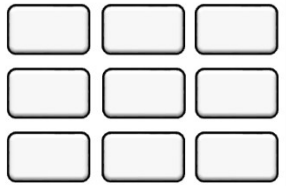
Chapter 3

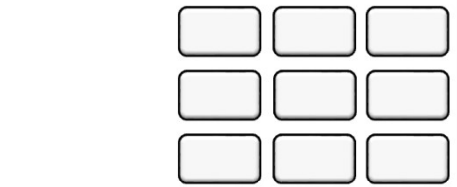


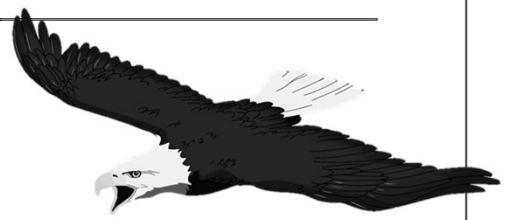
RT Level Modeling with C++

Zainalabedin Navabi

Slides prepared by: Hanieh Hashemi

RT Level Modeling in C++



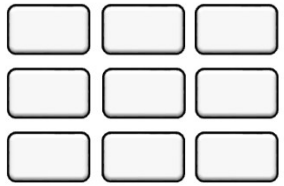


A BIRD'S-EYE VIEW OF OUTLINE

Introduction

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

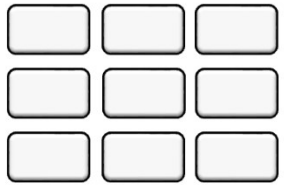
RT Level Modeling with C++



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

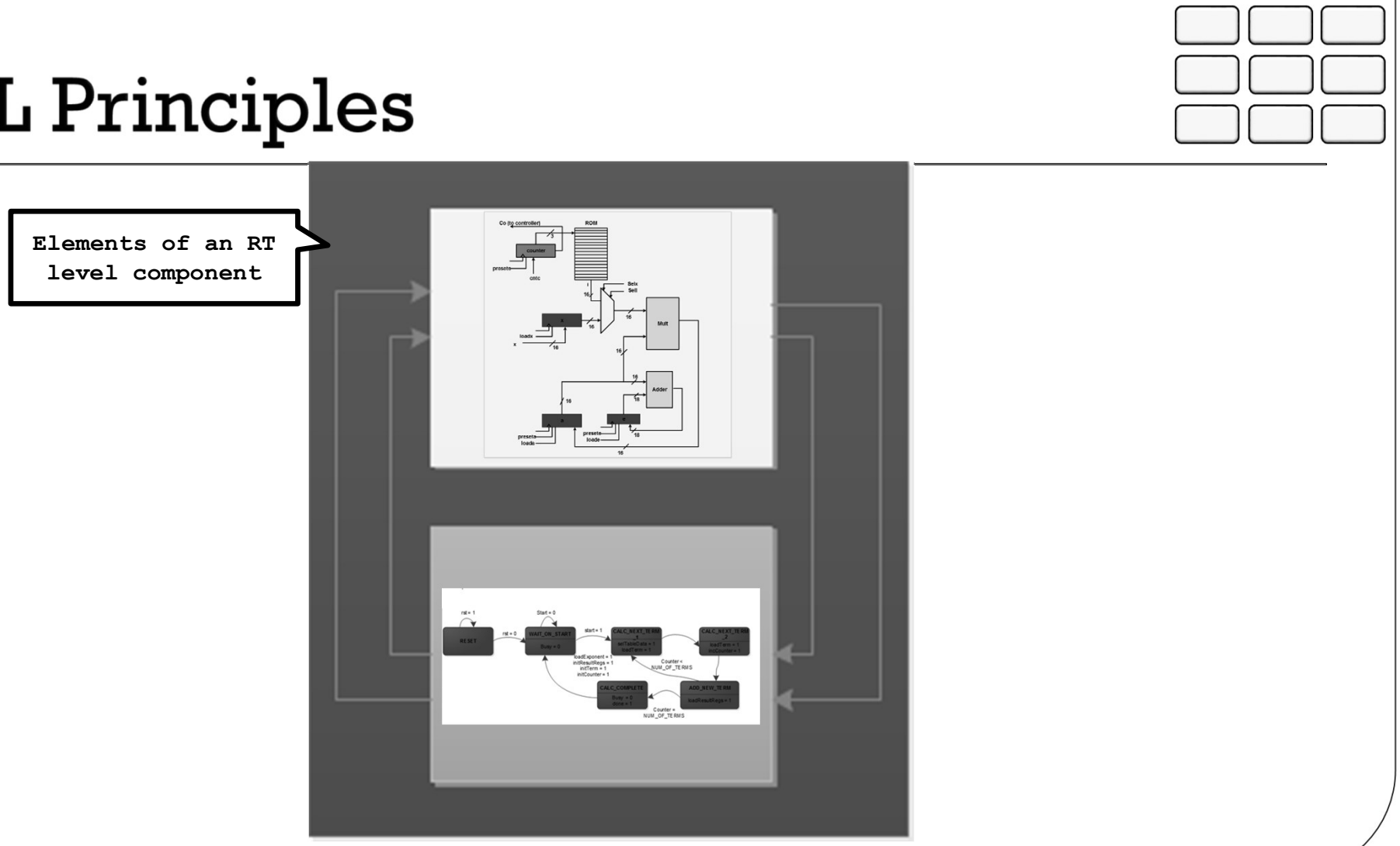
Outline



Introduction

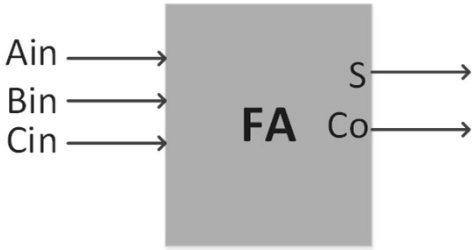
+ Bus Model and Operations
+ Basic Elements of RTL
+ RTL Design Examples

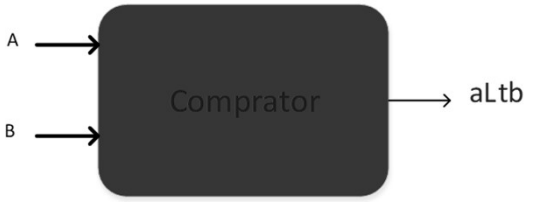
RTL Principles



Elements of Datapath

- Combinational Components
  - Adders
  - Comparators
  - Multiplexers
  - ALUs



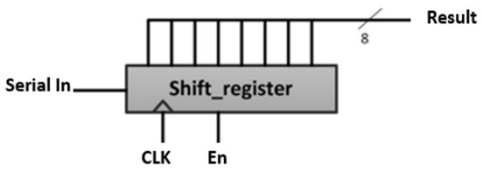


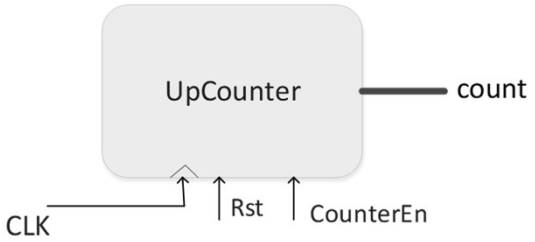
- Logical Operations
  - Vector based
  - Scalar operations
  - Mixed

Elements of Datapath

- RTL internal buses
  - Unconstrained
  - Represent multi-value logic system

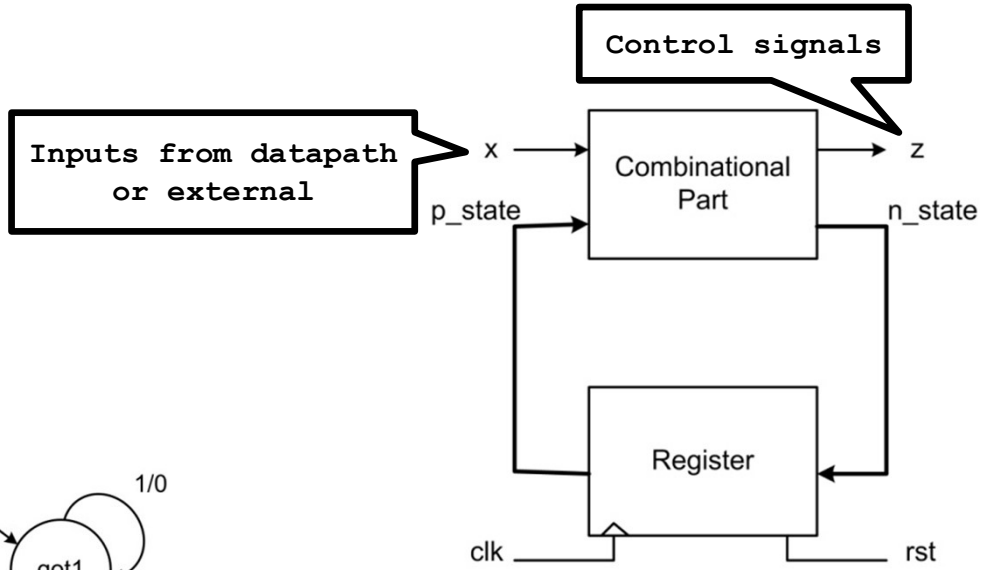
- Sequential component
  - Registers
  - Registers with some functionality
  - Register files

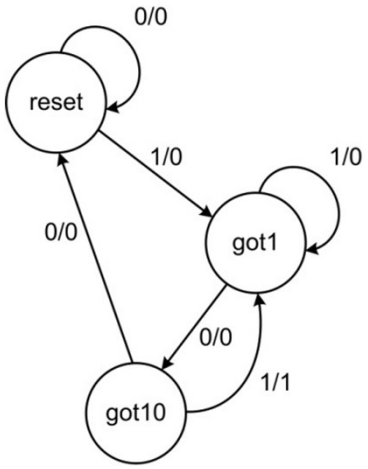




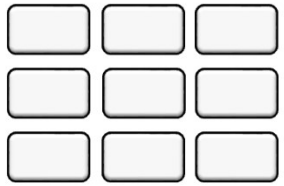
Elements of Control Unit

- One or more state machines
  - Huffman style



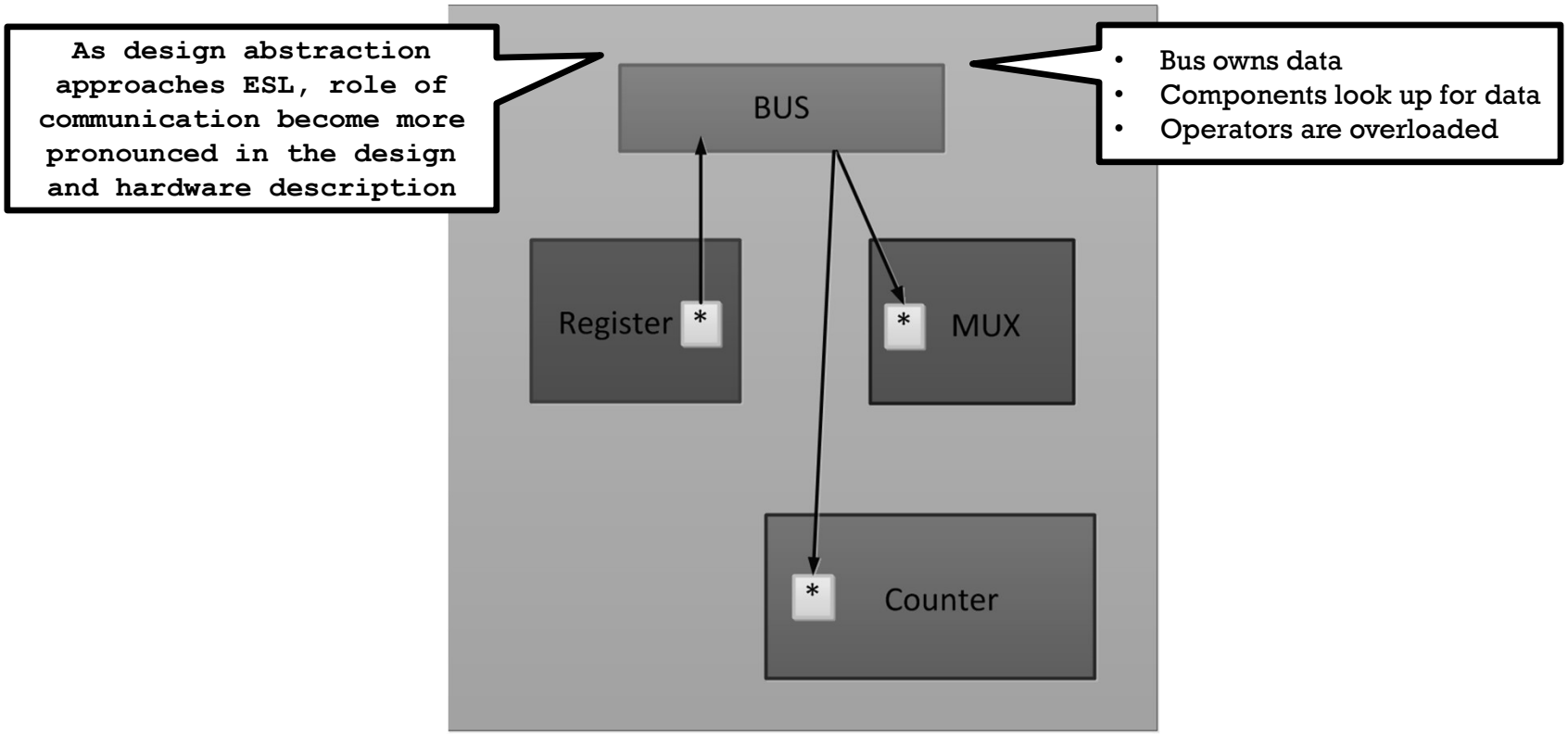


Outline

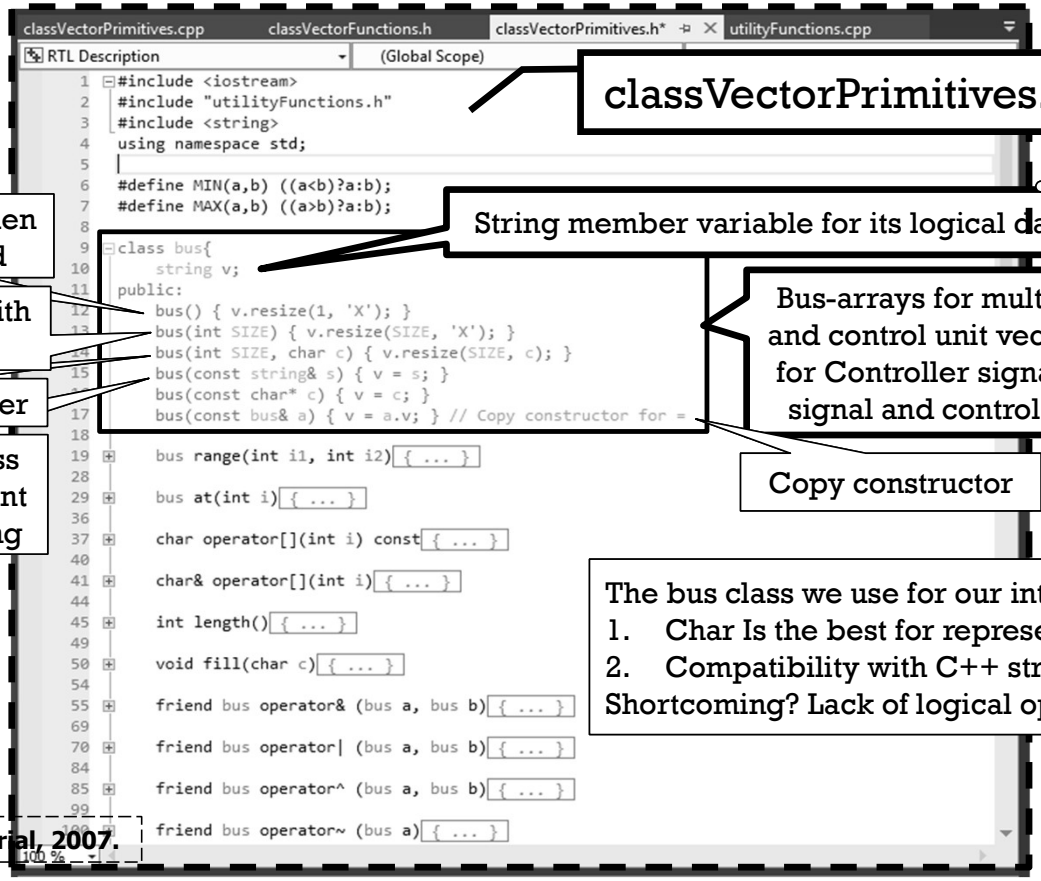


- Introduction
- – Bus Model and Operations
  - Bus Communications
  - Array Attributes
  - Logical Operations
  - Adding Operations
  - Relational Operations
  - IO Operations
- + Basic Elements of RTL
- + RTL Design Examples

Bus Communications



Bus Communications



classVectorPrimitives.h

Ref. [1] – Introduction to strings - P. 15-16

String member variable for its logical data

Default Constructor when no arguments passed

Multi-bit constructor with given bus size

Initial value is given as parameter

Using an object of bus class in an operation with constant character pointer and string

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.

