcir = {};

//bu fonksiyon yardımıyla tarayıcının cache belleğine yaptığımız işlemleri

//kaydediyoruz veya siliyoruz

simcir.$ = function() {

//değişken tanımlamalar

var debug = location.hash == '#debug';

var cacheIdKey = '.lessqCacheId';

var cacheIdSeq = 0;

var cache = {};

//cachedeki veriyi almak için getCache fonksiyonu

var getCache = function(elm) {

var cacheId = elm[cacheIdKey];

if (typeof cacheId == 'undefined') {

elm[cacheIdKey] = cacheId = cacheIdSeq++;

cache[cacheId] = debug? { e : elm } : {};

}

return cache[cacheId];

};

//cachede aradığımız verinin olup olmadığını kontrol eden hasCache fonksiyonu

var hasCache = function(elm) {

return typeof elm[cacheIdKey] != 'undefined';

};

if (debug) {

var lastKeys = {};

var showCacheCount = function() {

var cnt = 0;

var keys = {};

for (var k in cache) {

cnt += 1;

if (!lastKeys[k]) {

console.log(cache[k]);

}

keys[k] = true;

}

lastKeys = keys;

console.log('cacheCount:' + cnt);

window.setTimeout(showCacheCount, 5000);

};

showCacheCount();

}

//cachedeki veriyi silen fonksiyon

var removeCache = function(elm) {

if (typeof elm[cacheIdKey] != 'undefined') {

// işlem olup olmadığını kontrol eden listenerları silen fonksiyon

var cacheId = elm[cacheIdKey];

var listenerMap = cache[cacheId].listenerMap;

for (var type in listenerMap) {

var listeners = listenerMap[type];

for (var i = 0; i < listeners.length; i += 1) {

elm.removeEventListener(type, listeners[i]);

}

}

delete elm[cacheIdKey];

delete cache[cacheId];

}

//tüm veriler boşalana kadar cachedeki verileri silen fonksiyon

while (elm.firstChild) {

removeCache(elm.firstChild);

elm.removeChild(elm.firstChild);

}

};

//verileri çeken fonksiyon

var getData = function(elm) {

if (!getCache(elm).data) { getCache(elm).data = {}; }

return getCache(elm).data;

};

//listenerları çeken fonksiyon

var getListeners = function(elm, type) {

if (!getCache(elm).listenerMap) {

getCache(elm).listenerMap = {}; }

if (!getCache(elm).listenerMap[type]) {

getCache(elm).listenerMap[type] = []; }

return getCache(elm).listenerMap[type];

};

// event listener ekleyen fonksiyonlar

var addEventListener = function(elm, type, listener, add) {

var listeners = getListeners(elm, type);

var newListeners = [];

for (var i = 0; i < listeners.length; i += 1) {

if (listeners[i] != listener) {

newListeners.push(listeners[i]);

}

}

if (add) { newListeners.push(listener); }

getCache(elm).listenerMap[type] = newListeners;

return true;

};

var CustomEvent = {

preventDefault : function() { this.\_pD = true; },

stopPropagation : function() { this.\_sP = true; },

stopImmediatePropagation : function() { this.\_sIp = true; }

};

//hashmapte tutulan verileri kontrol eden fonksiyon

var trigger = function(elm, type, data) {

var event = { type : type, target : elm, currentTarget : null,

\_pD : false, \_sP : false, \_sIp : false, \_\_proto\_\_ : CustomEvent };

for (var e = elm; e != null; e = e.parentNode) {

if (!hasCache(e) ) { continue; }

if (!getCache(e).listenerMap) { continue; }

if (!getCache(e).listenerMap[type]) { continue; }

event.currentTarget = e;

var listeners = getCache(e).listenerMap[type];

for (var i = 0; i < listeners.length; i += 1) {

listeners[i].call(e, event, data);

if (event.\_sIp) { return; }

}

if (event.\_sP) { return; }

}

};

var data = function(elm, kv) {

if (arguments.length == 2) {

if (typeof kv == 'string') return getData(elm)[kv];

for (var k in kv) { getData(elm)[k] = kv[k]; }

} else if (arguments.length == 3) {

getData(elm)[kv] = arguments[2];

}

return elm;

};

//iki diziyi karşılaştıran fonksiyon

var extend = function(o1, o2) {

for (var k in o2) { o1[k] = o2[k]; } return o1;

};

//verinin fonksiyon olup olmadığını kontrol eden fonksiyon

var each = function(it, callback) {

if (typeof it.splice == 'function') {

for (var i = 0; i < it.length; i += 1) { callback(i, it[i]); }

} else {

for (var k in it) { callback(k, it[k]); }

}

};

//listedeki verileri kontrol edip doğru formda ise yeni listeye atan fonskiyon

var grep = function(list, accept) {

var newList = [];

for (var i = 0; i < list.length; i += 1) {

var item = list[i];

if (accept(item) ) {

newList.push(item);

}

}

return newList;

};

//yeni bir class ekleyen fonksiyon

var addClass = function(elm, className, add) {

var classes = (elm.getAttribute('class') || '').split(/\s+/g);

var newClasses = '';

for (var i = 0; i < classes.length; i+= 1) {

if (classes[i] == className) { continue; }

newClasses += ' ' + classes[i];

}

if (add) { newClasses += ' ' + className; }

elm.setAttribute('class', newClasses);

};

//classın var olup olmadığını kontrol eden fonksiyon

var hasClass = function(elm, className) {

var classes = (elm.getAttribute('class') || '').split(/\s+/g);

for (var i = 0; i < classes.length; i+= 1) {

if (classes[i] == className) { return true; }

}

return false;

};

//selectorlerin eşleşip eşleşmediğini kontrol eden fonksiyon

var matches = function(elm, selector) {

if (elm.nodeType != 1) {

return false;

} else if (!selector) {

return true;

}

var sels = selector.split(/[,\s]+/g);

for (var i = 0; i < sels.length; i += 1) {

var sel = sels[i];

if (sel.substring(0, 1) == '#') {

throw 'not supported:' + sel;

} else if (sel.substring(0, 1) == '.') {

if (hasClass(elm, sel.substring(1) ) ) {

return true;

}

} else {

if (elm.tagName.toUpperCase() == sel.toUpperCase() ) {

return true;

}

}

}

return false;

};

var parser = new window.DOMParser();

//html dosyası oluşturan bir fonksiyon

var html = function(html) {

var doc = parser.parseFromString(

'<div xmlns="http://www.w3.org/1999/xhtml">' + html + '</div>',

'text/xml').firstChild;

var elms = [];

while (doc.firstChild) {

elms.push(doc.firstChild);

doc.removeChild(doc.firstChild);

}

elms.\_\_proto\_\_ = fn;

return elms;

};

var pxToNum = function(px) {

if (typeof px != 'string' || px.length <= 2 ||

px.charAt(px.length - 2) != 'p' ||

px.charAt(px.length - 1) != 'x') {

throw 'illegal px:' + px;

}

return +px.substring(0, px.length - 2);

};

//sorgu oluşturan fonksiyon

var buildQuery = function(data) {

var query = '';

for (var k in data) {

if (query.length > 0) {

query += '&';

}

query += window.encodeURIComponent(k);

query += '=';

query += window.encodeURIComponent(data[k]);

}

return query;

};

var parseResponse = function() {

var contentType = this.getResponseHeader('content-type');

if (contentType != null) {

contentType = contentType.replace(/\s\*;.+$/, '').toLowerCase();

}

if (contentType == 'text/xml' ||

contentType == 'application/xml') {

return parser.parseFromString(this.responseText, 'text/xml');

} else if (contentType == 'text/json' ||

contentType == 'application/json') {

return JSON.parse(this.responseText);

} else {

return this.response;

}

};

//html ile javascript arasında veri alışverisini şağlayan fonksiyon

var ajax = function(params) {

params = extend({

url: '',

method : 'GET',

contentType : 'application/x-www-form-urlencoded;charset=UTF-8',

cache: true,

processData: true,

async : true

}, params);

if (!params.async) {

throw 'not supported.';

}

var method = params.method.toUpperCase();

var data = null;

var contentType = params.contentType;

if (method == 'POST' || method == 'PUT') {

data = (typeof params.data == 'object' && params.processData)?

buildQuery(params.data) : params.data;

} else {

contentType = false;

}

var xhr = params.xhr? params.xhr() : new window.XMLHttpRequest();

xhr.open(method, params.url, params.async);

if (contentType !== false) {

xhr.setRequestHeader('Content-Type', contentType);

}

xhr.onreadystatechange = function() {

if(xhr.readyState == window.XMLHttpRequest.DONE) {

try {

if (xhr.status == 200) {

done.call(xhr, parseResponse.call(this) );

} else {

fail.call(xhr);

}

} finally {

always.call(xhr);

}

}

};

window.setTimeout(function() { xhr.send(data); }, 0);

// geri çağırışlar

var done = function(data) {};

var fail = function() {};

var always = function() {};

var $ = {

done : function(callback) { done = callback; return $; },

fail : function(callback) { fail = callback; return $; },

always : function(callback) { always = callback; return $; },

abort : function() { xhr.abort(); return $; }

};

return $;

};

// 1. tek eleman için verileri işleyen fonksiyon

var fn = {

attr : function(kv) {

if (arguments.length == 1) {

if (typeof kv == 'string') return this.getAttribute(kv);

for (var k in kv) { this.setAttribute(k, kv[k]); }

} else if (arguments.length == 2) {

this.setAttribute(kv, arguments[1]);

}

return this;

},

prop : function(kv) {

if (arguments.length == 1) {

if (typeof kv == 'string') return this[kv];

for (var k in kv) { this[k] = kv[k]; }

} else if (arguments.length == 2) {

this[kv] = arguments[1];

}

return this;

},

css : function(kv) {

if (arguments.length == 1) {

if (typeof kv == 'string') return this.style[kv];

for (var k in kv) { this.style[k] = kv[k]; }

} else if (arguments.length == 2) {

this.style[kv] = arguments[1];

