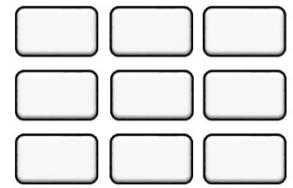


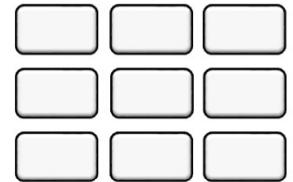
Chapter 3



RT Level Modeling with C++

Zainalabedin Navabi

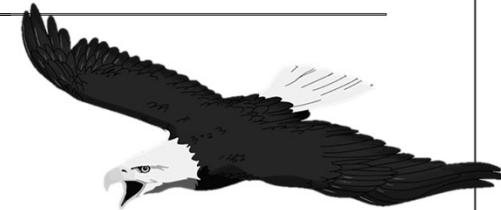
Slides prepared by: Hanieh Hashemi



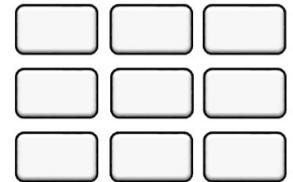
RT Level Modeling in C++

Introduction

- + Bus Model and Operations
- + Basic Elements of RTL
- + RTL Design Examples



A BIRD'S-EYE VIEW
OF OUTLINE



RT Level Modeling with C++

Introduction

– Bus Model and Operations

 Array Attributes

 Logical Operations

 Adding Operations

 Relational Operations

 IO Operations

– Basic Elements of RTL

 Combinational elements

 Sequential elements

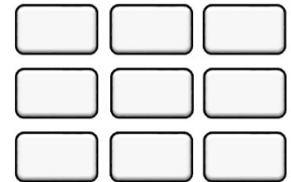
– RTL Design Examples

 Memory Structure

 Moore Sequence Detector (11011)

 Least- Recently-Used (LRU) Circuit

 Exponential Circuit



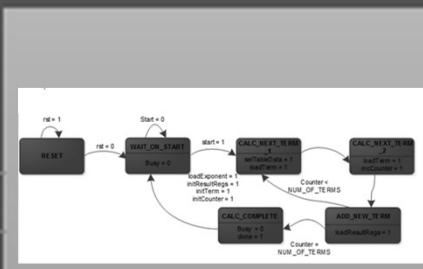
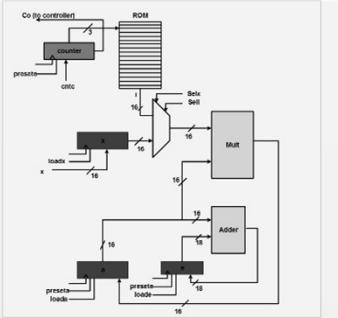
Outline

Introduction

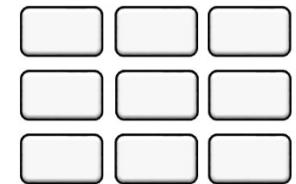
- + Bus Model and Operations
- + Basic Elements of RTL
- + RTL Design Examples

RTL Principles

Elements of an RT level component

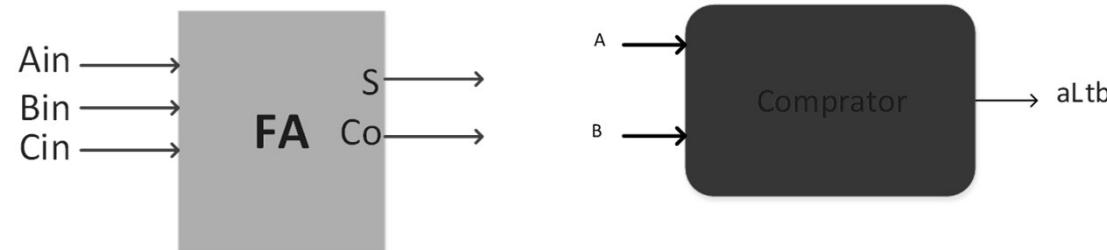


Elements of Datapath



- Combinational Components

- Adders
- Comparators
- Multiplexers
- ALUs

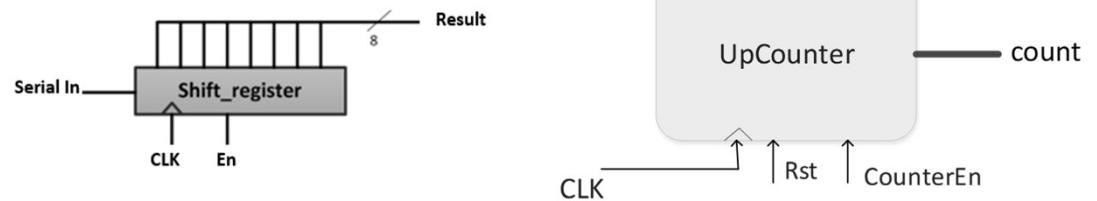


- Logical Operations

- Vector based
- Scalar operations
- Mixed

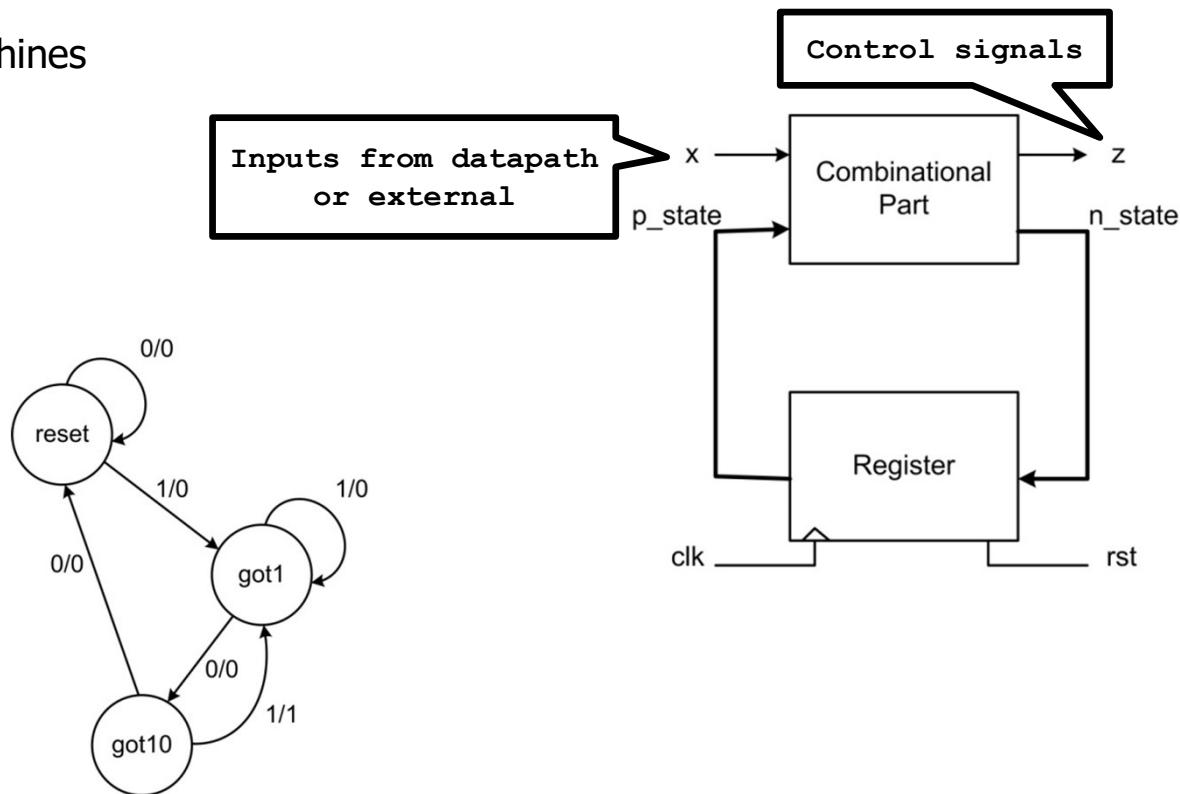
Elements of Datapath

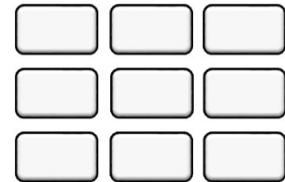
- RTL internal buses
 - Unconstrained
 - Represent multi-value logic system
- Sequential component
 - Registers
 - Registers with some functionality
 - Register files



Elements of Control Unit

- One or more state machines
 - Huffman style





Outline

Introduction

- Bus Model and Operations

 Bus Communications

 Array Attributes

 Logical Operations

 Adding Operations

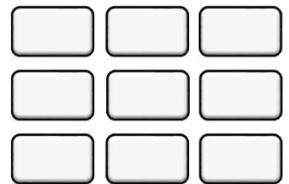
 Relational Operations

 IO Operations

+ Basic Elements of RTL

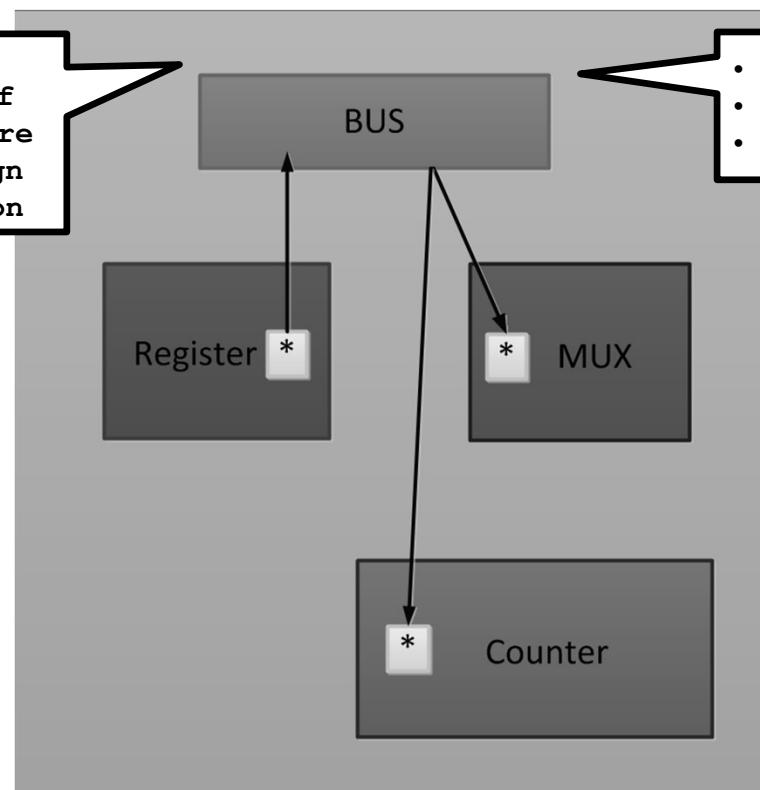
+ RTL Design Examples

Bus Communications



As design abstraction approaches ESL, role of communication become more pronounced in the design and hardware description

- Bus owns data
- Components look up for data
- Operators are overloaded



Bus Communications

The screenshot shows a code editor with two tabs: `classVectorPrimitives.h` and `utilityFunctions.cpp`. The `classVectorPrimitives.h` tab is active, displaying the following code:

```
#include <iostream>
#include "utilityFunctions.h"
#include <string>
using namespace std;

#define MIN(a,b) ((a<b)?a:b);
#define MAX(a,b) ((a>b)?a:b);

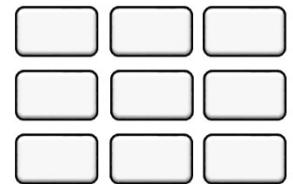
class bus{
    string v;
public:
    bus() { v.resize(1, 'X'); }
    bus(int SIZE) { v.resize(SIZE, 'X'); }
    bus(int SIZE, char c) { v.resize(SIZE, c); }
    bus(const string& s) { v = s; }
    bus(const char* c) { v = c; }
    bus(const bus& a) { v = a.v; } // Copy constructor for =
    bus range(int i1, int i2){ ... }
    bus at(int i){ ... }
    char operator[](int i) const{ ... }
    char& operator[](int i){ ... }
    int length(){ ... }
    void fill(char c){ ... }
    friend bus operator& (bus a, bus b){ ... }
    friend bus operator| (bus a, bus b){ ... }
    friend bus operator^ (bus a, bus b){ ... }
    friend bus operator~ (bus a){ ... }
};
```

Callouts from the slide point to specific parts of the code:

- Default Constructor when no arguments passed**: Points to the first constructor definition.
- Multi-bit constructor with given bus size**: Points to the second constructor definition.
- Initial value is given as parameter**: Points to the third constructor definition.
- Using an object of bus class in an operation with constant character pointer and string**: Points to the fourth constructor definition.
- String member variable for its logical data**: Points to the `v` member variable declaration in the `bus` class.
- Ref. [1] – Introduction to strings - P. 15-16**: A thought bubble pointing to the `#include <string>` and `using namespace std;` directives.
- Bus-arrays for multi-bit datapath buses and control unit vector while 1-bit buses for Controller signals, one-bit datapath signal and control unit internal wires.**: Points to the `range`, `at`, and `operator[]` methods.
- Copy constructor**: Points to the `bus` constructor that takes another `bus` object as a parameter.
- The bus class we use for our interconnection uses string variable**: Points to the `bus` class definition.
- 1. Char Is the best for representation of various logic values**: Part of a numbered list.
- 2. Compatibility with C++ string class**: Part of a numbered list.
- Shortcoming? Lack of logical operation**: Part of a numbered list.

Ref. [1]: J. Soulie, C++ Language Tutorial, 2007.

Utility Functions



The screenshot shows a code editor window with multiple tabs at the top: classVectorPrimitives.h, classVectorFunctions.h, classVectorPrimitives.cpp, and utilityFunctions.h. The utilityFunctions.h tab is active, displaying the following C++ code:

```
1 char and(char a, char b);
2 char or(char a, char b);
3 char not(char a);
4 char tri(char a, char c);
5 char resolve(char a, char c);
6 char xor(char a, char b);

7
8 void fullAdder(char a, char b, char ci, char& co, char& sum);
9
```

A callout box labeled "utilityFunctions.h" points to the tab. Another callout box contains the following text:

The bus class we use for our interconnection uses string variable.
Why char?
1. Char Is the best for representation of various logic values
2. Compatibility with c++ string class
Shortcoming?
Lack of logical operation

Utility Functions

The screenshot shows a code editor window with several tabs at the top: "classVectorPrimitives.cpp", "classVectorFunctions.h", "classVectorPrimitives.h*", and "utilityFunctions.cpp". The "utilityFunctions.cpp" tab is active, displaying the following C++ code:

```
1 char and(char a, char b)
2 {
3     if ((a == '0') || (b == '0')) return '0';
4     else if ((a == '1') && (b == '1')) return '1';
5     else return 'X';
6 }
7
8 char or(char a, char b){ ... }
14
15 char not(char a){ ... }
21
22 char tri(char a, char c)
23 {
24     if (c == '1') return a;
25     else return 'Z';
26 }
27
28 char resolve(char a, char b)
29 {
30     if (a == 'Z' || a == b) return b;
31     else if (b == 'Z') return a;
32     else return 'X';
33 }
34
35 char xor(char a, char b){ ... }
41
42 void fullAdder(char a, char b, char ci, char & co, char & sum)
43 {
44     char axb, ab, abc;
45
46     axb = xor(a, b);
47     ab = and(a, b);
48     abc = and(axb, ci);
49     co = or(ab, abc);
50     sum = xor(axb, ci);
51 }
52
```

A callout box points from the text "utilityFunctions.cpp" to the code editor window. Another callout box points from the text "Full adder implementation using primitives" to the "fullAdder" function definition.

Array Attributes

Bus slicing

```
bus range(int i1, int i2)
{
    int left = MAX(i1, i2);
    int rite = MIN(i1, i2);
    bus slice(left-rite, 'X');
    int vSize = v.length();
    slice.v = v.substr(vSize - left, vSize - rite);
    return slice;
}

bus at(int i)
{
    bus bit(1, 'X');
    int vSize = v.length();
    bit.v = v.at(vSize -1 - i);
    return bit;
}

char operator[](int i) const { ... }

char& operator[](int i) { ... }

int length() { ... }

void fill(char c) { ... }

friend bus operator& (bus a, bus b) { ... }

friend bus operator| (bus a, bus b) { ... }

friend bus operator^ (bus a, bus b) { ... }

friend bus operator~ (bus a) { ... }

friend bus operator+ (const bus a, const bus b)
{
    int aSize = a.v.length();
```

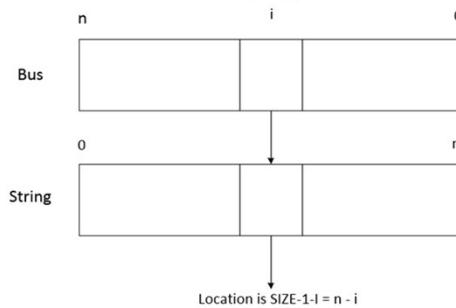
String uses 0 for its left character but range takes the larger index for left character

Array Attributes

The diagram illustrates the relationship between the `classVectorPrimitives.h` header file and the concept of **Bus indexing**. A bracket on the right side of the code editor highlights the `at` method, which is then enclosed in a box labeled **Bus indexing**. Another bracket on the right side of the code editor highlights the `operator[]` methods, which are also enclosed in a box labeled **Bus indexing**. This indicates that both the `at` method and the `operator[]` methods implement bus indexing.

```
classVectorPrimitives.h  X  classVectorFunctions.h  utilityFunctions.h  utilityFunctions.cpp
RTL Description  (Global Scope)

19  bus range(int i1, int i2)
20  {
21      int left = MAX(i1, i2);
22      int rite = MIN(i1, i2);
23      bus slice(left-rite, 'X');
24      int vSize = v.length();
25      slice.v = v.substr(vSize - left, vSize - rite);
26      return slice;
27  }
28
29  bus at(int i)
30  {
31      bus bit(1, 'X');
32      int vSize = v.length();
33      bit.v = v.at(vSize - 1 - i);
34      return bit;
35  }
36
37  char operator[](int i) const { ... }
38
39  char& operator[](int i) { ... }
40
41  int length() { ... }
42
43  void fill(char c) { ... }
44
45  friend bus operator& (bus a, bus b) { ... }
46
47  friend bus operator| (bus a, bus b) { ... }
48
49  friend bus operator^ (bus a, bus b) { ... }
50
51  friend bus operator~ (bus a) { ... }
52
53  friend bus operator+ (const bus a, const bus b)
54  {
55      int aSize = a.v.length();
```



Array Attributes

The screenshot shows a code editor window with several tabs at the top: `classVectorPrimitives.cpp`, `classVectorFunctions.h`, `classVectorPrimitives.h*`, and `utilityFunctions.cpp`. The `classVectorPrimitives.h*` tab is active. The code in the editor is as follows:

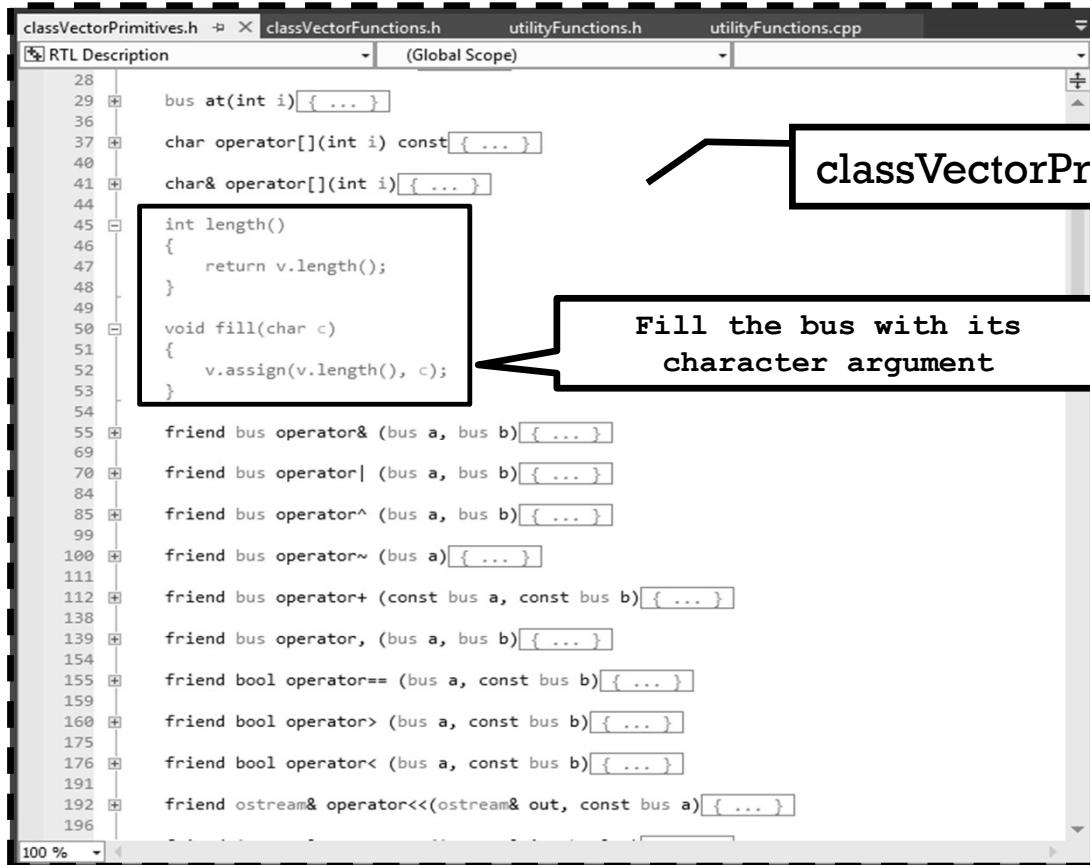
```
28 bus at(int i){ ... }
29
30 char operator[](int i) const {
31     return v[v.length()-i-1];
32 }
33 char& operator[](int i) {
34     return v[v.length()-i-1];
35 }
36 int length(){ ... }
37
38 void fill(char c){ ... }
39
40 friend bus operator& (bus a, bus b){ ... }
41
42 friend bus operator| (bus a, bus b){ ... }
43
44 friend bus operator^ (bus a, bus b){ ... }
45
46 friend bus operator~ (bus a){ ... }
47
48 friend bus operator+ (const bus a, const bus b){ ... }
49
50 friend bus operator, (bus a, bus b){ ... }
51
52 friend bool operator== (bus a, const bus b){ ... }
53
54 friend bool operator> (bus a, const bus b){ ... }
55
56 friend bool operator< (bus a, const bus b){ ... }
57
58 bool operator&& (bus b){ ... }
59
60 bool operator|| (bus b){ ... }
```

Annotations in callout boxes explain specific parts of the code:

- It returns `char` type instead of `bus` type in `at()`**: Points to the declaration `char operator[](int i) const { return v[v.length()-i-1]; }`.
- Bracket overloading for using indexing on the left hand side**: Points to the declaration `char& operator[](int i) { return v[v.length()-i-1]; }`.

A large bracket on the right side of the editor window points to the file name **classVectorPrimitives.h**.

Array Attributes



```
classVectorPrimitives.h  X  classVectorFunctions.h  utilityFunctions.h  utilityFunctions.cpp
RTL Description  (Global Scope)

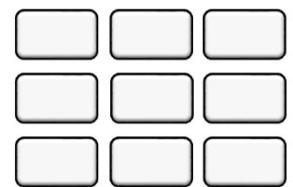
28 bus at(int i){ ... }
29
30 char operator[](int i) const{ ... }
31
32 char& operator[](int i){ ... }
33
34 int length()
35 {
36     return v.length();
37 }
38
39 void fill(char c)
40 {
41     v.assign(v.length(), c);
42 }

43 friend bus operator& (bus a, bus b){ ... }
44
45 friend bus operator| (bus a, bus b){ ... }
46
47 friend bus operator^ (bus a, bus b){ ... }
48
49 friend bus operator~ (bus a){ ... }
50
51 friend bus operator+ (const bus a, const bus b){ ... }
52
53 friend bus operator, (bus a, bus b){ ... }
54
55 friend bool operator== (bus a, const bus b){ ... }
56
57 friend bool operator> (bus a, const bus b){ ... }
58
59 friend bool operator< (bus a, const bus b){ ... }
60
61 friend ostream& operator<<(ostream& out, const bus a){ ... }
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
```

classVectorPrimitives.h

Fill the bus with its
character argument

Logical Operations



Declared as
friends inside
the bus class

```
100 friend bus operator~ (bus a) { ... }  
142 friend bus operator+ (const bus a, const bus b) { ... }  
157 friend bus operator, (bus a, bus b) { ... }  
158 friend bool operator== (bus a, const bus b) { ... }  
162 friend bool operator> (bus a, const bus b) { ... }  
178 friend bool operator< (bus a, const bus b) { ... }  
179  
194
```

classVectorPrimitives.h

Logical Operations

classVectorPrimitives.h

```
50 void fill(char c) { ... }
54
55 friend bus operator& (bus a, bus b)
56 {
57     int aSize = a.v.length();
58     int bSize = b.v.length();
59     int rSize;
60     if (bSize == 1) rSize = aSize; else rSize = MIN(aSize, bSize);
61     bus r(rSize, 'X');
62     int i;
63     for (i = rSize - 1; i >= 0; i--) {
64         if (bSize == 1) r.v[i] = and(a.v.at(i), b.v.at(0));
65         else r.v[i] = and(a.v.at(i), b.v.at(i));
66     }
67     return r;
68 }
69
70 friend bus operator| (bus a, bus b) { ... }
71
72 friend bus operator^ (bus a, bus b)
73 {
74     int aSize = a.v.length();
75     int bSize = b.v.length();
76     int rSize;
77     if (bSize == 1) rSize = aSize; else rSize = MIN(aSize, bSize);
78     bus r(rSize, 'X');
79     int i;
80     for (i = rSize - 1; i >= 0; i--) {
81         if (bSize == 1) r.v[i] = xor(a.v.at(i), b.v.at(0));
82         else r.v[i] = xor(a.v.at(i), b.v.at(i));
83     }
84     return r;
85 }
86
87 friend bus operator~ (bus a) { ... }
88
89 friend bus operator+ (const bus a, const bus b) { ... }
```

Adding Operations

The screenshot shows a code editor window with several tabs at the top: SeqDetector.cpp, utilityFunctions.h, SeqDetector.h, classVectorPrimitives.h, and classVectorFunctions.h. The classVectorPrimitives.h tab is active. The code in the editor is as follows:

```
111 friend bus operator+ (const bus a, const bus b)
112 {
113     int aSize = a.v.length();
114     int bSize = b.v.length();
115     int rSize;
116     int min = MIN(aSize, bSize);
117     if (bSize == 1) rSize = aSize; else rSize = min + 1;
118     bus r(rSize, 'X');
119
120     char ci('0');
121     char co('0'), sum('0');
122     if (bSize == 1){
123         for (int i = rSize - 1; i >= 0; i--) {
124             if (i == rSize - 1) fullAdder(a.v.at(i), b.v.at(0), ci, co, sum);
125             else fullAdder(a.v.at(i), '0', ci, co, sum);
126             ci = co;
127             r.v[i] = sum;
128         }
129     } else {
130         for (int i = rSize - 1; i >= 1; i--) {
131             fullAdder(a.v.at(i - 1), b.v.at(i - 1), ci, co, sum);
132             ci = co;
133             r.v[i] = sum;
134         }
135         r.v[0] = co;
136     }
137     return r;
138 }
139
140
141 friend bus operator* (bus a, bus b){ ... }
142
143 friend bool operator== (bus a, const bus b){ ... }
144
145 friend bool operator> (bus a, const bus b){ ... }
```

Annotations with callouts point to specific parts of the code:

- A callout labeled "String indexing" points to the line where a character is indexed from a string: `a.v[i]`.
- A callout labeled "Overloading + operator using full adder" points to the main loop of the addition operator implementation.
- A callout labeled "classVectorPrimitives.h" points to the tab bar where the file is currently open.