Copy constructor

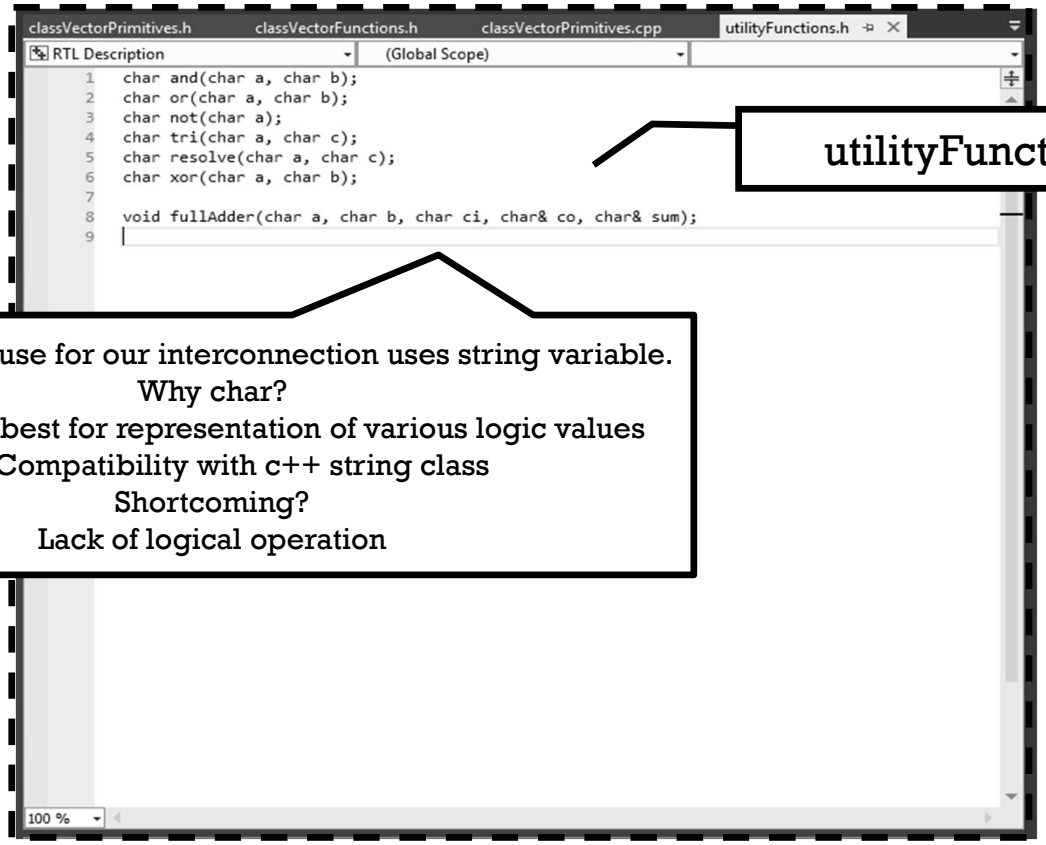
The bus class we use for our interconnection uses string variable

1. Char Is the best for representation of various logic values
2. Compatibility with C++ string class

Shortcoming? Lack of logical operation

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

Utility Functions



utilityFunctions.h

The bus class we use for our interconnection uses string variable.

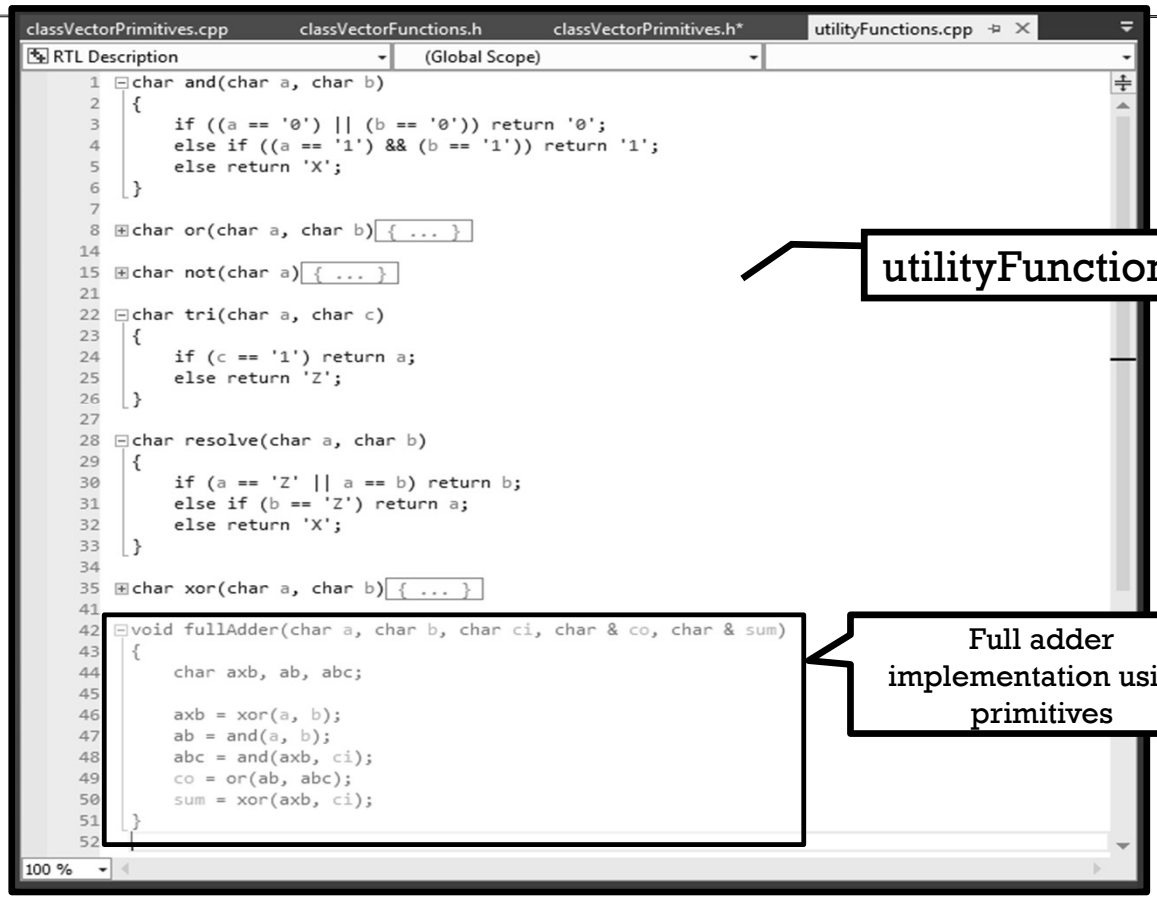
Why char?

1. Char Is the best for representation of various logic values
2. Compatibility with c++ string class

Shortcoming?

Lack of logical operation

Utility Functions

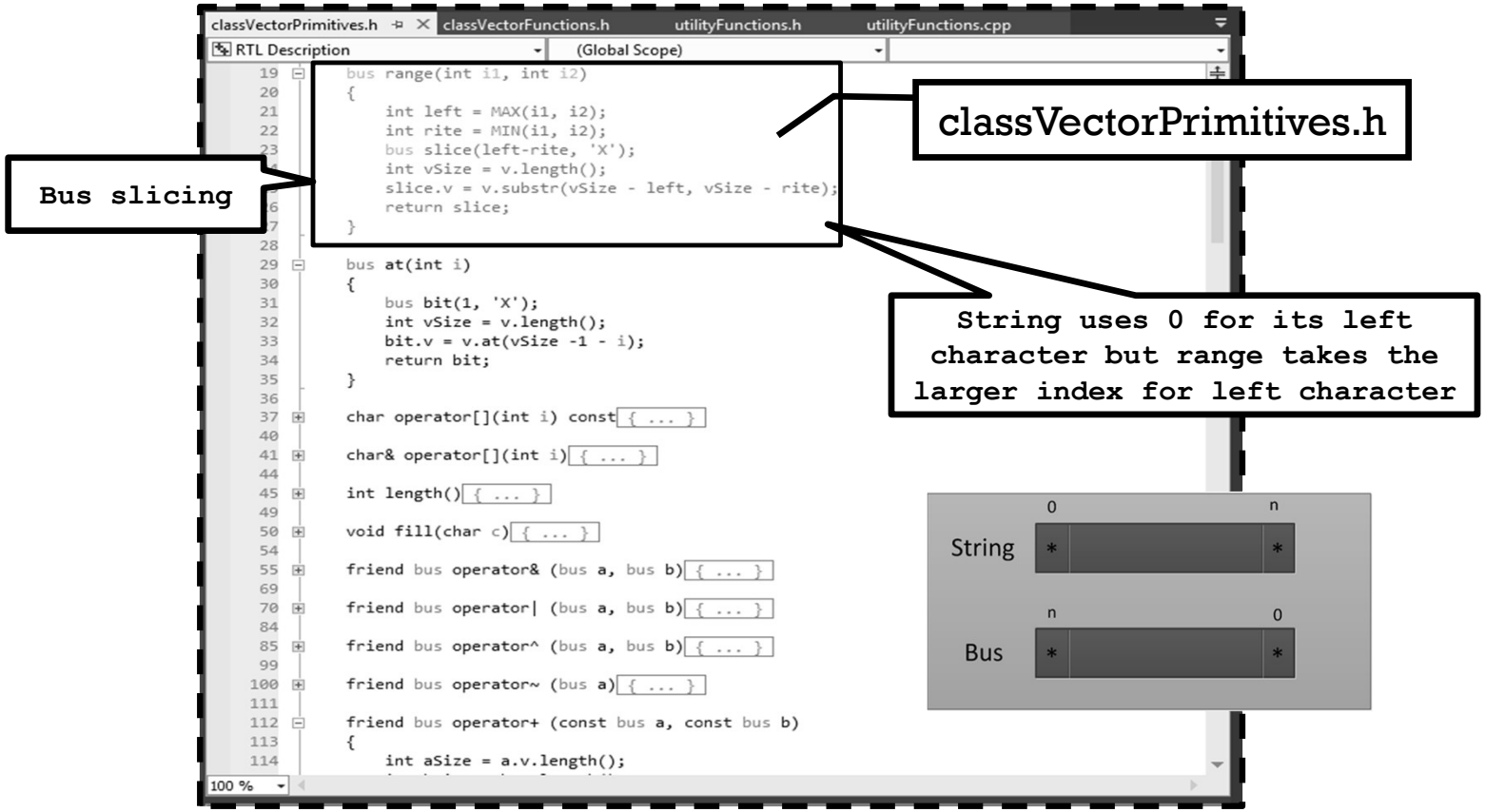



```
1  char and(char a, char b)
2  {
3      if ((a == '0') || (b == '0')) return '0';
4      else if ((a == '1') && (b == '1')) return '1';
5      else return 'X';
6  }
7
8  char or(char a, char b){ ... }
14
15 char not(char a){ ... }
21
22 char tri(char a, char c)
23 {
24     if (c == '1') return a;
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
```