}

return this;

},

data : function(kv) {

var args = [ this ];

for (var i = 0; i < arguments.length; i += 1) {

args.push(arguments[i]);

};

return data.apply(null, args);

},

val : function() {

if (arguments.length == 0) {

return this.value || '';

} else if (arguments.length == 1) {

this.value = arguments[0];

}

return this;

},

on : function(type, listener) {

var types = type.split(/\s+/g);

for (var i = 0; i < types.length; i += 1) {

this.addEventListener(types[i], listener);

addEventListener(this, types[i], listener, true);

}

return this;

},

off : function(type, listener) {

var types = type.split(/\s+/g);

for (var i = 0; i < types.length; i += 1) {

this.removeEventListener(types[i], listener);

addEventListener(this, types[i], listener, false);

}

return this;

},

trigger : function(type, data) {

trigger(this, type, data);

return this;

},

offset : function() {

var off = { left : 0, top : 0 };

var base = null;

for (var e = this; e.parentNode != null; e = e.parentNode) {

if (e.offsetParent != null) {

base = e;

break;

}

}

if (base != null) {

for (var e = base; e.offsetParent != null; e = e.offsetParent) {

off.left += e.offsetLeft;

off.top += e.offsetTop;

}

}

for (var e = this; e.parentNode != null &&

e != document.body; e = e.parentNode) {

off.left -= e.scrollLeft;

off.top -= e.scrollTop;

}

return off;

},

append : function(elms) {

if (typeof elms == 'string') {

elms = html(elms);

}

for (var i = 0; i < elms.length; i += 1) {

this.appendChild(elms[i]);

}

return this;

},

prepend : function(elms) {

if (typeof elms == 'string') {

elms = html(elms);

}

for (var i = 0; i < elms.length; i += 1) {

if (this.firstChild) {

this.insertBefore(elms[i], this.firstChild);

} else {

this.appendChild(elms[i]);

}

}

return this;

},

insertBefore : function(elms) {

var elm = elms[0];

elm.parentNode.insertBefore(this, elm);

return this;

},

insertAfter : function(elms) {

var elm = elms[0];

if (elm.nextSibling) {

elm.parentNode.insertBefore(this, elm.nextSibling);

} else {

elm.parentNode.appendChild(this);

}

return this;

},

remove : function() {

if (this.parentNode) { this.parentNode.removeChild(this); }

removeCache(this);

return this;

},

detach : function() {

if (this.parentNode) { this.parentNode.removeChild(this); }

return this;

},

parent : function() {

return $(this.parentNode);

},

closest : function(selector) {

for (var e = this; e != null; e = e.parentNode) {

if (matches(e, selector) ) {

return $(e);

}

}

return $();

},

find : function(selector) {

var elms = [];

var childNodes = this.querySelectorAll(selector);

for (var i = 0; i < childNodes.length; i += 1) {

elms.push(childNodes.item(i) );

}

elms.\_\_proto\_\_ = fn;

return elms;

},

children : function(selector) {

var elms = [];

var childNodes = this.childNodes;

for (var i = 0; i < childNodes.length; i += 1) {

if (matches(childNodes.item(i), selector) ) {

elms.push(childNodes.item(i) );

}

}

elms.\_\_proto\_\_ = fn;

return elms;

},

index : function(selector) {

return Array.prototype.indexOf.call(

$(this).parent().children(selector), this);

},

clone : function() { return $(this.cloneNode(true) ); },

focus : function() { this.focus(); return this; },

select : function() { this.select(); return this; },

submit : function() { this.submit(); return this; },

scrollLeft : function() {

if (arguments.length == 0) return this.scrollLeft;

this.scrollLeft = arguments[0]; return this;

},

scrollTop : function() {

if (arguments.length == 0) return this.scrollTop;

this.scrollTop = arguments[0]; return this;

},

html : function() {

if (arguments.length == 0) return this.innerHTML;

this.innerHTML = arguments[0]; return this;

},

text : function() {

if (typeof this.textContent != 'undefined') {

if (arguments.length == 0) return this.textContent;

this.textContent = arguments[0]; return this;

} else {

if (arguments.length == 0) return this.innerText;

this.innerText = arguments[0]; return this;

}

},

outerWidth : function(margin) {

var w = this.offsetWidth;

if (margin) {

var cs = window.getComputedStyle(this, null);

return w + pxToNum(cs.marginLeft) + pxToNum(cs.marginRight);

}

return w;

},

innerWidth : function() {

var cs = window.getComputedStyle(this, null);

return this.offsetWidth -

pxToNum(cs.borderLeftWidth) - pxToNum(cs.borderRightWidth);

},

width : function() {

if (this == window) return this.innerWidth;

var cs = window.getComputedStyle(this, null);

return this.offsetWidth -

pxToNum(cs.borderLeftWidth) - pxToNum(cs.borderRightWidth) -

pxToNum(cs.paddingLeft) - pxToNum(cs.paddingRight);

},

outerHeight : function(margin) {

var h = this.offsetHeight;

if (margin) {

var cs = window.getComputedStyle(this, null);

return h + pxToNum(cs.marginTop) + pxToNum(cs.marginBottom);

}

return h;

},

innerHeight : function() {

var cs = window.getComputedStyle(this, null);

return this.offsetHeight -

pxToNum(cs.borderTopWidth) - pxToNum(cs.borderBottomWidth);

},

height : function() {

if (this == window) return this.innerHeight;

var cs = window.getComputedStyle(this, null);

return this.offsetHeight -

pxToNum(cs.borderTopWidth) - pxToNum(cs.borderBottomWidth) -

pxToNum(cs.paddingTop) - pxToNum(cs.paddingBottom);

},

addClass : function(className) {

addClass(this, className, true); return this;

},

removeClass : function(className) {

addClass(this, className, false); return this;

},

hasClass : function(className) {

return hasClass(this, className);

}

};

// 2. birden fazla elemanı fn fonksiyonu ile kontol edecek fonksiyon

each(fn, function(name, func) {

fn[name] = function() {

var newRet = null;

for (var i = 0; i < this.length; i += 1) {

var elm = this[i];

var ret = func.apply(elm, arguments);

if (elm !== ret) {

if (ret != null && ret.\_\_proto\_\_ == fn) {

if (newRet == null) { newRet = []; }

newRet = newRet.concat(ret);

} else {

return ret;

}

}

}

if (newRet != null) {

newRet.\_\_proto\_\_ = fn;

return newRet;

}

return this;

};

});

// 3. Array için verileri işleyen fonksiyon

fn = extend(fn, {

each : function(callback) {

for (var i = 0; i < this.length; i += 1) {

callback.call(this[i], i);

}

return this;

},

first : function() {

return $(this.length > 0? this[0] : null);

},

last : function() {

return $(this.length > 0? this[this.length - 1] : null);

}

});

var $ = function(target) {

if (typeof target == 'function') {

// ready

return $(document).on('DOMContentLoaded', target);

} else if (typeof target == 'string') {

if (target.charAt(0) == '<') {

// dom creation

return html(target);

} else {

// query

var childNodes = document.querySelectorAll(target);

var elms = [];

for (var i = 0; i < childNodes.length; i += 1) {

elms.push(childNodes.item(i) );

}

elms.\_\_proto\_\_ = fn;

return elms;

}

} else if (typeof target == 'object' && target != null) {

if (target.\_\_proto\_\_ == fn) {

return target;

} else {

var elms = [];

elms.push(target);

elms.\_\_proto\_\_ = fn;

return elms;

}

} else {

var elms = [];

elms.\_\_proto\_\_ = fn;

return elms;

}

};

return extend($, {

fn : fn, extend : extend, each : each, grep : grep,

data : data, ajax : ajax });

}();

// çizimlerin yapıldığı fonksiyonlar

!function($s) {

var $ = $s.$;

var createSVGElement = function(tagName) {

return $(document.createElementNS(

'http://www.w3.org/2000/svg', tagName) );

};

var createSVG = function(w, h) {

return createSVGElement('svg').attr({

version: '1.1',

width: w, height: h,

viewBox: '0 0 ' + w + ' ' + h

});

};

var graphics = function($target) {

var attr = {};

var buf = '';

var moveTo = function(x, y) {

buf += ' M ' + x + ' ' + y;

};

var lineTo = function(x, y) {

buf += ' L ' + x + ' ' + y;

};

var curveTo = function(x1, y1, x, y) {

buf += ' Q ' + x1 + ' ' + y1 + ' ' + x + ' ' + y;

};

var closePath = function(close) {

if (close) {

// really close path.

buf += ' Z';

}

$target.append(createSVGElement('path').

attr('d', buf).attr(attr) );

buf = '';

};

var drawRect = function(x, y, width, height) {

$target.append(createSVGElement('rect').

attr({x: x, y: y, width: width, height: height}).attr(attr) );

};

var drawCircle = function(x, y, r) {

$target.append(createSVGElement('circle').

attr({cx: x, cy: y, r: r}).attr(attr) );

};

return {

attr: attr,

moveTo: moveTo,

lineTo: lineTo,

curveTo: curveTo,

closePath: closePath,

drawRect: drawRect,

drawCircle: drawCircle

};

};

var transform = function() {

var attrX = 'simcir-transform-x';

var attrY = 'simcir-transform-y';

var attrRotate = 'simcir-transform-rotate';

var num = function($o, k) {

var v = $o.attr(k);

return v? +v : 0;

};

return function($o, x, y, rotate) {

if (arguments.length >= 3) {

var transform = 'translate(' + x + ' ' + y + ')';

if (rotate) {

transform += ' rotate(' + rotate + ')';

}

$o.attr('transform', transform);

$o.attr(attrX, x);

$o.attr(attrY, y);

$o.attr(attrRotate, rotate);

} else if (arguments.length == 1) {

return {x: num($o, attrX), y: num($o, attrY),

rotate: num($o, attrRotate)};