Logical Operations

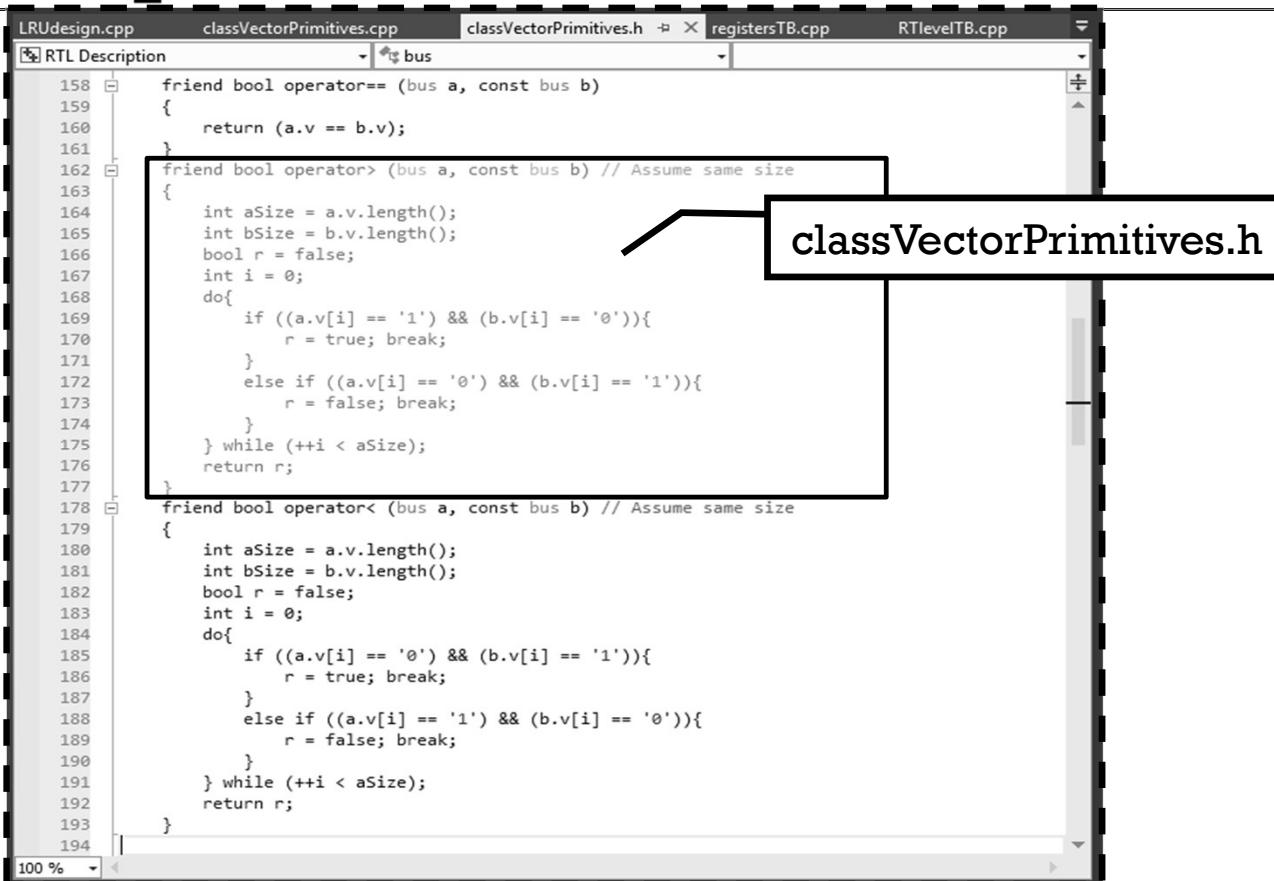
The screenshot shows a code editor window with several tabs at the top: `classVectorPrimitives.cpp`, `classVectorFunctions.h`, `classVectorPrimitives.h*`, and `utilityFunctions.cpp`. The `classVectorPrimitives.h*` tab is active. A callout box labeled "Overloading concatenation operator" points to the implementation of the `operator+` function. Another callout box labeled "classVectorPrimitives.h" points to the declaration of the same function in the header file.

```
friend bus operator& (bus a, bus b){ ... }
friend bus operator| (bus a, bus b){ ... }
friend bus operator^ (bus a, bus b){ ... }
friend bus operator~ (bus a){ ... }
friend bus operator+ (const bus a, const bus b){ ... }

friend bus operator, (bus a, bus b)
{
    int aSize = a.v.length();
    int bSize = b.v.length();
    int rSize = aSize + bSize;
    bus r(rSize, 'X');
    int i;
    for (i = bSize - 1; i >= 0; i--) {
        r.v[aSize + i] = b.v.at(i);
    }
    for (i = aSize - 1; i >= 0; i--) {
        r.v[i] = a.v.at(i);
    }
    return r;
}

friend bool operator== (bus a, const bus b){ ... }
friend bool operator> (bus a, const bus b){ ... }
friend bool operator< (bus a, const bus b){ ... }
bool operator&& (bus b){ ... }
bool operator|| (bus b){ ... }
```

Rational Operations



The screenshot shows a code editor window with several tabs at the top: LRUdesign.cpp, classVectorPrimitives.cpp, classVectorPrimitives.h, registersTB.cpp, and RTLevelTB.cpp. The current tab is classVectorPrimitives.h. The code in this tab defines comparison operators for bus objects:

```
158     friend bool operator== (bus a, const bus b)
159     {
160         return (a.v == b.v);
161     }
162     friend bool operator> (bus a, const bus b) // Assume same size
163     {
164         int aSize = a.v.length();
165         int bSize = b.v.length();
166         bool r = false;
167         int i = 0;
168         do{
169             if ((a.v[i] == '1') && (b.v[i] == '0')){
170                 r = true; break;
171             }
172             else if ((a.v[i] == '0') && (b.v[i] == '1')){
173                 r = false; break;
174             }
175         } while (++i < aSize);
176         return r;
177     }
178     friend bool operator< (bus a, const bus b) // Assume same size
179     {
180         int aSize = a.v.length();
181         int bSize = b.v.length();
182         bool r = false;
183         int i = 0;
184         do{
185             if ((a.v[i] == '0') && (b.v[i] == '1')){
186                 r = true; break;
187             }
188             else if ((a.v[i] == '1') && (b.v[i] == '0')){
189                 r = false; break;
190             }
191         } while (++i < aSize);
192         return r;
193     }
194 }
```

A callout box points from the text "classVectorPrimitives.h" to the tab bar where the file is currently selected.

Logical Operations

The screenshot shows a code editor with several tabs: `classVectorPrimitives.cpp`, `classVectorFunctions.h`, `classVectorPrimitives.h*`, and `utilityFunctions.cpp`. The `classVectorPrimitives.h*` tab is active, displaying the following C++ code:

```
163 friend bool operator> (bus a, const bus b) { ... }
178
179 friend bool operator< (bus a, const bus b) { ... }
194
195 bool operator&& (bus b) // Must be member function for second argument
196 {
197     int aSize = this->v.length();
198     int bSize = b.v.length();
199     int rSize;
200     if (bSize == 1) rSize = aSize; else rSize = MIN(aSize, bSize);
201     bool r = false;
202     char c = '0';
203     int i;
204     for (i = rSize - 1; i >= 0; i--) {
205         if (bSize == 1) c = and(this->v.at(i), b.v.at(0));
206         else c = and(this->v.at(i), b.v.at(i));
207         if (c == '1') r = true;
208     }
209     return r;
210 }
211
212 bool operator|| (bus b)
213 {
214     int aSize = this->v.length();
215     int bSize = b.v.length();
216     int rSize;
217     if (bSize == 1) rSize = aSize; else rSize = MIN(aSize, bSize);
218     bool r = false;
219     char c = '0';
220     int i;
221     for (i = rSize - 1; i >= 0; i--) {
222         if (bSize == 1) c = or(this->v.at(i), b.v.at(0));
223         else c = or(this->v.at(i), b.v.at(i));
224         if (c == '1') r = true;
225     }
226     return r;
227 }
```

Annotations on the code:

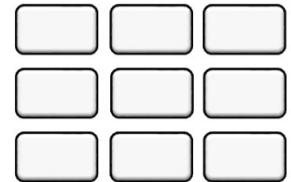
- A callout box points to the first two lines of the `&&` operator implementation: "this" is the pointer to the first operand.
- A callout box points to the first two lines of the `||` operator implementation: Functions return true if contains at least one "1".
- A callout box points to the `operator&&` line: classVectorPrimitives.h
- A callout box points to the `operator||` line: Member functions of the bus -> only one arguments is passed.

IO Operations

The screenshot shows a code editor with several tabs at the top: classVectorPrimitives.cpp, LRUdesignTB.cpp, LRUdesign.cpp, and classVectorPrimitives.h. The classVectorPrimitives.h tab is active. The code in the editor is as follows:

```
100 friend bus operator~ (bus a){ ... }
112 friend bus operator+ (const bus a, const bus b){ ... }
141 friend bus operator, (bus a, bus b){ ... }
157
158 friend bool operator== (bus a, const bus b){ ... }
162
163 friend bool operator> (bus a, const bus b){ ... }
178
179 friend bool operator< (bus a, const bus b){ ... }
194
195 bool operator&& (bus b){ ... }
211
212 bool operator|| (bus b){ ... }
228
229 friend ostream& operator<<(ostream& out, const bus a)
230 {
231     return(out << a.v);
232 }
233
234 friend istream& operator>>(istream& in, bus& a)
235 {
236     return(in >> a.v);
237 }
238
239 int ival ()
240 {
241     int aSize = v.length();
242     int ia=0;
243     for (int i = aSize - 1; i >= 0; i--) {
244         if (v.at(i) == '1') ia = ia + int(pow(2, (aSize - 1 - i)));
245     }
246     return ia;
247 }
248 void resize(int i, char c) { v.resize(i, c); }
249 };
```

A callout box points to the `friend` keyword in the `ostream& operator<<` function definition. The text in the callout box reads: "Because the first operand is not of the bus class, we need both arguments and so we use friend".



Outline

Introduction

- + Bus Model and Operations
- Basic Elements of RTL

Combinational elements

Mux

Tri

Adder

Comparator

Sequential elements

Registers

Up-Counters

- + RTL Design Examples

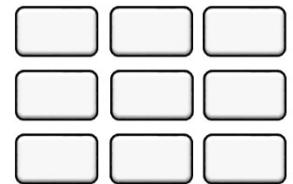
Combinational Elements

The screenshot shows a code editor window with several tabs at the top: `classVectorPrimitives.cpp`, `classVectorFunctions.h`, `classVectorPrimitives.h*`, and `utilityFunctions.cpp`. The main pane displays C++ code for combinational logic elements:

```
249
250 class Mux {
251     bus *i1, *i2, *i3, *o1;
252 public:
253     Mux(bus& a, bus& b, bus& sel, bus& w);
254     ~Mux(); // destructor
255     void evl();
256 };
257
258 class Tri {
259     bus *i1, *i2, *o1;
260 public:
261     Tri (bus& a, bus& tri, bus& w);
262     ~Tri(); // destructor
263     void evl ();
264 };
265
266 class Adder {
267     bus *i1, *i2, *i3, *o1, *o2;
268 public:
269     Adder(bus& a, bus& b, bus& ci, bus& co, bus& sum);
270     ~Adder();
271     void evl();
272 };
273
274 class Comparator {
275     bus *i1, *i2, *o1, *o2, *o3;
276 public:
277     Comparator(bus& a, bus& b, bus& lt, bus& eq, bus& gt);
278     ~Comparator();
279     void evl();
280 };
281
282 //class signExtend {
283 //class rightShift {
284
285 class nBitFunctionalRegister { ... };
```

A callout box labeled "classVectorPrimitives.h" points to the `#include "classVectorPrimitives.h"` directive at the top of the file. Another callout box labeled "Typical Combinational RTL elements" points to the declarations of the `Mux`, `Tri`, `Adder`, and `Comparator` classes.

Combinational Elements

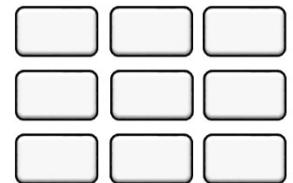


```
classVectorPrimitives.cpp  X SeqDetector.cpp  utilityFunctions.h  SeqDetector.h
RTL Description  (Global Scope)

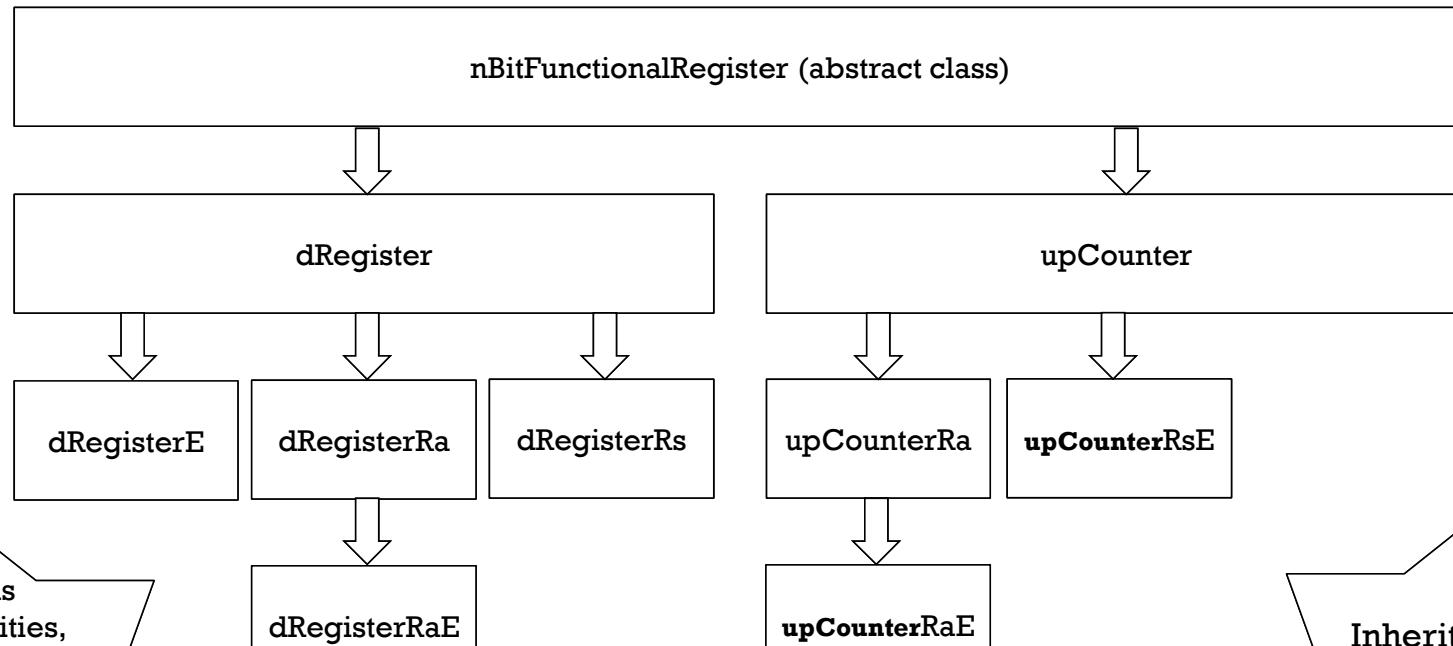
1 #include "classVectorPrimitives.h"
2 #include <iostream>
3 #include <fstream>
4 #include <string>
5 #include <math.h>
6 using namespace std;
7
8 Mux::Mux (bus& a, bus& b, bus& sel, bus& w) : i1(&a), i2(&b), i3(&sel), o1(&w)
9 {
10     o1->fill('X');
11 }
12 void Mux::evl () {
13     if (*i3 == "0") *o1 = *i1; else *o1 = *i2;
14 }
15
16 Tri::Tri (bus& a, bus& tri, bus& w) : i1(&a), i2(&tri), o1(&w)
17 {
18     o1->fill('X');
19 }
20 void Tri::evl () {
21     if (*i2 == "1") *o1 = *i1; else o1->fill('Z');
22 }
23
24 Adder::Adder(bus& a, bus& b, bus& ci, bus& co, bus& sum) :
25     i1(&a), i2(&b), i3(&ci), o1(&sum), o2(&co)
26 {
27     o1->fill('X'); o2->fill('X');
28 }
29 void Adder::evl () {
30     bus result(i1->length() + 1);
31     result = *i1 + *i2 + *i3;
32     int leftIndex = result.length() - 1;
33     *o2 = result.at(leftIndex);
34     *o1 = result.range(leftIndex, 0);
35 }
36
```

classVectorPrimitives.cpp

"+" operator is
overloaded before



Registers and Counters



Various
functionalities,
clocking schemes,
and resetting
mechanisms

Inheritance and
Polymorphism

Functional Registers

The diagram shows a code editor window displaying `classVectorPrimitives.h`. The code defines an abstract base class `nBitFunctionalRegister` and two derived classes, `dRegister` and `dRegisterE`.

```
285 class nBitFunctionalRegister {
286     public:
287         bus *d, *c, *q;
288         int size;
289         string rtype; // = "Register information";
290
291     public:
292         nBitFunctionalRegister () : size(0) {}
293         ~nBitFunctionalRegister () {}
294         void info (bus& D, bus& C, bus& Q, int& N, string& typ);
295         void init (string typ);
296         virtual void evl ()=0;
297     };
298
299 class dRegister : public nBitFunctionalRegister {
300     public:
301         dRegister(bus& D, bus& C, bus& Q);
302         ~dRegister();
303         virtual void evl ();
304     };
305
306 class dRegisterE : public dRegister { //Enable
307     bus* e;
308     public:
309         dRegisterE (bus& D, bus& C, bus& E, bus& Q);
310         ~dRegisterE ();
311         void evl ();
312     };
313
314 class dRegisterRa : public dRegister { //Reset-asynch
315     bus* r;
316     public:
317         dRegisterRa(bus& D, bus& C, bus& R, bus& Q);
318         ~dRegisterRa();
319         virtual void evl ();
320     };
321
```

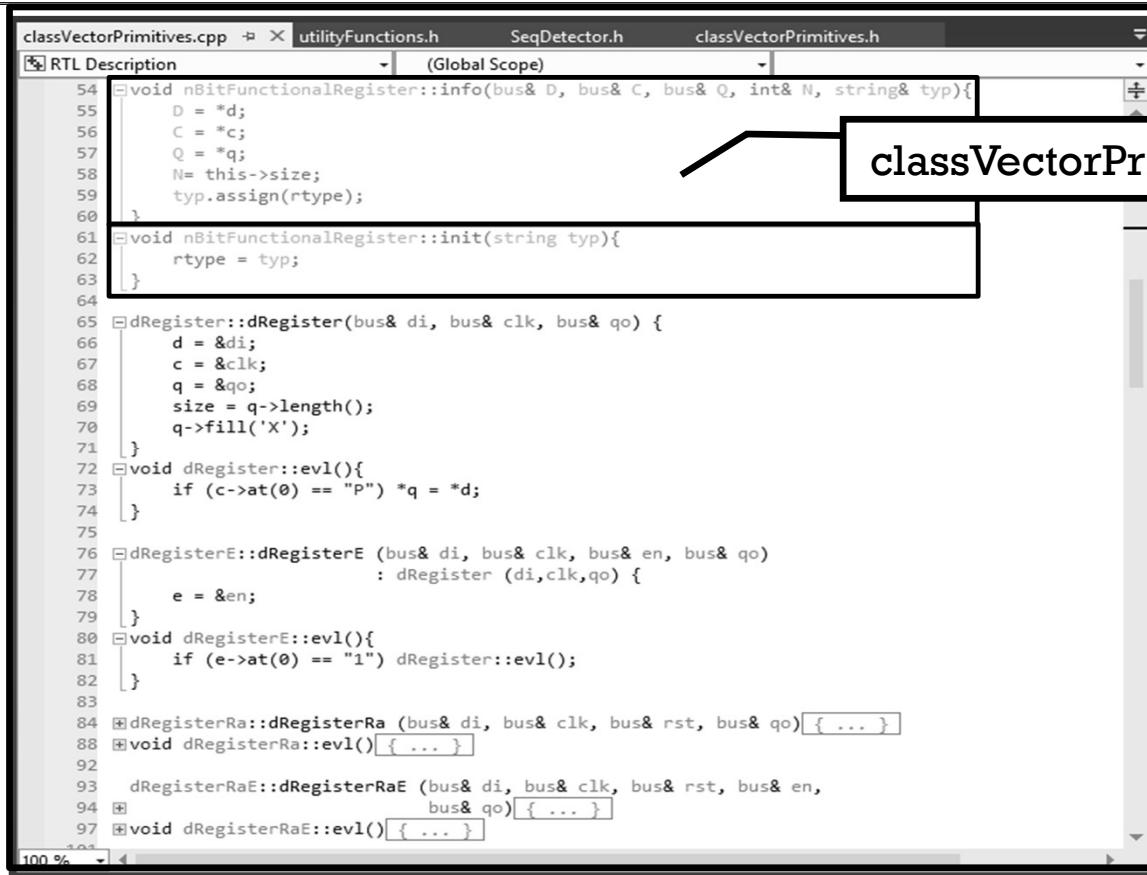
Initializing member variables. (Annotation pointing to line 295)

Pure virtual method:
Derived classes must define it. This is an abstract class because of this. (Annotation pointing to line 296)

This is a default constructor. It must be since this is an abstract class. (Annotation pointing to line 292)

The base class for all registers and counters (Annotation pointing to the `nBitFunctionalRegister` class definition)

Functional Registers



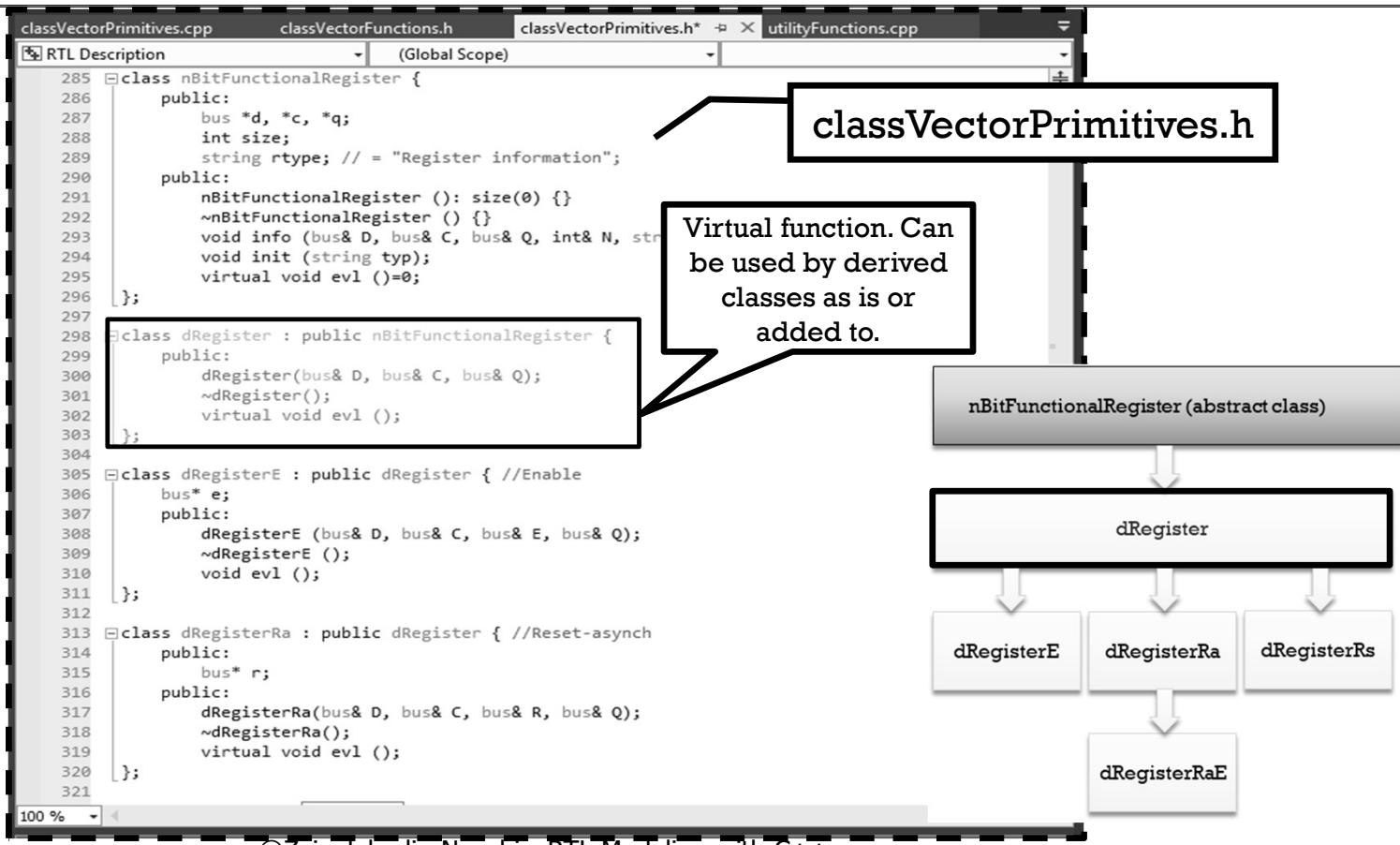
The screenshot shows a code editor window with the following details:

- File tabs: classVectorPrimitives.cpp, utilityFunctions.h, SeqDetector.h, classVectorPrimitives.h
- Search bar: RTL Description (Global Scope)
- Code content:

```
54 void nBitFunctionalRegister::info(bus& D, bus& C, bus& Q, int& N, string& typ){  
55     D = *d;  
56     C = *c;  
57     Q = *q;  
58     N= this->size;  
59     typ.assign(rtype);  
60 }  
61 void nBitFunctionalRegister::init(string typ){  
62     rtype = typ;  
63 }  
64  
65 dRegister::dRegister(bus& di, bus& clk, bus& qo) {  
66     d = &di;  
67     c = &clk;  
68     q = &qo;  
69     size = q->length();  
70     q->fill('X');  
71 }  
72 dRegister::evl(){  
73     if (c->at(0) == "P") *q = *d;  
74 }  
75  
76 dRegisterE::dRegisterE (bus& di, bus& clk, bus& en, bus& qo)  
77     : dRegister (di,clk,qo) {  
78     e = &en;  
79 }  
80 dRegisterE::evl(){  
81     if (e->at(0) == "1") dRegister::evl();  
82 }  
83  
84 dRegisterRa::dRegisterRa (bus& di, bus& clk, bus& rst, bus& qo){ ... }  
85 void dRegisterRa::evl(){ ... }  
86  
87 dRegisterRaE::dRegisterRaE (bus& di, bus& clk, bus& rst, bus& en,  
88                             bus& qo){ ... }  
89 void dRegisterRaE::evl(){ ... }
```
- Bottom status bar: 100 %

A callout box highlights the file name "classVectorPrimitives.cpp".