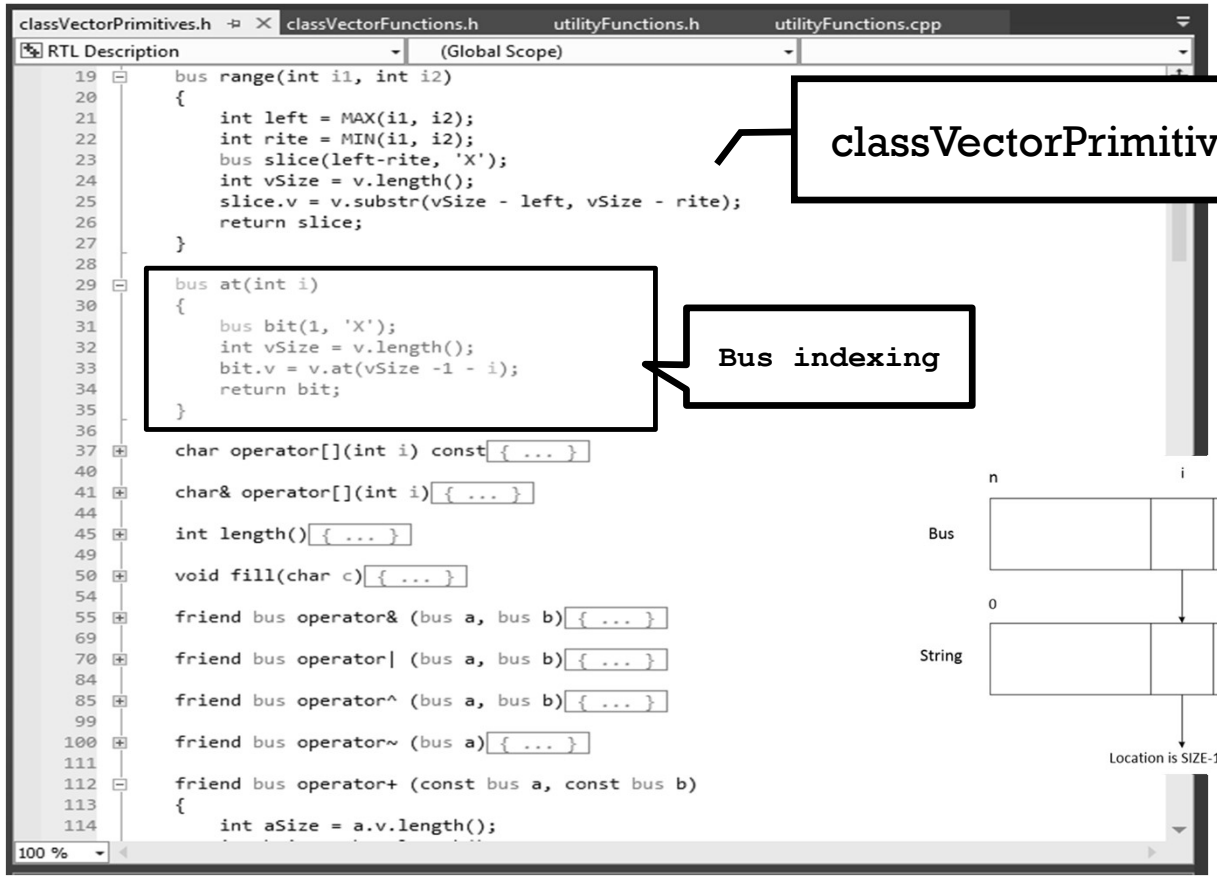
utilityFunctions.cpp

Full adder implementation using primitives

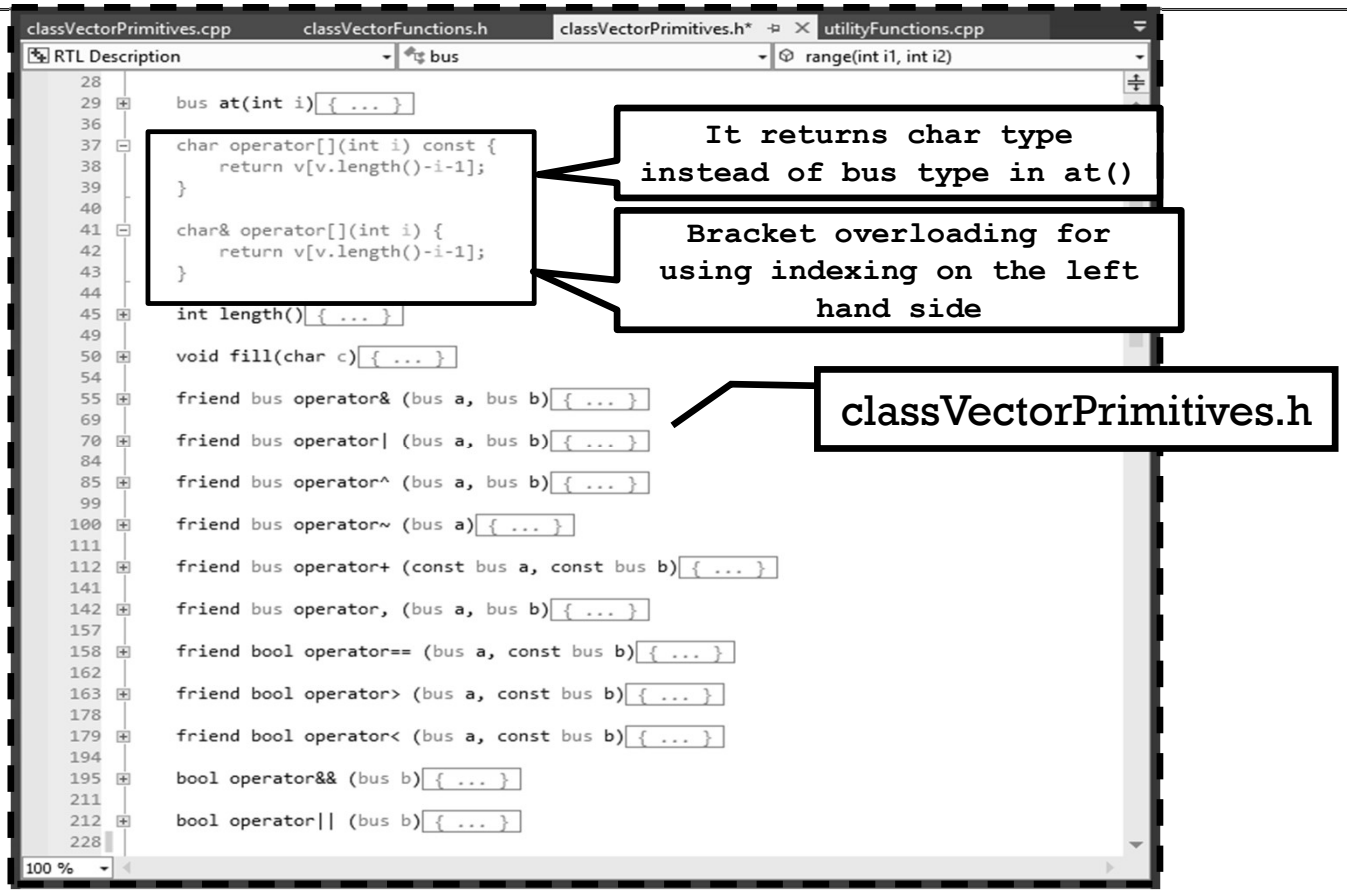
Array Attributes



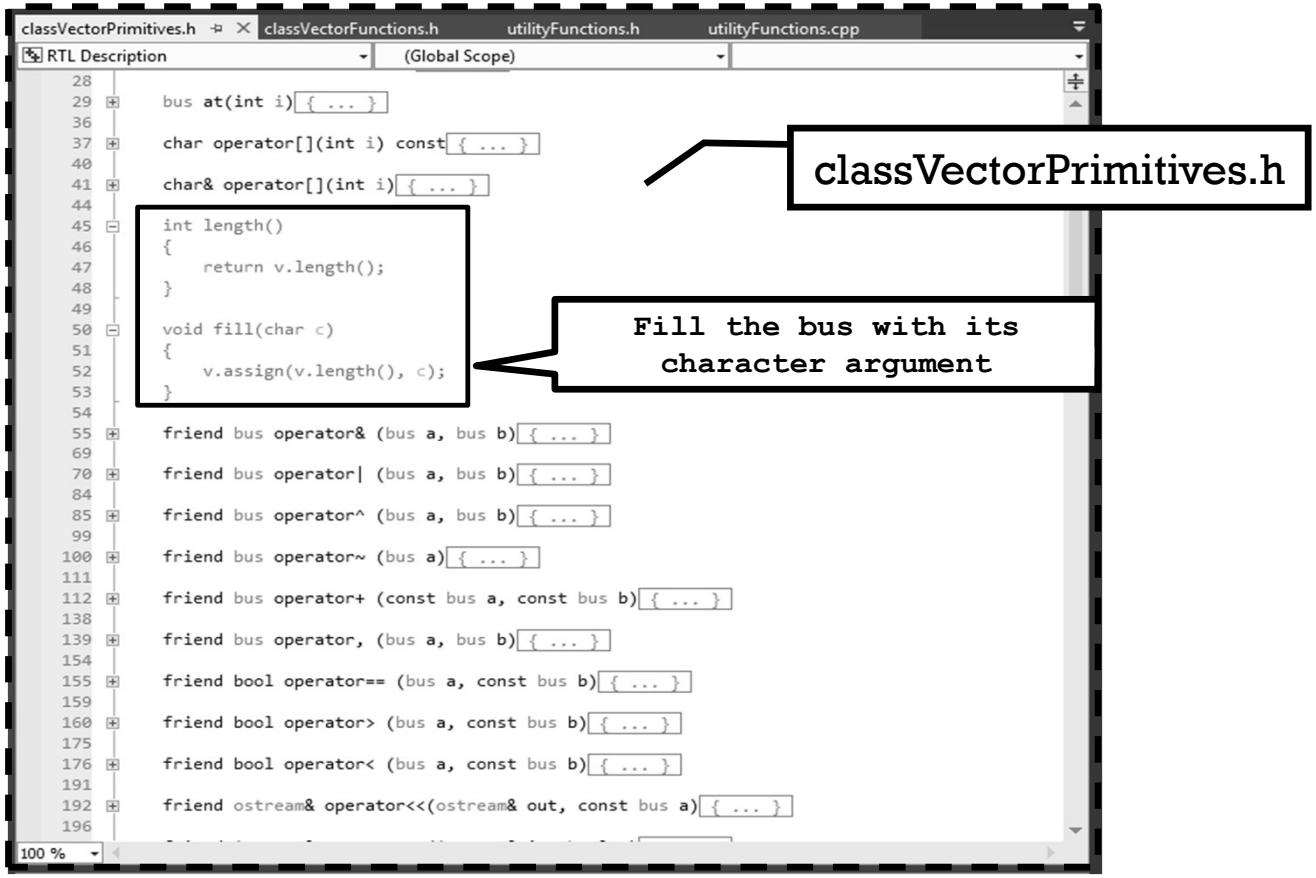
Array Attributes



Array Attributes

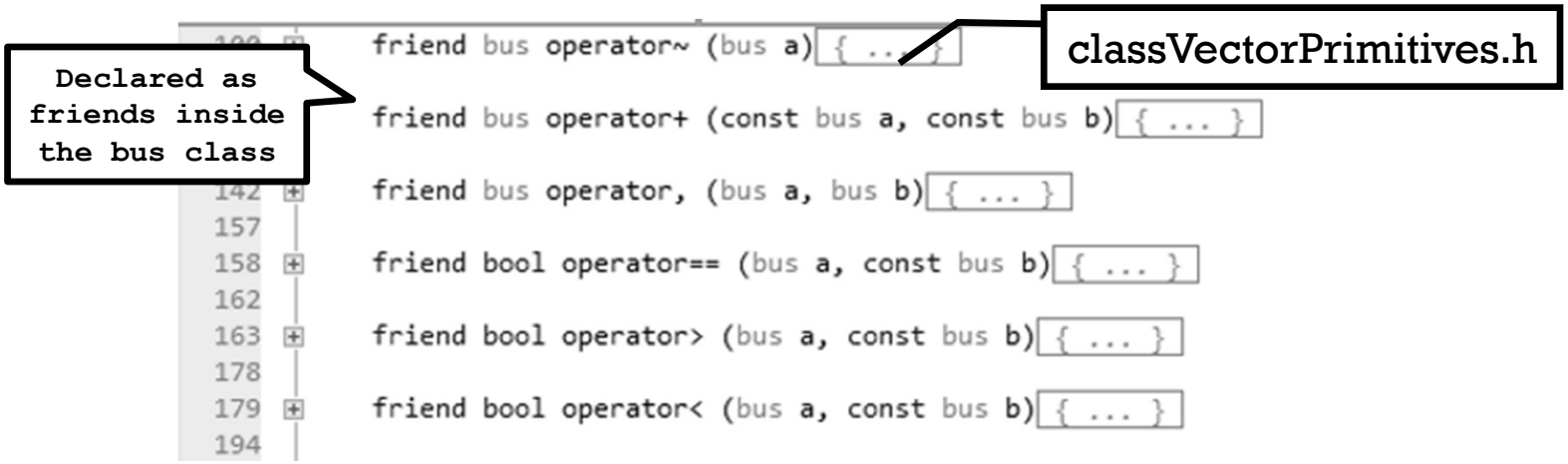


Array Attributes

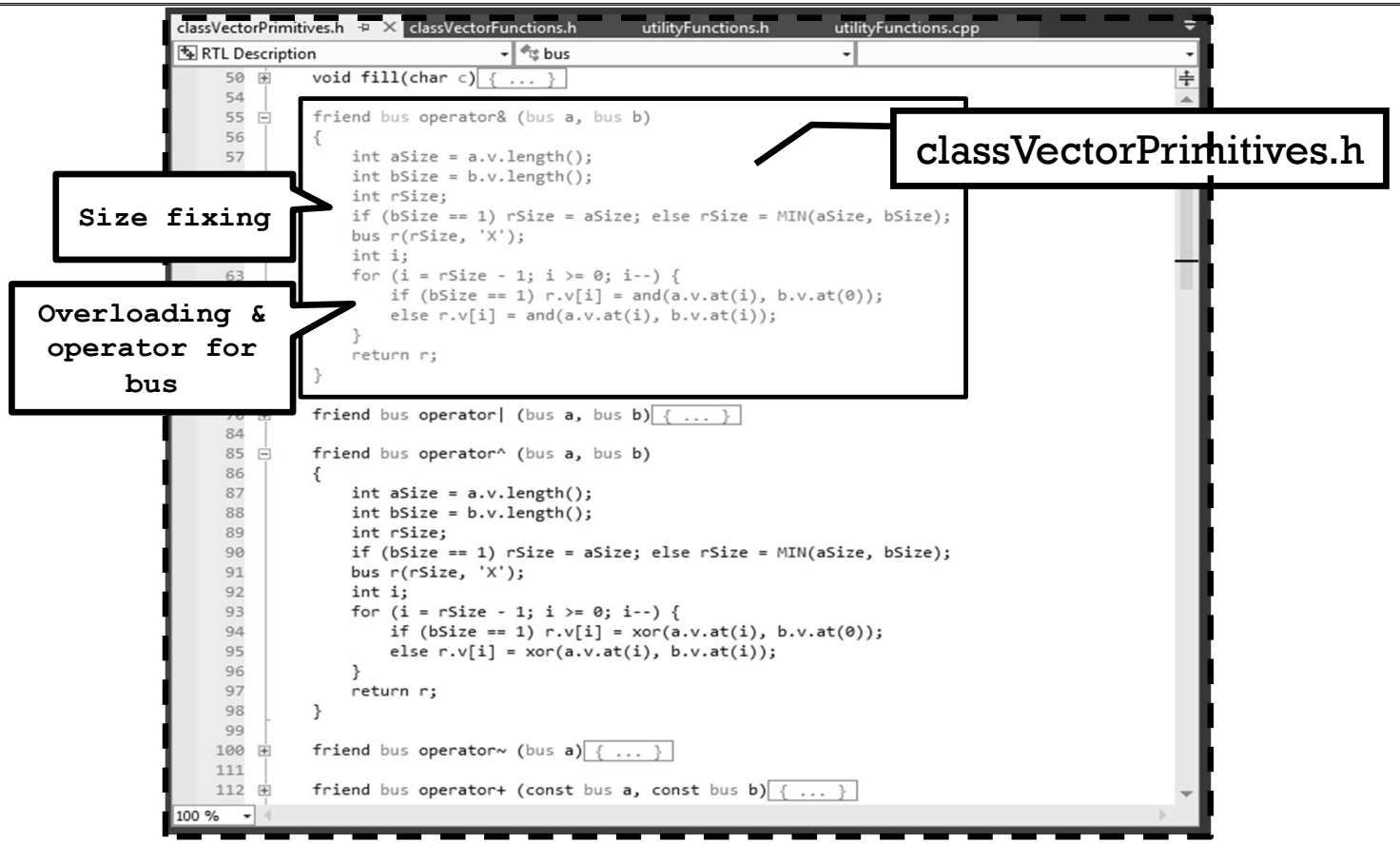



```
classVectorPrimitives.h | classVectorFunctions.h | utilityFunctions.h | utilityFunctions.cpp
RTL Description | (Global Scope)
28
29   bus at(int i){ ... }
36
37   char operator[](int i) const{ ... }
40
41   char& operator[](int i){ ... }
44
45   int length()
46   {
47       return v.length();
48   }
49
50   void fill(char c)
51   {
52       v.assign(v.length(), c);
53   }
54
55   friend bus operator& (bus a, bus b){ ... }
69
70   friend bus operator| (bus a, bus b){ ... }
84
85   friend bus operator^ (bus a, bus b){ ... }
99
100  friend bus operator~ (bus a){ ... }
111
112  friend bus operator+ (const bus a, const bus b){ ... }
138
139  friend bus operator, (bus a, bus b){ ... }
154
155  friend bool operator== (bus a, const bus b){ ... }
159
160  friend bool operator> (bus a, const bus b){ ... }
175
176  friend bool operator< (bus a, const bus b){ ... }
191
192  friend ostream& operator<<(ostream& out, const bus a){ ... }
196
100 %
```