}

};

}();

//bağlantıların yapıldığı fonksiyon

var offset = function($o) {

var x = 0;

var y = 0;

while ($o[0].nodeName != 'svg') {

var pos = transform($o);

x += pos.x;

y += pos.y;

$o = $o.parent();

}

return {x: x, y: y};

};

//eventları aktifleştiren fonksiyon

var enableEvents = function($o, enable) {

$o.css('pointer-events', enable? 'visiblePainted' : 'none');

};

//bağlantıyı kaldıran fonksiyon

var disableSelection = function($o) {

$o.each(function() {

this.onselectstart = function() { return false; };

}).css('-webkit-user-select', 'none');

};

var controller = function() {

var id = 'controller';

return function($ui, controller) {

if (arguments.length == 1) {

return $.data($ui[0], id);

} else if (arguments.length == 2) {

$.data($ui[0], id, controller);

}

};

}();

//event kuyruğunu oluşturan fonksiyon

var eventQueue = function() {

var delay = 50; // ms

var limit = 40; // ms

var \_queue = null;

var postEvent = function(event) {

if (\_queue == null) {

\_queue = [];

}

\_queue.push(event);

};

var dispatchEvent = function() {

var queue = \_queue;

\_queue = null;

while (queue.length > 0) {

var e = queue.shift();

e.target.trigger(e.type);

}

};

var getTime = function() {

return new Date().getTime();

};

var timerHandler = function() {

var start = getTime();

while (\_queue != null && getTime() - start < limit) {

dispatchEvent();

}

window.setTimeout(timerHandler,

Math.max(delay - limit, delay - (getTime() - start) ) );

};

timerHandler();

return {

postEvent: postEvent

};

}();

var unit = 16;

var fontSize = 12;

//label oluşturan fonksiyon

var createLabel = function(text) {

return createSVGElement('text').

text(text).

css('font-size', fontSize + 'px');

};

//input ve output nodelarını oluşturan fonksiyon

var createNode = function(type, label, description, headless) {

var $node = createSVGElement('g').

attr('simcir-node-type', type);

if (!headless) {

$node.attr('class', 'simcir-node');

}

var node = createNodeController({

$ui: $node, type: type, label: label,

description: description, headless: headless});

if (type == 'in') {

controller($node, createInputNodeController(node) );

} else if (type == 'out') {

controller($node, createOutputNodeController(node) );

} else {

throw 'unknown type:' + type;

}

return $node;

};

//nodun aktif olup olmadığını kontrol eden fonksiyon

var isActiveNode = function($o) {

return $o.closest('.simcir-node').length == 1 &&

$o.closest('.simcir-toolbox').length == 0;

};

//node kontrol etmek için controller oluhşturan fonksiyon

var createNodeController = function(node) {

var \_value = null;

var setValue = function(value, force) {

if (\_value === value && !force) {

return;

}

\_value = value;

eventQueue.postEvent({target: node.$ui, type: 'nodeValueChange'});

};

var getValue = function() {

return \_value;

};

if (!node.headless) {

node.$ui.attr('class', 'simcir-node simcir-node-type-' + node.type);

var $circle = createSVGElement('circle').

attr({cx: 0, cy: 0, r: 4});

node.$ui.on('mouseover', function(event) {

if (isActiveNode(node.$ui) ) {

node.$ui.addClass('simcir-node-hover');

}

});

node.$ui.on('mouseout', function(event) {

if (isActiveNode(node.$ui) ) {

node.$ui.removeClass('simcir-node-hover');

}

});

node.$ui.append($circle);

var appendLabel = function(text, align) {

var $label = createLabel(text).

attr('class', 'simcir-node-label');

enableEvents($label, false);

if (align == 'right') {

$label.attr('text-anchor', 'start').

attr('x', 6).

attr('y', fontSize / 2);

} else if (align == 'left') {

$label.attr('text-anchor', 'end').

attr('x', -6).

attr('y', fontSize / 2);

}

node.$ui.append($label);

};

if (node.label) {

if (node.type == 'in') {

appendLabel(node.label, 'right');

} else if (node.type == 'out') {

appendLabel(node.label, 'left');

}

}

if (node.description) {

if (node.type == 'in') {

appendLabel(node.description, 'left');

} else if (node.type == 'out') {

appendLabel(node.description, 'right');

}

}

node.$ui.on('nodeValueChange', function(event) {

if (\_value != null) {

node.$ui.addClass('simcir-node-hot');

} else {

node.$ui.removeClass('simcir-node-hot');

}

});

}

return $.extend(node, {

setValue: setValue,

getValue: getValue

});

};

//input nodelarını kontrol edecek fonksiyouun oluşturulması

var createInputNodeController = function(node) {

var output = null;

var setOutput = function(outNode) {

output = outNode;

};

var getOutput = function() {

return output;

};

return $.extend(node, {

setOutput: setOutput,

getOutput: getOutput

});

};

//output nodelarını kontrol edecek fonksiyouun oluşturulması

var createOutputNodeController = function(node) {

var inputs = [];

var super\_setValue = node.setValue;

var setValue = function(value) {

super\_setValue(value);

for (var i = 0; i < inputs.length; i += 1) {

inputs[i].setValue(value);

}

};

var connectTo = function(inNode) {

if (inNode.getOutput() != null) {

inNode.getOutput().disconnectFrom(inNode);

}

inNode.setOutput(node);

inputs.push(inNode);

inNode.setValue(node.getValue(), true);

};

var disconnectFrom = function(inNode) {

if (inNode.getOutput() != node) {

throw 'not connected.';

}

inNode.setOutput(null);

inNode.setValue(null, true);

inputs = $.grep(inputs, function(v) {

return v != inNode;

});

};

var getInputs = function() {

return inputs;

};

return $.extend(node, {

setValue: setValue,

getInputs: getInputs,

connectTo: connectTo,

disconnectFrom: disconnectFrom

});

};

//kapıları oluşturan fonksiyon

var createDevice = function(deviceDef, headless, scope) {

headless = headless || false;

scope = scope || null;

var $dev = createSVGElement('g');

if (!headless) {

$dev.attr('class', 'simcir-device');

}

controller($dev, createDeviceController(

{$ui: $dev, deviceDef: deviceDef,

headless: headless, scope: scope, doc: null}) );

var factory = factories[deviceDef.type];

if (factory) {

factory(controller($dev) );

}

if (!headless) {

controller($dev).createUI();

}

return $dev;

};

//kapıları kontrol edecek controllerı oluşturan fonksiyon

var createDeviceController = function(device) {

var inputs = [];

var outputs = [];

var addInput = function(label, description) {

var $node = createNode('in', label, description, device.headless);

$node.on('nodeValueChange', function(event) {

device.$ui.trigger('inputValueChange');

});

if (!device.headless) {

device.$ui.append($node);

}

var node = controller($node);

inputs.push(node);

return node;

};

var addOutput = function(label, description) {

var $node = createNode('out', label, description, device.headless);

if (!device.headless) {

device.$ui.append($node);

}

var node = controller($node);

outputs.push(node);

return node;

};

var getInputs = function() {

return inputs;

};

var getOutputs = function() {

return outputs;

};

var disconnectAll = function() {

$.each(getInputs(), function(i, inNode) {

var outNode = inNode.getOutput();

if (outNode != null) {

outNode.disconnectFrom(inNode);

}

});

$.each(getOutputs(), function(i, outNode) {

$.each(outNode.getInputs(), function(i, inNode) {

outNode.disconnectFrom(inNode);

});

});

};

device.$ui.on('dispose', function() {

$.each(getInputs(), function(i, inNode) {

inNode.$ui.remove();

});

$.each(getOutputs(), function(i, outNode) {

outNode.$ui.remove();

});

device.$ui.remove();

} );

var selected = false;

var setSelected = function(value) {

selected = value;

device.$ui.trigger('deviceSelect');

};

var isSelected = function() {

return selected;

};

var label = device.deviceDef.label;

var defaultLabel = device.deviceDef.type;

if (typeof label == 'undefined') {

label = defaultLabel;

}

var setLabel = function(value) {

value = value.replace(/^\s+|\s+$/g, '');

label = value || defaultLabel;

device.$ui.trigger('deviceLabelChange');

};

var getLabel = function() {

return label;

};

var getSize = function() {

var nodes = Math.max(device.getInputs().length,

device.getOutputs().length);

return { width: unit \* 2.5,

height: unit \* Math.max(3.5, device.halfPitch?