DRegister



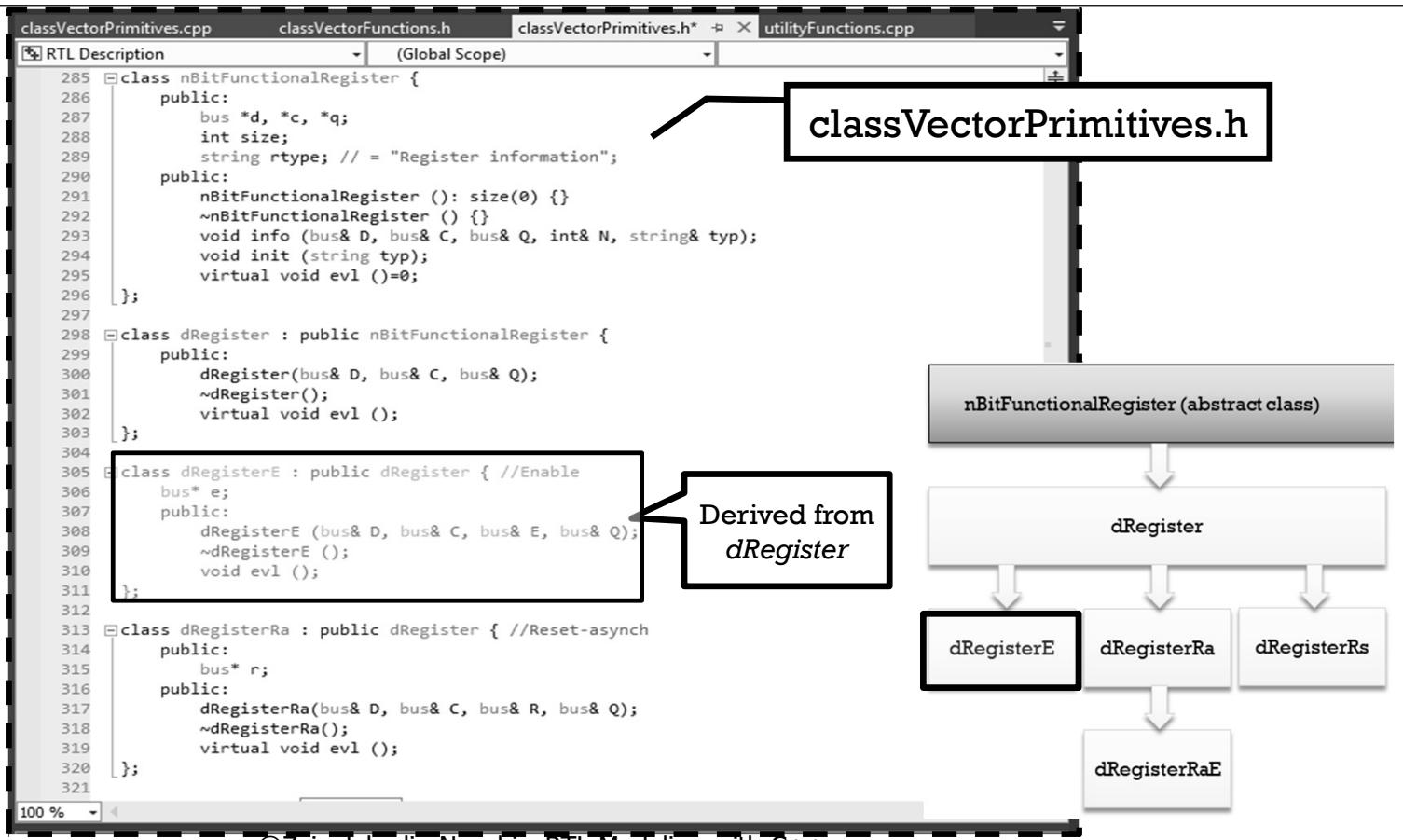
DRegister

```
classVectorPrimitives.cpp  X utilityFunctions.h      SeqDetector.h      classVectorPrimitives.h
RTL Description      (Global Scope)      classVectorPrimitives.h
54 void nBitFunctionalRegister::info(bus& D, bus& C, bus& Q, int& N, string& typ){
55     D = *d;
56     C = *c;
57     Q = *q;
58     N= this->size;
59     typ.assign(rtype);
60 }
61 void nBitFunctionalRegister::init(string typ){
62     rtype = typ;
63 }
64
65 void dRegister::dRegister(bus& di, bus& clk, bus& qo) {
66     d = &di;
67     c = &clk;
68     q = &qo;
69     size = q->length();
70     q->fill('X');
71 }
72 void dRegister::evl(){
73     if (c->at(0) == "P") *q = *d;
74 }
75
76 void dRegisterE::dRegisterE (bus& di, bus& clk, bus& en, bus& qo)
77     : dRegister (di,clk,qo) {
78     e = &en;
79 }
80 void dRegisterE::evl(){
81     if (e->at(0) == "1") dRegister::evl();
82 }
83
84 void dRegisterRa::dRegisterRa (bus& di, bus& clk, bus& rst, bus& qo){ ... }
85 void dRegisterRa::evl(){ ... }
86
87 void dRegisterRaE::dRegisterRaE (bus& di, bus& clk, bus& rst, bus& en,
88                                 bus& qo){ ... }
89 void dRegisterRaE::evl(){ ... }
```

classVectorPrimitives.cpp

Assign register pointer
Set register size
Initialize the bus connected to the output

DRegisterE



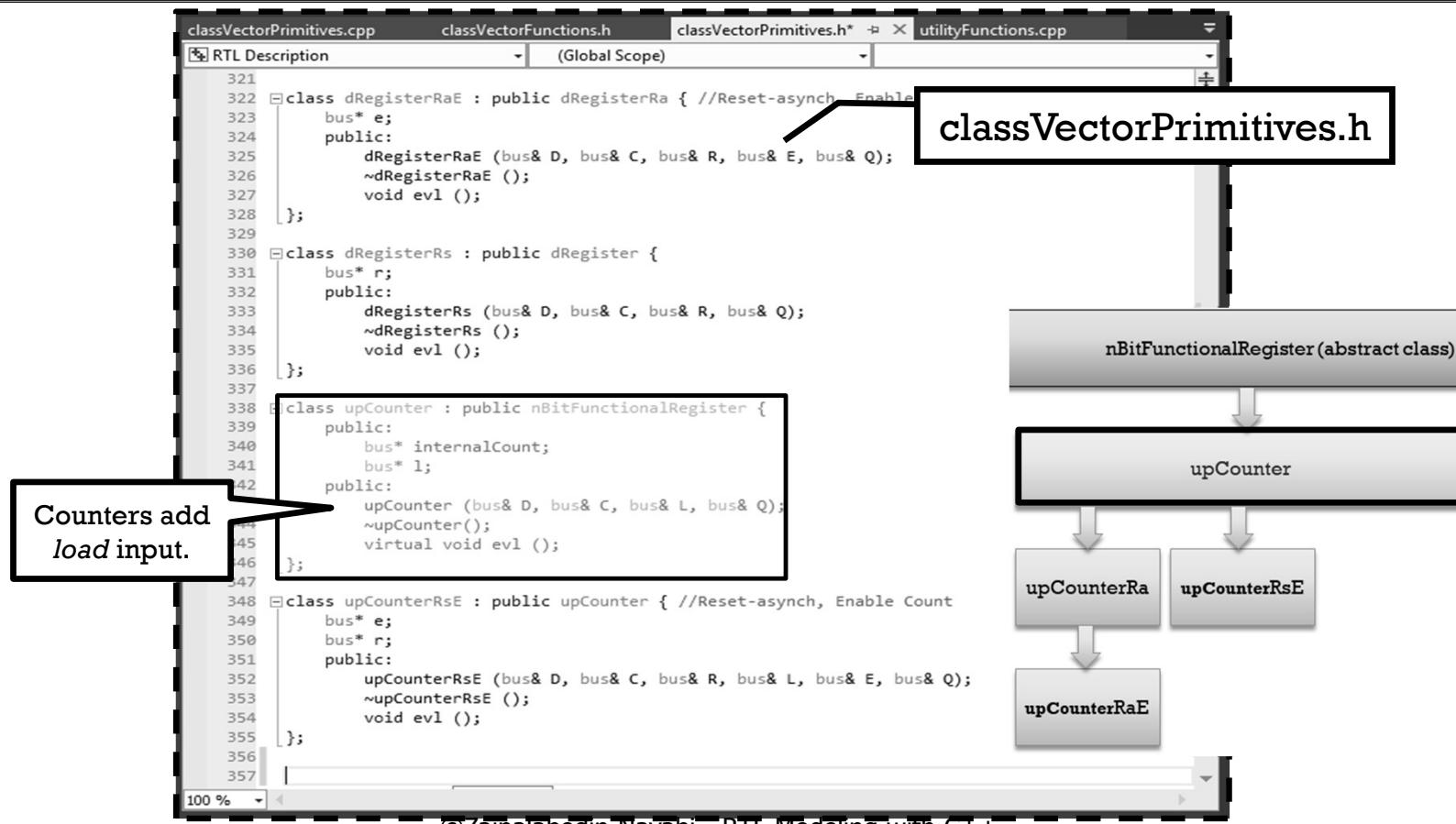
DRegisterE

```
classVectorPrimitives.cpp  X utilityFunctions.h      SeqDetector.h      classVectorPrimitives.h
RTL Description      (Global Scope)      classVectorPrimitives.h
54 void nBitFunctionalRegister::info(bus& D, bus& C, bus& Q, int& N, string& typ){
55     D = *d;
56     C = *c;
57     Q = *q;
58     N= this->size;
59     typ.assign(rtype);
60 }
61 void nBitFunctionalRegister::init(string typ){
62     rtype = typ;
63 }
64
65 dRegister::dRegister(bus& di, bus& clk, bus& qo) {
66     d = &di;
67     c = &clk;
68     q = &qo;
69     size = q->length();
70     q->fill('X');
71 }
72 void dRegister::evl(){
73     if (c->at(0) == "P") *q = *d;
74 }
75
76 dRegisterE::dRegisterE (bus& di, bus& clk, bus& en, bus& qo)
77     : dRegister (di,clk,qo) {
78     e = &en;
79 }
80 void dRegisterE::evl(){
81     if (e->at(0) == "1") dRegister::evl();
82 }
83
84 dRegisterRa::dRegisterRa (bus& di, bus& clk, bus& rst, bus& qo){ ... }
85 void dRegisterRa::evl(){ ... }
86
87 dRegisterRaE::dRegisterRaE (bus& di, bus& clk, bus& rst, bus& en,
88                             bus& qo){ ... }
89 void dRegisterRaE::evl(){ ... }
```

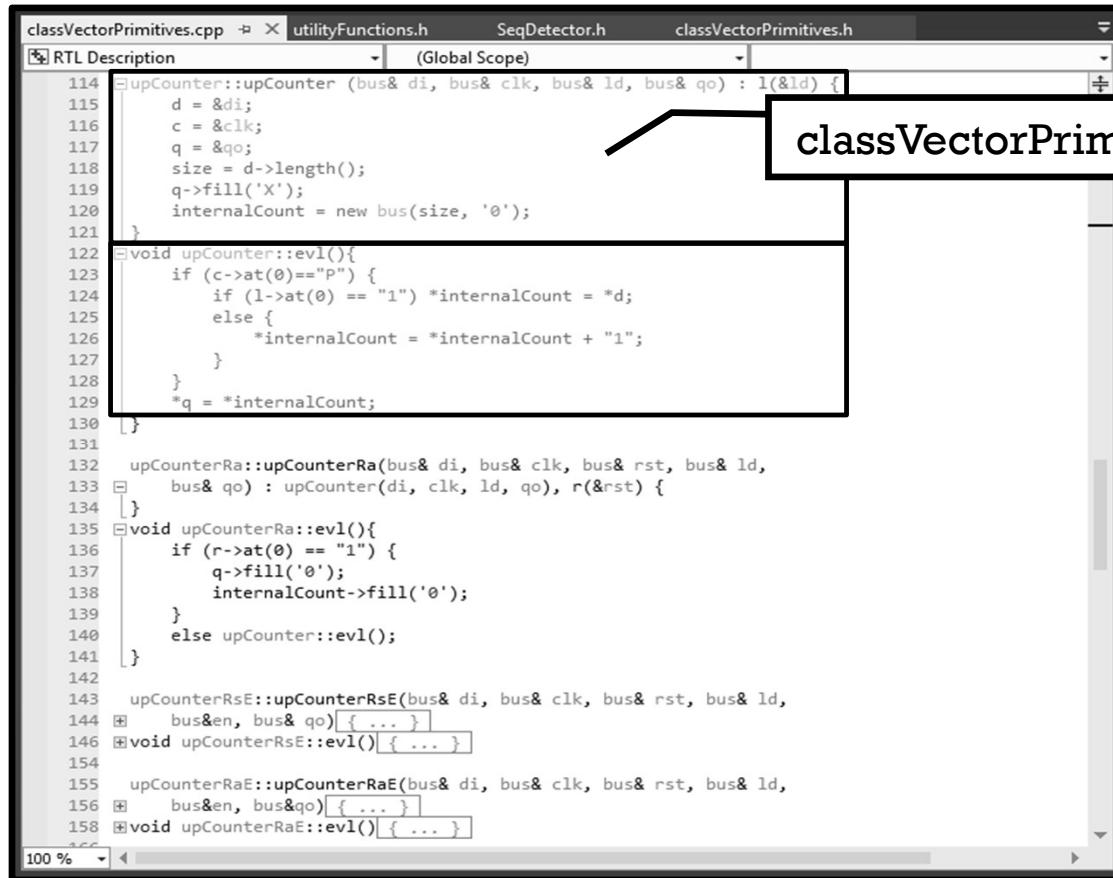
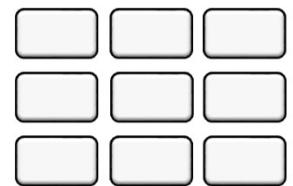
classVectorPrimitives.cpp

Dregister constructor is called
Enable port is assigned

UpCounter



UpCounter

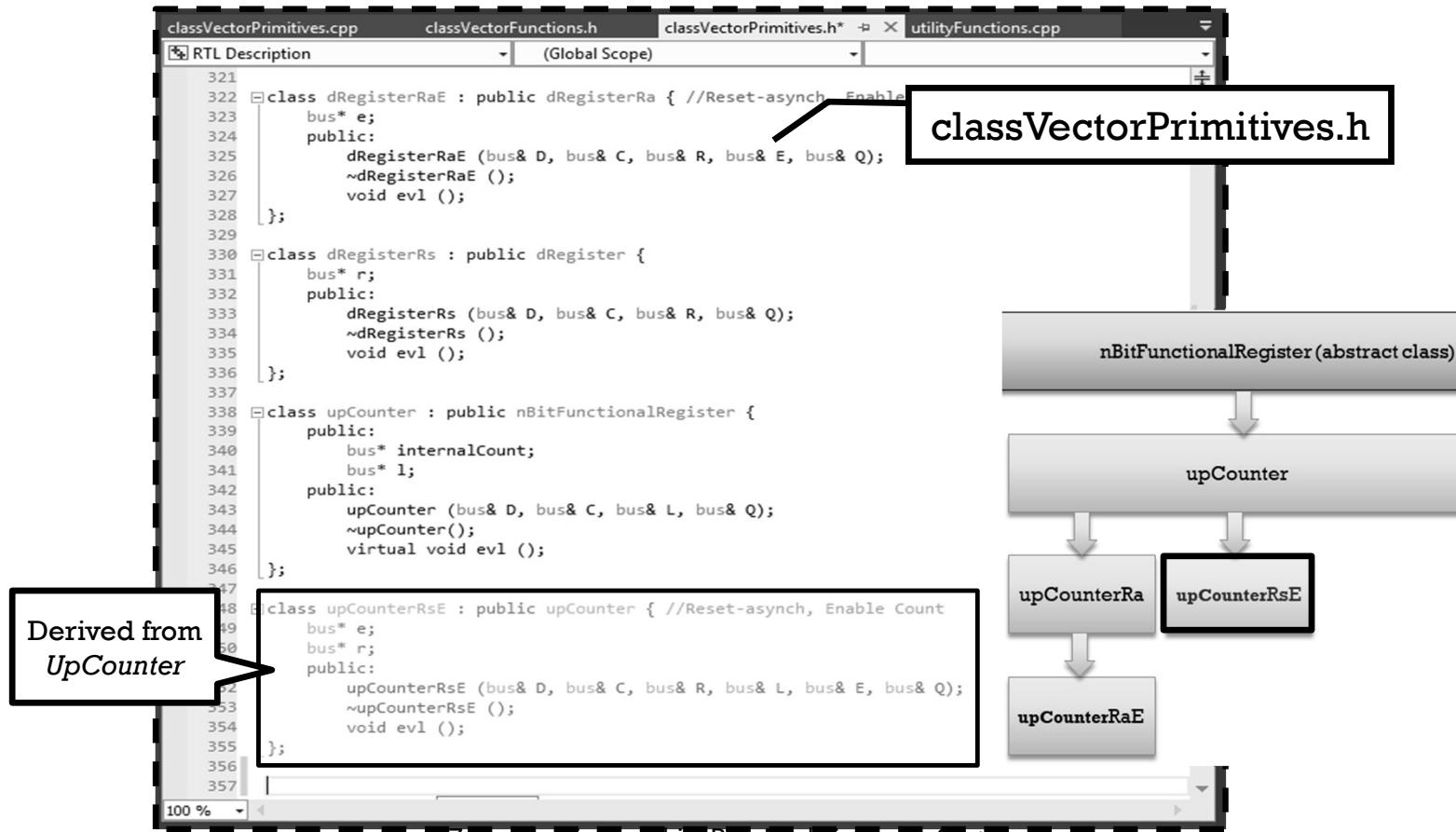


```
classVectorPrimitives.cpp  X utilityFunctions.h      SeqDetector.h      classVectorPrimitives.h
RTL Description          (Global Scope)           -|
```

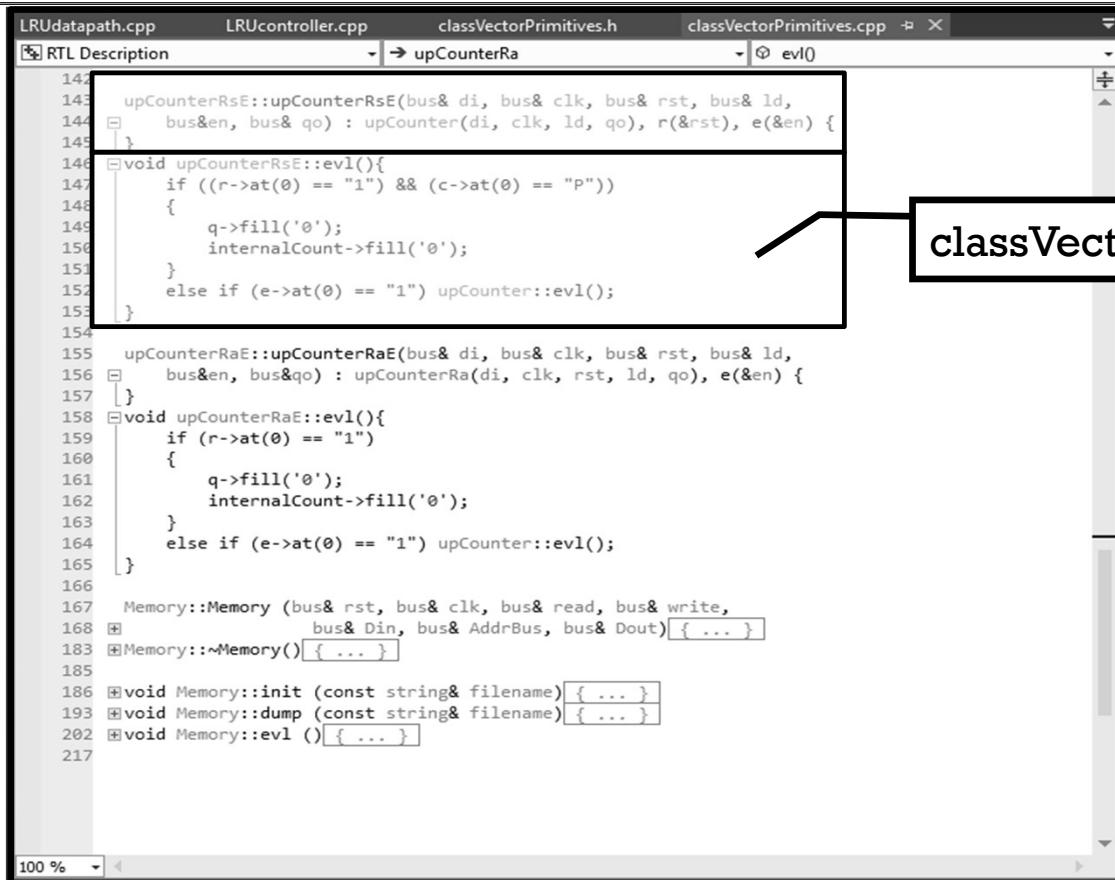
```
114     upCounter (bus& di, bus& clk, bus& ld, bus& qo) : l(&ld) {
115         d = &di;
116         c = &clk;
117         q = &qo;
118         size = d->length();
119         q->fill('X');
120         internalCount = new bus(size, '0');
121     }
122     void upCounter::evl(){
123         if (c->at(0)=="P") {
124             if (l->at(0) == "1") *internalCount = *d;
125             else {
126                 *internalCount = *internalCount + "1";
127             }
128         }
129         *q = *internalCount;
130     }
131
132     upCounterRa::upCounterRa(bus& di, bus& clk, bus& rst, bus& ld,
133     bus& qo) : upCounter(di, clk, ld, qo), r(&rst) {
134     }
135     void upCounterRa::evl(){
136         if (r->at(0) == "1") {
137             q->fill('0');
138             internalCount->fill('0');
139         }
140         else upCounter::evl();
141     }
142
143     upCounterRsE::upCounterRsE(bus& di, bus& clk, bus& rst, bus& ld,
144     bus&en, bus& qo){ ... }
145     void upCounterRsE::evl(){ ... }
146
147     upCounterRaE::upCounterRaE(bus& di, bus& clk, bus& rst, bus& ld,
148     bus&en, bus&qo){ ... }
149     void upCounterRaE::evl(){ ... }
```

©Zainalabedin Navabi - RTL Modeling with C++

UpCounterRsE



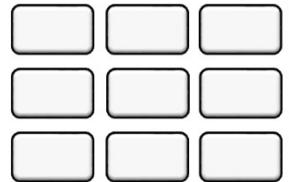
UpCounterRsE



The screenshot shows a code editor with multiple tabs. The active tab is 'classVectorPrimitives.cpp'. The code is as follows:

```
142     upCounterRsE::upCounterRsE(bus& di, bus& clk, bus& rst, bus& ld,
143     bus&en, bus& qo) : upCounter(di, clk, ld, qo), r(&rst), e(&en) {
144   }
145
146   void upCounterRsE::evl(){
147     if ((r->at(0) == "1") && (c->at(0) == "P"))
148     {
149       q->fill('0');
150       internalCount->fill('0');
151     }
152     else if (e->at(0) == "1") upCounter::evl();
153   }
154
155   upCounterRaE::upCounterRaE(bus& di, bus& clk, bus& rst, bus& ld,
156     bus&en, bus&qo) : upCounterRa(di, clk, rst, ld, qo), e(&en) {
157   }
158   void upCounterRaE::evl(){
159     if (r->at(0) == "1")
160     {
161       q->fill('0');
162       internalCount->fill('0');
163     }
164     else if (e->at(0) == "1") upCounter::evl();
165   }
166
167   Memory::Memory (bus& rst, bus& clk, bus& read, bus& write,
168     bus& Din, bus& AddrBus, bus& Dout){ ... }
169   ~Memory(){ ... }
170
171   void Memory::init (const string& filename){ ... }
172   void Memory::dump (const string& filename){ ... }
173   void Memory::evl (){ ... }
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
```

classVectorPrimitives.cpp



Outline

Introduction

- + Bus Model and Operations
- + Basic Elements of RTL
- RTL Design Examples

Memory Structure

Moore Sequence Detector (11011)