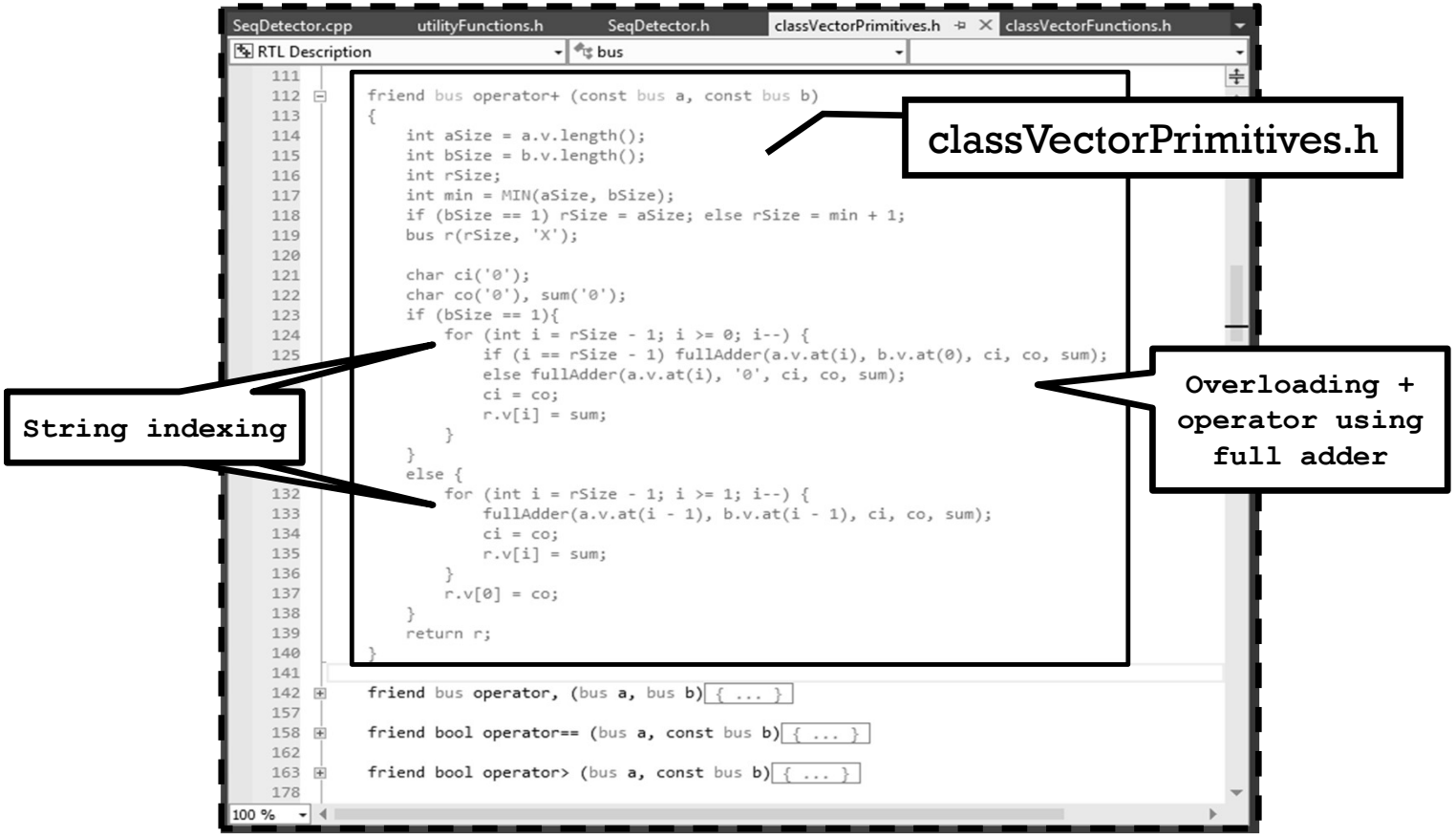
Logical Operations



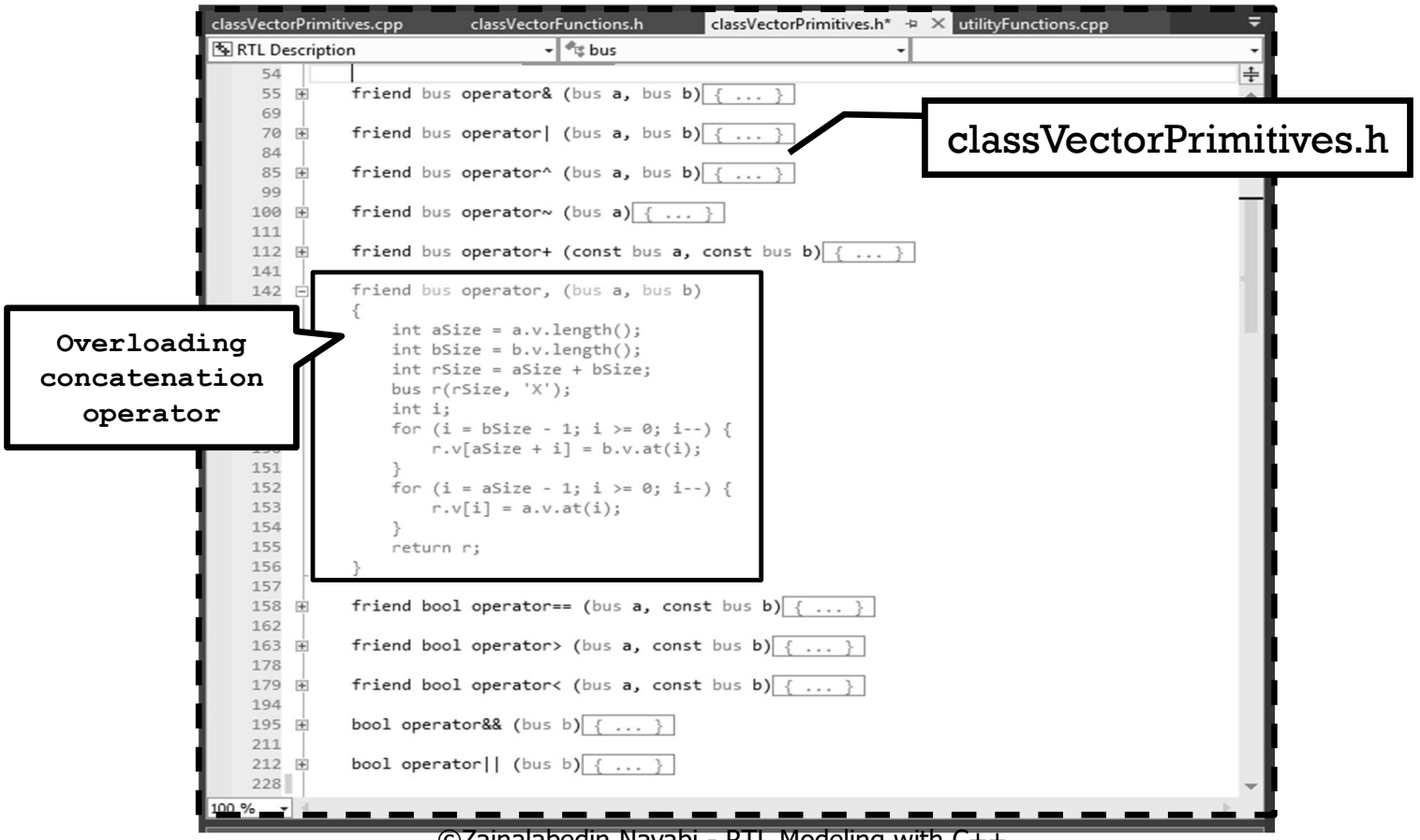
Logical Operations



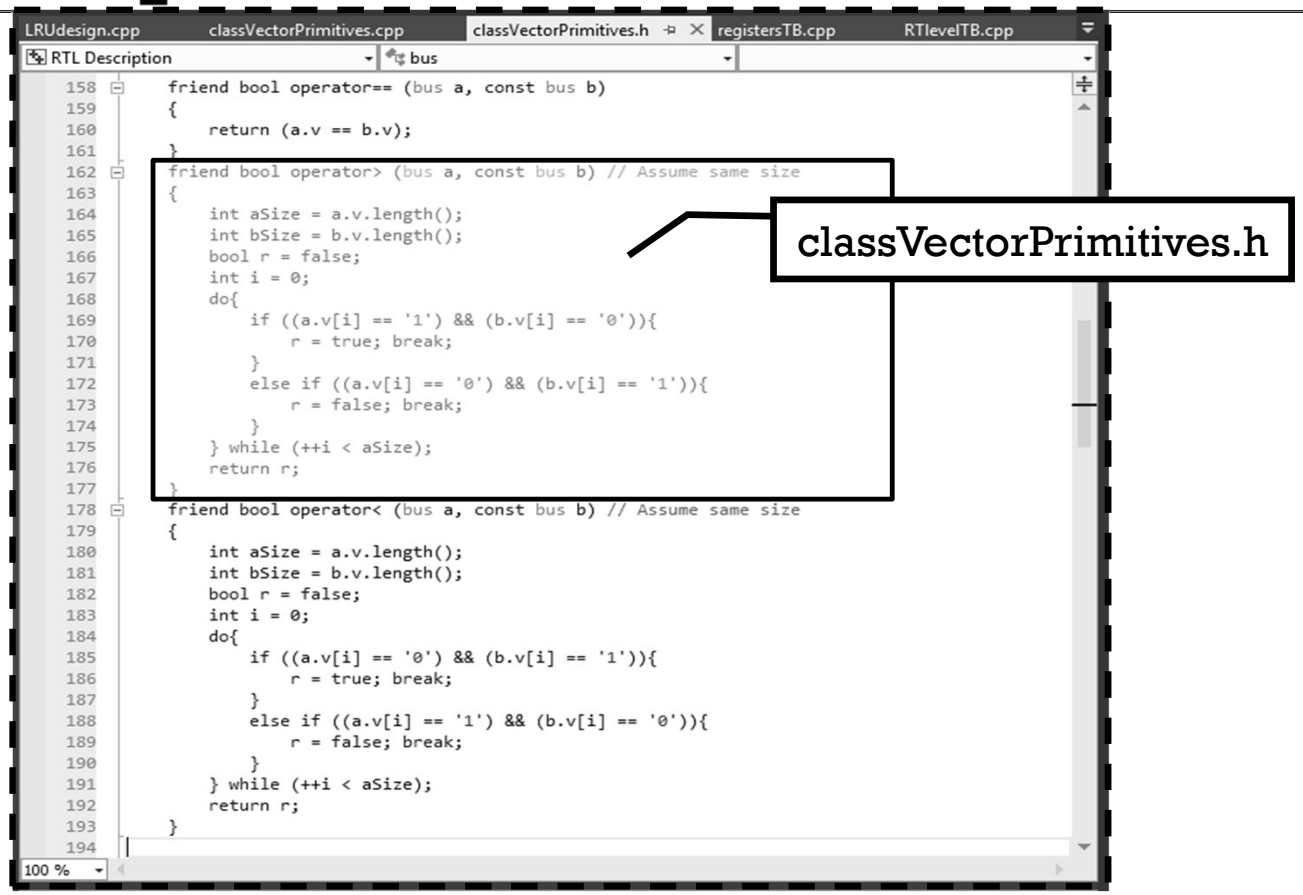
Adding Operations



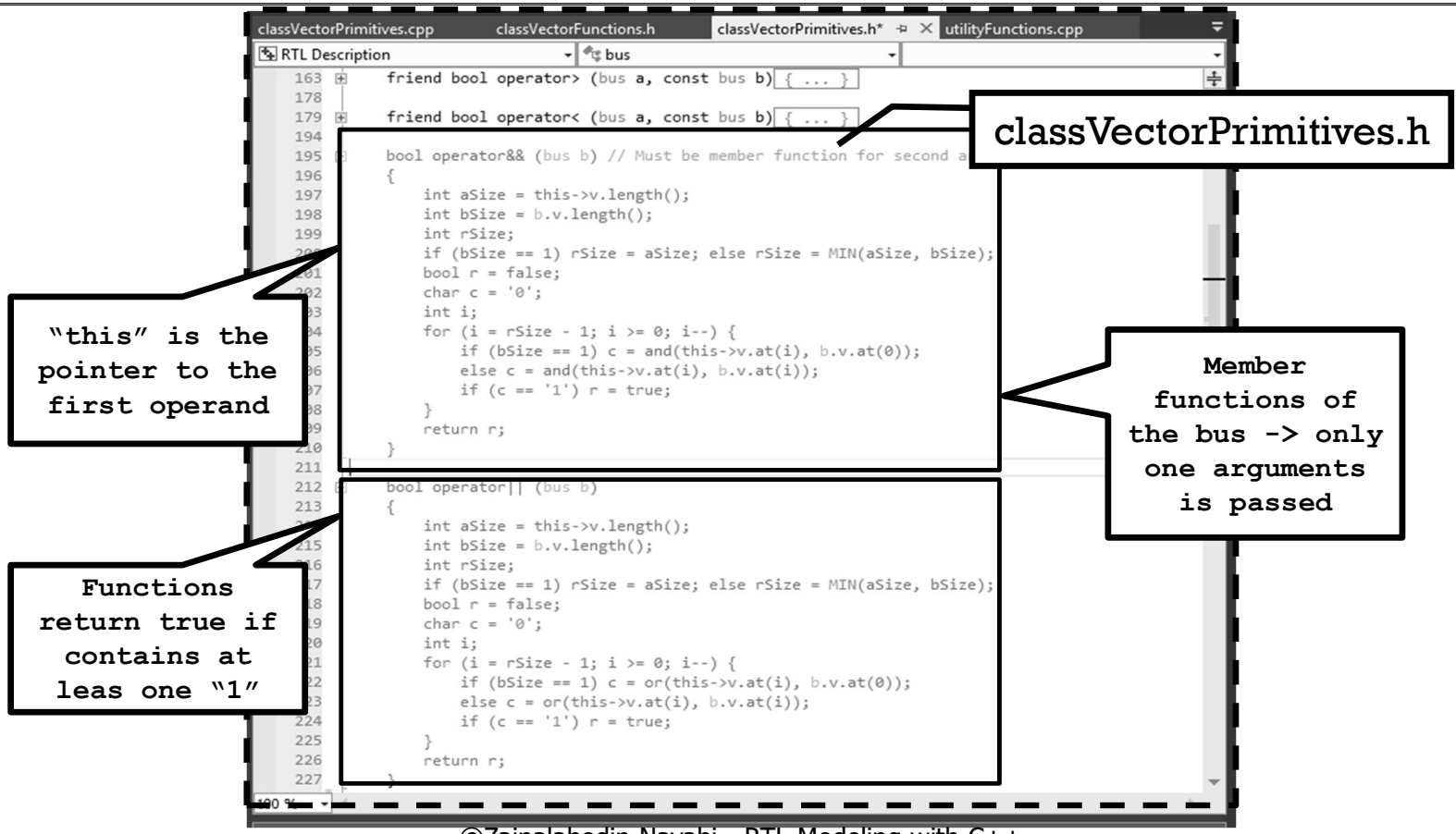
Logical Operations



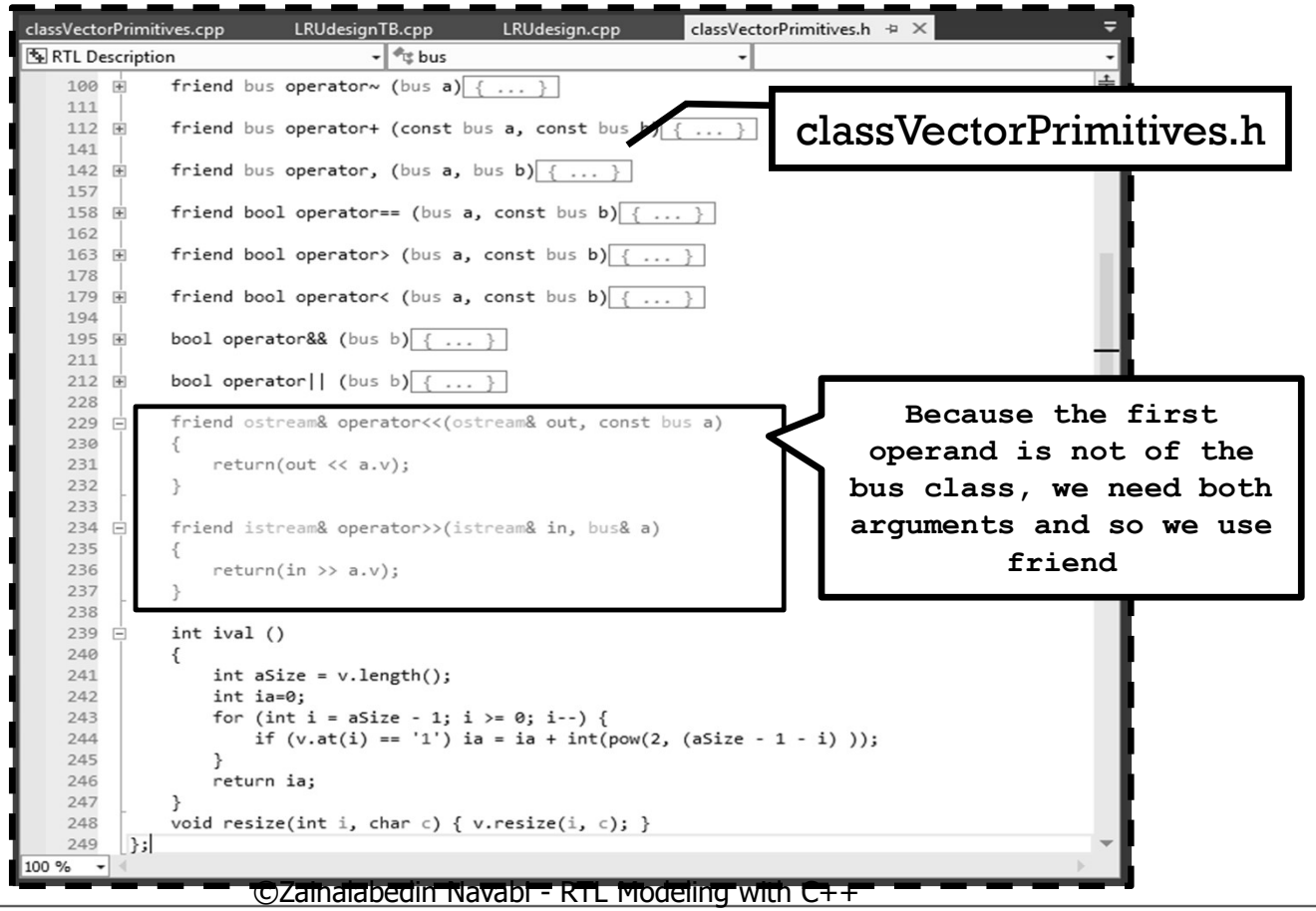
Rational Operations



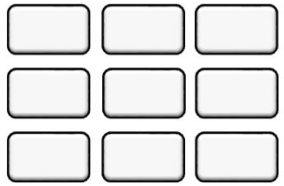
Logical Operations



IO Operations

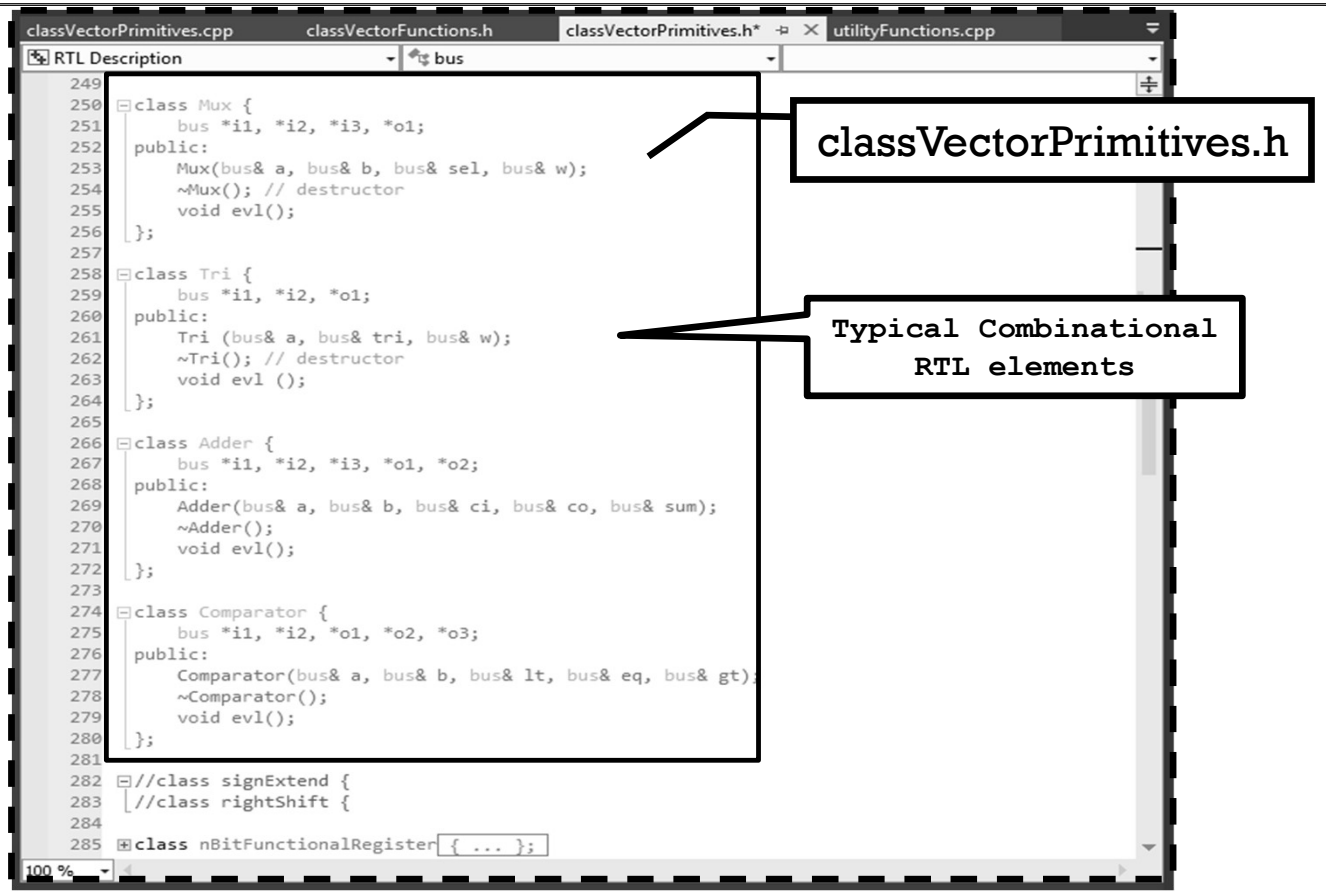


Outline

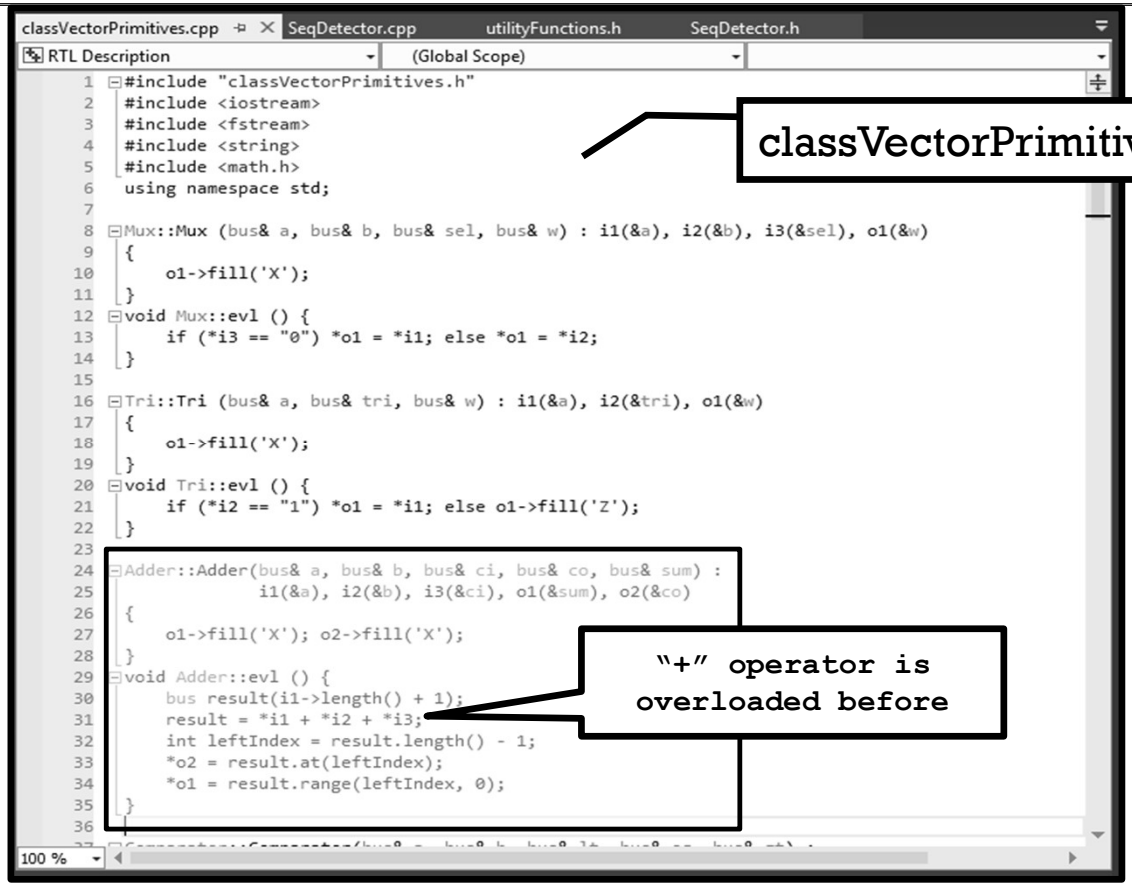


- Introduction
- + Bus Model and Operations
- – Basic Elements of RTL
  - Combinational elements
    - Mux
    - Tri
    - Adder
    - Comparator
  - Sequential elements
    - Registers
    - Up-Counters
- + RTL Design Examples