(nodes + 1) / 2 : nodes)};

};

var layoutUI = function() {

var size = device.getSize();

var w = size.width;

var h = size.height;

device.$ui.children('.simcir-device-body').

attr({x: 0, y: 0, width: w, height: h});

var pitch = device.halfPitch? unit / 2 : unit;

var layoutNodes = function(nodes, x) {

var offset = (h - pitch \* (nodes.length - 1) ) / 2;

$.each(nodes, function(i, node) {

transform(node.$ui, x, pitch \* i + offset);

});

};

layoutNodes(getInputs(), 0);

layoutNodes(getOutputs(), w);

device.$ui.children('.simcir-device-label').

attr({x: w / 2, y: h + fontSize});

};

var createUI = function() {

device.$ui.attr('class', 'simcir-device');

device.$ui.on('deviceSelect', function() {

if (selected) {

$(this).addClass('simcir-device-selected');

} else {

$(this).removeClass('simcir-device-selected');

}

});

var $body = createSVGElement('rect').

attr('class', 'simcir-device-body').

attr('rx', 2).attr('ry', 2);

device.$ui.prepend($body);

var $label = createLabel(label).

attr('class', 'simcir-device-label').

attr('text-anchor', 'middle');

device.$ui.on('deviceLabelChange', function() {

$label.text(getLabel() );

});

var label\_dblClickHandler = function(event) {

event.preventDefault();

event.stopPropagation();

var $workspace = $(event.target).closest('.simcir-workspace');

if (!controller($workspace).data().editable) {

return;

}

var title = 'Enter device name ';

var $labelEditor = $('<input type="text"/>').

addClass('simcir-label-editor').

val($label.text() ).

on('keydown', function(event) {

if (event.keyCode == 13) {

// ENTER

setLabel($(this).val() );

$dlg.remove();

} else if (event.keyCode == 27) {

// ESC

$dlg.remove();

}

} );

var $placeHolder = $('<div></div>').

append($labelEditor);

var $dlg = showDialog(title, $placeHolder);

$labelEditor.focus();

};

device.$ui.on('deviceAdd', function() {

$label.on('dblclick', label\_dblClickHandler);

} );

device.$ui.on('deviceRemove', function() {

$label.off('dblclick', label\_dblClickHandler);

} );

device.$ui.append($label);

layoutUI();

};

var getState = function() { return null; };

return $.extend(device, {

addInput: addInput,

addOutput: addOutput,

getInputs: getInputs,

getOutputs: getOutputs,

disconnectAll: disconnectAll,

setSelected: setSelected,

isSelected: isSelected,

getLabel: getLabel,

halfPitch: false,

getSize: getSize,

createUI: createUI,

layoutUI: layoutUI,

getState: getState

});

};

//Bağlantıların oluşturulması

var createConnector = function(x1, y1, x2, y2) {

return createSVGElement('path').

attr('d', 'M ' + x1 + ' ' + y1 + ' L ' + x2 + ' ' + y2).

attr('class', 'simcir-connector');

};

//bağlantılar

var connect = function($node1, $node2) {

var type1 = $node1.attr('simcir-node-type');

var type2 = $node2.attr('simcir-node-type');

if (type1 == 'in' && type2 == 'out') {

controller($node2).connectTo(controller($node1) );

} else if (type1 == 'out' && type2 == 'in') {

controller($node1).connectTo(controller($node2) );

}

};

//devreyi oluşturan fonksiyon

//cacheden gelen verileri işleyerek devreyi oluşturan fonksiyon

var buildCircuit = function(data, headless, scope) {

var $devices = [];

var $devMap = {};

var getNode = function(path) {

if (!path.match(/^(\w+)\.(in|out)([0-9]+)$/g) ) {

throw 'unknown path:' + path;

}

var devId = RegExp.$1;

var type = RegExp.$2;

var index = +RegExp.$3;

return (type == 'in')?

controller($devMap[devId]).getInputs()[index] :

controller($devMap[devId]).getOutputs()[index];

};

$.each(data.devices, function(i, deviceDef) {

var $dev = createDevice(deviceDef, headless, scope);

transform($dev, deviceDef.x, deviceDef.y);

$devices.push($dev);

$devMap[deviceDef.id] = $dev;

});

$.each(data.connectors, function(i, conn) {

var nodeFrom = getNode(conn.from);

var nodeTo = getNode(conn.to);

if (nodeFrom && nodeTo) {

connect(nodeFrom.$ui, nodeTo.$ui);

}

});

return $devices;

};

//boarda kapıları ekleyen silen yönetim fonksiyonu

var dialogManager = function() {

var dialogs = [];

var updateDialogs = function($dlg, remove) {

var newDialogs = [];

$.each(dialogs, function(i) {

if (dialogs[i] != $dlg) {

newDialogs.push(dialogs[i]);

}

});

if (!remove) {

newDialogs.push($dlg);

}

$.each(newDialogs, function(i) {

newDialogs[i].css('z-index', '' + (i + 1) );

});

dialogs = newDialogs;

};

return {

add : function($dlg) {

updateDialogs($dlg);

},

remove : function($dlg) {

updateDialogs($dlg, true);

},

toFront : function($dlg) {

updateDialogs($dlg);

}

};

}();

//pencereyi şekillendiren fonksiyonlar

var showDialog = function(title, $content) {

var $closeButton = function() {

var r = 16;

var pad = 4;

var $btn = createSVG(r, r).

attr('class', 'simcir-dialog-close-button');

var g = graphics($btn);

g.drawRect(0, 0, r, r);

g.attr['class'] = 'simcir-dialog-close-button-symbol';

g.moveTo(pad, pad);

g.lineTo(r - pad, r - pad);

g.closePath();

g.moveTo(r - pad, pad);

g.lineTo(pad, r - pad);

g.closePath();

return $btn;

}();

var $title = $('<div></div>').

addClass('simcir-dialog-title').

text(title).

css('cursor', 'default').

on('mousedown', function(event) {

event.preventDefault();

});

var $dlg = $('<div></div>').

addClass('simcir-dialog').

css({position:'absolute'}).

append($title.css('float', 'left') ).

append($closeButton.css('float', 'right') ).

append($('<br/>').css('clear', 'both') ).

append($content);

$('BODY').append($dlg);

dialogManager.add($dlg);

var dragPoint = null;

var dlg\_mouseDownHandler = function(event) {

if (!$(event.target).hasClass('simcir-dialog') &&

!$(event.target).hasClass('simcir-dialog-title') ) {

return;

}

event.preventDefault();

dialogManager.toFront($dlg);

var off = $dlg.offset();

dragPoint = {

x: event.pageX - off.left,

y: event.pageY - off.top};

$(document).on('mousemove', dlg\_mouseMoveHandler);

$(document).on('mouseup', dlg\_mouseUpHandler);

};

var dlg\_mouseMoveHandler = function(event) {

moveTo(

event.pageX - dragPoint.x,

event.pageY - dragPoint.y);

};

var dlg\_mouseUpHandler = function(event) {

$(document).off('mousemove', dlg\_mouseMoveHandler);

$(document).off('mouseup', dlg\_mouseUpHandler);

};

$dlg.on('mousedown', dlg\_mouseDownHandler);

$closeButton.on('mousedown', function() {

$dlg.trigger('close');

$dlg.remove();

dialogManager.remove($dlg);

});

var w = $dlg.width();

var h = $dlg.height();

var cw = $(window).width();

var ch = $(window).height();

var getProp = function(id) {

return $('HTML')[id]() || $('BODY')[id]();

};

var x = (cw - w) / 2 + getProp('scrollLeft');

var y = (ch - h) / 2 + getProp('scrollTop');

var moveTo = function(x, y) {

$dlg.css({left: x + 'px', top: y + 'px'});

};

moveTo(x, y);

return $dlg;

};

var createDeviceRefFactory = function(data) {

return function(device) {

var $devs = buildCircuit(data, true, {});

var $ports = [];

$.each($devs, function(i, $dev) {

var deviceDef = controller($dev).deviceDef;

if (deviceDef.type == 'In' || deviceDef.type == 'Out') {

$ports.push($dev);

}

});

$ports.sort(function($p1, $p2) {

var x1 = controller($p1).deviceDef.x;

var y1 = controller($p1).deviceDef.y;

var x2 = controller($p2).deviceDef.x;

var y2 = controller($p2).deviceDef.y;

if (x1 == x2) {

return (y1 < y2)? -1 : 1;

}

return (x1 < x2)? -1 : 1;

});

var getDesc = function(port) {

return port? port.description : '';

};

$.each($ports, function(i, $port) {

var port = controller($port);

var portDef = port.deviceDef;

var inPort;

var outPort;

if (portDef.type == 'In') {

outPort = port.getOutputs()[0];

inPort = device.addInput(portDef.label,

getDesc(outPort.getInputs()[0]) );

var inNode = port.getInputs()[0];

if (inNode.getOutput() != null) {

inNode.getOutput().disconnectFrom(inNode);

}

} else if (portDef.type == 'Out') {

inPort = port.getInputs()[0];

outPort = device.addOutput(portDef.label,

getDesc(inPort.getOutput() ) );

var outNode = port.getOutputs()[0];

$.each(outNode.getInputs(), function(i, inNode) {

if (inNode.getOutput() != null) {

inNode.getOutput().disconnectFrom(inNode);

}

} );

}

inPort.$ui.on('nodeValueChange', function() {

outPort.setValue(inPort.getValue() );

});

});

var super\_getSize = device.getSize;

device.getSize = function() {

var size = super\_getSize();

return {width: unit \* 4, height: size.height};

};

device.$ui.on('dispose', function() {

$.each($devs, function(i, $dev) {

$dev.trigger('dispose');

});

} );

device.$ui.on('dblclick', function(event) {

// open library,

event.preventDefault();

event.stopPropagation();

showDialog(device.deviceDef.label || device.deviceDef.type,

setupSimcir($('<div></div>'), data) ).on('close', function() {

$(this).find('.simcir-workspace').trigger('dispose');

});

});

};

};

//toolbox oluşturan fonksiyonlar

var createCustomLayoutDeviceRefFactory = function(data) {

return function(device) {

var $devs = buildCircuit(data, true, {});

var $ports = [];

var intfs = [];

$.each($devs, function(i, $dev) {

var deviceDef = controller($dev).deviceDef;

if (deviceDef.type == 'In' || deviceDef.type == 'Out') {

$ports.push($dev);

}

});

var getDesc = function(port) {

return port? port.description : '';

};

$.each($ports, function(i, $port) {

var port = controller($port);

var portDef = port.deviceDef;

var inPort;

var outPort;

if (portDef.type == 'In') {

outPort = port.getOutputs()[0];

inPort = device.addInput();

intfs.push({ node : inPort, label : portDef.label,

desc : getDesc(outPort.getInputs()[0]) });

// force disconnect test devices that connected to In-port

var inNode = port.getInputs()[0];

if (inNode.getOutput() != null) {

inNode.getOutput().disconnectFrom(inNode);

}

} else if (portDef.type == 'Out') {

inPort = port.getInputs()[0];

outPort = device.addOutput();

intfs.push({ node : outPort, label : portDef.label,

desc : getDesc(inPort.getOutput() ) });

// force disconnect test devices that connected to Out-port

var outNode = port.getOutputs()[0];

$.each(outNode.getInputs(), function(i, inNode) {

if (inNode.getOutput() != null) {

inNode.getOutput().disconnectFrom(inNode);