Least- Recently-Used (LRU) Circuit

Exponential Circuit

Memory Structure

```
LRUdatapath.cpp LRUcontroller.cpp classVectorPrimitives.h X classVectorPrimitives.cpp
RTL Description (Global Scope)

356
357 class upCounterRa : public upCounter { //Reset-asynch
358     public:
359         bus* r;
360     public:
361         upCounterRa(bus& D, bus& C, bus& R, bus& L, bus& Q);
362         ~upCounterRa ();
363         virtual void evl ();
364     };
365
366 class upCounterRaE : public upCounterRa { //Reset-asynch, Enable Count
367     bus* e;
368     public:
369         upCounterRaE (bus& D, bus& C, bus& R, bus& L, bus& E, bus& Q);
370         ~upCounterRaE ();
371         void evl ();
372     };
373
374 class Memory{
375     bus *rst, *clk, *read, *write;
376     bus *Din, *AddrBus;
377     bus *Dout;
378     bus *mem;
379     int N;
380
381     public:
382         Memory (bus& rst, bus& clk, bus& read, bus& write,
383                 bus& Din, bus& AddrBus, bus& Dout);
384         Memory() {};
385         ~Memory ();
386         void evl ();
387         void init (const string& filename);
388         void dump (const string& filename);
389     };
390
391 #endif |
```

classVectorPrimitives.h

Use bus pointers for all memory ports

Memory Structure

```
LRUdatapath.cpp LRUcontroller.cpp classVectorPrimitives.h classVectorPrimitives.cpp* evl()
RTL Description Memory evl()

166     Memory::Memory (bus& rst, bus& clk, bus& read, bus& write,
167                         bus& Din, bus& AddrBus, bus& Dout) :
168         rst(&rst), read(&read), write(&write), clk(&clk), Din(&Din),
169         AddrBus(&AddrBus), Dout(&Dout)
170     {
171         this->write->fill('X');
172         this->Dout->fill('X');
173         int LEN = this->Dout->length();
174         N = int(pow(2, LEN));
175
176         mem = new bus[N];
177
178         for (int i=0; i < N; i++) {
179             mem[i].resize(LEN, '0');
180         }
181     }
182
183     ~Memory() { ... }
184
185
186     void Memory::init (const string& filename) { ... }
187     void Memory::dump (const string& filename) { ... }
188
189     void Memory::evl () {
190         if (rst->at(0) == "1") {
191             for (int i = 0; i < N; i++) mem[i].fill('0');
192         }
193         else if (read->at(0) == "1") {
194             *Dout = mem[AddrBus->ival()];
195         }
196         else if (clk->at(0) == "P") {
197             if (write->at(0) == "1") {
198                 mem[AddrBus->ival()] = *Din;
199             }
200         }
201     }
202
203 }
```

classVectorPrimitives.h

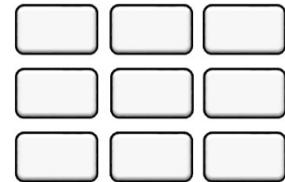
Memory resetting,
read and write

Memory Structure

```
LRUdatapath.cpp LRUcontroller.cpp classVectorPrimitives.h classVectorPrimitives.cpp + X
RTL Description → upCounterRa ⊕ evl()
167 Memory::Memory (bus& rst, bus& clk, bus& read, bus& write,
168     bus& Din, bus& AddrBus, bus& Dout) :
169     rst(&rst), read(&read), write(&write), clk(&clk), Din(&Din),
170     AddrBus(&AddrBus), Dout(&Dout)
171 {
172     this->write->fill('X');
173     this->Dout->fill('X');
174     int LEN = this->Dout->length();
175     N = int(pow(2, LEN));
176
177     mem = new bus[N];
178
179     for (int i=0; i < N; i++) {
180         mem[i].resize(LEN, '0');
181     }
182 }
183 Memory::~Memory() {
184 }
185
186 void Memory::init (const string& filename){
187     ifstream finp(filename);
188     for (int i = 0; i < N; i++) {
189         finp >> mem[i];
190         cout << mem[i] << "\n";
191     }
192 }
193 void Memory::dump (const string& filename){
194     ofstream fout(filename);
195     fout << "listing follows:\n";
196     cout << "listing follows:\n";
197     for (int i = 0; i < N; i++) {
198         fout << i << ":" << mem[i] << "\n";
199         cout << i << ":" << mem[i] << "\n";
200     }
201 }
202 void Memory::evl () { ... }
```

classVectorPrimitives.h

Use binary file passed to
them and use overloaded
“<<” and “>>”



Outline

Introduction

- + Bus Model and Operations
- + Basic Elements of RTL
- RTL Design Examples

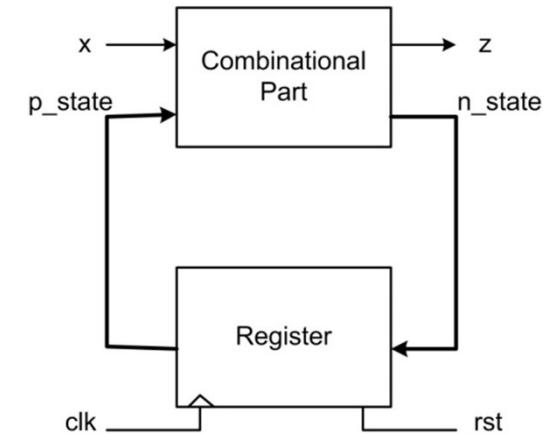
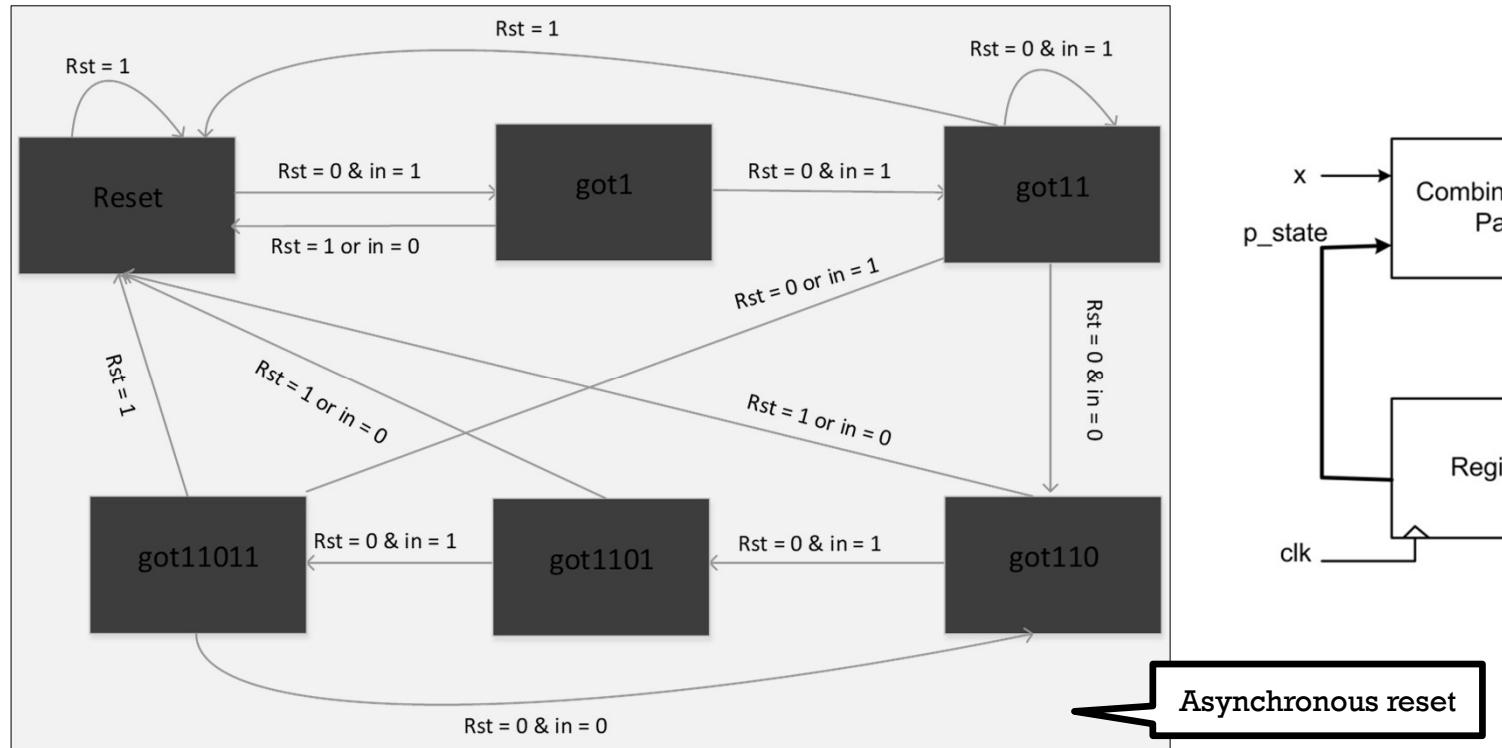
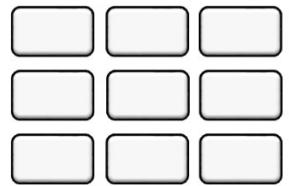
Memory Structure

Moore Sequence Detector (11011)

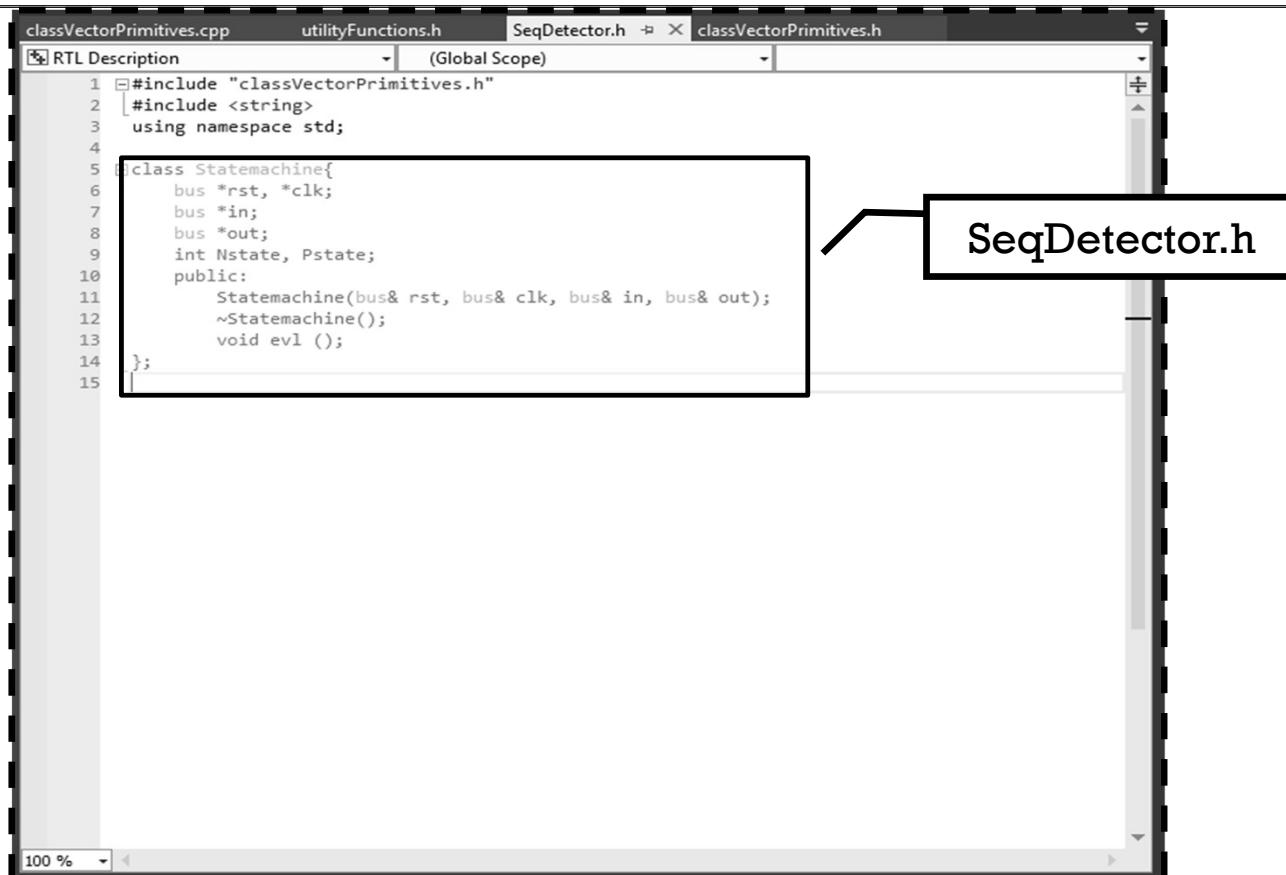
Least- Recently-Used (LRU) Circuit

Exponential Circuit

Moore Sequence Detector (11011)



Moore Sequence Detector (11011)



```
#include "classVectorPrimitives.h"
#include <string>
using namespace std;

class Statemachine{
    bus *rst, *clk;
    bus *in;
    bus *out;
    int Nstate, Pstate;
public:
    Statemachine(bus& rst, bus& clk, bus& in, bus& out);
    ~Statemachine();
    void evl ();
};

SeqDetector.h
```

Moore Sequence Detector (11011)

SeqDetector.cpp

```
3  Statemachine::Statemachine (bus& rst, bus& clk, bus& in, bus& out)
4  {
5      this->rst = &rst; this->clk = &clk; this->in = &in; this->out = &out;
6      Nstate = 0;
7      Pstate= 0;
8  }
9
10 void Statemachine::evl () {
11     *out = "0";
12
13     switch (Pstate){
14         case 0:
15             if(*in == "1") Nstate = 1;
16             else Nstate = 0; break;
17         case 1:
18             if (*in == "1") Nstate = 2;
19             else Nstate = 0; break;
20         case 2:
21             if(*in == "1") Nstate = 2;
22             else Nstate = 3; break;
23         case 3:
24             if(*in == "1") Nstate = 4;
25             else Nstate = 0; break;
26         case 4:
27             if(*in == "1") Nstate = 5;
28             else Nstate = 0; break;
29         case 5:
30             if (*in == "1") Nstate = 5;
31             else Nstate = 3; break;
32     }
33
34     if (*rst == "1") Pstate = 0;
35     else if (*clk == "P") Pstate = Nstate;
36
37     if (Pstate == 5) *out = "1";
38     else *out = "0";
39 }
```

Switch statement handle state transition

Sequential part implementation

Moore Sequence Detector (11011)



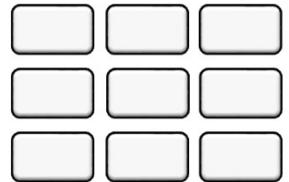
```
#include "SeqDetector.h"
#include "classVectorFunctions.h"

int main ()
{
    int ij;
    bus clk, rst;
    bus in;
    bus out;

    Statemachine* Statemachine1 = new Statemachine(rst, clk, in, out);

    // Statemachine resetting
    rst = "1";
    Statemachine1->evl();
    rst = "0";

    do{
        for ( int i =0; i<5; i++){
            cout << "\n Enter 1 bits for the input: "; cin >> in;
            clk = "P";
            Statemachine1 -> evl();
            cout << "\n" << out;
        }
        cout << "\n" << "Continue (0 or 1)?"; cin >> ij;
    }while (ij >0);
}
```



Outline

Introduction

- + Bus Model and Operations
- + Basic Elements of RTL
- RTL Design Examples

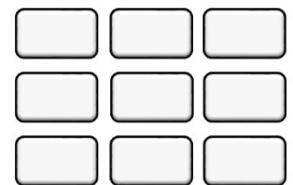
Memory Structure

Moore Sequence Detector (11011)

Least- Recently-Used (LRU) Circuit

Exponential Circuit

LRU Updater



Assign queue positions
to items based on
frequency of their use

Before I9 is accessed

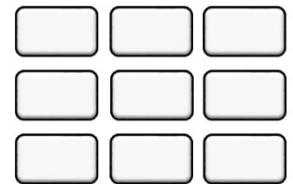
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

P8	P15	P1	P11	P3	P5	P9	P13	P0	P12	P6	P14	P7	P2	P10	P4
----	-----	----	-----	----	----	----	-----	----	-----	----	-----	----	----	-----	----

P9	P8	P15	P1	P11	P3	P5	P13	P0	P12	P6	P14	P7	P2	P10	P4
----	----	-----	----	-----	----	----	-----	----	-----	----	-----	----	----	-----	----

After I9 is accessed

Queue Memory File

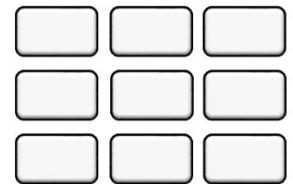


Before I9 is accessed

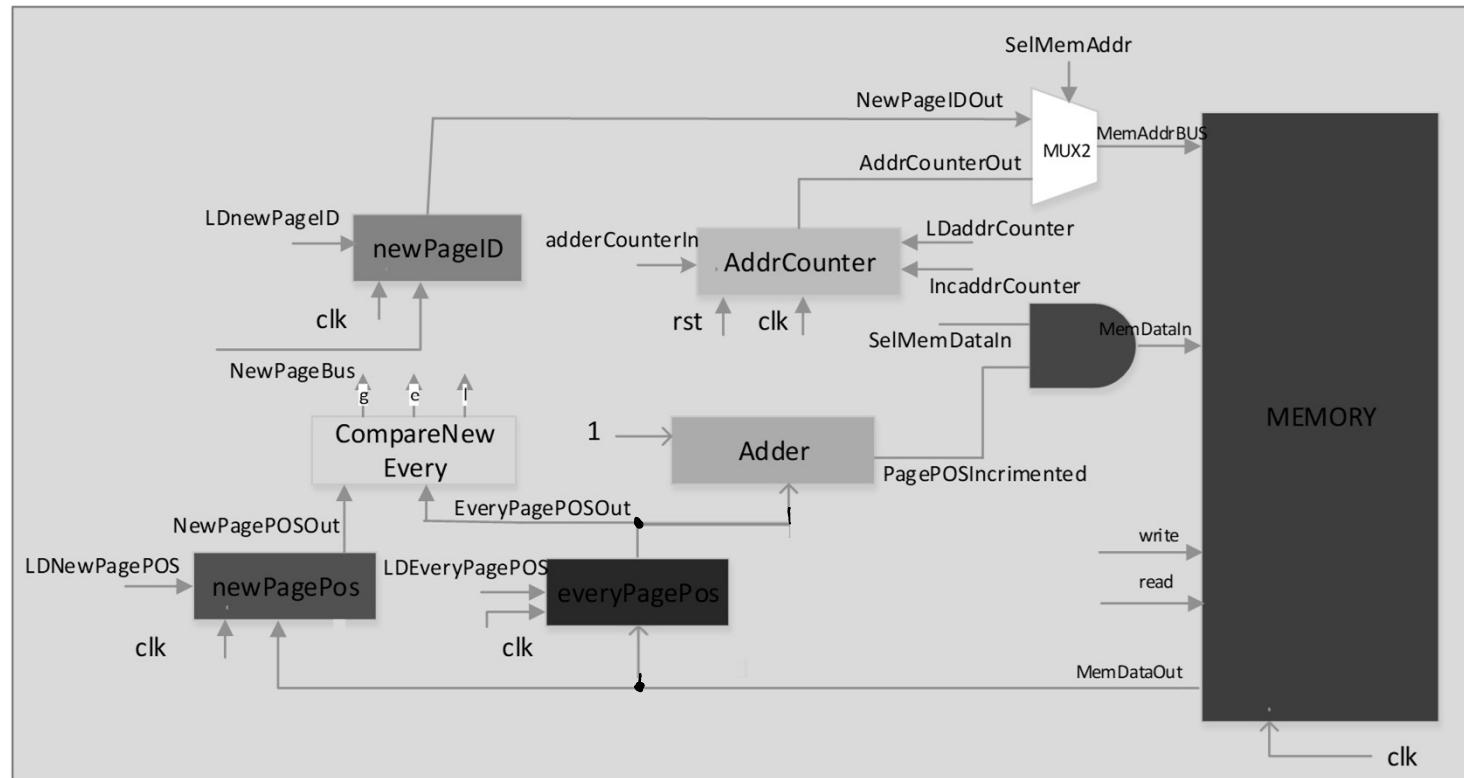
P0	8
P1	2
P2	13
P3	4
P4	15
P5	5
P6	10
P7	12
P8	0
P9	6
P10	14
P11	3
P12	9
P13	7
P14	11
P15	1

After I9 is accessed

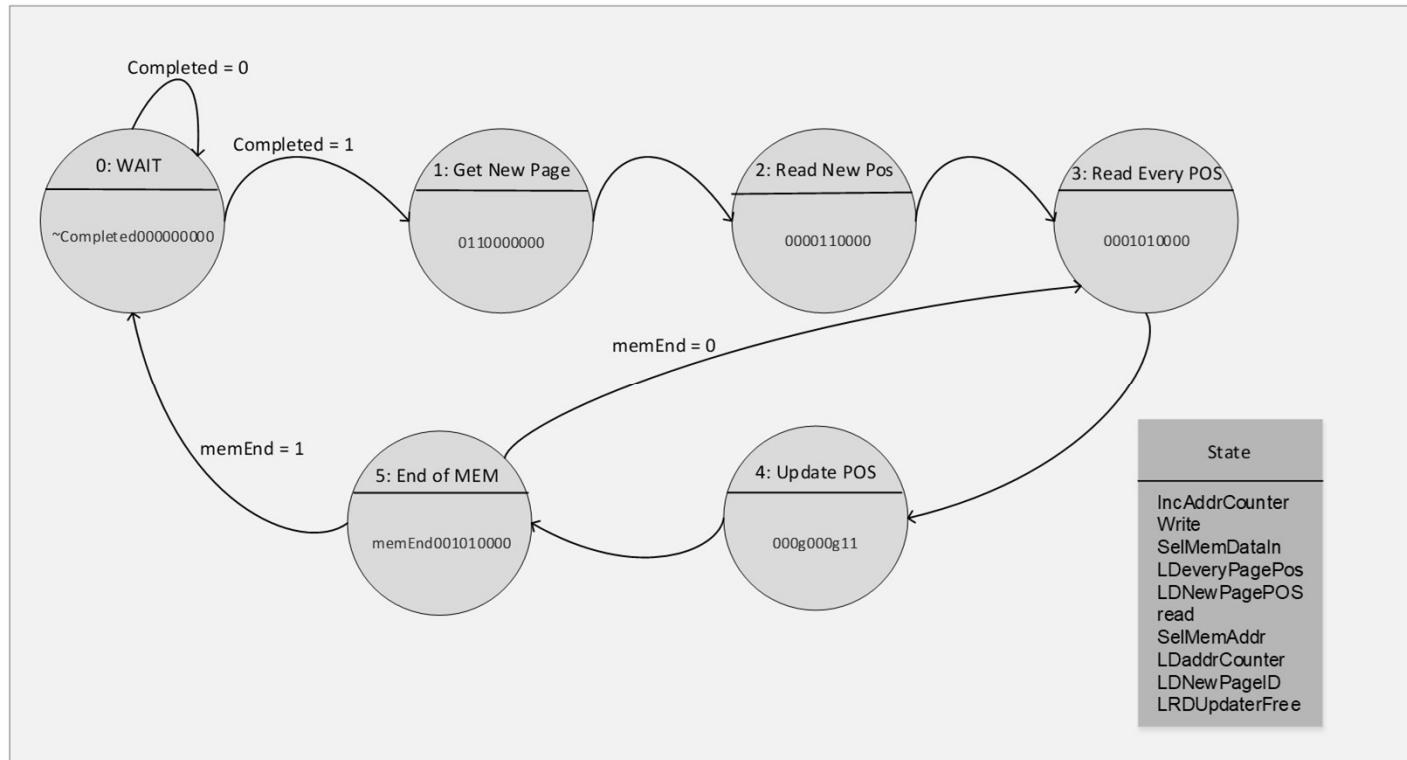
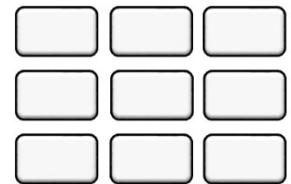
8
+ 3
13
+ 5
15
+ 6
10
12
+ 1
0
14
4
9
7
11
+ 2



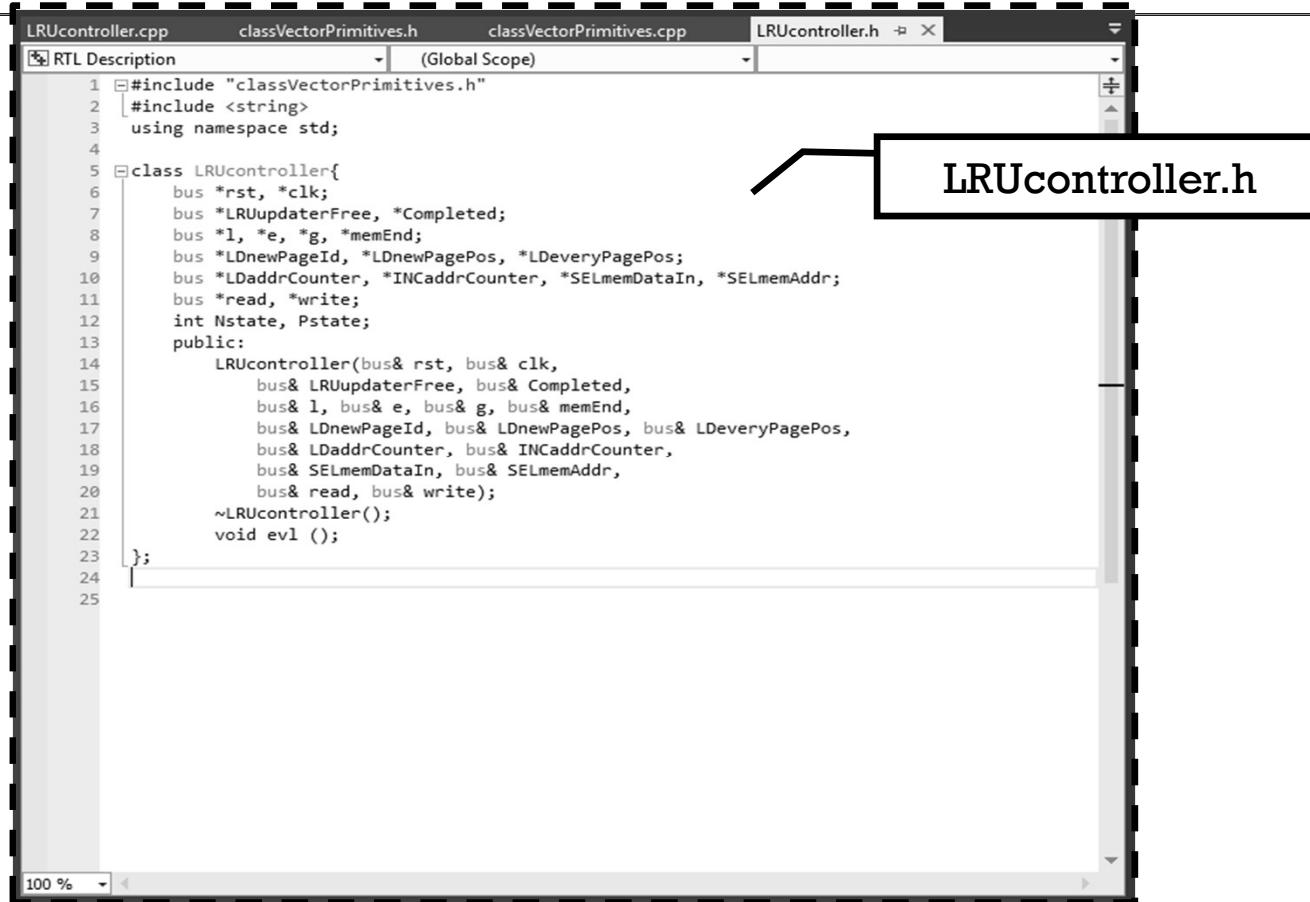
LRU Updater Datapath



LRU Controller



LRU Controller



```
1 #include "classVectorPrimitives.h"
2 #include <string>
3 using namespace std;
4
5 class LRUController{
6     bus *rst, *clk;
7     bus *LRUupdateFree, *Completed;
8     bus *l, *e, *g, *memEnd;
9     bus *LDnewPageId, *LDnewPagePos, *LDeveryPagePos;
10    bus *LDaddrCounter, *INCAddrCounter, *SELmemDataIn, *SELmemAddr;
11    bus *read, *write;
12    int Nstate, Pstate;
13
14    LRUcontroller(bus& rst, bus& clk,
15                  bus& LRUupdateFree, bus& Completed,
16                  bus& l, bus& e, bus& g, bus& memEnd,
17                  bus& LDnewPageId, bus& LDnewPagePos, bus& LDeveryPagePos,
18                  bus& LDaddrCounter, bus& INCAddrCounter,
19                  bus& SELmemDataIn, bus& SELmemAddr,
20                  bus& read, bus& write);
21    ~LRUcontroller();
22    void evl ();
23};
24
25
```