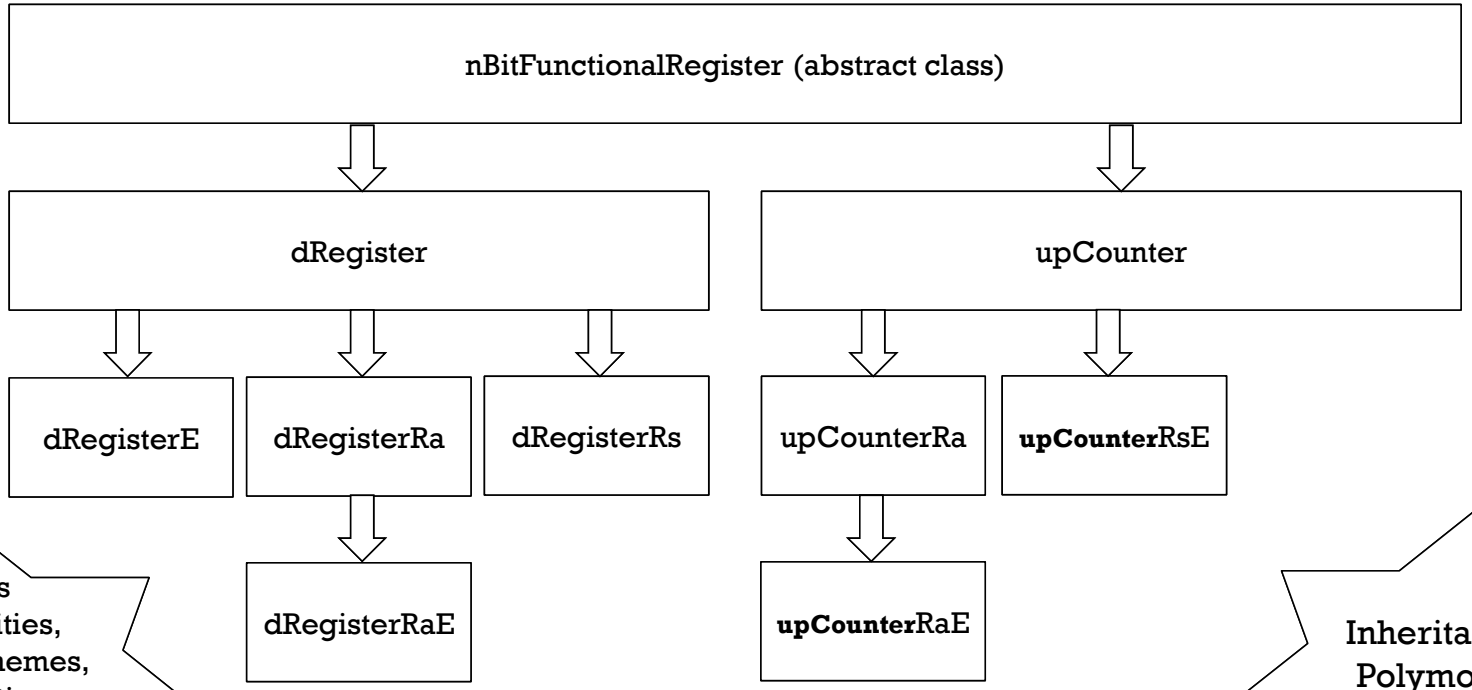
Combinational Elements



Combinational Elements



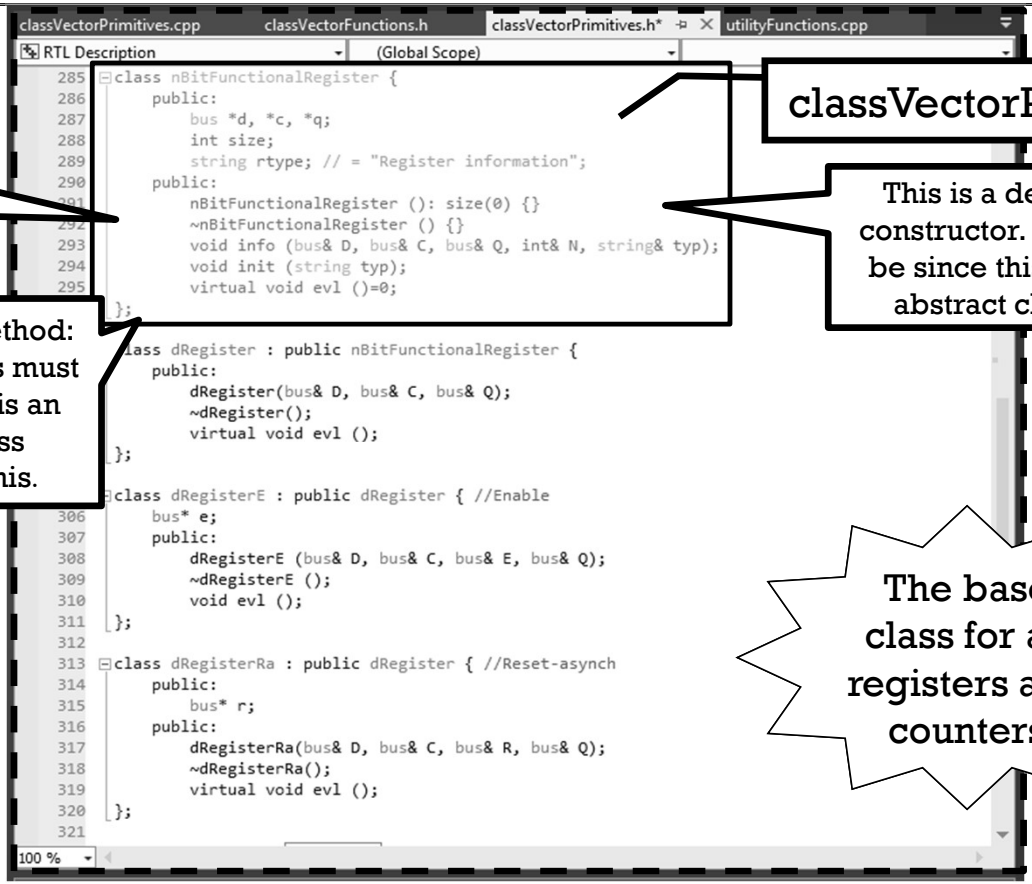
Registers and Counters



Various functionalities, clocking schemes, and resetting mechanisms

Inheritance and Polymorphism

Functional Registers



classVectorPrimitives.h

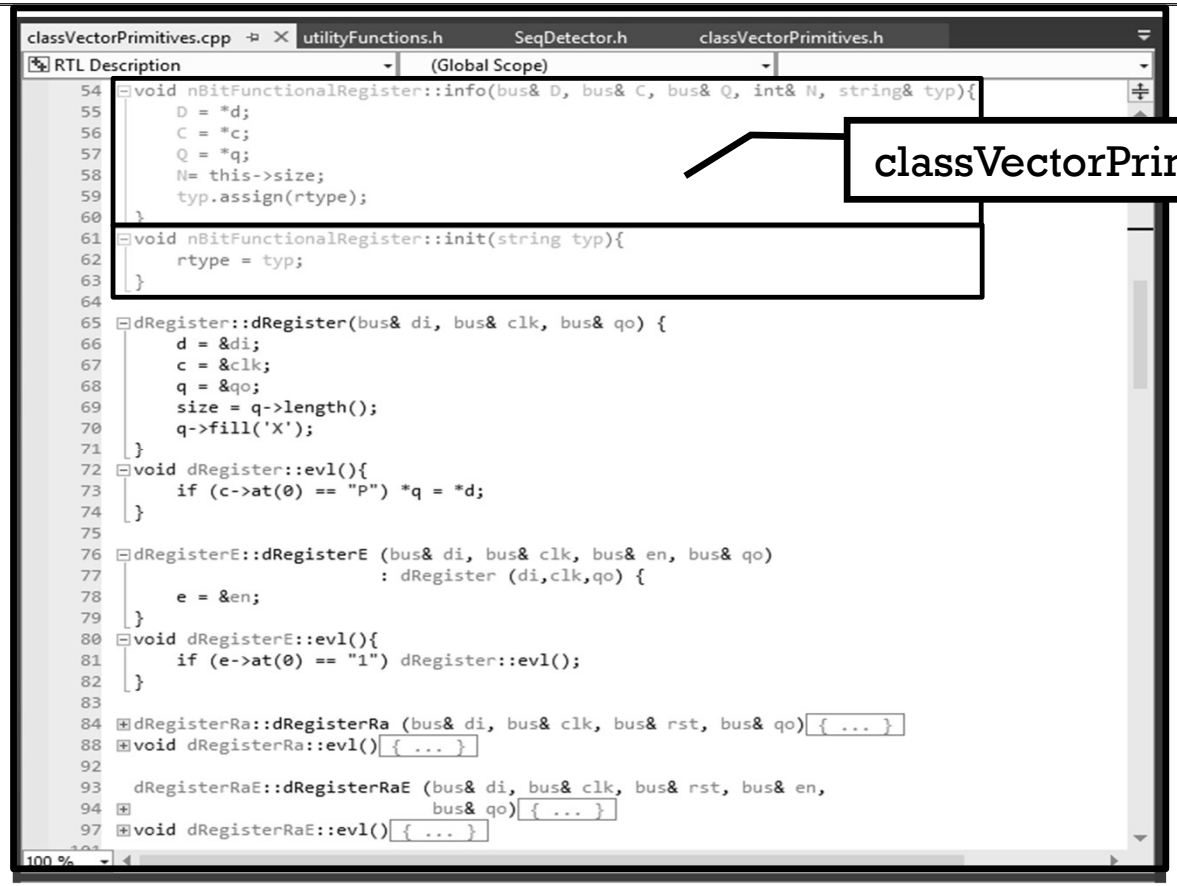
Initializing member variables.

This is a default constructor. It must be since this is an abstract class.

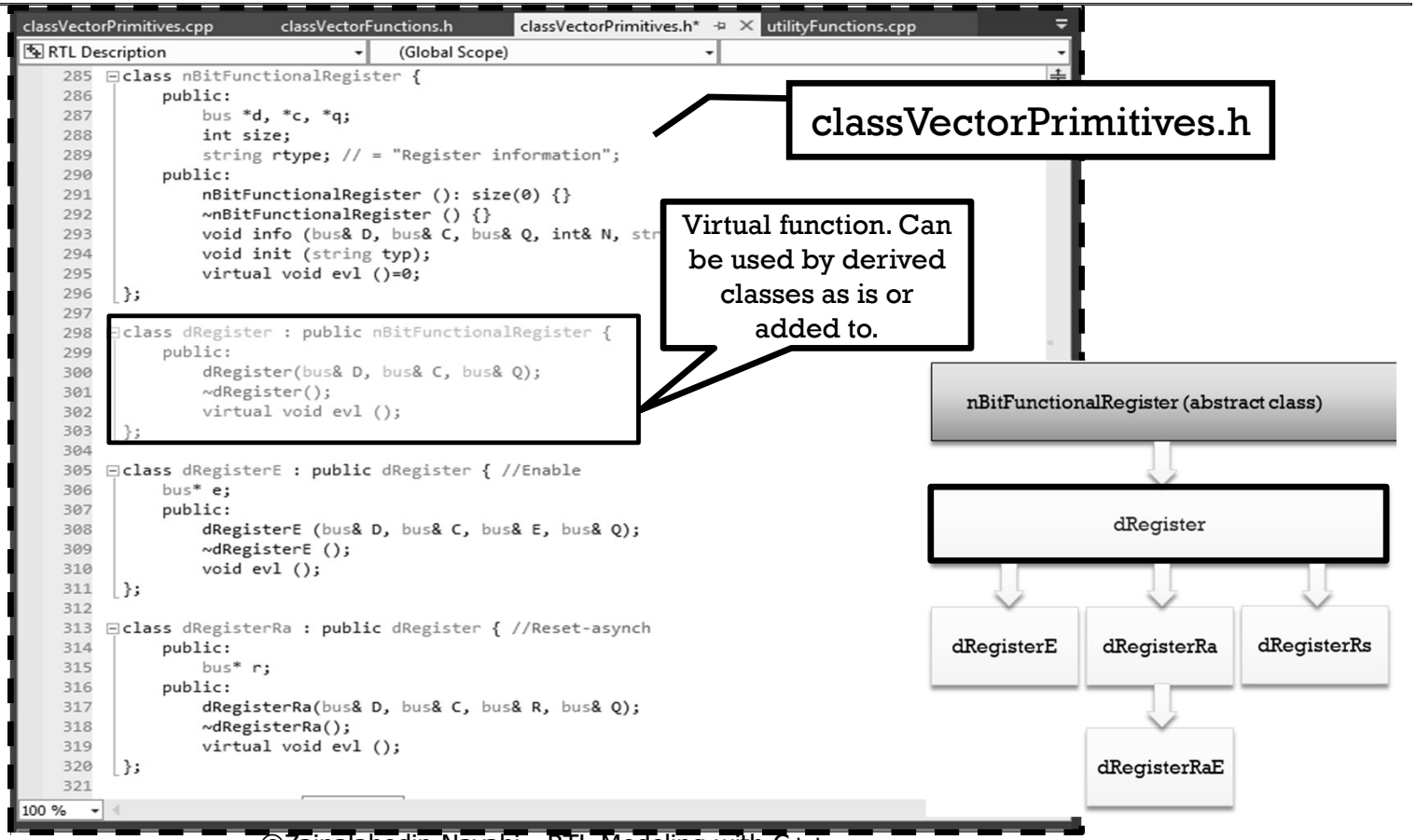
Pure virtual method: Derived classes must define it. This is an abstract class because of this.