}

} );

}

inPort.$ui.on('nodeValueChange', function() {

outPort.setValue(inPort.getValue() );

});

});

var layout = data.layout;

var cols = layout.cols;

var rows = layout.rows;

rows = ~~( (Math.max(1, rows) + 1) / 2) \* 2;

cols = ~~( (Math.max(1, cols) + 1) / 2) \* 2;

var updateIntf = function(intf, x, y, align) {

transform(intf.node.$ui, x, y);

if (!intf.$label) {

intf.$label = createLabel(intf.label).

attr('class', 'simcir-node-label');

enableEvents(intf.$label, false);

intf.node.$ui.append(intf.$label);

}

if (align == 'right') {

intf.$label.attr('text-anchor', 'start').

attr('x', 6).

attr('y', fontSize / 2);

} else if (align == 'left') {

intf.$label.attr('text-anchor', 'end').

attr('x', -6).

attr('y', fontSize / 2);

} else if (align == 'top') {

intf.$label.attr('text-anchor', 'middle').

attr('x', 0).

attr('y', -6);

} else if (align == 'bottom') {

intf.$label.attr('text-anchor', 'middle').

attr('x', 0).

attr('y', fontSize + 6);

}

};

var doLayout = function() {

var x = 0;

var y = 0;

var w = unit \* cols / 2;

var h = unit \* rows / 2;

device.$ui.children('.simcir-device-label').

attr({y : y + h + fontSize});

device.$ui.children('.simcir-device-body').

attr({x: x, y: y, width: w, height: h});

$.each(intfs, function(i, intf) {

if (layout.nodes[intf.label] &&

layout.nodes[intf.label].match(/^([TBLR])([0-9]+)$/) ) {

var off = +RegExp.$2 \* unit / 2;

switch(RegExp.$1) {

case 'T' : updateIntf(intf, x + off, y, 'bottom'); break;

case 'B' : updateIntf(intf, x + off, y + h, 'top'); break;

case 'L' : updateIntf(intf, x, y + off, 'right'); break;

case 'R' : updateIntf(intf, x + w, y + off, 'left'); break;

}

} else {

transform(intf.node.$ui, 0, 0);

}

});

};

device.getSize = function() {

return {width: unit \* cols / 2, height: unit \* rows / 2};

};

device.$ui.on('dispose', function() {

$.each($devs, function(i, $dev) {

$dev.trigger('dispose');

});

} );

if (data.layout.hideLabelOnWorkspace) {

device.$ui.on('deviceAdd', function() {

device.$ui.children('.simcir-device-label').css('display', 'none');

}).on('deviceRemove', function() {

device.$ui.children('.simcir-device-label').css('display', '');

});

}

device.$ui.on('dblclick', function(event) {

// open library,

event.preventDefault();

event.stopPropagation();

showDialog(device.deviceDef.label || device.deviceDef.type,

setupSimcir($('<div></div>'), data) ).on('close', function() {

$(this).find('.simcir-workspace').trigger('dispose');

});

});

var super\_createUI = device.createUI;

device.createUI = function() {

super\_createUI();

doLayout();

};

};

};

var factories = {};

var defaultToolbox = [];

//toolboxda bulunacak kapıları ekleyen fonksiyon

var registerDevice = function(type, factory, deprecated) {

if (typeof factory == 'object') {

if (typeof factory.layout == 'object') {

factory = createCustomLayoutDeviceRefFactory(factory);

} else {

factory = createDeviceRefFactory(factory);

}

}

factories[type] = factory;

if (!deprecated) {

defaultToolbox.push({type: type});

}

};

//scrollbar fonksiyonu

var createScrollbar = function() {

// vertical only.

var \_value = 0;

var \_min = 0;

var \_max = 0;

var \_barSize = 0;

var \_width = 0;

var \_height = 0;

var $body = createSVGElement('rect');

var $bar = createSVGElement('g').

append(createSVGElement('rect') ).

attr('class', 'simcir-scrollbar-bar');

var $scrollbar = createSVGElement('g').

attr('class', 'simcir-scrollbar').

append($body).append($bar).

on('unitup', function(event) {

setValue(\_value - unit \* 2);

}).on('unitdown', function(event) {

setValue(\_value + unit \* 2);

}).on('rollup', function(event) {

setValue(\_value - \_barSize);

}).on('rolldown', function(event) {

setValue(\_value + \_barSize);

});

var dragPoint = null;

var bar\_mouseDownHandler = function(event) {

event.preventDefault();

event.stopPropagation();

var pos = transform($bar);

dragPoint = {

x: event.pageX - pos.x,

y: event.pageY - pos.y};

$(document).on('mousemove', bar\_mouseMoveHandler);

$(document).on('mouseup', bar\_mouseUpHandler);

};

var bar\_mouseMoveHandler = function(event) {

calc(function(unitSize) {

setValue( (event.pageY - dragPoint.y) / unitSize);

});

};

var bar\_mouseUpHandler = function(event) {

$(document).off('mousemove', bar\_mouseMoveHandler);

$(document).off('mouseup', bar\_mouseUpHandler);

};

$bar.on('mousedown', bar\_mouseDownHandler);

var body\_mouseDownHandler = function(event) {

event.preventDefault();

event.stopPropagation();

var off = $scrollbar.parent('svg').offset();

var pos = transform($scrollbar);

var y = event.pageY - off.top - pos.y;

var barPos = transform($bar);

if (y < barPos.y) {

$scrollbar.trigger('rollup');

} else {

$scrollbar.trigger('rolldown');

}

};

$body.on('mousedown', body\_mouseDownHandler);

var setSize = function(width, height) {

\_width = width;

\_height = height;

layout();

};

var layout = function() {

$body.attr({x: 0, y: 0, width: \_width, height: \_height});

var visible = \_max - \_min > \_barSize;

$bar.css('display', visible? 'inline' : 'none');

if (!visible) {

return;

}

calc(function(unitSize) {

$bar.children('rect').

attr({x: 0, y: 0, width: \_width, height: \_barSize \* unitSize});

transform($bar, 0, \_value \* unitSize);

});

};

var calc = function(f) {

f(\_height / (\_max - \_min) );

};

var setValue = function(value) {

setValues(value, \_min, \_max, \_barSize);

};

var setValues = function(value, min, max, barSize) {

value = Math.max(min, Math.min(value, max - barSize) );

var changed = (value != \_value);

\_value = value;

\_min = min;

\_max = max;

\_barSize = barSize;

layout();

if (changed) {

$scrollbar.trigger('scrollValueChange');

}

};

var getValue = function() {

return \_value;

};

controller($scrollbar, {

setSize: setSize,

setValues: setValues,

getValue: getValue

});

return $scrollbar;

};

var getUniqueId = function() {

var uniqueIdCount = 0;

return function() {

return 'simcir-id' + uniqueIdCount++;

};

}();

//board oluşturan fonksiyon

var createWorkspace = function(data) {

data = $.extend({

width: 400,

height: 200,

showToolbox: true,

editable: true,

toolbox: defaultToolbox,

devices: [],

connectors: [],

}, data);

var scope = {};

var workspaceWidth = data.width;

var workspaceHeight = data.height;

var barWidth = unit;

var toolboxWidth = data.showToolbox? unit \* 6 + barWidth : 0;

var connectorsValid = true;

var connectorsValidator = function() {

if (!connectorsValid) {

updateConnectors();

connectorsValid = true;

}

};

var $workspace = createSVG(

workspaceWidth, workspaceHeight).

attr('class', 'simcir-workspace').

on('nodeValueChange', function(event) {

connectorsValid = false;

window.setTimeout(connectorsValidator, 0);

}).

on('dispose', function() {

$(this).find('.simcir-device').trigger('dispose');

$toolboxPane.remove();

$workspace.remove();

});

disableSelection($workspace);

var $defs = createSVGElement('defs');

$workspace.append($defs);

!function() {

var patId = getUniqueId();

var pitch = unit / 2;

var w = workspaceWidth - toolboxWidth;

var h = workspaceHeight;

$defs.append(createSVGElement('pattern').

attr({id: patId, x: 0, y: 0,

width: pitch / w, height: pitch / h}).append(

createSVGElement('rect').attr('class', 'simcir-pin-hole').

attr({x: 0, y: 0, width: 1, height: 1}) ) );

$workspace.append(createSVGElement('rect').

attr({x: toolboxWidth, y: 0, width: w, height: h}).

css({fill: 'url(#' + patId + ')'}) );

}();

var $toolboxDevicePane = createSVGElement('g');

var $scrollbar = createScrollbar();

$scrollbar.on('scrollValueChange', function(event) {

transform($toolboxDevicePane, 0,

-controller($scrollbar).getValue() );

});

controller($scrollbar).setSize(barWidth, workspaceHeight);

transform($scrollbar, toolboxWidth - barWidth, 0);

var $toolboxPane = createSVGElement('g').

attr('class', 'simcir-toolbox').

append(createSVGElement('rect').

attr({x: 0, y: 0,

width: toolboxWidth,

height: workspaceHeight}) ).

append($toolboxDevicePane).

append($scrollbar).on('wheel', function(event) {

event.preventDefault();

var oe = event.originalEvent || event;

if (oe.deltaY < 0) {

$scrollbar.trigger('unitup');

} else if (oe.deltaY > 0) {

$scrollbar.trigger('unitdown');

}

});

var $devicePane = createSVGElement('g');

transform($devicePane, toolboxWidth, 0);

var $connectorPane = createSVGElement('g');

var $temporaryPane = createSVGElement('g');

enableEvents($connectorPane, false);

enableEvents($temporaryPane, false);

if (data.showToolbox) {

$workspace.append($toolboxPane);

}

$workspace.append($devicePane);

$workspace.append($connectorPane);

$workspace.append($temporaryPane);

//boarda kapı ekleyen fonksiyon

var addDevice = function($dev) {

$devicePane.append($dev);