LRUController.h

LRU Controller



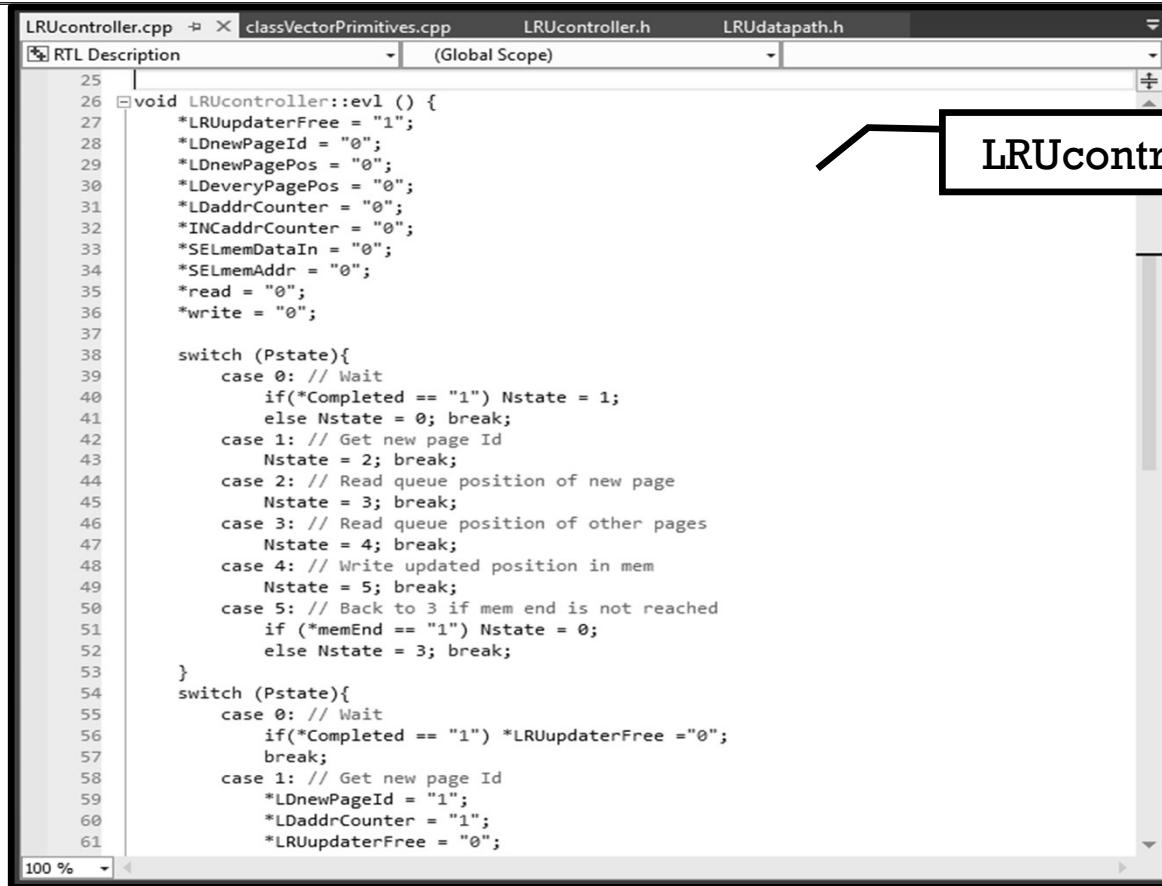
LRUController.cpp

```
#include "LRUcontroller.h"

LRUcontroller::LRUcontroller (bus& rst, bus& clk,
    bus& LRUupdateFree, bus& Completed,
    bus& l, bus& e, bus& g, bus& memEnd,
    bus& LDnewPageId, bus& LDnewPagePos, bus& LDeveryPagePos,
    bus& LDaddrCounter, bus& INCaddrCounter,
    bus& SELmemDataIn, bus& SELmemAddr,
    bus& read, bus& write) :  
    rst(&rst), clk(&clk),
    LRUupdateFree(&LRUupdateFree), Completed(&Completed),
    l(&l), e(&e), g(&g), memEnd(&memEnd),
    LDnewPageId(&LDnewPageId), LDnewPagePos(&LDnewPagePos),
    LDeveryPagePos(&LDeveryPagePos),
    LDaddrCounter(&LDaddrCounter), INCaddrCounter(&INCaddrCounter),
    SELmemDataIn(&SELmemDataIn), SELmemAddr(&SELmemAddr),
    read(&read), write(&write)  
{  
    this->e->fill('0');  
    this->g->fill('0');  
    Nstate = 0;  
    Pstate= 0;  
}  
  
void LRUcontroller::evl () {  
    *LRUupdateFree = "1";  
    *LDnewPageId = "0";  
    *LDnewPagePos = "0";  
    *LDeveryPagePos = "0";  
    *LDaddrCounter = "0";  
    *INCaddrCounter = "0";  
    *SELmemDataIn = "0";  
    *SELmemAddr = "0";  
    *read = "0";  
    *write = "0";  
}
```

LRUcontroller.cpp

LRU Controller

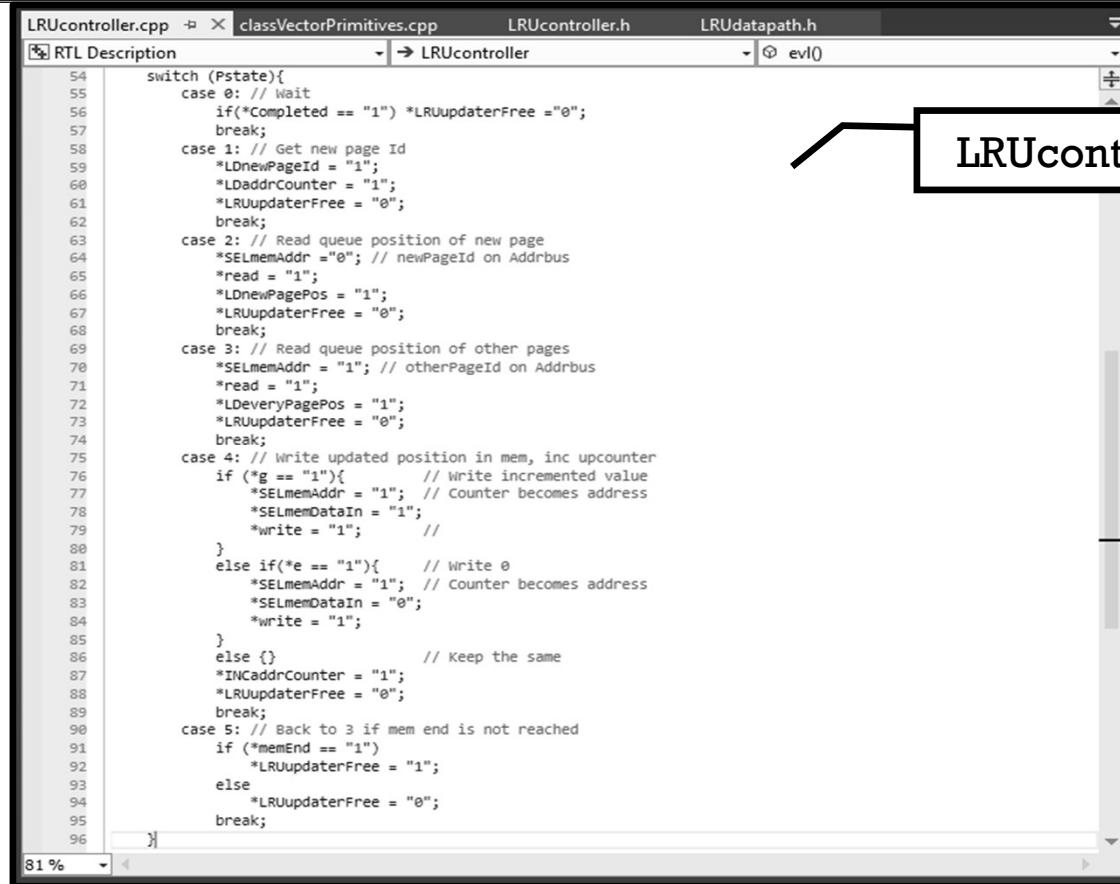


LRUcontroller.cpp (Global Scope)

```
25
26 void LRUcontroller::ev1 () {
27     *LRUupdateFree = "1";
28     *LDnewPageId = "0";
29     *LDnewPagePos = "0";
30     *LDeveryPagePos = "0";
31     *LDaddrCounter = "0";
32     *INCaddrCounter = "0";
33     *SELmemDataIn = "0";
34     *SELmemAddr = "0";
35     *read = "0";
36     *write = "0";
37
38     switch (Pstate){
39         case 0: // Wait
40             if(*Completed == "1") Nstate = 1;
41             else Nstate = 0; break;
42         case 1: // Get new page Id
43             Nstate = 2; break;
44         case 2: // Read queue position of new page
45             Nstate = 3; break;
46         case 3: // Read queue position of other pages
47             Nstate = 4; break;
48         case 4: // Write updated position in mem
49             Nstate = 5; break;
50         case 5: // Back to 3 if mem end is not reached
51             if (*memEnd == "1") Nstate = 0;
52             else Nstate = 3; break;
53     }
54     switch (Pstate){
55         case 0: // Wait
56             if(*Completed == "1") *LRUupdateFree ="0";
57             break;
58         case 1: // Get new page Id
59             *LDnewPageId = "1";
60             *LDaddrCounter = "1";
61             *LRUupdateFree = "0";
62     }
63 }
```

LRUcontroller.cpp

LRU Controller

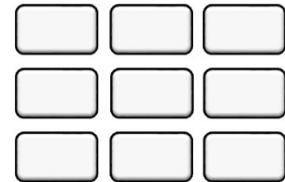


The screenshot shows a software interface with multiple tabs at the top: 'LRUController.cpp', 'classVectorPrimitives.cpp', 'LRUcontroller.h', and 'LRUdatapath.h'. The 'LRUController.cpp' tab is active. Below the tabs is a dropdown menu labeled 'RTL Description' with an arrow pointing to 'LRUController'. To the right of the dropdown is a small icon with a question mark and the text 'evl()'. The main area contains the C++ code for the LRU controller. A callout box with a black border and white background points from the bottom right towards the file name 'LRUcontroller.cpp' in the title bar.

```
54 switch (Pstate){  
55     case 0: // Wait  
56         if("Completed == "1") *LRUupdateFree ="0";  
57         break;  
58     case 1: // Get new page Id  
59         *LDnewPageId = "1";  
60         *LDaddrCounter = "1";  
61         *LRUupdateFree = "0";  
62         break;  
63     case 2: // Read queue position of new page  
64         *SELmemAddr ="0"; // newPageId on Addrbus  
65         *read = "1";  
66         *LDnewPagePOS = "1";  
67         *LRUupdateFree = "0";  
68         break;  
69     case 3: // Read queue position of other pages  
70         *SELmemAddr = "1"; // otherPageId on Addrbus  
71         *read = "1";  
72         *LDeveryPagePos = "1";  
73         *LRUupdateFree = "0";  
74         break;  
75     case 4: // Write updated position in mem, inc upcounter  
76         if (*g == "1"){ // Write incremented value  
77             *SELmemAddr = "1"; // Counter becomes address  
78             *SELmemDataIn = "1";  
79             *write = "1"; //  
80         }  
81         else if(*e == "1"){ // Write 0  
82             *SELmemAddr = "1"; // Counter becomes address  
83             *SELmemDataIn = "0";  
84             *write = "1";  
85         }  
86         else {} // Keep the same  
87         *INCaddrCounter = "1";  
88         *LRUupdateFree = "0";  
89         break;  
90     case 5: // Back to 3 if mem end is not reached  
91         if (*memEnd == "1")  
92             *LRUupdateFree = "1";  
93         else  
94             *LRUupdateFree = "0";  
95         break;  
96     }  
81 %
```

©Zainalabedin Navabi - RTL Modeling with C++

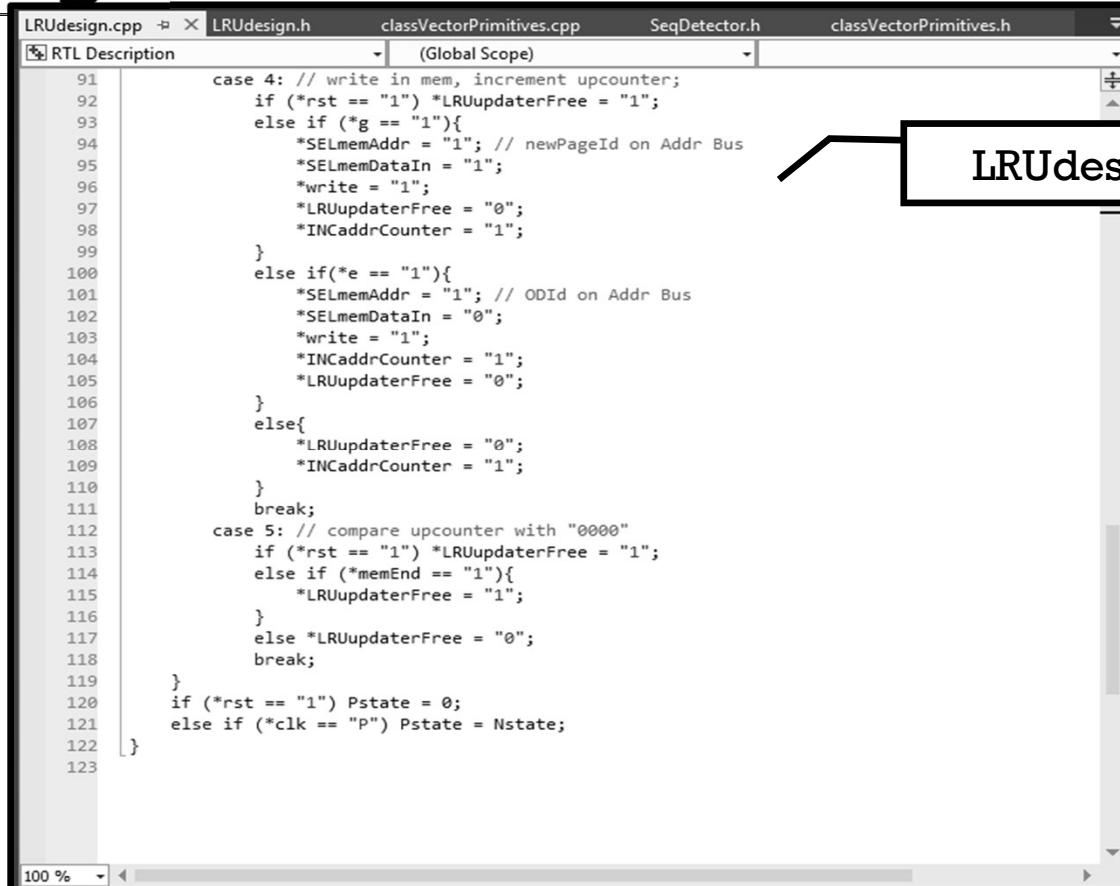
LRU Design



```
LRUdesign.h      LRUdesign.cpp*  X  classVectorPrimitives.cpp    classVectorPrimitives.h    registersTB.cpp
[RTL Description]  (Global Scope)
```

```
51     switch (Pstate){
52         case 0: // Wait
53             if(*Completed == "1") *LRUupdaterFree ="0";
54             break;
55         case 1: // Get new page Id
56             *LDnewPageId = "1";
57             *LAddrCounter = "1";
58             *LRUupdaterFree = "0";
59             break;
60         case 2: // Read queue position of new page
61             *SELmemAddr = "0"; // newPageId on Addrbus
62             *read = "1";
63             *LDnewPagePos = "1";
64             *LRUupdaterFree = "0";
65             break;
66         case 3: // Read queue position of other pages
67             *SELmemAddr = "1"; // otherPageId on Addrbus
68             *read = "1";
69             *LDeveryPagePos = "1";
70             *LRUupdaterFree = "0";
71             break;
72         case 4: // Write updated position in mem, inc upcounter
73             if (*g == "1"){
74                 // Write incremented value
75                 *SELmemAddr = "1"; // Counter becomes address
76                 *SELmemDataIn = "1";
77                 *write = "1";
78             }
79             else if(*e == "1"){
80                 // Write 0
81                 *SELmemAddr = "1"; // Counter becomes address
82                 *SELmemDataIn = "0";
83                 *write = "1";
84             }
85             else {}
86                 // Keep the same
87                 *INCAddrCounter = "1";
88                 *LRUupdaterFree = "0";
89                 break;
90         case 5: // Back to 3 if mem end is not reached
```

LRU Design

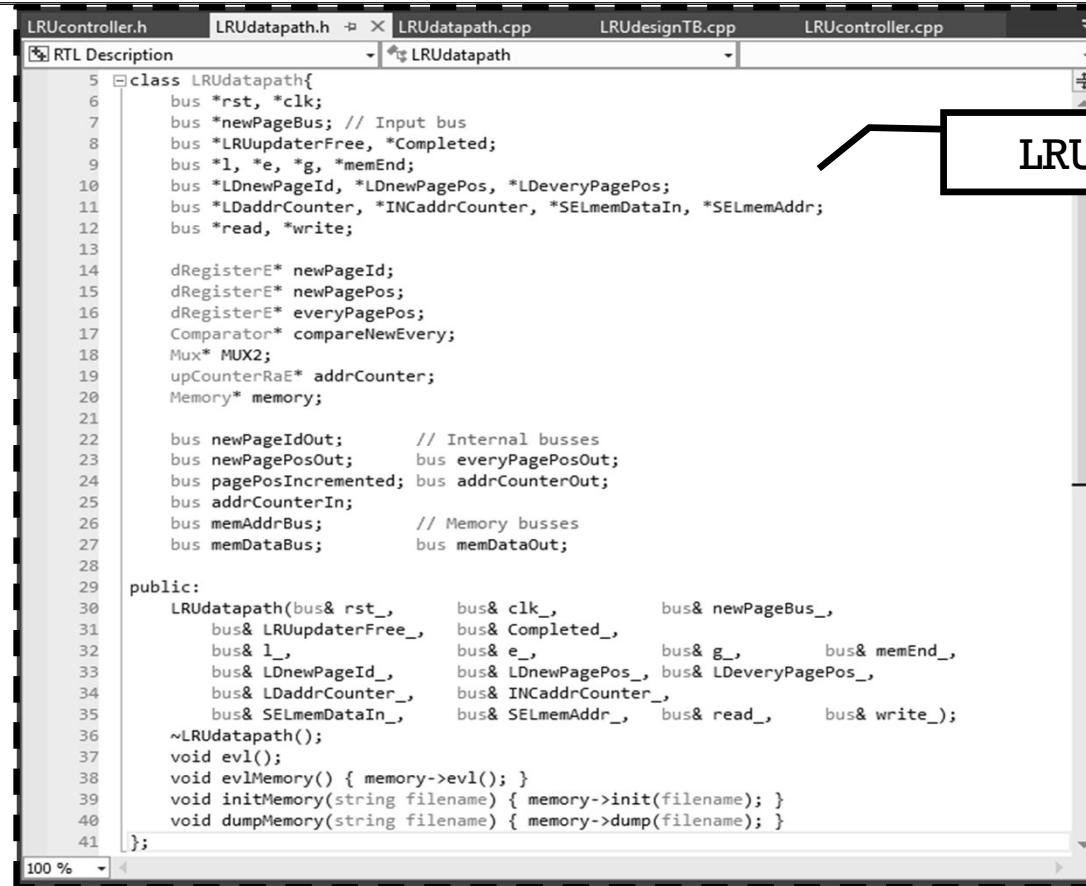


The screenshot shows a code editor window with multiple tabs at the top: LRUdesign.cpp, LRUdesign.h, classVectorPrimitives.cpp, SeqDetector.h, and classVectorPrimitives.h. The active tab is LRUdesign.cpp, which contains C++ code for an LRU cache. A callout box points from the text "LRUdesign.cpp" to the tab bar.

```
case 4: // write in mem, increment upcounter;
    if (*rst == "1") *LRUUpdaterFree = "1";
    else if (*g == "1"){
        *SELmemAddr = "1"; // newPageId on Addr Bus
        *SELmemDataIn = "1";
        *write = "1";
        *LRUUpdaterFree = "0";
        *INCaddrCounter = "1";
    }
    else if(*e == "1"){
        *SELmemAddr = "1"; // ODid on Addr Bus
        *SELmemDataIn = "0";
        *write = "1";
        *INCaddrCounter = "1";
        *LRUUpdaterFree = "0";
    }
    else{
        *LRUUpdaterFree = "0";
        *INCaddrCounter = "1";
    }
    break;
case 5: // compare upcounter with "0000"
    if (*rst == "1") *LRUUpdaterFree = "1";
    else if (*memEnd == "1"){
        *LRUUpdaterFree = "1";
    }
    else *LRUUpdaterFree = "0";
    break;
}
if (*rst == "1") Pstate = 0;
else if (*clk == "P") Pstate = Nstate;
```

LRUdesign.cpp

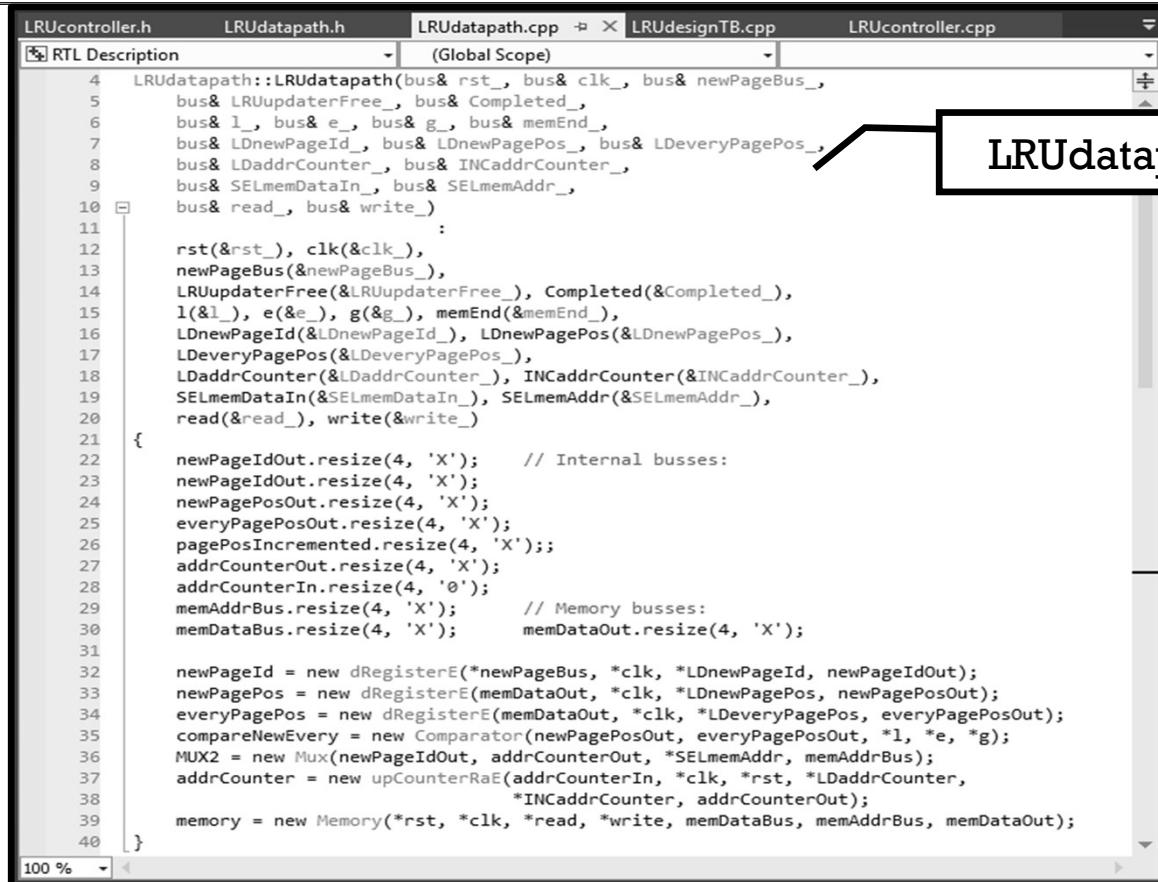
LRU Datapath



The screenshot shows a software interface with multiple tabs at the top: LRUController.h, LRUdatapath.h (highlighted with a black border), LRUdatapath.cpp, LRUdesignTB.cpp, and LRUController.cpp. The central area displays the content of the LRUdatapath.h file. A callout box with a black border points from the right side towards the file name 'LRUdatapath.h'.

```
5 class LRUdatapath{
6     bus *rst, *clk;
7     bus *newPageBus; // Input bus
8     bus *LRUupdateFree, *Completed;
9     bus *l, *e, *g, *memEnd;
10    bus *LDnewPageId, *LDnewPagePos, *LDeveryPagePos;
11    bus *LaddrCounter, *INCaddrCounter, *SELmemDataIn, *SELmemAddr;
12    bus *read, *write;
13
14    dRegisterE* newPageId;
15    dRegisterE* newPagePos;
16    dRegisterE* everyPagePos;
17    Comparator* compareNewEvery;
18    Mux* MUX2;
19    upCounterRaE* addrCounter;
20    Memory* memory;
21
22    bus newPageIdOut;      // Internal busses
23    bus newPagePosOut;    bus everyPagePosOut;
24    bus pagePosIncremented; bus addrCounterOut;
25    bus addrCounterIn;
26    bus memAddrBus;        // Memory busses
27    bus memDataBus;        bus memDataOut;
28
29 public:
30     LRUdatapath(bus& rst_,           bus& clk_,           bus& newPageBus_,
31                 bus& LRUupdateFree_, bus& Completed_, bus& memEnd_,
32                 bus& l_,             bus& e_,             bus& g_,             bus& memEnd_,
33                 bus& LDnewPageId_,  bus& LDnewPagePos_, bus& LDeveryPagePos_,
34                 bus& LaddrCounter_, bus& INCaddrCounter_, bus& SELmemDataIn_, bus& SELmemAddr_,
35                 bus& read_,         bus& write_);
36     ~LRUdatapath();
37     void evl();
38     void evlMemory() { memory->evl(); }
39     void initMemory(string filename) { memory->init(filename); }
40     void dumpMemory(string filename) { memory->dump(filename); }
41};
```