The base class for all registers and counters

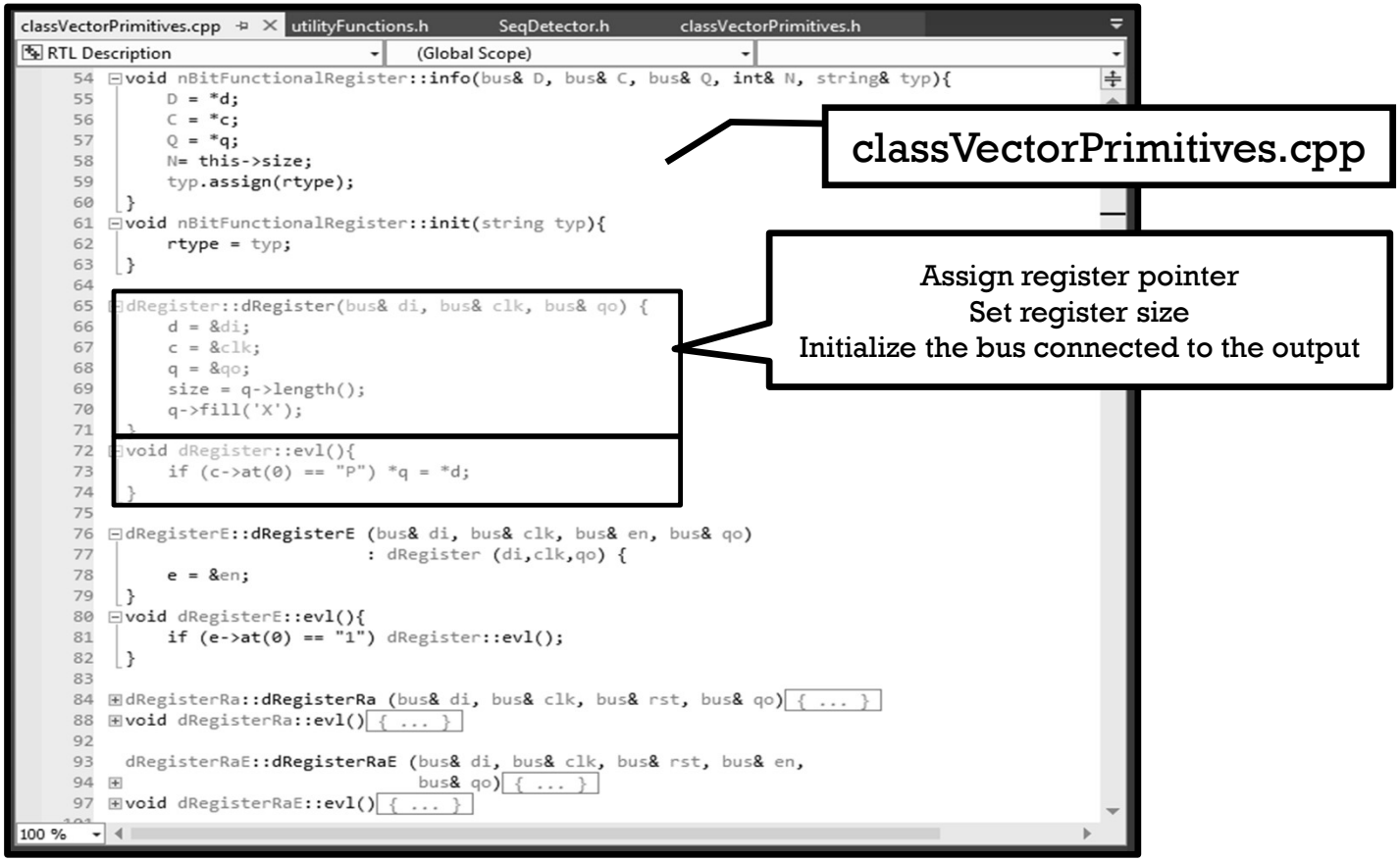
Functional Registers



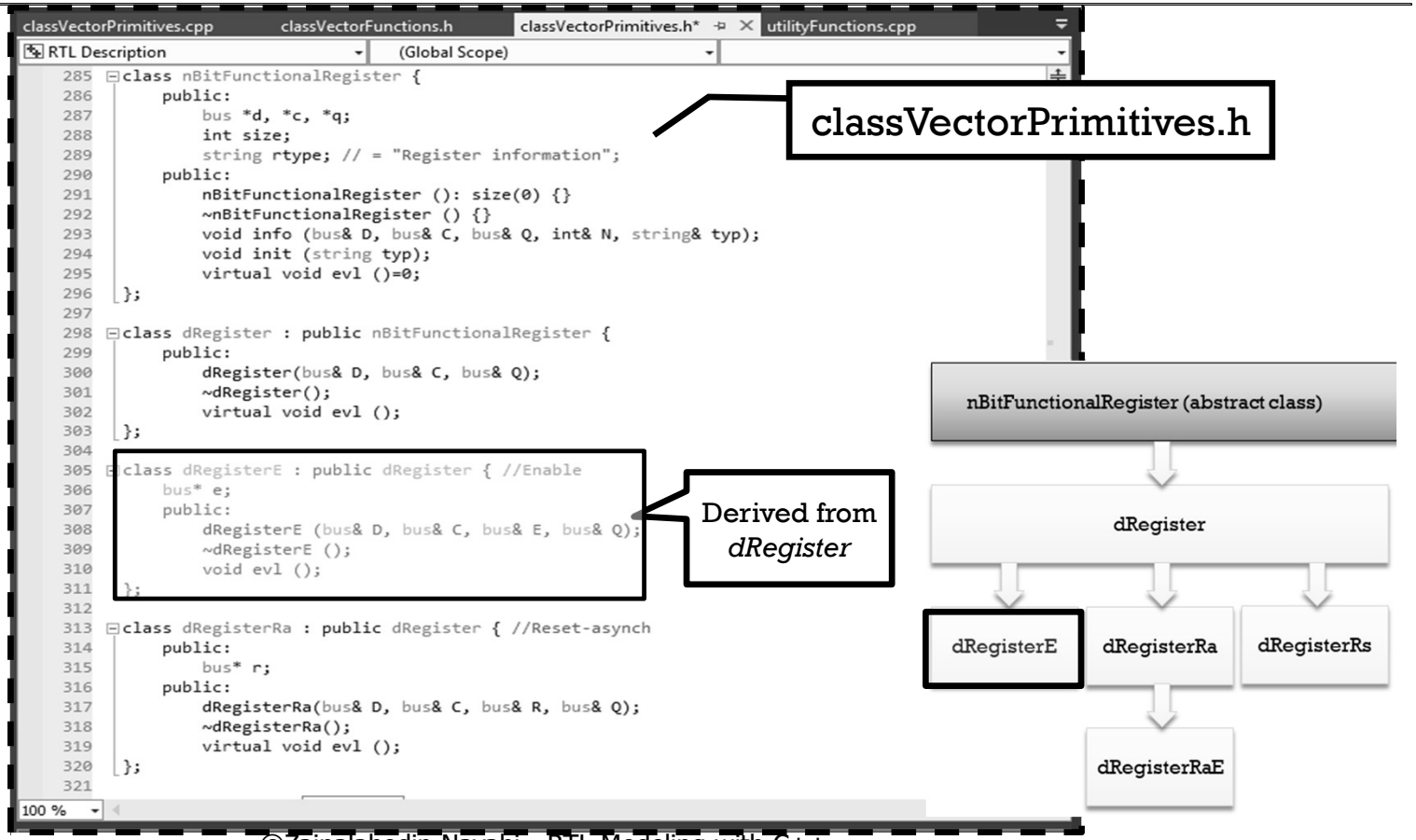
DRegister


classVectorPrimitives.cpp
classVectorFunctions.h
classVectorPrimitives.h*
utilityFunctions.cpp
RTL Description
(Global Scope)
285 class nBitFunctionalRegister {
286 public:
287 bus *d, *c, *q;
288 int size;
289 string rtype; // = "Register information";
290 public:
291 nBitFunctionalRegister (): size(0) {}
292 ~nBitFunctionalRegister () {}
293 void info (bus& D, bus& C, bus& Q, int& N, str
294 void init (string typ);
295 virtual void evl ()=0;
296 };
297
298 class dRegister : public nBitFunctionalRegister {
299 public:
300 dRegister(bus& D, bus& C, bus& Q);
301 ~dRegister();
302 virtual void evl ();
303 };
304
305 class dRegisterE : public dRegister { //Enable
306 bus* e;
307 public:
308 dRegisterE (bus& D, bus& C, bus& E, bus& Q);
309 ~dRegisterE ();
310 void evl ();
311 };
312
313 class dRegisterRa : public dRegister { //Reset-asynch
314 public:
315 bus* r;
316 public:
317 dRegisterRa(bus& D, bus& C, bus& R, bus& Q);
318 ~dRegisterRa();
319 virtual void evl ();
320 };
321
100 %
classVectorPrimitives.h
Virtual function. Can be used by derived classes as is or added to.
nBitFunctionalRegister (abstract class)
dRegister
dRegisterE
dRegisterRa
dRegisterRs
dRegisterRaE

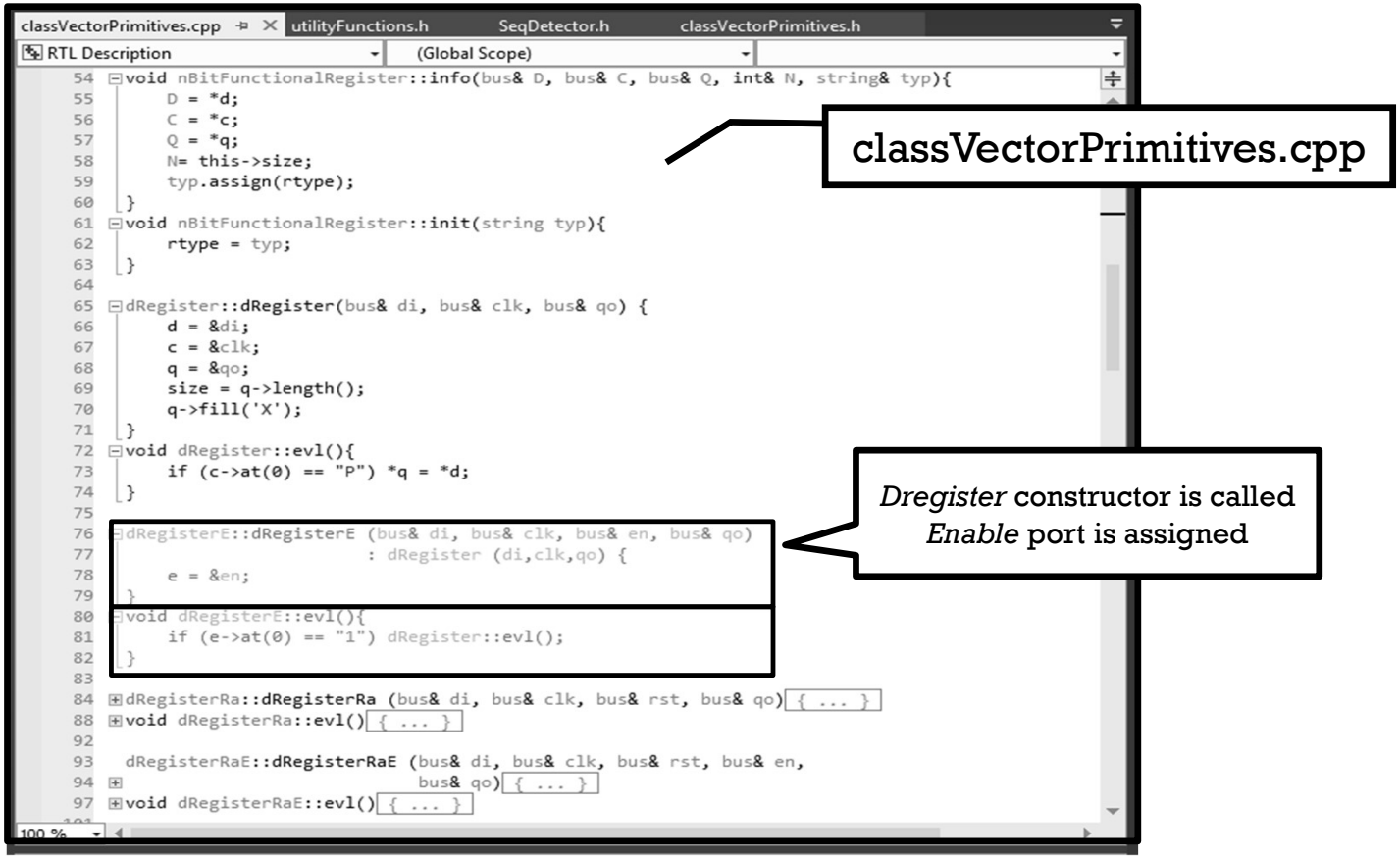
DRegister


classVectorPrimitives.cpp
Assign register pointer
Set register size
Initialize the bus connected to the output

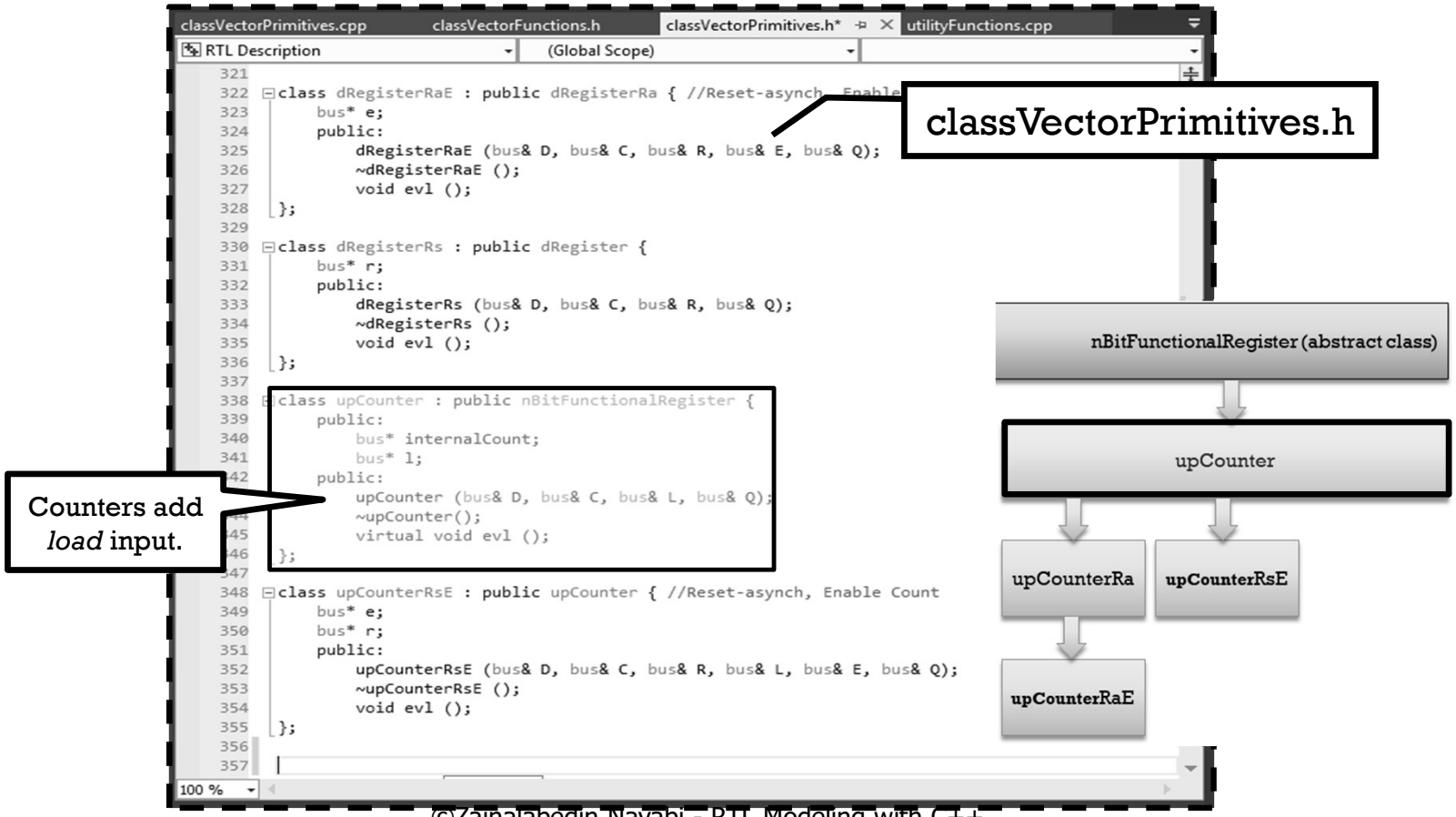
DRegisterE


classVectorPrimitives.cpp
classVectorFunctions.h
classVectorPrimitives.h*
utilityFunctions.cpp
RTL Description
(Global Scope)
285 class nBitFunctionalRegister {
286 public:
287 bus *d, *c, *q;
288 int size;
289 string rtype; // = "Register information";
290 public:
291 nBitFunctionalRegister (): size(0) {}
292 ~nBitFunctionalRegister () {}
293 void info (bus& D, bus& C, bus& Q, int& N, string& typ);
294 void init (string typ);
295 virtual void evl ()=0;
296 };
297
298 class dRegister : public nBitFunctionalRegister {
299 public:
300 dRegister(bus& D, bus& C, bus& Q);
301 ~dRegister();
302 virtual void evl ();
303 };
304
305 class dRegisterE : public dRegister { //Enable
306 bus* e;
307 public:
308 dRegisterE (bus& D, bus& C, bus& E, bus& Q);
309 ~dRegisterE ();
310 void evl ();
311 };
312
313 class dRegisterRa : public dRegister { //Reset-asynch
314 public:
315 bus* r;
316 public:
317 dRegisterRa(bus& D, bus& C, bus& R, bus& Q);
318 ~dRegisterRa();
319 virtual void evl ();
320 };
321
100 %
classVectorPrimitives.h
Derived from dRegister
nBitFunctionalRegister (abstract class)
dRegister
dRegisterE
dRegisterRa
dRegisterRs
dRegisterRaE

DRegisterE


classVectorPrimitives.cpp
utilityFunctions.h
SeqDetector.h
classVectorPrimitives.h
RTL Description
(Global Scope)
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) { ... }
88 void dRegisterRa::evl() { ... }
92
93 dRegisterRaE::dRegisterRaE (bus& di, bus& clk, bus& rst, bus& en,
94 bus& qo) { ... }
97 void dRegisterRaE::evl() { ... }
100 %
Dregister constructor is called
Enable port is assigned

UpCounter


classVectorPrimitives.cpp
classVectorFunctions.h
classVectorPrimitives.h*
utilityFunctions.cpp
RTL Description
(Global Scope)
321
322 class dRegisterRaE : public dRegisterRa { //Reset-asynch, Enable
323 bus* e;
324 public:
325 dRegisterRaE (bus& D, bus& C, bus& R, bus& E, bus& Q);
326 ~dRegisterRaE ();
327 void evl ();
328 };
329
330 class dRegisterRs : public dRegister {
331 bus* r;
332 public:
333 dRegisterRs (bus& D, bus& C, bus& R, bus& Q);
334 ~dRegisterRs ();
335 void evl ();
336 };
337
338 class upCounter : public nBitFunctionalRegister {
339 public:
340 bus* internalCount;
341 bus* l;
342 public:
344 upCounter (bus& D, bus& C, bus& L, bus& Q);
345 ~upCounter();
346 virtual void evl ();
347 };
348 class upCounterRsE : public upCounter { //Reset-asynch, Enable Count
349 bus* e;
350 bus* r;
351 public:
352 upCounterRsE (bus& D, bus& C, bus& R, bus& L, bus& E, bus& Q);
353 ~upCounterRsE ();
354 void evl ();
355 };
356
357
100 %
classVectorPrimitives.h
Counters add load input.
nBitFunctionalRegister (abstract class)
upCounter
upCounterRa
upCounterRsE
upCounterRaE

UpCounter

classVectorPrimitives.cpp | utilityFunctions.h | SeqDetector.h | classVectorPrimitives.h