$dev.trigger('deviceAdd');

};

//boarddan kapıları silen fonksiyon

var removeDevice = function($dev) {

$dev.trigger('deviceRemove');

// before remove, disconnect all

controller($dev).disconnectAll();

$dev.trigger('dispose');

updateConnectors();

};

//bağlantıları kaldıran fonksiyon

var disconnect = function($inNode) {

var inNode = controller($inNode);

if (inNode.getOutput() != null) {

inNode.getOutput().disconnectFrom(inNode);

}

updateConnectors();

};

var updateConnectors = function() {

$connectorPane.children().remove();

$devicePane.children('.simcir-device').each(function() {

var device = controller($(this) );

$.each(device.getInputs(), function(i, inNode) {

if (inNode.getOutput() != null) {

var p1 = offset(inNode.$ui);

var p2 = offset(inNode.getOutput().$ui);

var $conn = createConnector(p1.x, p1.y, p2.x, p2.y);

if (inNode.getOutput().getValue() != null) {

$conn.addClass('simcir-connector-hot');

}

$connectorPane.append($conn);

}

});

});

};

//toolboxı gösteren fonksiyon

var loadToolbox = function(data) {

var vgap = 8;

var y = vgap;

$.each(data.toolbox, function(i, deviceDef) {

var $dev = createDevice(deviceDef);

$toolboxDevicePane.append($dev);

var size = controller($dev).getSize();

transform($dev, (toolboxWidth - barWidth - size.width) / 2, y);

y += (size.height + fontSize + vgap);

});

controller($scrollbar).setValues(0, 0, y, workspaceHeight);

};

var getData = function() {

var devIdCount = 0;

$devicePane.children('.simcir-device').each(function() {

var $dev = $(this);

var device = controller($dev);

var devId = 'dev' + devIdCount++;

device.id = devId;

$.each(device.getInputs(), function(i, node) {

node.id = devId + '.in' + i;

});

$.each(device.getOutputs(), function(i, node) {

node.id = devId + '.out' + i;

});

});

var toolbox = [];

var devices = [];

var connectors = [];

var clone = function(obj) {

return JSON.parse(JSON.stringify(obj) );

};

$toolboxDevicePane.children('.simcir-device').each(function() {

var $dev = $(this);

var device = controller($dev);

toolbox.push(device.deviceDef);

});

$devicePane.children('.simcir-device').each(function() {

var $dev = $(this);

var device = controller($dev);

$.each(device.getInputs(), function(i, inNode) {

if (inNode.getOutput() != null) {

connectors.push({from:inNode.id, to:inNode.getOutput().id});

}

});

var pos = transform($dev);

var deviceDef = clone(device.deviceDef);

deviceDef.id = device.id;

deviceDef.x = pos.x;

deviceDef.y = pos.y;

deviceDef.label = device.getLabel();

var state = device.getState();

if (state != null) {

deviceDef.state = state;

}

devices.push(deviceDef);

});

return {

width: data.width,

height: data.height,

showToolbox: data.showToolbox,

editable: data.editable,

toolbox: toolbox,

devices: devices,

connectors: connectors

};

};

var getText = function() {

var data = getData();

var buf = '';

var print = function(s) {

buf += s;

};

var println = function(s) {

print(s);

buf += '\r\n';

};

var printArray = function(array) {

$.each(array, function(i, item) {

println(' ' + JSON.stringify(item).

replace(/</g, '\\u003c').replace(/>/g, '\\u003e') +

(i + 1 < array.length? ',' : '') );

});

};

println('{');

println(' "width":' + data.width + ',');

println(' "height":' + data.height + ',');

println(' "showToolbox":' + data.showToolbox + ',');

println(' "toolbox":[');

printArray(data.toolbox);

println(' ],');

println(' "devices":[');

printArray(data.devices);

println(' ],');

println(' "connectors":[');

printArray(data.connectors);

println(' ]');

print('}');

return buf;

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// mouse işlemleri

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

var dragMoveHandler = null;

var dragCompleteHandler = null;

var adjustDevice = function($dev) {

var pitch = unit / 2;

var adjust = function(v) { return Math.round(v / pitch) \* pitch; };

var pos = transform($dev);

var size = controller($dev).getSize();

var x = Math.max(0, Math.min(pos.x,

workspaceWidth - toolboxWidth - size.width) );

var y = Math.max(0, Math.min(pos.y,

workspaceHeight - size.height) );

transform($dev, adjust(x), adjust(y) );

};

var beginConnect = function(event, $target) {

var $srcNode = $target.closest('.simcir-node');

var off = $workspace.offset();

var pos = offset($srcNode);

if ($srcNode.attr('simcir-node-type') == 'in') {

disconnect($srcNode);

}

dragMoveHandler = function(event) {

var x = event.pageX - off.left;

var y = event.pageY - off.top;

$temporaryPane.children().remove();

$temporaryPane.append(createConnector(pos.x, pos.y, x, y) );

};

dragCompleteHandler = function(event) {

$temporaryPane.children().remove();

var $dst = $(event.target);

if (isActiveNode($dst) ) {

var $dstNode = $dst.closest('.simcir-node');

connect($srcNode, $dstNode);

updateConnectors();

}

};

};

var beginNewDevice = function(event, $target) {

var $dev = $target.closest('.simcir-device');

var pos = offset($dev);

$dev = createDevice(controller($dev).deviceDef, false, scope);

transform($dev, pos.x, pos.y);

$temporaryPane.append($dev);

var dragPoint = {

x: event.pageX - pos.x,

y: event.pageY - pos.y};

dragMoveHandler = function(event) {

transform($dev,

event.pageX - dragPoint.x,

event.pageY - dragPoint.y);

};

dragCompleteHandler = function(event) {

var $target = $(event.target);

if ($target.closest('.simcir-toolbox').length == 0) {

$dev.detach();

var pos = transform($dev);

transform($dev, pos.x - toolboxWidth, pos.y);

adjustDevice($dev);

addDevice($dev);

} else {

$dev.trigger('dispose');

}

};

};

var $selectedDevices = [];

var addSelected = function($dev) {

controller($dev).setSelected(true);

$selectedDevices.push($dev);

};

var deselectAll = function() {

$devicePane.children('.simcir-device').each(function() {

controller($(this) ).setSelected(false);

});

$selectedDevices = [];

};

var beginMoveDevice = function(event, $target) {

var $dev = $target.closest('.simcir-device');

var pos = transform($dev);

if (!controller($dev).isSelected() ) {

deselectAll();

addSelected($dev);

// to front.

$dev.parent().append($dev.detach() );

}

var dragPoint = {

x: event.pageX - pos.x,

y: event.pageY - pos.y};

dragMoveHandler = function(event) {

// disable events while dragging.

enableEvents($dev, false);

var curPos = transform($dev);

var deltaPos = {

x: event.pageX - dragPoint.x - curPos.x,

y: event.pageY - dragPoint.y - curPos.y};

$.each($selectedDevices, function(i, $dev) {

var curPos = transform($dev);

transform($dev,

curPos.x + deltaPos.x,

curPos.y + deltaPos.y);

});

updateConnectors();

};

dragCompleteHandler = function(event) {

var $target = $(event.target);

enableEvents($dev, true);

$.each($selectedDevices, function(i, $dev) {

if ($target.closest('.simcir-toolbox').length == 0) {

adjustDevice($dev);

updateConnectors();

} else {

removeDevice($dev);

}

});

};

};

var beginSelectDevice = function(event, $target) {

var intersect = function(rect1, rect2) {

return !(

rect1.x > rect2.x + rect2.width ||

rect1.y > rect2.y + rect2.height ||

rect1.x + rect1.width < rect2.x ||

rect1.y + rect1.height < rect2.y);

};

var pointToRect = function(p1, p2) {

return {

x: Math.min(p1.x, p2.x),

y: Math.min(p1.y, p2.y),

width: Math.abs(p1.x - p2.x),

height: Math.abs(p1.y - p2.y)};

};

deselectAll();

var off = $workspace.offset();

var pos = offset($devicePane);

var p1 = {x: event.pageX - off.left, y: event.pageY - off.top};

dragMoveHandler = function(event) {

deselectAll();

var p2 = {x: event.pageX - off.left, y: event.pageY - off.top};

var selRect = pointToRect(p1, p2);

$devicePane.children('.simcir-device').each(function() {

var $dev = $(this);

var devPos = transform($dev);

var devSize = controller($dev).getSize();

var devRect = {

x: devPos.x + pos.x,

y: devPos.y + pos.y,

width: devSize.width,

height: devSize.height};

if (intersect(selRect, devRect) ) {

addSelected($dev);

}

});

$temporaryPane.children().remove();

$temporaryPane.append(createSVGElement('rect').

attr(selRect).

attr('class', 'simcir-selection-rect') );

};

};

var mouseDownHandler = function(event) {

event.preventDefault();

event.stopPropagation();

var $target = $(event.target);

if (!data.editable) {

return;

}

if (isActiveNode($target) ) {

beginConnect(event, $target);

} else if ($target.closest('.simcir-device').length == 1) {

if ($target.closest('.simcir-toolbox').length == 1) {

beginNewDevice(event, $target);

} else {

beginMoveDevice(event, $target);

}

} else {

beginSelectDevice(event, $target);

}

$(document).on('mousemove', mouseMoveHandler);

$(document).on('mouseup', mouseUpHandler);

};

var mouseMoveHandler = function(event) {

if (dragMoveHandler != null) {

dragMoveHandler(event);

}

};

var mouseUpHandler = function(event) {

if (dragCompleteHandler != null) {

dragCompleteHandler(event);

}

dragMoveHandler = null;

dragCompleteHandler = null;

$devicePane.children('.simcir-device').each(function() {

enableEvents($(this), true);

});

$temporaryPane.children().remove();

$(document).off('mousemove', mouseMoveHandler);

$(document).off('mouseup', mouseUpHandler);