LRU Datapath

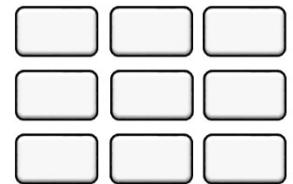


The screenshot shows a code editor with multiple tabs. The active tab is `LRUdatapath.cpp`. The code in the editor is as follows:

```
4  LRUdatapath::LRUdatapath(bus& rst_, bus& clk_, bus& newPageBus_,
5    bus& LRUUpdaterFree_, bus& Completed_,
6    bus& l_, bus& e_, bus& g_, bus& memEnd_,
7    bus& LDnewPageId_, bus& LDnewPagePos_, bus& LDeveryPagePos_,
8    bus& LDaddrCounter_, bus& INCaddrCounter_,
9    bus& SELmemDataIn_, bus& SELmemAddr_,
10   bus& read_, bus& write_)
11  :
12  rst(&rst_), clk(&clk_),
13  newPageBus(&newPageBus_),
14  LRUUpdaterFree(&LRUUpdaterFree_), Completed(&Completed_),
15  l(&l_), e(&e_), g(&g_), memEnd(&memEnd_),
16  LDnewPageId(&LDnewPageId_), LDnewPagePos(&LDnewPagePos_),
17  LDeveryPagePos(&LDeveryPagePos_),
18  LDaddrCounter(&LDaddrCounter_), INCaddrCounter(&INCaddrCounter_),
19  SELmemDataIn(&SELmemDataIn_), SELmemAddr(&SELmemAddr_),
20  read(&read_), write(&write_)
21 {
22  newPageIdOut.resize(4, 'X'); // Internal busses:
23  newPageIdOut.resize(4, 'X');
24  newPagePosOut.resize(4, 'X');
25  everyPagePosOut.resize(4, 'X');
26  pagePosIncremented.resize(4, 'X');
27  addrCounterOut.resize(4, 'X');
28  addrCounterIn.resize(4, '0');
29  memAddrBus.resize(4, 'X'); // Memory busses:
30  memDataBus.resize(4, 'X'); memDataOut.resize(4, 'X');
31
32  newPageId = new dRegisterE(*newPageBus, *clk, *LDnewPageId, newPageIdOut);
33  newPagePos = new dRegisterE(memDataOut, *clk, *LDnewPagePos, newPagePosOut);
34  everyPagePos = new dRegisterE(memDataOut, *clk, *LDeveryPagePos, everyPagePosOut);
35  compareNewEvery = new Comparator(newPagePosOut, everyPagePosOut, *l, *e, *g);
36  MUX2 = new Mux(newPageIdOut, addrCounterOut, *SELmemAddr, memAddrBus);
37  addrCounter = new upCounterRaE(addrCounterIn, *clk, *rst, *LDaddrCounter,
38                                *INCaddrCounter, addrCounterOut);
39  memory = new Memory(*rst, *clk, *read, *write, memDataBus, memDataOut);
40 }
```

LRUdatapath.cpp

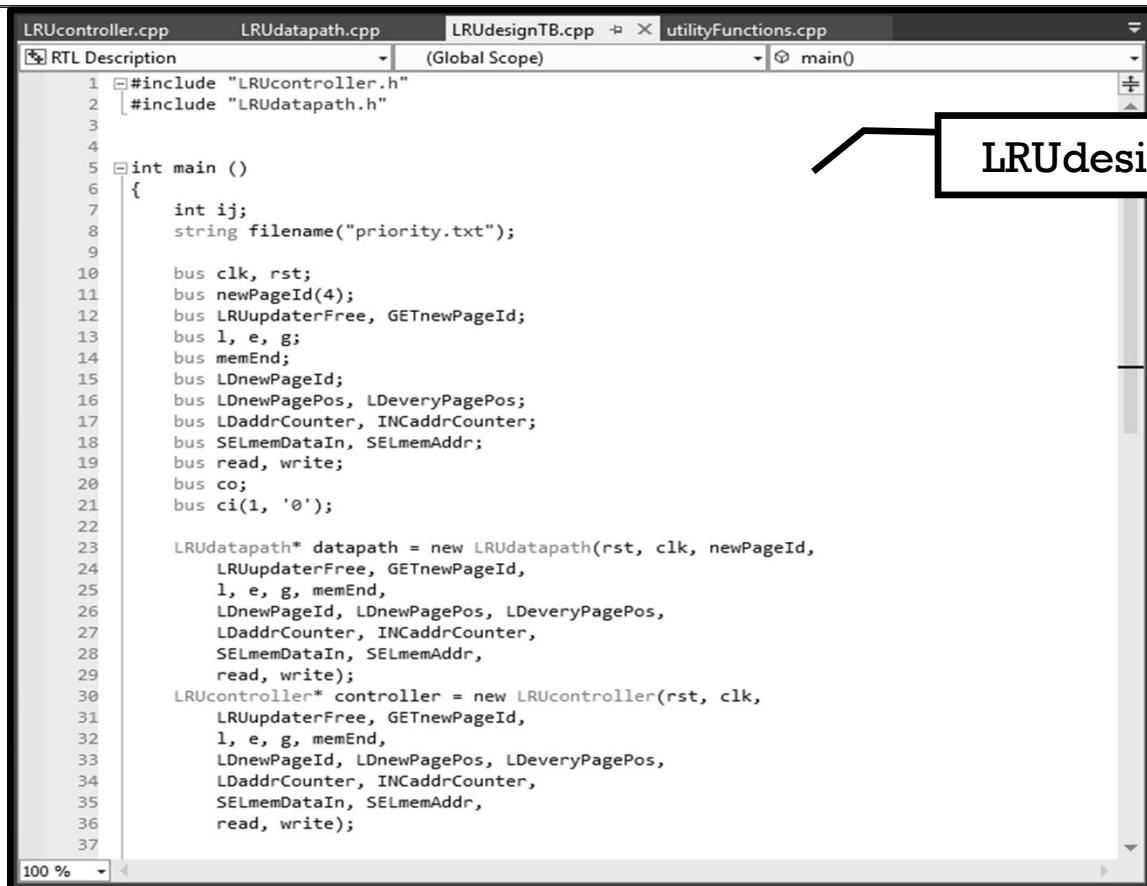
LRU Datapath



The screenshot shows a code editor window with multiple tabs. The active tab is `LRUdatapath.cpp`, which contains C++ code for an LRU datapath. The code includes declarations for various bus and memory components and an implementation of the `evl()` method. A black callout box with a white border and a black arrow points from the text "LRUdatapath.cpp" to the tab bar where `LRUdatapath.cpp` is selected.

```
3
4 LRUdatapath::LRUdatapath(bus& rst_, bus& clk_, bus& newPageBus_,
5 bus& LRUupdateFree_, bus& Completed_,
6 bus& l_, bus& e_, bus& g_, bus& memEnd_,
7 bus& LDnewPageId_, bus& LDnewPagePos_, bus& LDeveryPagePos_,
8 bus& LDaddrCounter_, bus& INCaddrCounter_,
9 bus& SELmemDataIn_, bus& SELmemAddr_,
10 bus& read_, bus& write_) { ... }
11
12 void LRUdatapath::evl()
13 {
14     newPageId->evl();
15     memDataBus = pagePosIncremented & *SELmemDataIn;
16     MUX2->evl();
17     memory->evl();
18     newPagePos->evl();
19     everyPagePos->evl();
20     pagePosIncremented = everyPagePosOut + "1";
21     addrCounter->evl();
22     compareNewEvery->evl();
23     *memEnd = ((addrCounterOut == "0000") ? "1" : "0");
24 }
25
26
```

LRU Testbench



The screenshot shows a code editor window with multiple tabs. The active tab is `LRUdesignTB.cpp`. The code in the editor is as follows:

```
#include "LRUcontroller.h"
#include "LRUdatapath.h"

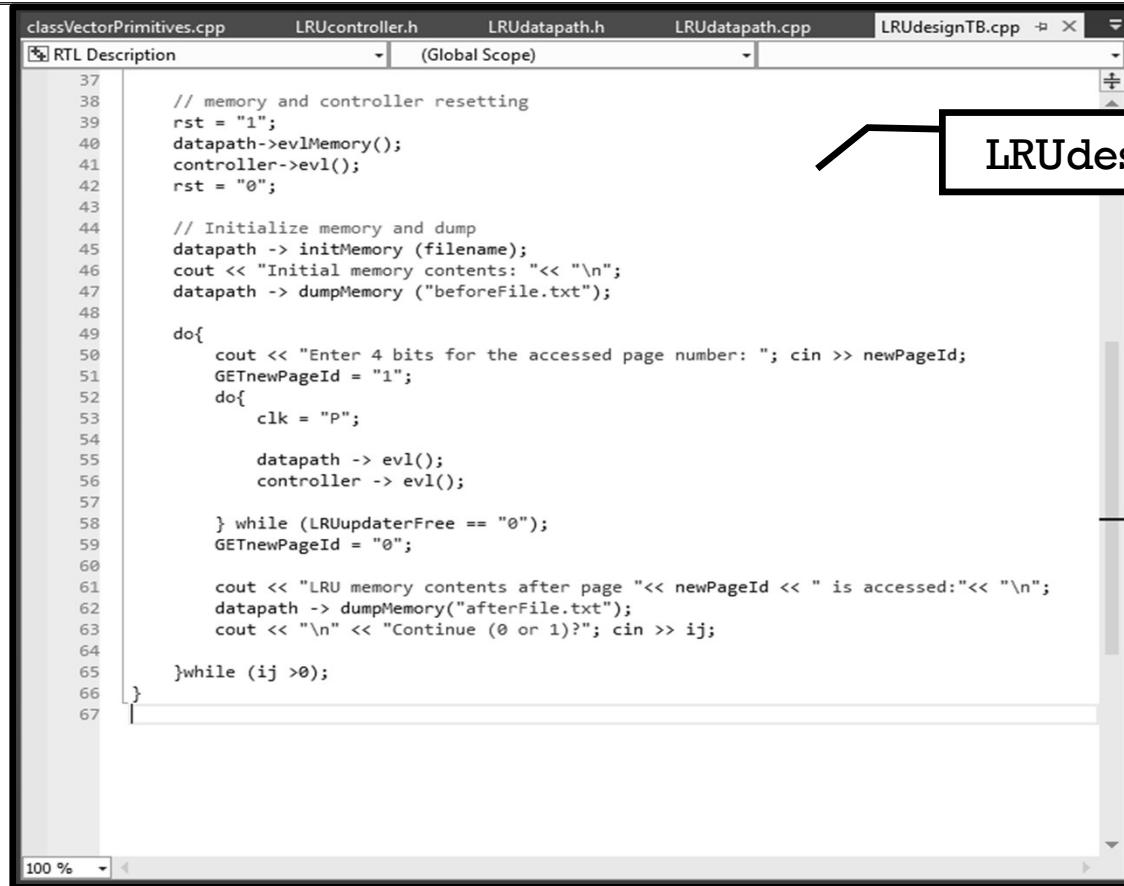
int main ()
{
    int ij;
    string filename("priority.txt");

    bus clk, rst;
    bus newPageId(4);
    bus LRUupdateFree, GETnewPageId;
    bus l, e, g;
    bus memEnd;
    bus LDnewPageId;
    bus LDnewPagePos, LDeveryPagePos;
    bus LDaddrCounter, INCaddrCounter;
    bus SELmemDataIn, SELmemAddr;
    bus read, write;
    bus co;
    bus ci(1, '0');

    LRUdatapath* datapath = new LRUdatapath(rst, clk, newPageId,
        LRUupdateFree, GETnewPageId,
        l, e, g, memEnd,
        LDnewPageId, LDnewPagePos, LDeveryPagePos,
        LDaddrCounter, INCaddrCounter,
        SELmemDataIn, SELmemAddr,
        read, write);
    LRUcontroller* controller = new LRUcontroller(rst, clk,
        LRUupdateFree, GETnewPageId,
        l, e, g, memEnd,
        LDnewPageId, LDnewPagePos, LDeveryPagePos,
        LDaddrCounter, INCaddrCounter,
        SELmemDataIn, SELmemAddr,
        read, write);
}
```

LRUdesignTB.cpp

LRU Testbench

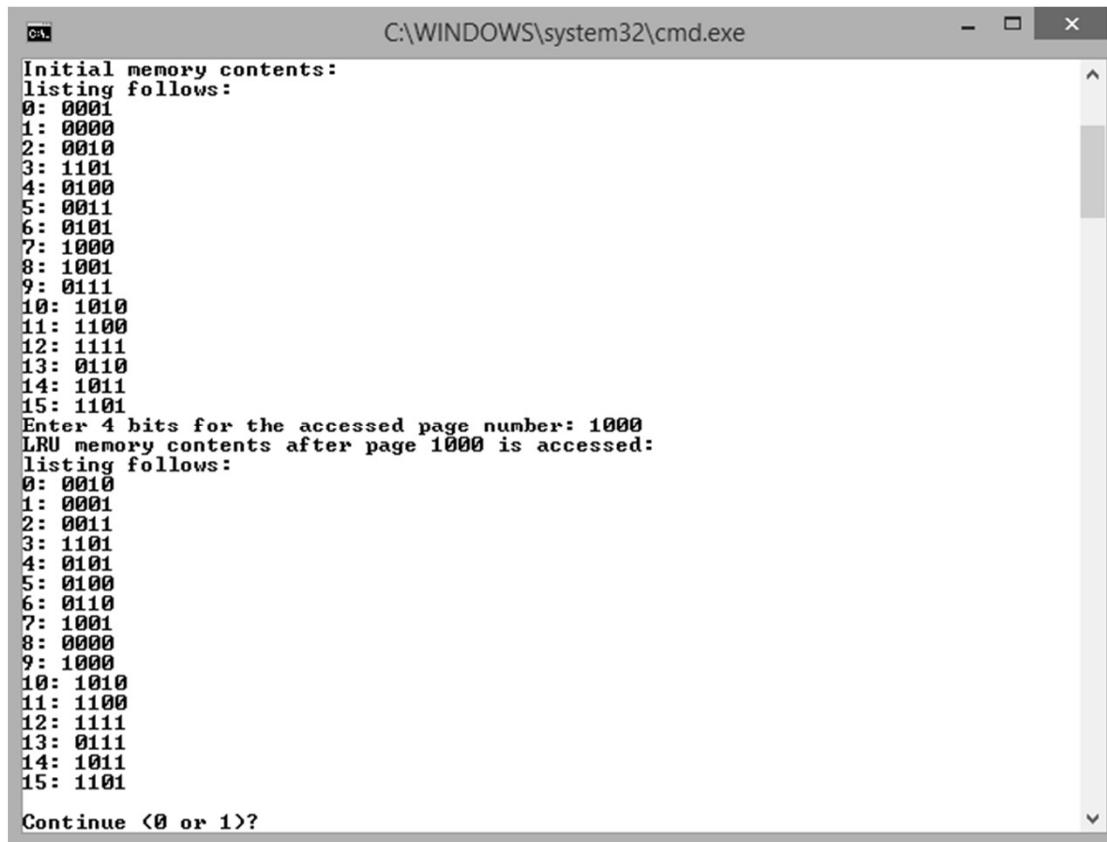


The screenshot shows a code editor window with multiple tabs. The active tab is `LRUdesignTB.cpp`. The code in the editor is as follows:

```
37 // memory and controller resetting
38 rst = "1";
39 datapath->evlMemory();
40 controller->evl();
41 rst = "0";
42
43 // Initialize memory and dump
44 datapath -> initMemory (filename);
45 cout << "Initial memory contents: "<< endl;
46 datapath -> dumpMemory ("beforeFile.txt");
47
48 do{
49     cout << "Enter 4 bits for the accessed page number: "; cin >> newPageId;
50     GETnewPageId = "1";
51     do{
52         clk = "P";
53
54         datapath -> evl();
55         controller -> evl();
56
57     } while (LRUupdateFree == "0");
58     GETnewPageId = "0";
59
60     cout << "LRU memory contents after page " << newPageId << " is accessed:<< endl;
61     datapath -> dumpMemory("afterFile.txt");
62     cout << "\n" << "Continue (0 or 1)?"; cin >> ij;
63
64     }while (ij >0);
65 }
66
67
```

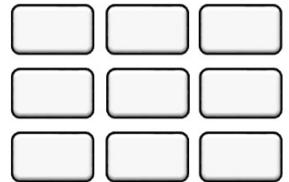
LRUdesignTB.cpp

LRU Output



A screenshot of a Windows Command Prompt window titled "cmd" with the path "C:\WINDOWS\system32\cmd.exe". The window displays the following text:

```
Initial memory contents:  
listing follows:  
0: 0001  
1: 0000  
2: 0010  
3: 1101  
4: 0100  
5: 0011  
6: 0101  
7: 1000  
8: 1001  
9: 0111  
10: 1010  
11: 1100  
12: 1111  
13: 0110  
14: 1011  
15: 1101  
Enter 4 bits for the accessed page number: 1000  
LRU memory contents after page 1000 is accessed:  
listing follows:  
0: 0010  
1: 0001  
2: 0011  
3: 1101  
4: 0101  
5: 0100  
6: 0110  
7: 1001  
8: 0000  
9: 1000  
10: 1010  
11: 1100  
12: 1111  
13: 0111  
14: 1011  
15: 1101  
Continue <0 or 1>?
```



Outline

Introduction

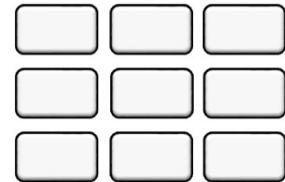
- + Bus Model and Operations
- + Basic Elements of RTL
- RTL Design Examples

Memory Structure

Moore Sequence Detector (11011)

Least- Recently-Used (LRU) Circuit

Exponential Circuit



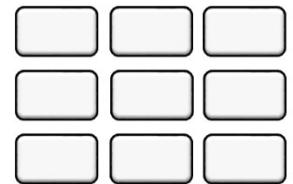
Exponential Circuit

◎ Taylor Series

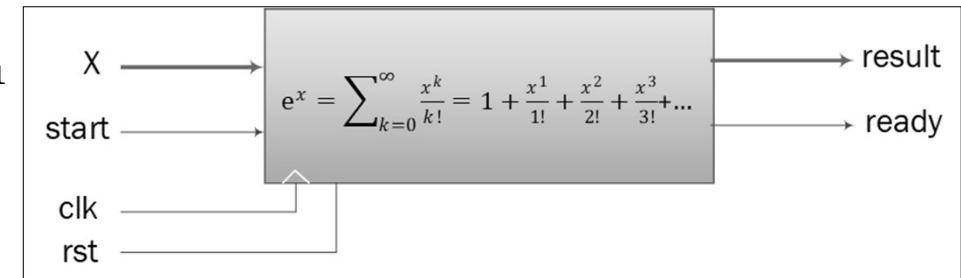
$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

```
e = 1;  
a = 1;  
for( i = 1; i < n; i++ )  
{  
    a = a * x * ( 1 / i );  
    e = e + a;  
}
```

Exponential Circuit



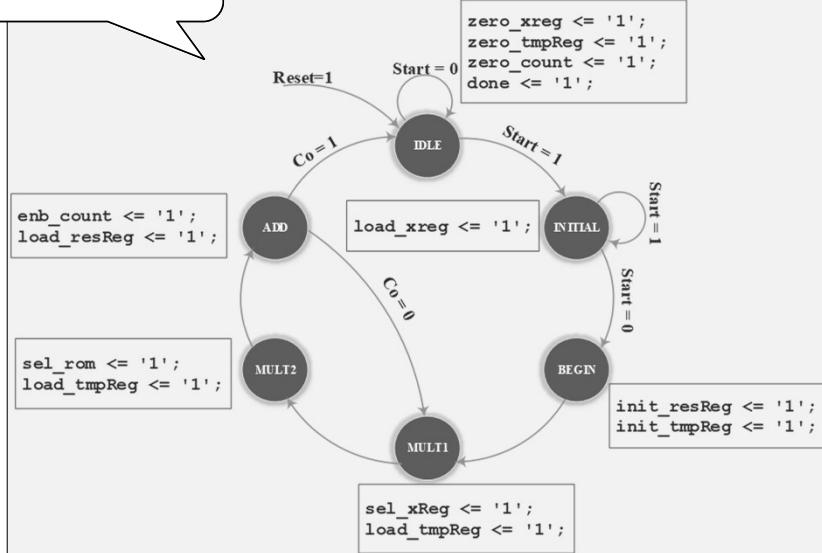
- The circuit calculates e^x using Taylor expansion.
- The input is an 8-bit fixed-point number.
- The output is a 10-bit fixed-point number including 2 integer bits and 8 fractional bits.
- The circuit receives x as the input with the pulse on the start signal.
- The calculation continues for 8 iterations.
 - With 8-bit input size, x is in the range between 0.00000000 for the smallest and 0.11111111 for the largest value.
- When the result becomes ready, done signal will be issued.
 - Smallest output = 1 when e^0
 - Largest output = 10.1010111 (2.68357) when e^1



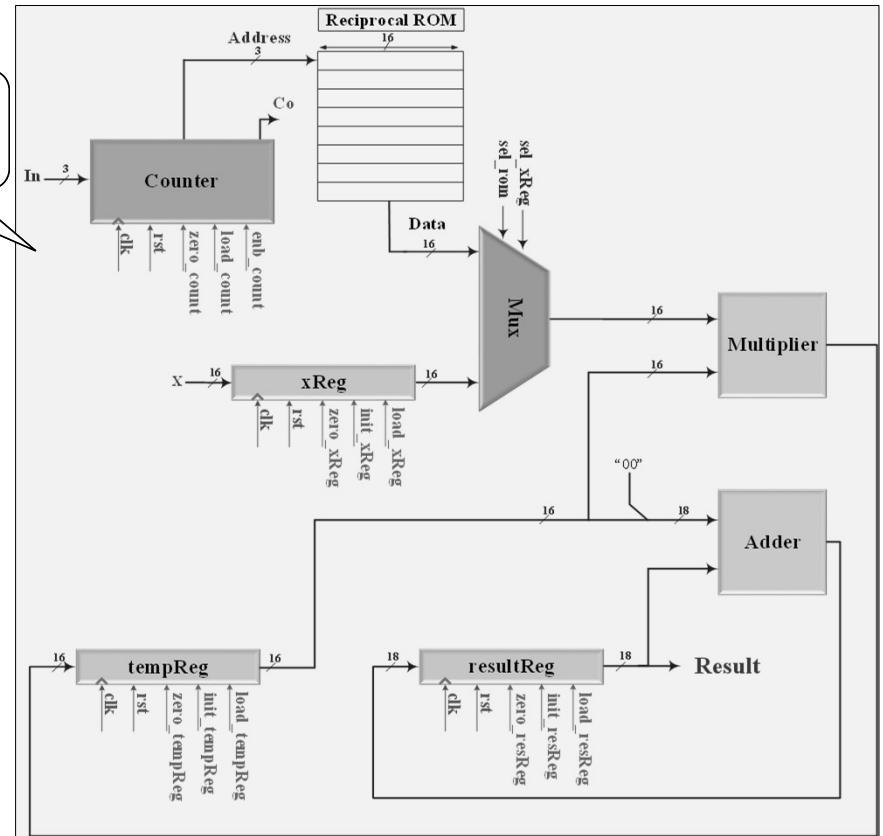
Exponential Circuit

◎ RTL design

Controller



Datapath



Exponential Circuit – Datapath

EXPController.cpp EXPdatapath.cpp EXPDesign.cpp EXPdesignTB.cpp EXPdatapath.h

```

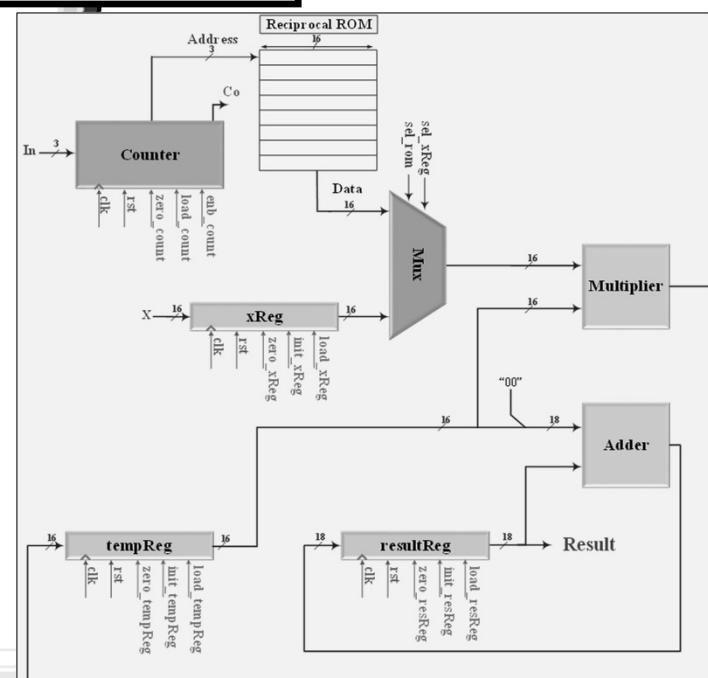
5 class expDatapath
6 {
7     bus *clk, *rst, *read;
8     bus *loadExponent, *rstExponent;
9     bus *loadTerm, *initTerm, *rstTerm;
10    bus *selTableData;
11    bus *rstResultReg, *initResultReg, *loadResultReg;
12    bus *enCounter, *rstCounter, *initCounter;
13    bus *x, *co, *addr, *result;
14
15    dRegisterRaE* expReg;
16    dRegisterRaE* termReg;
17    dRegisterRaE* resultReg;
18
19    Mux* M1;
20    upCounterRsE* indexCounter;
21    Memory* fractionsMemory;
22    Multiplier* Mult;
23    Adder* Add;
24
25    // internal busses
26    bus exponent;
27    bus term;
28    bus countValue;
29    bus initDCounter;
30    bus multRInput;
31    bus addResult;
32    bus resultInput;
33    bus multResult;
34    bus termInput;
35    bus initDataCounter;
36    bus memDatabase;
37    bus tableData;
38    bus addInput1;
39
40    bus cin, cout1, write;
41    bus enableResultReg;
42    bus enableTermReg;
43    bus enableExponent;
44    bus exponentInput;
45