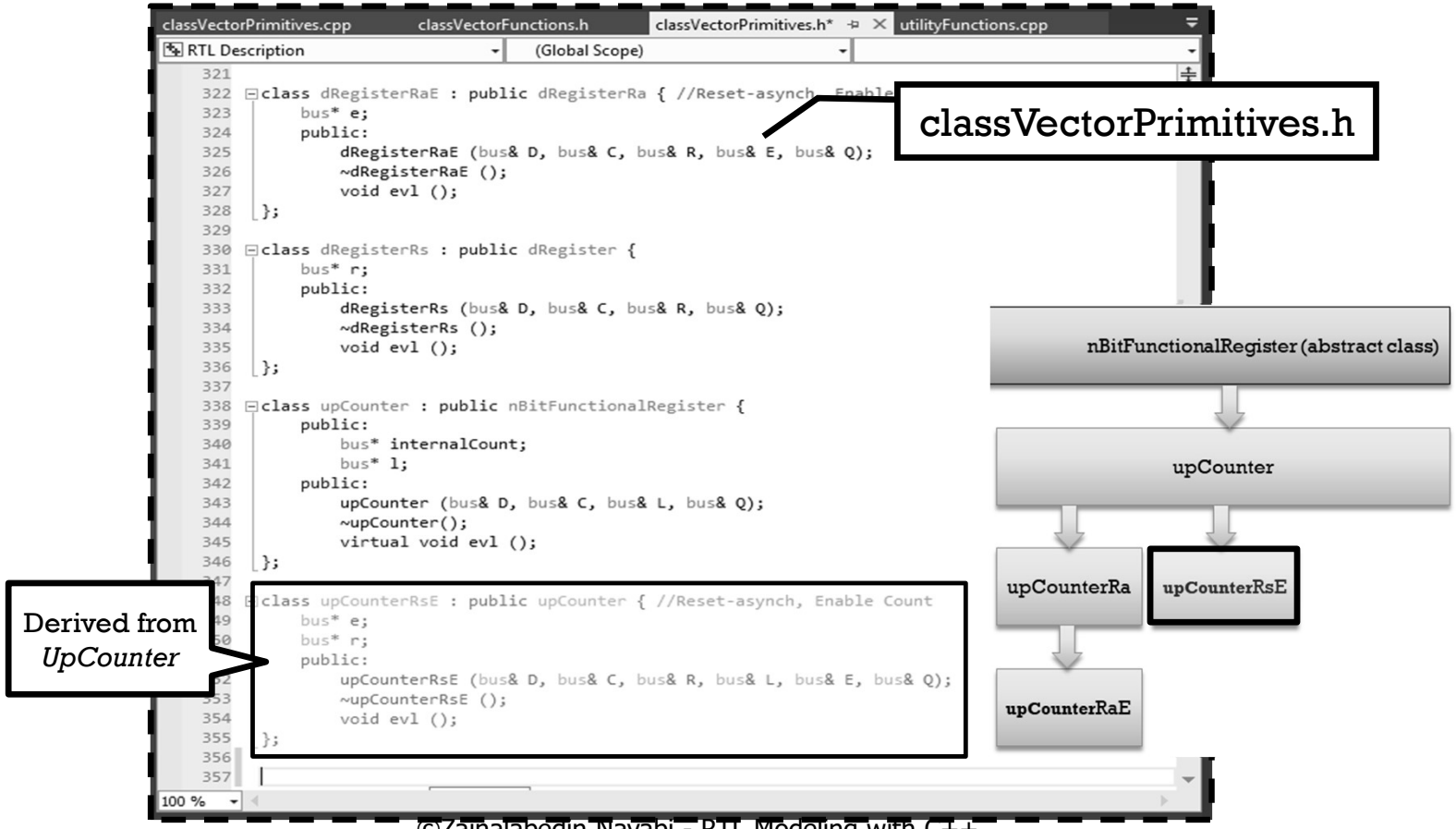
RTL Description | (Global Scope)

```
114 upCounter::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) { ... }
146 void upCounterRsE::evl() { ... }
154
155 upCounterRaE::upCounterRaE(bus& di, bus& clk, bus& rst, bus& ld,
156     bus&en, bus&qo) { ... }
158 void upCounterRaE::evl() { ... }
```

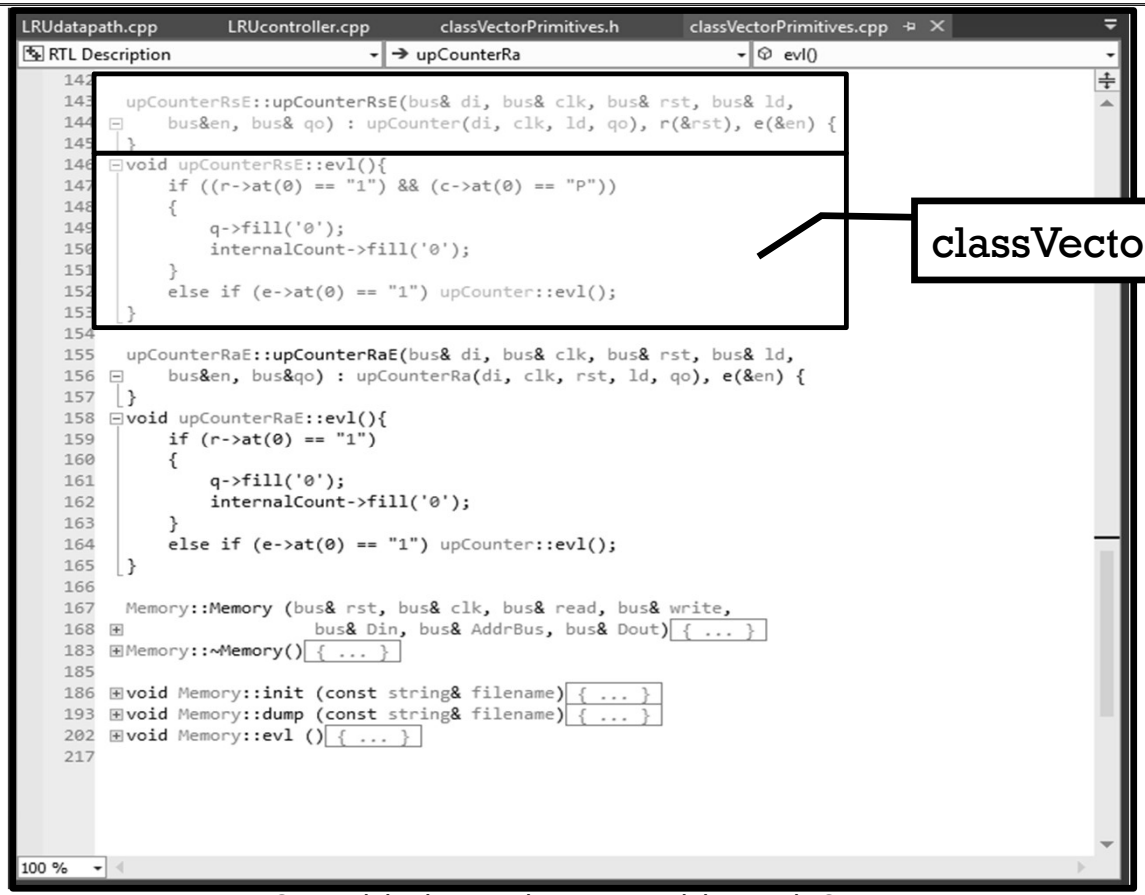
100 %

classVectorPrimitives.cpp

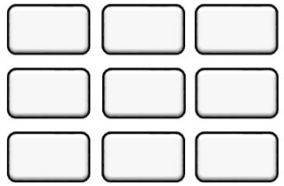
UpCounterRsE



UpCounterRsE



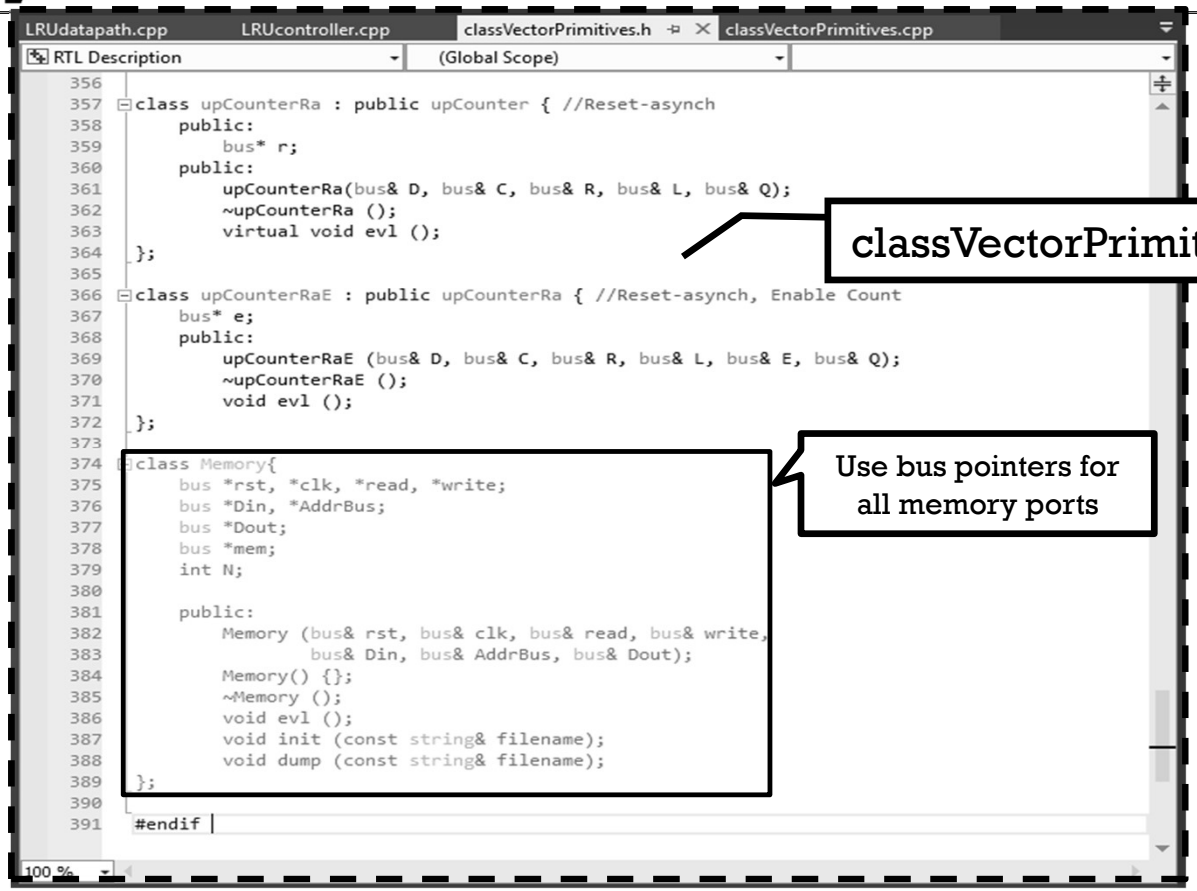
Outline



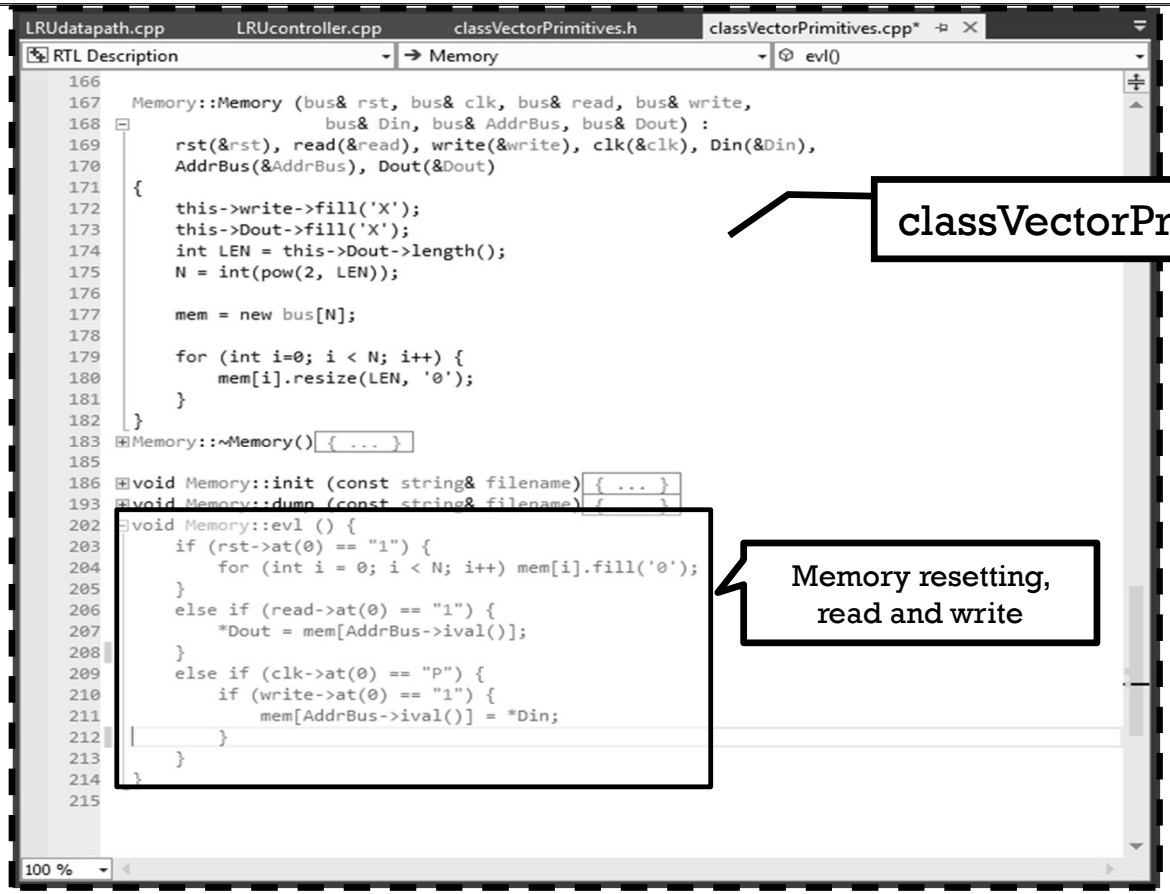
Introduction

+ Bus Model and Operations
+ Basic Elements of RTL
– RTL Design Examples
  - Memory Structure
  - Moore Sequence Detector (11011)
  - Least- Recently-Used (LRU) Circuit
  - Exponential Circuit