};

$workspace.on('mousedown', mouseDownHandler);

loadToolbox(data);

$.each(buildCircuit(data, false, scope), function(i, $dev) {

addDevice($dev);

});

updateConnectors();

controller($workspace, {

data: getData,

text: getText

});

return $workspace;

};

var clearSimcir = function($placeHolder) {

$placeHolder = $($placeHolder[0]);

$placeHolder.find('.simcir-workspace').trigger('dispose');

$placeHolder.children().remove();

return $placeHolder;

};

var setupSimcir = function($placeHolder, data) {

$placeHolder = clearSimcir($placeHolder);

var $workspace = simcir.createWorkspace(data);

var $dataArea = $('<textarea></textarea>').

addClass('simcir-json-data-area').

attr('readonly', 'readonly').

css('width', $workspace.attr('width') + 'px').

css('height', $workspace.attr('height') + 'px');

var showData = false;

var toggle = function() {

$workspace.css('display', !showData? 'inline' : 'none');

$dataArea.css('display', showData? 'inline' : 'none');

if (showData) {

$dataArea.val(controller($workspace).text() ).focus();

}

showData = !showData;

};

$placeHolder.text('');

$placeHolder.append($('<div></div>').

addClass('simcir-body').

append($workspace).

append($dataArea).

on('click', function(event) {

if (event.ctrlKey || event.metaKey) {

toggle();

}

}));

toggle();

return $placeHolder;

};

var setupSimcirDoc = function($placeHolder) {

var $table = $('<table><tbody></tbody></table>').

addClass('simcir-doc-table');

$.each(defaultToolbox, function(i, deviceDef) {

var $dev = createDevice(deviceDef);

var device = controller($dev);

if (!device.doc) {

return;

}

var doc = $.extend({description: '', params: []},device.doc);

var size = device.getSize();

var $tr = $('<tr></tr>');

var hgap = 32;

var vgap = 8;

var $view = createSVG(size.width + hgap \* 2,

size.height + vgap \* 2 + fontSize);

var $dev = createDevice(deviceDef);

transform($dev, hgap, vgap);

$view.append($dev);

$tr.append($('<td></td>').css('text-align', 'center').append($view) );

var $desc = $('<td></td>');

$tr.append($desc);

if (doc.description) {

$desc.append($('<span></span>').

text(doc.description) );

}

$desc.append($('<div>Params</div>').addClass('simcir-doc-title') );

var $paramsTable = $('<table><tbody></tbody></table>').

addClass('simcir-doc-params-table');

$paramsTable.children('tbody').append($('<tr></tr>').

append($('<th>Name</th>') ).

append($('<th>Type</th>') ).

append($('<th>Default</th>') ).

append($('<th>Description</th>') ) );

$paramsTable.children('tbody').append($('<tr></tr>').

append($('<td>type</td>') ).

append($('<td>string</td>') ).

append($('<td>-</td>').

css('text-align', 'center') ).

append($('<td>"' + deviceDef.type + '"</td>') ) );

if (!doc.labelless) {

$paramsTable.children('tbody').append($('<tr></tr>').

append($('<td>label</td>') ).

append($('<td>string</td>') ).

append($('<td>same with type</td>').css('text-align', 'center') ).

append($('<td>label for a device.</td>') ) );

}

if (doc.params) {

$.each(doc.params, function(i, param) {

$paramsTable.children('tbody').append($('<tr></tr>').

append($('<td></td>').text(param.name) ).

append($('<td></td>').text(param.type) ).

append($('<td></td>').css('text-align', 'center').

text(param.defaultValue) ).

append($('<td></td>').text(param.description) ) );

});

}

$desc.append($paramsTable);

if (doc.code) {

$desc.append($('<div>Code</div>').addClass('simcir-doc-title') );

$desc.append($('<div></div>').

addClass('simcir-doc-code').text(doc.code) );

}

$table.children('tbody').append($tr);

});

$placeHolder.append($table);

};

$(function() {

$('.simcir').each(function() {

var $placeHolder = $(this);

var text = $placeHolder.text().replace(/^\s+|\s+$/g, '');

setupSimcir($placeHolder, JSON.parse(text || '{}') );

});

});

$(function() {

$('.simcir-doc').each(function() {

setupSimcirDoc($(this) );

});

});

$.extend($s, {

registerDevice: registerDevice,

clearSimcir: clearSimcir,

setupSimcir: setupSimcir,

createWorkspace: createWorkspace,

createSVGElement: createSVGElement,

offset: offset,

transform: transform,

enableEvents: enableEvents,

graphics: graphics,

controller: controller,

unit: unit

});

}(simcir);

//

// devrelerin oluşturulması

//

!function($s) {

'use strict';

var $ = $s.$;

// birim boyutu

var unit = $s.unit;

var connectNode = function(in1, out1) {

// input değeri ekleme

var in1\_super\_setValue = in1.setValue;

in1.setValue = function(value, force) {

var changed = in1.getValue() !== value;

in1\_super\_setValue(value, force);

if (changed || force) {

out1.setValue(in1.getValue() );

}

};

};

var createPortFactory = function(type) {

return function(device) {

var in1 = device.addInput();

var out1 = device.addOutput();

connectNode(in1, out1);

var super\_createUI = device.createUI;

device.createUI = function() {

super\_createUI();

var size = device.getSize();

var cx = size.width / 2;

var cy = size.height / 2;

device.$ui.append($s.createSVGElement('circle').

attr({cx: cx, cy: cy, r: unit / 2}).

attr('class', 'simcir-port simcir-node-type-' + type) );

device.$ui.append($s.createSVGElement('circle').

attr({cx: cx, cy: cy, r: unit / 4}).

attr('class', 'simcir-port-hole') );

};

};

};

var createJointFactory = function() {

var maxFadeCount = 16;

var fadeTimeout = 100;

var Direction = { WE : 0, NS : 1, EW : 2, SN : 3 };

return function(device) {

var in1 = device.addInput();

var out1 = device.addOutput();

connectNode(in1, out1);

var state = device.deviceDef.state || { direction : Direction.WE };

device.getState = function() {

return state;

};

device.getSize = function() {

return { width : unit, height : unit };

};

var super\_createUI = device.createUI;

device.createUI = function() {

super\_createUI();

var $label = device.$ui.children('.simcir-device-label');

$label.attr('y', $label.attr('y') - unit / 4);

var $point = $s.createSVGElement('circle').

css('pointer-events', 'none').css('opacity', 0).attr('r', 2).

addClass('simcir-connector').addClass('simcir-joint-point');

device.$ui.append($point);

var $path = $s.createSVGElement('path').

css('pointer-events', 'none').css('opacity', 0).

addClass('simcir-connector');

device.$ui.append($path);

var $title = $s.createSVGElement('title').

text('Double-Click to change a direction.');

var updatePoint = function() {

$point.css('display', out1.getInputs().length > 1? '' : 'none');

};

updatePoint();

var super\_connectTo = out1.connectTo;

out1.connectTo = function(inNode) {

super\_connectTo(inNode);

updatePoint();

};

var super\_disconnectFrom = out1.disconnectFrom;

out1.disconnectFrom = function(inNode) {

super\_disconnectFrom(inNode);

updatePoint();

};

var updateUI = function() {

var x0, y0, x1, y1;

x0 = y0 = x1 = y1 = unit / 2;

var d = unit / 2;

var direction = state.direction;

if (direction == Direction.WE) {

x0 -= d;

x1 += d;

} else if (direction == Direction.NS) {

y0 -= d;

y1 += d;

} else if (direction == Direction.EW) {

x0 += d;

x1 -= d;

} else if (direction == Direction.SN) {

y0 += d;

y1 -= d;

}

$path.attr('d', 'M' + x0 + ' ' + y0 + 'L' + x1 + ' ' + y1);

$s.transform(in1.$ui, x0, y0);

$s.transform(out1.$ui, x1, y1);

$point.attr({cx : x1, cy : y1});

if (direction == Direction.EW || direction == Direction.WE) {

device.$ui.children('.simcir-device-body').

attr({x: 0, y: unit / 4, width: unit, height: unit / 2});

} else {

device.$ui.children('.simcir-device-body').

attr({x: unit / 4, y: 0, width: unit / 2, height: unit});

}

};

updateUI();

// fadeout a body.

var fadeCount = 0;

var setOpacity = function(opacity) {

device.$ui.children('.simcir-device-body,.simcir-node').

css('opacity', opacity);

$path.css('opacity', 1 - opacity);

$point.css('opacity', 1 - opacity);

};

var fadeout = function() {

window.setTimeout(function() {

if (fadeCount > 0) {

fadeCount -= 1;

setOpacity(fadeCount / maxFadeCount);

fadeout();

}

}, fadeTimeout);

};

var isEditable = function($dev) {

var $workspace = $dev.closest('.simcir-workspace');

return !!$s.controller($workspace).data().editable;

};

var device\_mouseoutHandler = function(event) {

if (!isEditable($(event.target) ) ) {

return;

}

if (!device.isSelected() ) {

fadeCount = maxFadeCount;

fadeout();

}

};

var device\_dblclickHandler = function(event) {

if (!isEditable($(event.target) ) ) {

return;

}

state.direction = (state.direction + 1) % 4;

updateUI();

// update connectors.

$(this).trigger('mousedown').trigger('mouseup');

};

device.$ui.on('mouseover', function(event) {

if (!isEditable($(event.target) ) ) {

$title.text('');

return;

}

setOpacity(1);

fadeCount = 0;

}).on('deviceAdd', function() {

if ($(this).closest('BODY').length == 0) {

setOpacity(0);

}

$(this).append($title).on('mouseout', device\_mouseoutHandler).

on('dblclick', device\_dblclickHandler);

// hide a label

$label.css('display', 'none');

}).on('deviceRemove', function() {

$(this).off('mouseout', device\_mouseoutHandler).

off('dblclick', device\_dblclickHandler);

$title.remove();

// show a label

$label.css('display', '');

}).on('deviceSelect', function() {

if (device.isSelected() ) {

setOpacity(1);

fadeCount = 0;

} else {

if (fadeCount == 0) {

setOpacity(0);

}

}

});

};

};

};

// Nodeları oluşturan fonksiyonların çağırılması

$s.registerDevice('In', createPortFactory('in') );

$s.registerDevice('Out', createPortFactory('out') );

$s.registerDevice('Joint', createJointFactory() );

}(simcir);

Simcir-basicset.js dosyası, bu dosyayı temel mantık elemanlarımızı tanımlamak için kullanıyoruz.