```

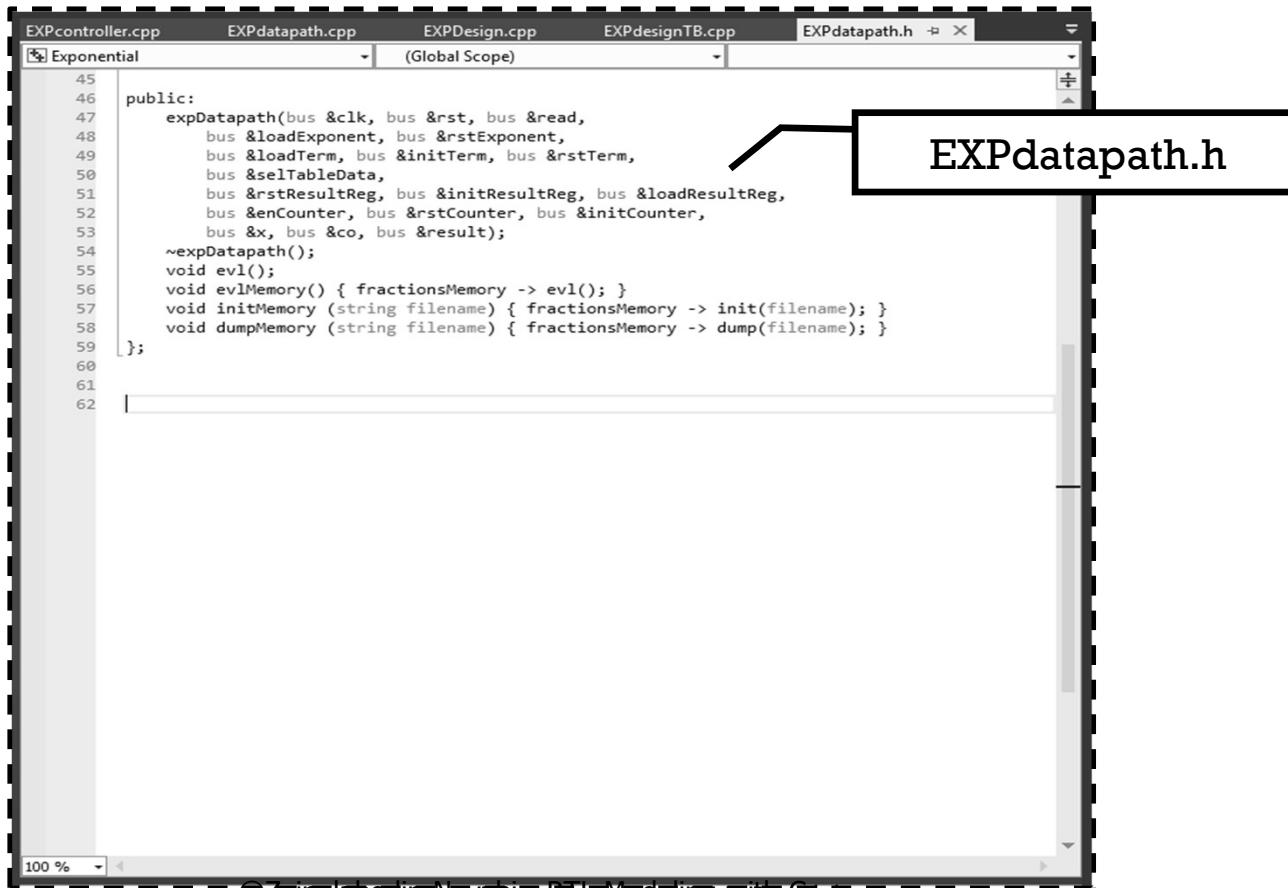
100 %

Register for saving partial results

EXPdatapath.h



Exponential Circuit – Datapath

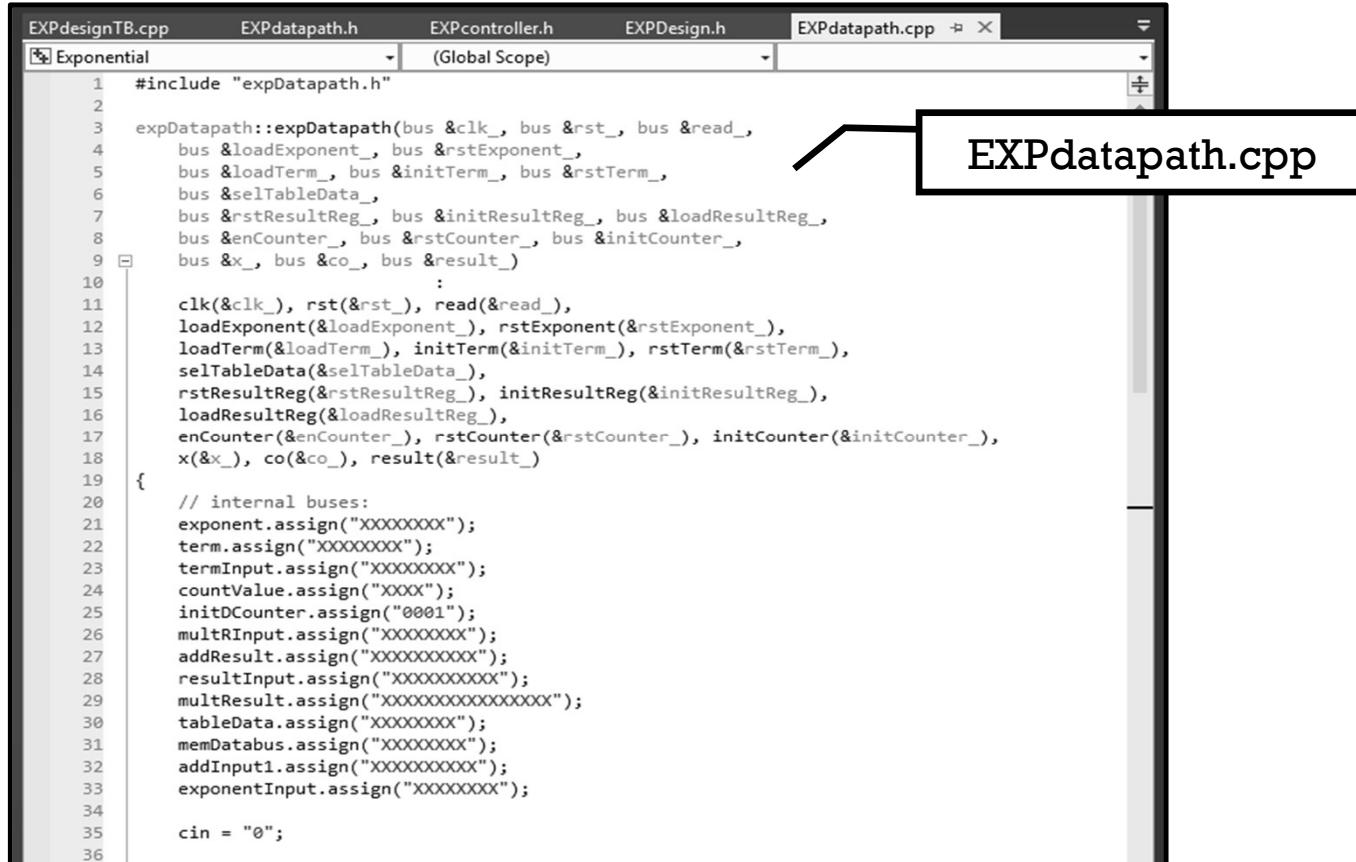


The screenshot shows a code editor window with multiple tabs at the top: EXPcontroller.cpp, EXPdatapath.cpp, EXPDesign.cpp, EXPdesignTB.cpp, and EXPdatapath.h. The EXPdatapath.h tab is active. A search bar below the tabs contains the text "Exponential". The main pane displays the following C++ code:

```
45 public:
46     expDatapath(bus &clk, bus &rst, bus &read,
47                 bus &loadExponent, bus &rstExponent,
48                 bus &loadTerm, bus &initTerm, bus &rstTerm,
49                 bus &selTableData,
50                 bus &rstResultReg, bus &initResultReg, bus &loadResultReg,
51                 bus &enCounter, bus &rstCounter, bus &initCounter,
52                 bus &x, bus &co, bus &result);
53     ~expDatapath();
54     void evl();
55     void evlMemory() { fractionsMemory -> evl(); }
56     void initMemory (string filename) { fractionsMemory -> init(filename); }
57     void dumpMemory (string filename) { fractionsMemory -> dump(filename); }
58 };
59
60
61
62
```

A callout box with a black border and white background points from the text "EXPdatapath.h" to the active tab in the code editor.

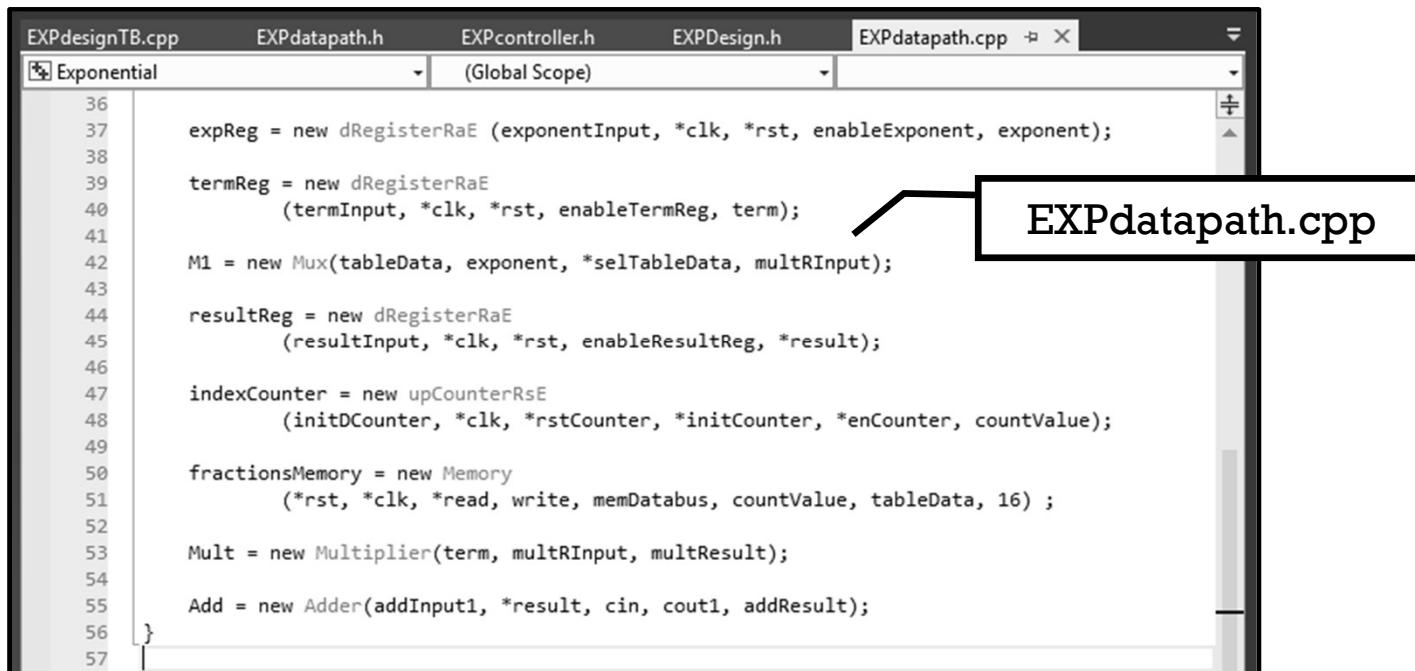
Exponential Circuit – Datapath



The screenshot shows a code editor window with multiple tabs. The active tab is `EXPdatapath.cpp`. The code itself is a constructor for a class named `expDatopath`, which takes various bus pointers as parameters. The code initializes these buses with specific values (e.g., "XXXXXXXX") and sets up internal variables like `exponent`, `term`, etc. A callout box with an arrow points from the text "EXPdatapath.cpp" to the tab bar where the file is listed.

```
1 #include "expDatopath.h"
2
3 expDatopath::expDatopath(bus &clk_, bus &rst_, bus &read_,
4     bus &loadExponent_, bus &rstExponent_,
5     bus &loadTerm_, bus &initTerm_, bus &rstTerm_,
6     bus &selTableData_,
7     bus &rstResultReg_, bus &initResultReg_, bus &loadResultReg_,
8     bus &enCounter_, bus &rstCounter_, bus &initCounter_,
9     bus &x_, bus &co_, bus &result_)
10    :
11    clk(&clk_), rst(&rst_), read(&read_),
12    loadExponent(&loadExponent_), rstExponent(&rstExponent_),
13    loadTerm(&loadTerm_), initTerm(&initTerm_), rstTerm(&rstTerm_),
14    selTableData(&selTableData_),
15    rstResultReg(&rstResultReg_), initResultReg(&initResultReg_),
16    loadResultReg(&loadResultReg_),
17    enCounter(&enCounter_), rstCounter(&rstCounter_), initCounter(&initCounter_),
18    x(&x_), co(&co_), result(&result_)
19 {
20     // internal buses:
21     exponent.assign("XXXXXXXX");
22     term.assign("XXXXXXXX");
23     termInput.assign("XXXXXXXX");
24     countValue.assign("XXXX");
25     initDCounter.assign("0001");
26     multRInput.assign("XXXXXXXXXX");
27     addResult.assign("XXXXXXXXXX");
28     resultInput.assign("XXXXXXXXXX");
29     multResult.assign("XXXXXXXXXXXXXXXX");
30     tableData.assign("XXXXXXXX");
31     memDatabus.assign("XXXXXXXX");
32     addInput1.assign("XXXXXXXX");
33     exponentInput.assign("XXXXXXXX");
34
35     cin = "0";
36 }
```

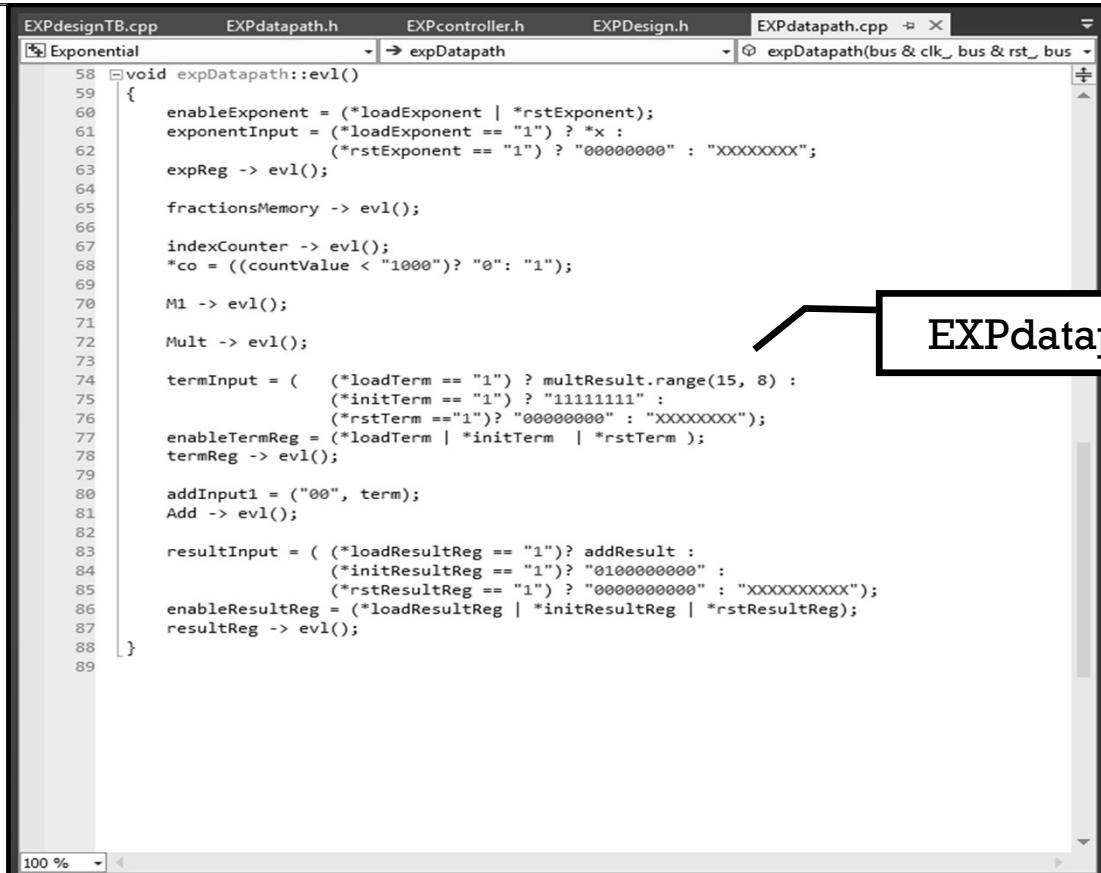
Exponential Circuit – Datapath



```
EXPdesignTB.cpp EXPdatapath.h EXPcontroller.h EXPDesign.h EXPdatapath.cpp
Exponential (Global Scope)
36 expReg = new dRegisterRaE (exponentInput, *clk, *rst, enableExponent, exponent);
37
38 termReg = new dRegisterRaE
39     (termInput, *clk, *rst, enableTermReg, term);
40
41 M1 = new Mux(tableData, exponent, *selTableData, multRInput);
42
43 resultReg = new dRegisterRaE
44     (resultInput, *clk, *rst, enableResultReg, *result);
45
46 indexCounter = new upCounterRsE
47     (initDCounter, *clk, *rstCounter, *initCounter, *enCounter, countValue);
48
49 fractionsMemory = new Memory
50     (*rst, *clk, *read, write, memDatabus, countValue, tableData, 16) ;
51
52 Mult = new Multiplier(term, multRInput, multResult);
53
54 Add = new Adder(addInput1, *result, cin, cout1, addResult);
55
56 }
57 }
```

EXPdatapath.cpp

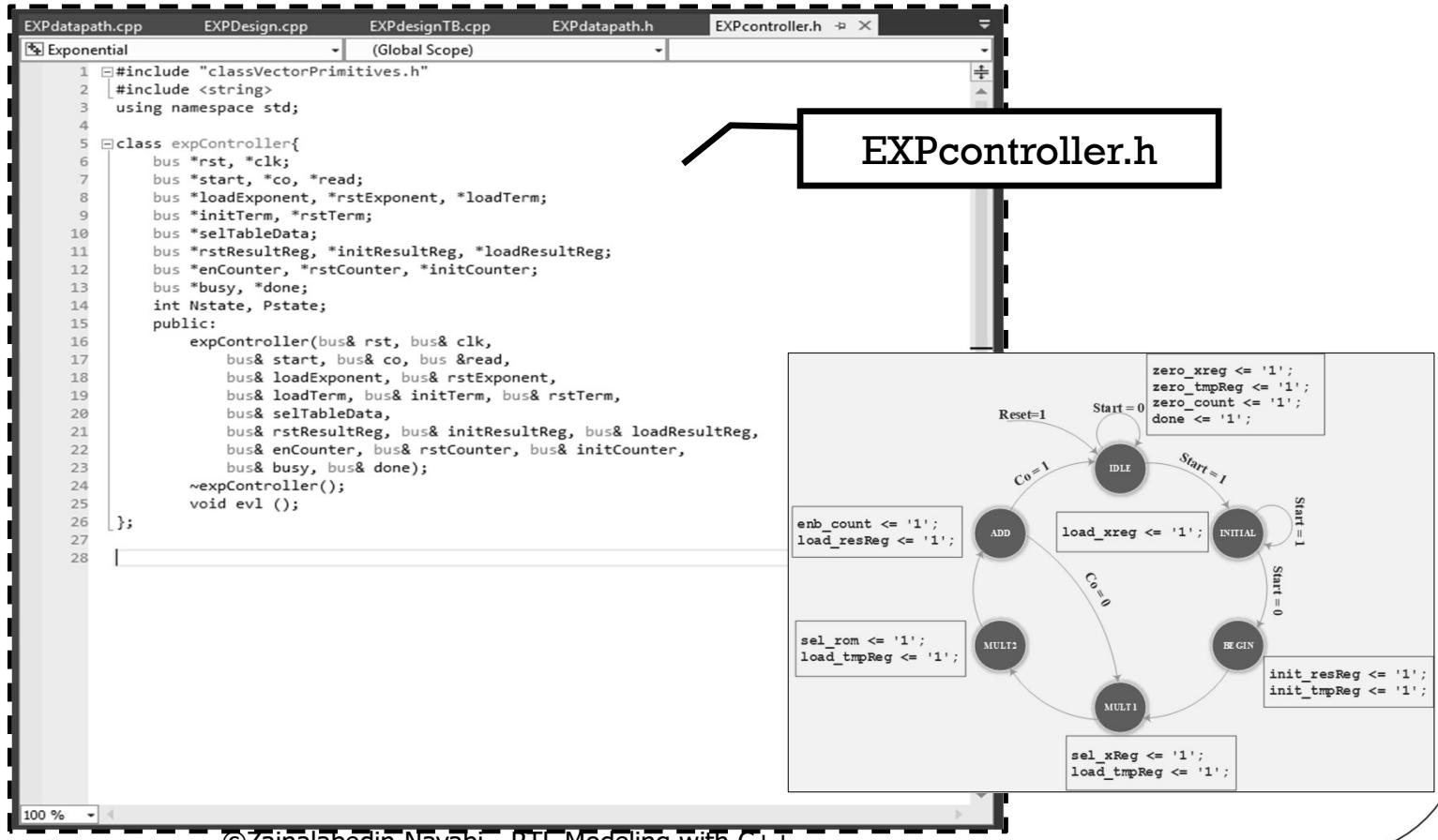
Exponential Circuit – Datapath



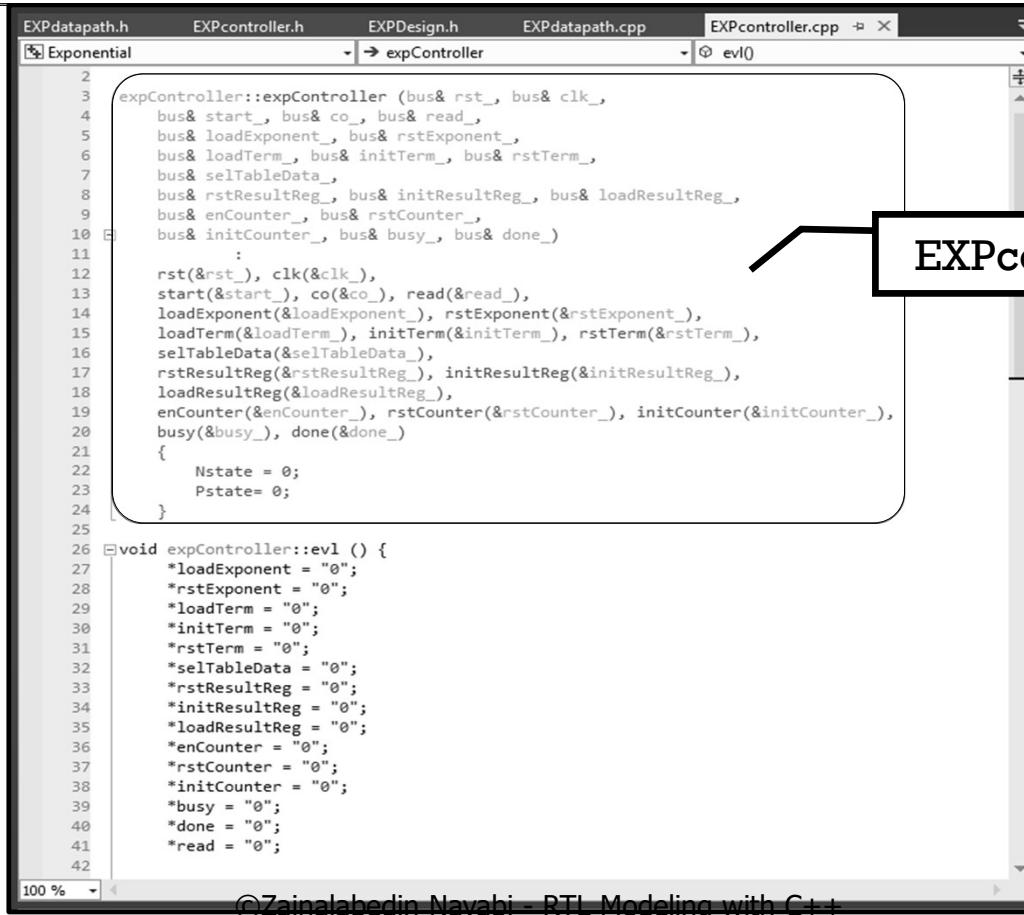
EXPdesignTB.cpp EXPdatapath.h EXPcontroller.h EXPDesign.h EXPdatapath.cpp expDatopath
Exponential expDatopath(bus & clk_, bus & rst_, bus)
58 void EXPdatapath::evl()
59 {
60 enableExponent = (*loadExponent | *rstExponent);
61 exponentInput = (*loadExponent == "1") ? "x" :
62 (*rstExponent == "1") ? "00000000" : "xxxxxxxx";
63 expReg -> evl();
64
65 fractionsMemory -> evl();
66
67 indexCounter -> evl();
68 *co = ((countValue < "1000")? "0": "1");
69
70 M1 -> evl();
71
72 Mult -> evl();
73
74 termInput = ((*loadTerm == "1") ? multResult.range(15, 8) :
75 (*initTerm == "1") ? "11111111" :
76 (*rstTerm == "1") ? "00000000" : "xxxxxxxx");
77 enableTermReg = (*loadTerm | *initTerm | *rstTerm);
78 termReg -> evl();
79
80 addInput1 = ("00", term);
81 Add -> evl();
82
83 resultInput = ((*loadResultReg == "1")? addResult :
84 (*initResultReg == "1")? "0100000000" :
85 (*rstResultReg == "1") ? "0000000000" : "xxxxxxxxxxxx");
86 enableResultReg = (*loadResultReg | *initResultReg | *rstResultReg);
87 resultReg -> evl();
88 }
89
100 %