!function($s) {

'use strict';

var $ = $s.$;

var unit = $s.unit;

var defaultLEDColor = '#ff0000';

var defaultLEDBgColor = '#000000';

var multiplyColor = function() {

var HEX = '0123456789abcdef';

var toIColor = function(sColor) {

if (!sColor) {

return 0;

}

sColor = sColor.toLowerCase();

if (sColor.match(/^#[0-9a-f]{3}$/i) ) {

var iColor = 0;

for (var i = 0; i < 6; i += 1) {

iColor = (iColor << 4) | HEX.indexOf(sColor.charAt( (i >> 1) + 1) );

}

return iColor;

} else if (sColor.match(/^#[0-9a-f]{6}$/i) ) {

var iColor = 0;

for (var i = 0; i < 6; i += 1) {

iColor = (iColor << 4) | HEX.indexOf(sColor.charAt(i + 1) );

}

return iColor;

}

return 0;

};

var toSColor = function(iColor) {

var sColor = '#';

for (var i = 0; i < 6; i += 1) {

sColor += HEX.charAt( (iColor >>> (5 - i) \* 4) & 0x0f);

}

return sColor;

};

var toRGB = function(iColor) {

return {

r: (iColor >>> 16) & 0xff,

g: (iColor >>> 8) & 0xff,

b: iColor & 0xff};

};

var multiplyColor = function(iColor1, iColor2, ratio) {

var c1 = toRGB(iColor1);

var c2 = toRGB(iColor2);

var mc = function(v1, v2, ratio) {

return ~~Math.max(0, Math.min( (v1 - v2) \* ratio + v2, 255) );

};

return (mc(c1.r, c2.r, ratio) << 16) |

(mc(c1.g, c2.g, ratio) << 8) | mc(c1.b, c2.b, ratio);

};

return function(color1, color2, ratio) {

return toSColor(multiplyColor(

toIColor(color1), toIColor(color2), ratio) );

};

}();

// kapı sembollerinin çizimleri

var drawBUF = function(g, x, y, width, height) {

g.moveTo(x, y);

g.lineTo(x + width, y + height / 2);

g.lineTo(x, y + height);

g.lineTo(x, y);

g.closePath(true);

};

var drawAND = function(g, x, y, width, height) {

g.moveTo(x, y);

g.curveTo(x + width, y, x + width, y + height / 2);

g.curveTo(x + width, y + height, x, y + height);

g.lineTo(x, y);

g.closePath(true);

};

var drawOR = function(g, x, y, width, height) {

var depth = width \* 0.2;

g.moveTo(x, y);

g.curveTo(x + width - depth, y, x + width, y + height / 2);

g.curveTo(x + width - depth, y + height, x, y + height);

g.curveTo(x + depth, y + height, x + depth, y + height / 2);

g.curveTo(x + depth, y, x, y);

g.closePath(true);

};

var drawNOT = function(g, x, y, width, height) {

drawBUF(g, x - 1, y, width - 2, height);

g.drawCircle(x + width - 1, y + height / 2, 2);

};

// logical functions

var AND = function(a, b) { return a & b; };

var OR = function(a, b) { return a | b; };

var BUF = function(a) { return (a == 1)? 1 : 0; };

var NOT = function(a) { return (a == 1)? 0 : 1; };

var onValue = 1;

var offValue = null;

var isHot = function(v) { return v != null; };

var intValue = function(v) { return isHot(v)? 1 : 0; };

//mantık kapılarını oluşturmak için kullandığımız fonksiyon

var createLogicGateFactory = function(op, out, draw) {

return function(device) {

var numInputs = (op == null)? 1 :

Math.max(2, device.deviceDef.numInputs || 2);

device.halfPitch = numInputs > 2;

for (var i = 0; i < numInputs; i += 1) {

device.addInput();

}

device.addOutput();

var inputs = device.getInputs();

var outputs = device.getOutputs();

device.$ui.on('inputValueChange', function() {

var b = intValue(inputs[0].getValue() );

if (op != null) {

for (var i = 1; i < inputs.length; i += 1) {

b = op(b, intValue(inputs[i].getValue() ) );

}

}

b = out(b);

outputs[0].setValue( (b == 1)? 1 : null);

});

var super\_createUI = device.createUI;

device.createUI = function() {

super\_createUI();

var size = device.getSize();

var g = $s.graphics(device.$ui);

g.attr['class'] = 'simcir-basicset-symbol';

draw(g,

(size.width - unit) / 2,

(size.height - unit) / 2,

unit, unit);

if (op != null) {

device.doc = {

params: [

{name: 'numInputs', type: 'number',

defaultValue: 2,

description: 'number of inputs.'}

],

code: '{"type":"' + device.deviceDef.type + '","numInputs":2}'

};

}

};

};

};

// 1 değeri gönderen devre elemanı

$s.registerDevice('1', function(device) {

device.addOutput();

var super\_createUI = device.createUI;

device.createUI = function() {

super\_createUI();

device.$ui.addClass('simcir-basicset-dc');

};

device.$ui.on('deviceAdd', function() {

device.getOutputs()[0].setValue(onValue);

});

device.$ui.on('deviceRemove', function() {

device.getOutputs()[0].setValue(null);

});

});

//göstermelik olarak da olsa 0 değeri gönderen devre elemanı

//boş bir input zaten 0 değerini alıyor fakat gösterebilmek için böyle bir

//devre elemanı tasarladım

$s.registerDevice('0', function(device) {

device.addOutput();

var super\_createUI = device.createUI;

device.createUI = function() {

super\_createUI();

device.$ui.addClass('simcir-basicset-dc');

};

device.$ui.on('deviceAdd', function() {

device.getOutputs()[0].setValue(null);

});

device.$ui.on('deviceRemove', function() {

device.getOutputs()[0].setValue(null);

});

});

// Led Devre eleamnını ekleyen fonksiyon

$s.registerDevice('LED', function(device) {

var in1 = device.addInput();

var super\_createUI = device.createUI;

device.createUI = function() {

super\_createUI();

var hiColor = device.deviceDef.color || defaultLEDColor;

var bgColor = device.deviceDef.bgColor || defaultLEDBgColor;

var loColor = multiplyColor(hiColor, bgColor, 0.25);

var bLoColor = multiplyColor(hiColor, bgColor, 0.2);

var bHiColor = multiplyColor(hiColor, bgColor, 0.8);

var size = device.getSize();

var $ledbase = $s.createSVGElement('circle').

attr({cx: size.width / 2, cy: size.height / 2, r: size.width / 4}).

attr('stroke', 'none').

attr('fill', bLoColor);

device.$ui.append($ledbase);

var $led = $s.createSVGElement('circle').

attr({cx: size.width / 2, cy: size.height / 2, r: size.width / 4 \* 0.8}).

attr('stroke', 'none').

attr('fill', loColor);

device.$ui.append($led);

device.$ui.on('inputValueChange', function() {

$ledbase.attr('fill', isHot(in1.getValue() )? bHiColor : bLoColor);

$led.attr('fill', isHot(in1.getValue() )? hiColor : loColor);

});

device.doc = {

params: [

{name: 'color', type: 'string',

defaultValue: defaultLEDColor,

description: 'color in hexadecimal.'},

{name: 'bgColor', type: 'string',

defaultValue: defaultLEDBgColor,

description: 'background color in hexadecimal.'}

],

code: '{"type":"' + device.deviceDef.type +

'","color":"' + defaultLEDColor + '"}'

};

};

});

// mantık kapılarını oluşturan fonksiyonların çağırılması

$s.registerDevice('NOT', createLogicGateFactory(null, NOT, drawNOT) );

$s.registerDevice('AND', createLogicGateFactory(AND, BUF, drawAND) );

$s.registerDevice('OR', createLogicGateFactory(OR, BUF, drawOR) );

}(simcir);

Devre simülasyonu.html dosyası

<!doctype html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />

<script type="text/javascript" src="simcir.js"></script>

<link rel="stylesheet" type="text/css" href="simcir.css" />

<script type="text/javascript" src="simcir-basicset.js"></script>

<link rel="stylesheet" type="text/css" href="simcir-basicset.css" />

<style type="text/css">

\* { font-family: arial, sans-serif; }

.code {

margin: 16px 0px 16px 0px;

padding: 4px;

border-width: 0px 0px 0px 4px;

border-color: #cccccc;

border-style: solid;

background-color: #f0f0f0;

white-space: pre;

}

</style>

<title>Logic Circuit Design</title>

</head>

<body>

<h1>Cumhuriyet Üniversitesi Mühendislik Fakültesi</h1>

<h1>Bilgisayar Mühendisliği</h1>

<h2>Mühendislik Projesi 3. Ödevi: Logic Devre Tasarım Simülasyonu</h2>

<div class="simcir">

{

"width":1100,

"height":550,

"showToolbox":true,

"toolbox":[

{"type":"1"},

{"type":"0"},

{"type":"LED","color":"#00ff00","label":"LED(G)"},

{"type":"AND","numInputs":"3","label":"AND(3in)"},

{"type":"OR","numInputs":"3","label":"OR(3in)"},

{"type":"AND"},

{"type":"OR"},

{"type":"NOT"}

],

"devices":[

],

"connectors":[

]

}

</div>

<h2>Enes Yıldız 2014123070 I. Öğretim</h2>

</body>

</html>