EXPdatapath.cpp

Exponential Circuit – Controller



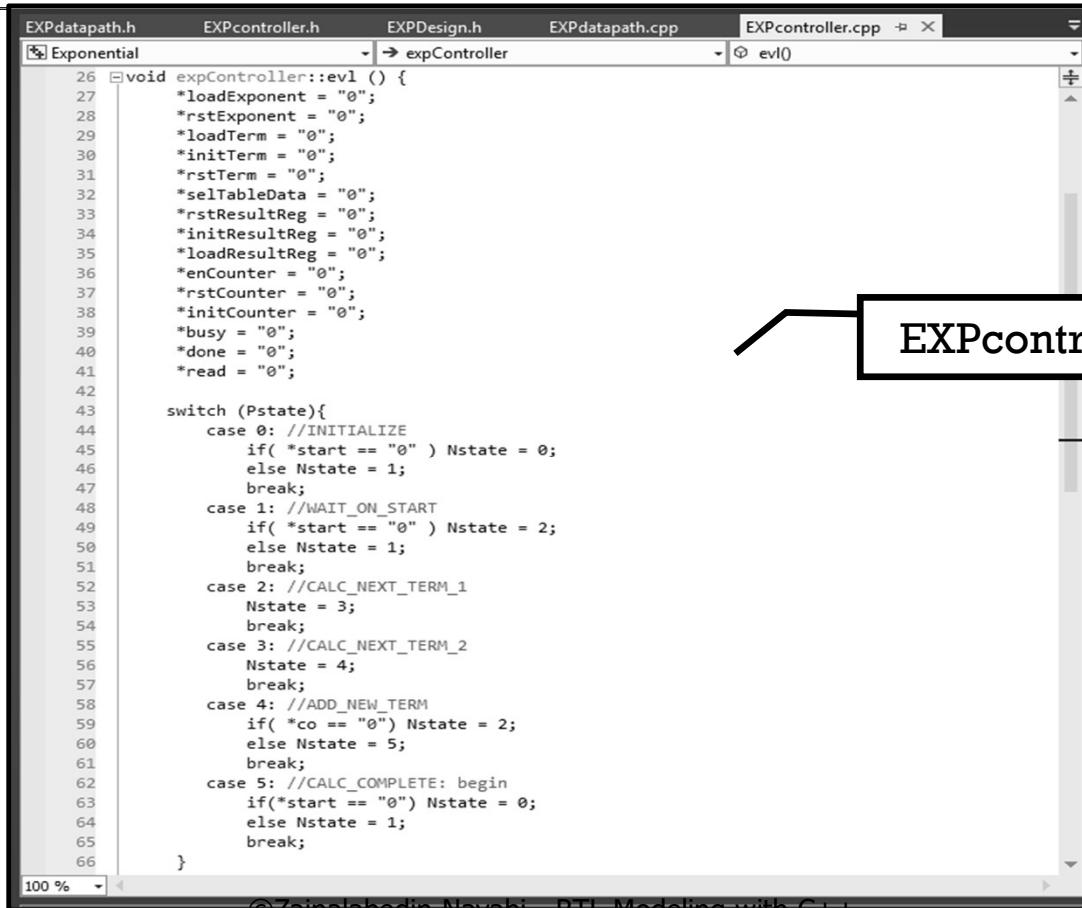
Exponential Circuit – Controller



```
EXPdatapath.h EXPController.h EXPDesign.h EXPdatapath.cpp EXPcontroller.cpp evl()
2
3 expController::expController (bus& rst_, bus& clk_,
4 bus& start_, bus& co_, bus& read_,
5 bus& loadExponent_, bus& rstExponent_,
6 bus& loadTerm_, bus& initTerm_, bus& rstTerm_,
7 bus& selTableData_,
8 bus& rstResultReg_, bus& initResultReg_, bus& loadResultReg_,
9 bus& enCounter_, bus& rstCounter_,
10 bus& initCounter_, bus& busy_, bus& done_)
11 :
12     rst(&rst_), clk(&clk_),
13     start(&start_), co(&co_), read(&read_),
14     loadExponent(&loadExponent_), rstExponent(&rstExponent_),
15     loadTerm(&loadTerm_), initTerm(&initTerm_), rstTerm(&rstTerm_),
16     selTableData(&selTableData_),
17     rstResultReg(&rstResultReg_), initResultReg(&initResultReg_),
18     loadResultReg(&loadResultReg_),
19     enCounter(&enCounter_), rstCounter(&rstCounter_), initCounter(&initCounter_),
20     busy(&busy_), done(&done_)
21 {
22     Nstate = 0;
23     Pstate= 0;
24 }
25
26 void expController::evl () {
27     *loadExponent = "0";
28     *rstExponent = "0";
29     *loadTerm = "0";
30     *initTerm = "0";
31     *rstTerm = "0";
32     *selTableData = "0";
33     *rstResultReg = "0";
34     *initResultReg = "0";
35     *loadResultReg = "0";
36     *enCounter = "0";
37     *rstCounter = "0";
38     *initCounter = "0";
39     *busy = "0";
40     *done = "0";
41     *read = "0";
42 }
```

EXPcontroller.cpp

Exponential Circuit – Controller



EXPdatapath.h EXPcontroller.h EXPDesign.h EXPdatapath.cpp EXPcontroller.cpp evl()

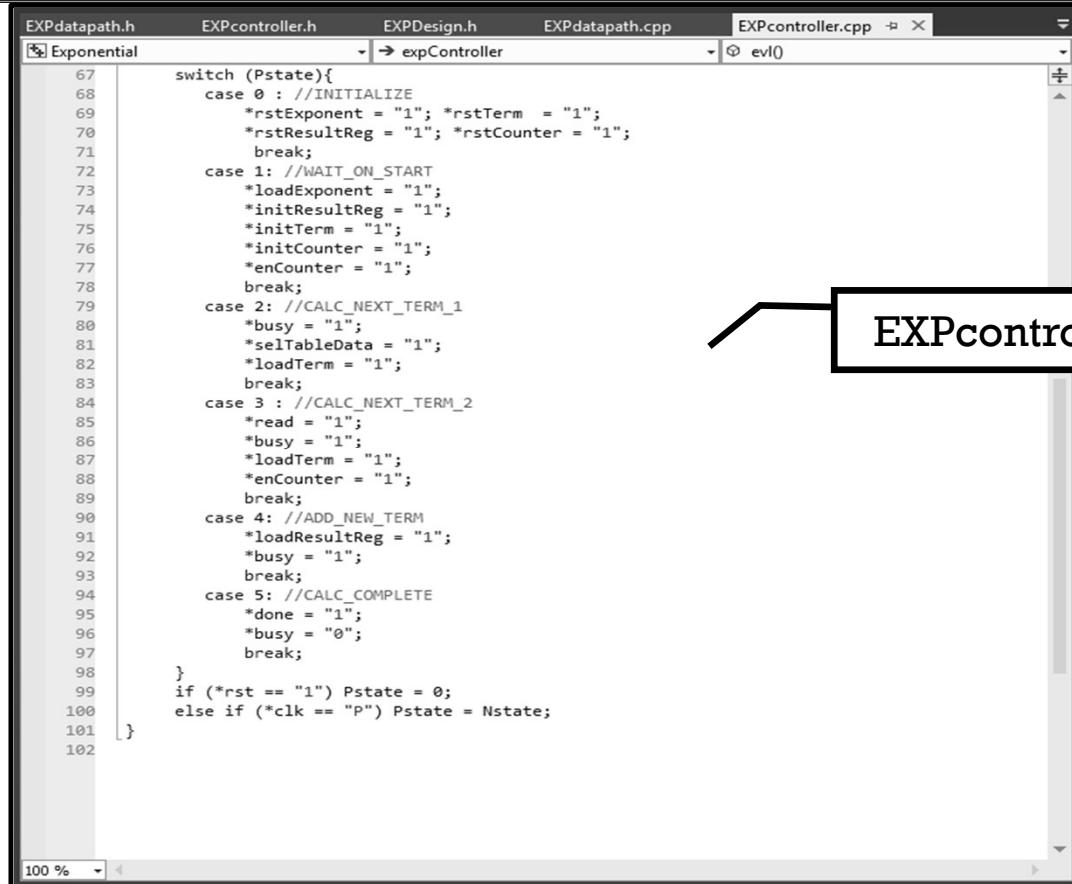
```
26 void expController::evl () {
27     *loadExponent = "0";
28     *rstExponent = "0";
29     *loadTerm = "0";
30     *initTerm = "0";
31     *rstTerm = "0";
32     *selTableData = "0";
33     *rstResultReg = "0";
34     *initResultReg = "0";
35     *loadResultReg = "0";
36     *enCounter = "0";
37     *rstCounter = "0";
38     *initCounter = "0";
39     *busy = "0";
40     *done = "0";
41     *read = "0";
42
43     switch (Pstate){
44         case 0: //INITIALIZE
45             if( *start == "0" ) Nstate = 0;
46             else Nstate = 1;
47             break;
48         case 1: //WAIT_ON_START
49             if( *start == "0" ) Nstate = 2;
50             else Nstate = 1;
51             break;
52         case 2: //CALC_NEXT_TERM_1
53             Nstate = 3;
54             break;
55         case 3: //CALC_NEXT_TERM_2
56             Nstate = 4;
57             break;
58         case 4: //ADD_NEW_TERM
59             if( *co == "0" ) Nstate = 2;
60             else Nstate = 5;
61             break;
62         case 5: //CALC_COMPLETE: begin
63             if(*start == "0") Nstate = 0;
64             else Nstate = 1;
65             break;
66     }
}
```

100 %

©Zainalabedin Navabi - RTL Modeling with C++

EXPcontroller.cpp

Exponential Circuit – Controller

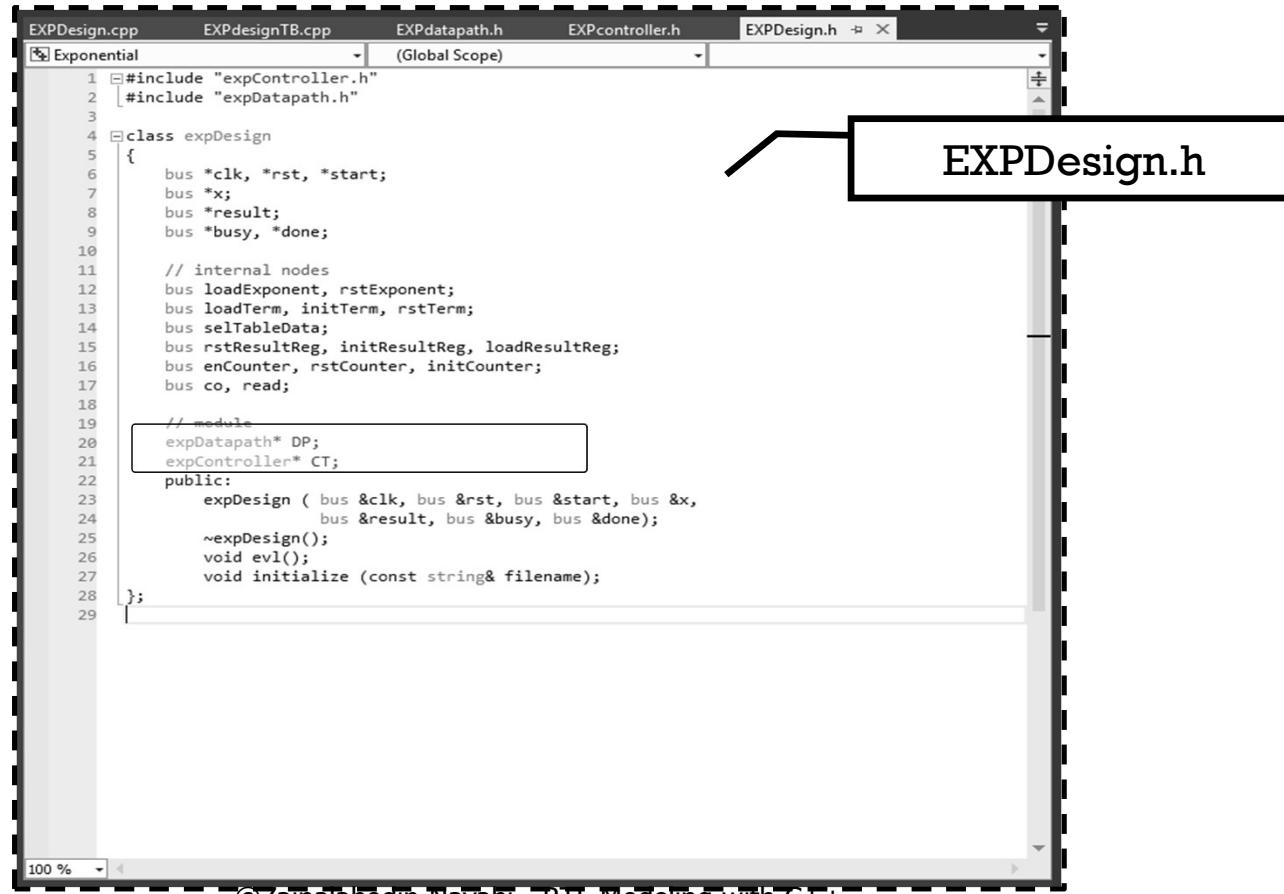


EXPcontroller.cpp

```
EXPdatapath.h EXPcontroller.h EXPDesign.h EXPdatapath.cpp EXPcontroller.cpp evl()
Exponential expController

67 switch (Pstate){
68     case 0 : //INITIALIZE
69         *rstExponent = "1"; *rstTerm = "1";
70         *rstResultReg = "1"; *rstCounter = "1";
71         break;
72     case 1: //WAIT_ON_START
73         *loadExponent = "1";
74         *initResultReg = "1";
75         *initTerm = "1";
76         *initCounter = "1";
77         *enCounter = "1";
78         break;
79     case 2: //CALC_NEXT_TERM_1
80         *busy = "1";
81         *selTableData = "1";
82         *loadTerm = "1";
83         break;
84     case 3 : //CALC_NEXT_TERM_2
85         *read = "1";
86         *busy = "1";
87         *loadTerm = "1";
88         *enCounter = "1";
89         break;
90     case 4: //ADD_NEW_TERM
91         *loadResultReg = "1";
92         *busy = "1";
93         break;
94     case 5: //CALC_COMPLETE
95         *done = "1";
96         *busy = "0";
97         break;
98     }
99     if (*rst == "1") Pstate = 0;
100    else if (*clk == "P") Pstate = Nstate;
101
102 }
```

Exponential Circuit – Design



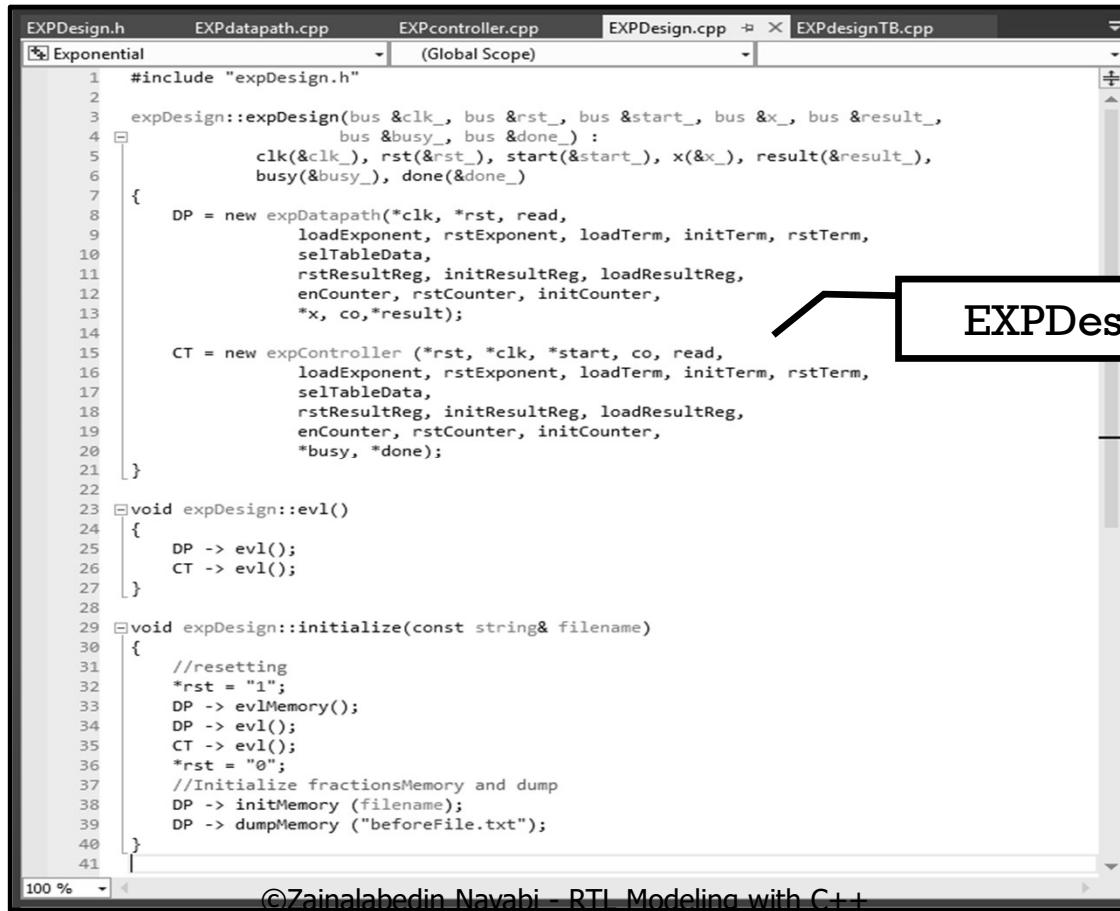
The image shows a screenshot of a code editor with multiple tabs at the top: EXPDesign.cpp, EXPdesignTB.cpp, EXPdatapath.h, EXPcontroller.h, and EXPDesign.h. The EXPDesign.h tab is active. A callout box points from the text "EXPDesign.h" to the active tab. The code editor displays the following C++ code:

```
EXPDesign.cpp EXPdesignTB.cpp EXPdatapath.h EXPcontroller.h EXPDesign.h
EXPDesign.h (Global Scope)

1 #include "expController.h"
2 #include "expDatapath.h"
3
4 class expDesign
5 {
6     bus *clk, *rst, *start;
7     bus *x;
8     bus *result;
9     bus *busy, *done;
10
11    // internal nodes
12    bus loadExponent, rstExponent;
13    bus loadTerm, initTerm, rstTerm;
14    bus selTableData;
15    bus rstResultReg, initResultReg, loadResultReg;
16    bus enCounter, rstCounter, initCounter;
17    bus co, read;
18
19    // module
20    expDatapath* DP;
21    expController* CT;
22
23 public:
24     expDesign ( bus &clk, bus &rst, bus &start, bus &x,
25                 bus &result, bus &busy, bus &done);
26     ~expDesign();
27     void eval();
28     void initialize (const string& filename);
29 }
```

The code defines a class `expDesign` with various bus pointers and methods for initializing and evaluating the design.

Exponential Circuit – Design



EXPDesign.h EXPdatapath.cpp EXPcontroller.cpp EXPDesign.cpp EXPdesignTB.cpp

Exponential (Global Scope)

```
1 #include "expDesign.h"
2
3 expDesign::expDesign(bus &clk_, bus &rst_, bus &start_, bus &x_, bus &result_,
4                      bus &busy_, bus &done_) :
5     clk(&clk_), rst(&rst_), start(&start_), x(&x_), result(&result_),
6     busy(&busy_), done(&done_)
7 {
8     DP = new expDatapath(*clk, *rst, read,
9                           loadExponent, rstExponent, loadTerm, initTerm, rstTerm,
10                          selTableData,
11                          rstResultReg, initResultReg, loadResultReg,
12                          enCounter, rstCounter, initCounter,
13                          *x, co, *result);
14
15     CT = new expController (*rst, *clk, *start, co, read,
16                            loadExponent, rstExponent, loadTerm, initTerm, rstTerm,
17                            selTableData,
18                            rstResultReg, initResultReg, loadResultReg,
19                            enCounter, rstCounter, initCounter,
20                            *busy, *done);
21 }
22
23 void expDesign::evl()
24 {
25     DP -> evl();
26     CT -> evl();
27 }
28
29 void expDesign::initialize(const string& filename)
30 {
31     //resetting
32     *rst = "1";
33     DP -> evlMemory();
34     DP -> evl();
35     CT -> evl();
36     *rst = "0";
37     //Initialize fractionsMemory and dump
38     DP -> initMemory (filename);
39     DP -> dumpMemory ("beforeFile.txt");
40 }
41 }
```

©Zainalabedin Navabi - RTL Modeling with C++

EXPDesign.cpp

Exponential Circuit – Testbench



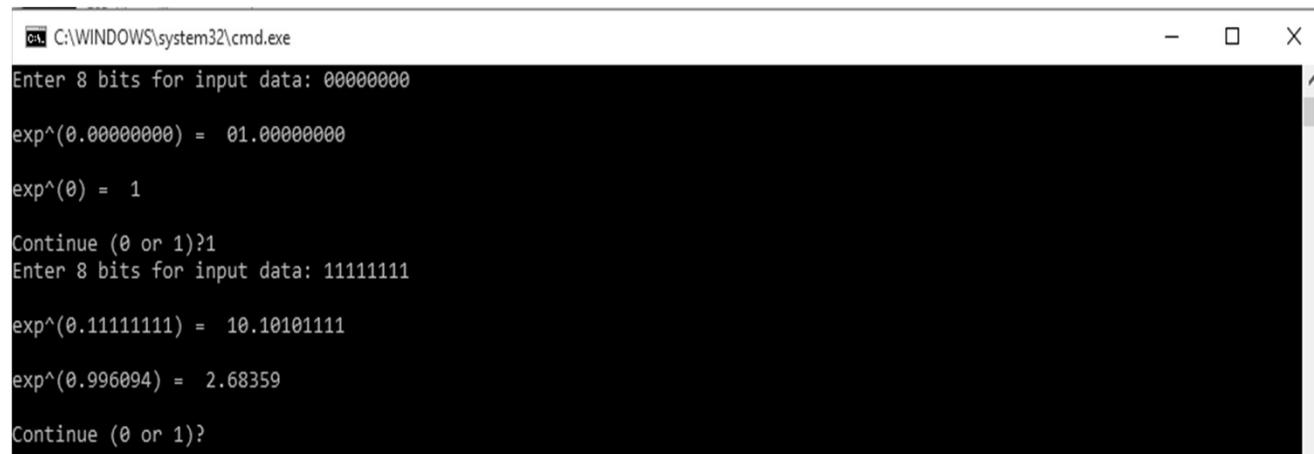
The diagram illustrates the connection between a testbench file and its corresponding C++ code. A black bracket labeled "EXPdesignTB.cpp" points to a code editor window. The window displays a C++ program named "EXPdesignTB.cpp" which includes an "Exponential" module. The code handles user input for 8-bit binary numbers, initializes the module, and performs iterative calculations until a condition is met.

```
#include "expDesign.h"
int main ()
{
    int ij = 0;
    string filename("coefficient.txt");
    bus clk, rst, start, done, busy;
    bus x(8);
    bus result(10);

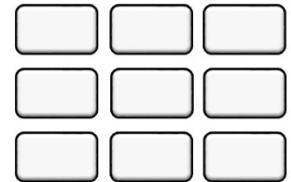
    // module instantiation
    expDesign *EXP = new expDesign(clk, rst, start, x, result, busy, done);
    EXP -> initialize(filename);

    do {
        cout << "Enter 8 bits for input data: ";
        cin >> x;
        cout << endl;
        start ="1";
        do{
            clk = "P";
            EXP -> evl();
            start ="0";
        } while (done == "0");
        cout << "exp^(8." << x << ") = "
             << result.range(9,8)<< "."<< result.range(7,0) << "\n";
        cout << endl << "exp^( " << fval(x) << " ) = "
             << double(result.range(9,8).ival()) + fval(result.range(7,0)) << "\n";
        cout << "\n" << "Continue (0 or 1)?";
        cin >> ij;
    } while (ij >0);
}
```

Exponential Circuit – Result



```
C:\WINDOWS\system32\cmd.exe
Enter 8 bits for input data: 00000000
exp^(0.00000000) =  01.00000000
exp^(0) =  1
Continue (0 or 1)?1
Enter 8 bits for input data: 11111111
exp^(0.11111111) =  10.10101111
exp^(0.996094) =  2.68359
Continue (0 or 1)?
```



Conclusions

⦿ We have covered these topics:

- Bus Model and Operations
 - Bus class
 - Array attributes
 - Logical operations
 - Adding operations
 - Relational operations
 - IO operations
- Basic Elements of RTL
 - Combinational elements
 - Sequential elements
- RTL Design Examples
 - Memory Structure
 - Moore Sequence Detector (11011)
 - Least- Recently-Used (LRU) Circuit
 - Exponential Circuit