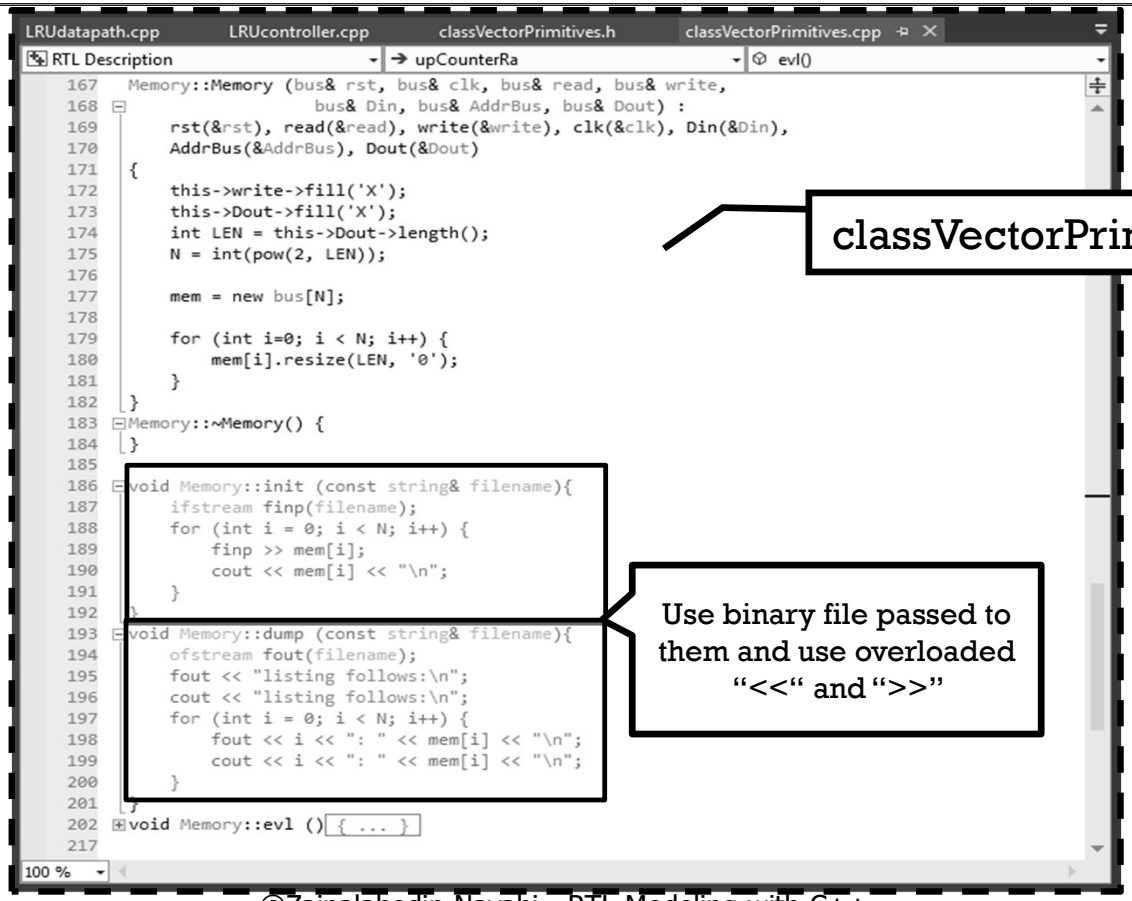
Memory Structure



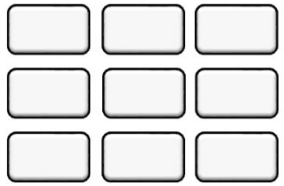
Memory Structure



Memory Structure

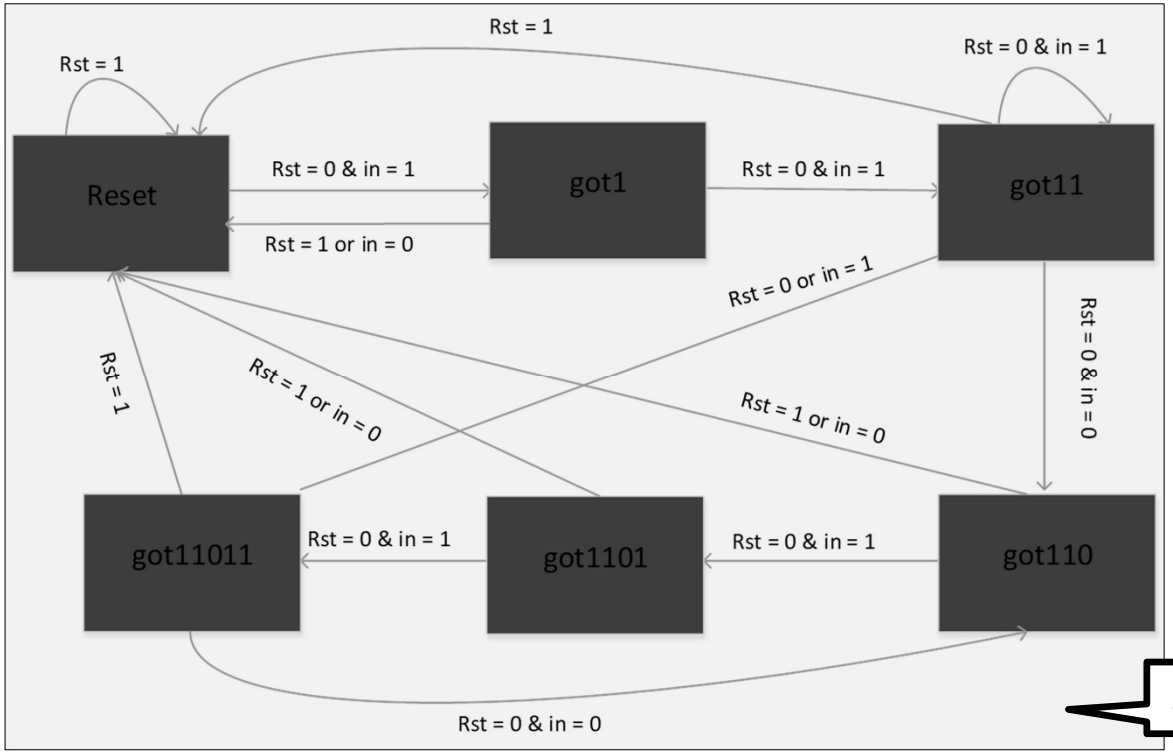


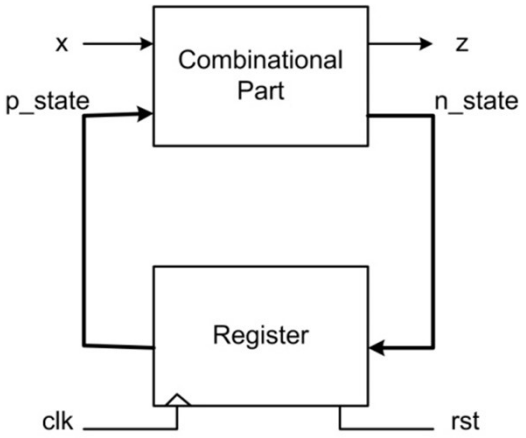
Outline



- 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

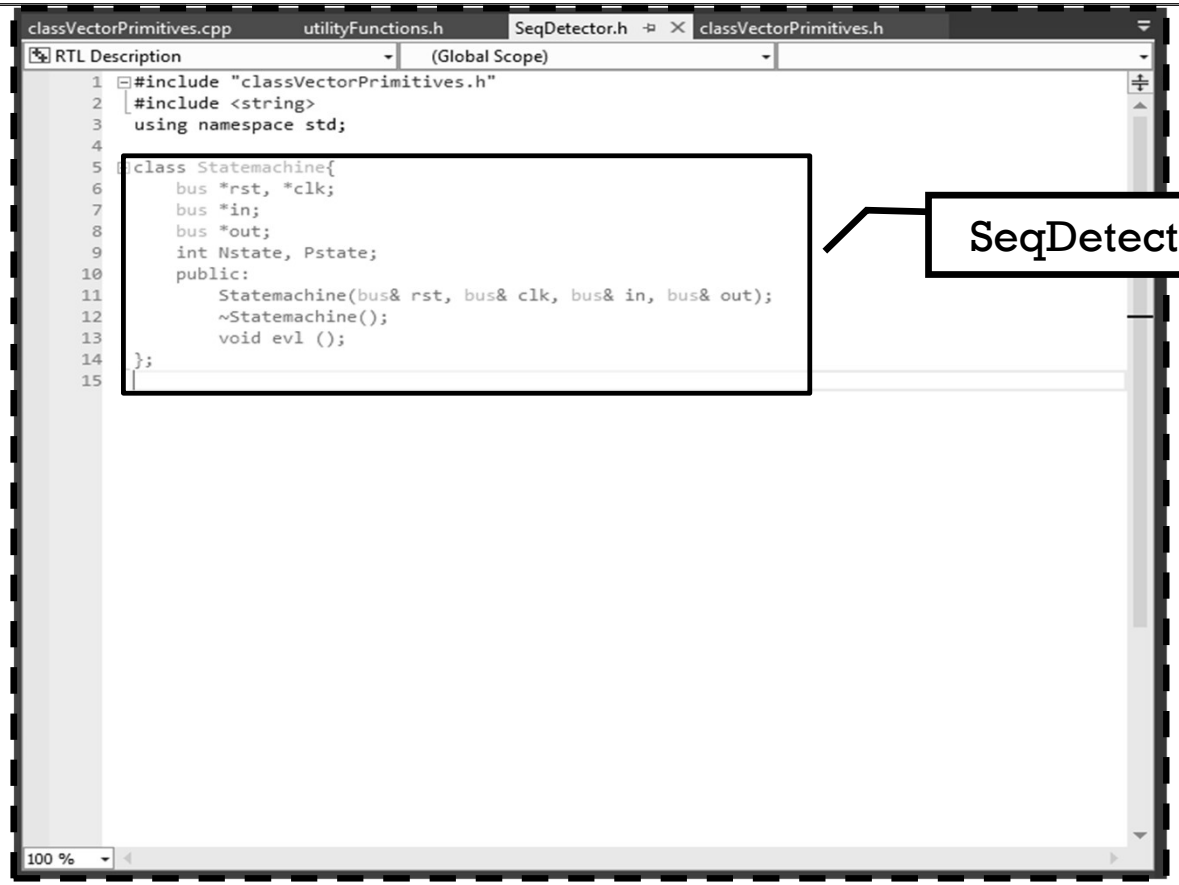
Moore Sequence Detector (11011)



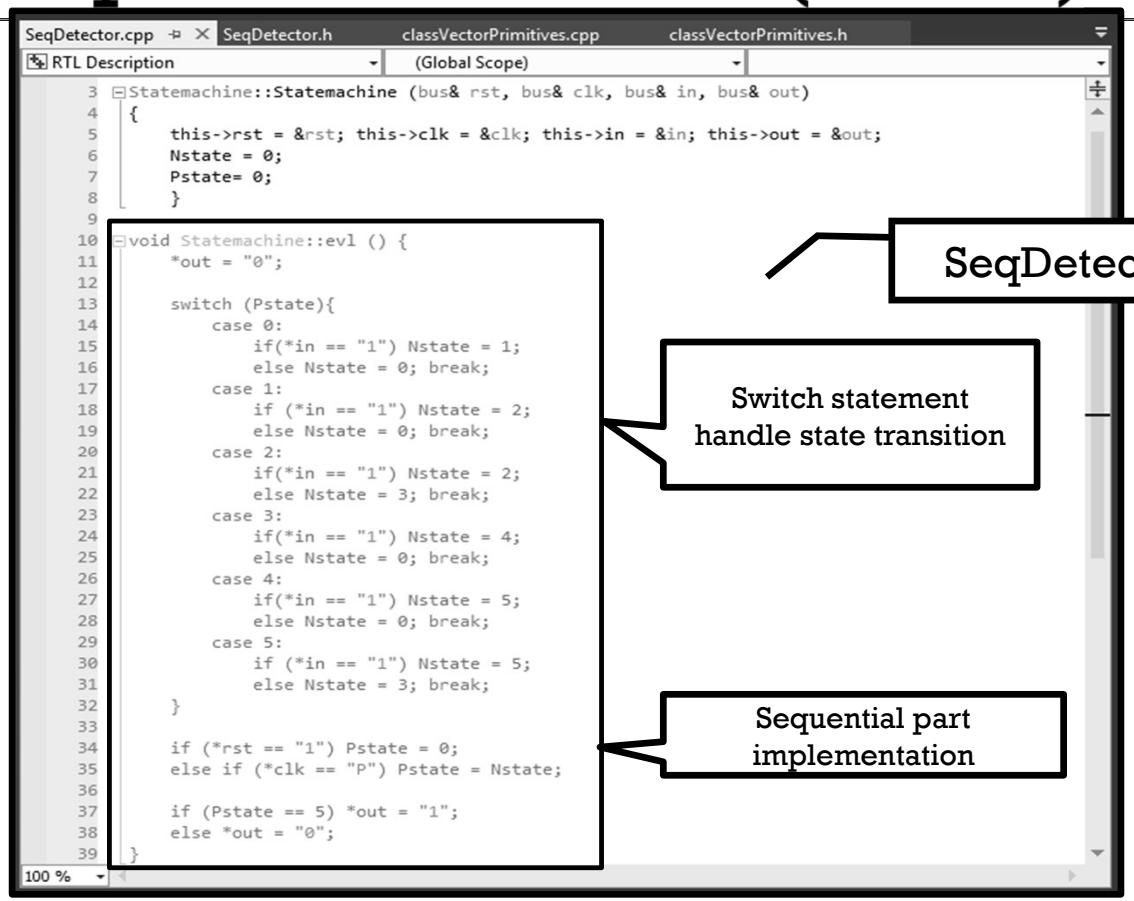


Asynchronous reset

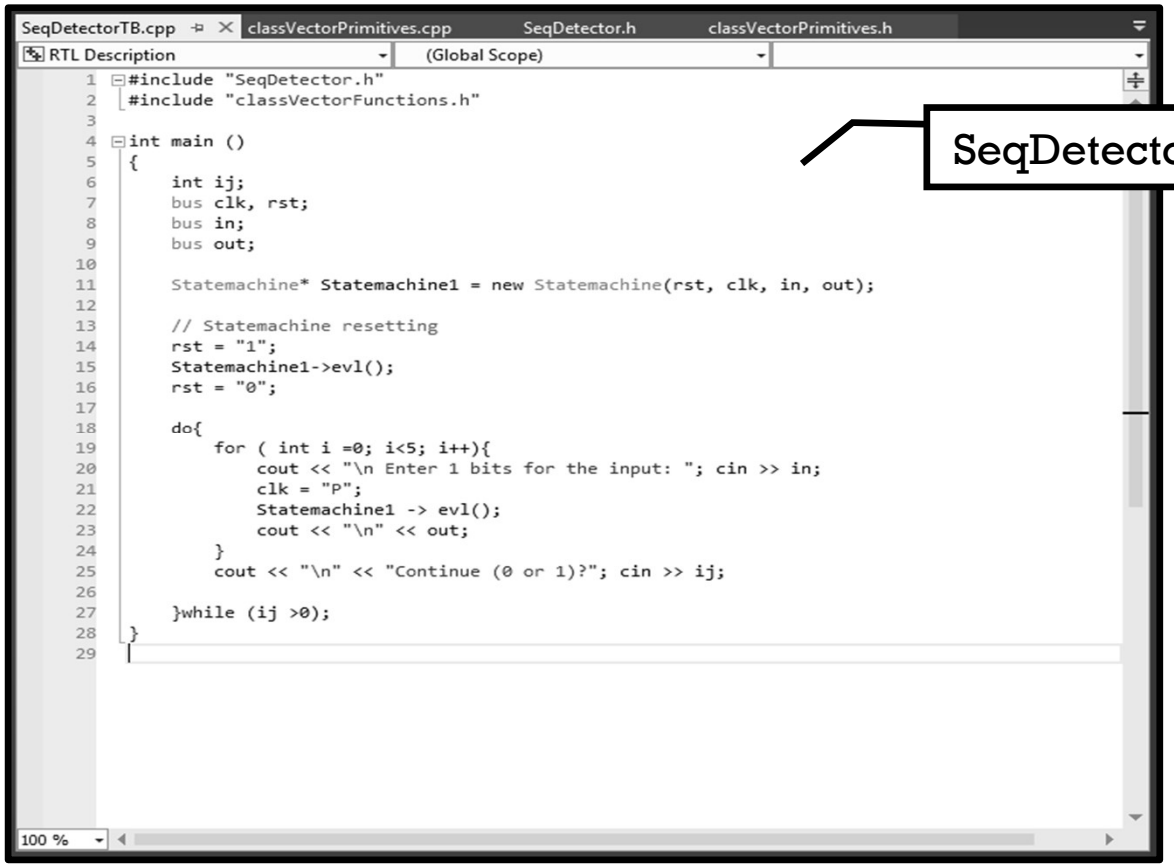
Moore Sequence Detector (11011)



Moore Sequence Detector (11011)

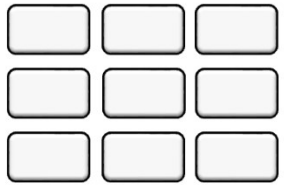


Moore Sequence Detector (11011)



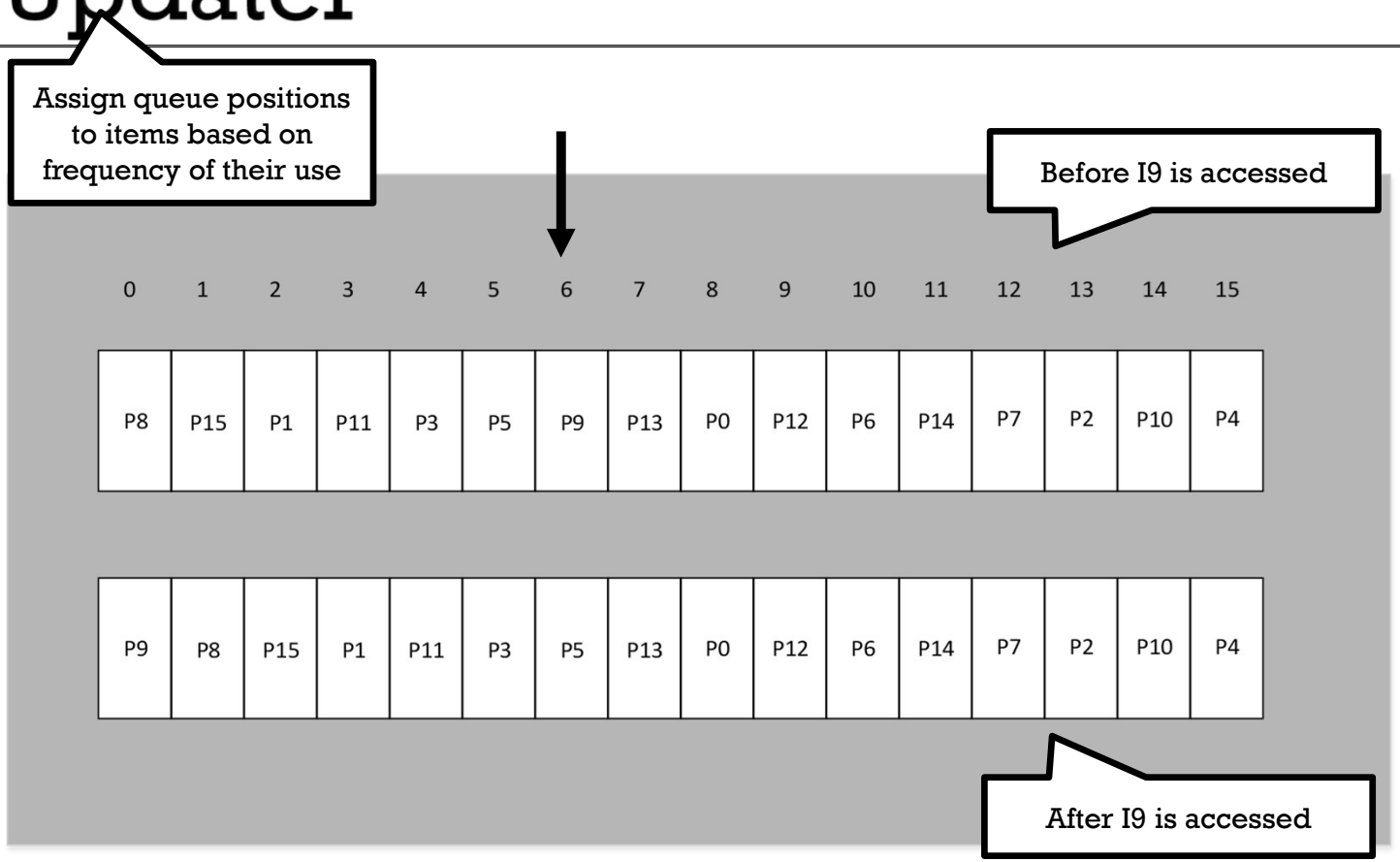

```
#include "SeqDetector.h"
#include "classVectorFunctions.h"

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

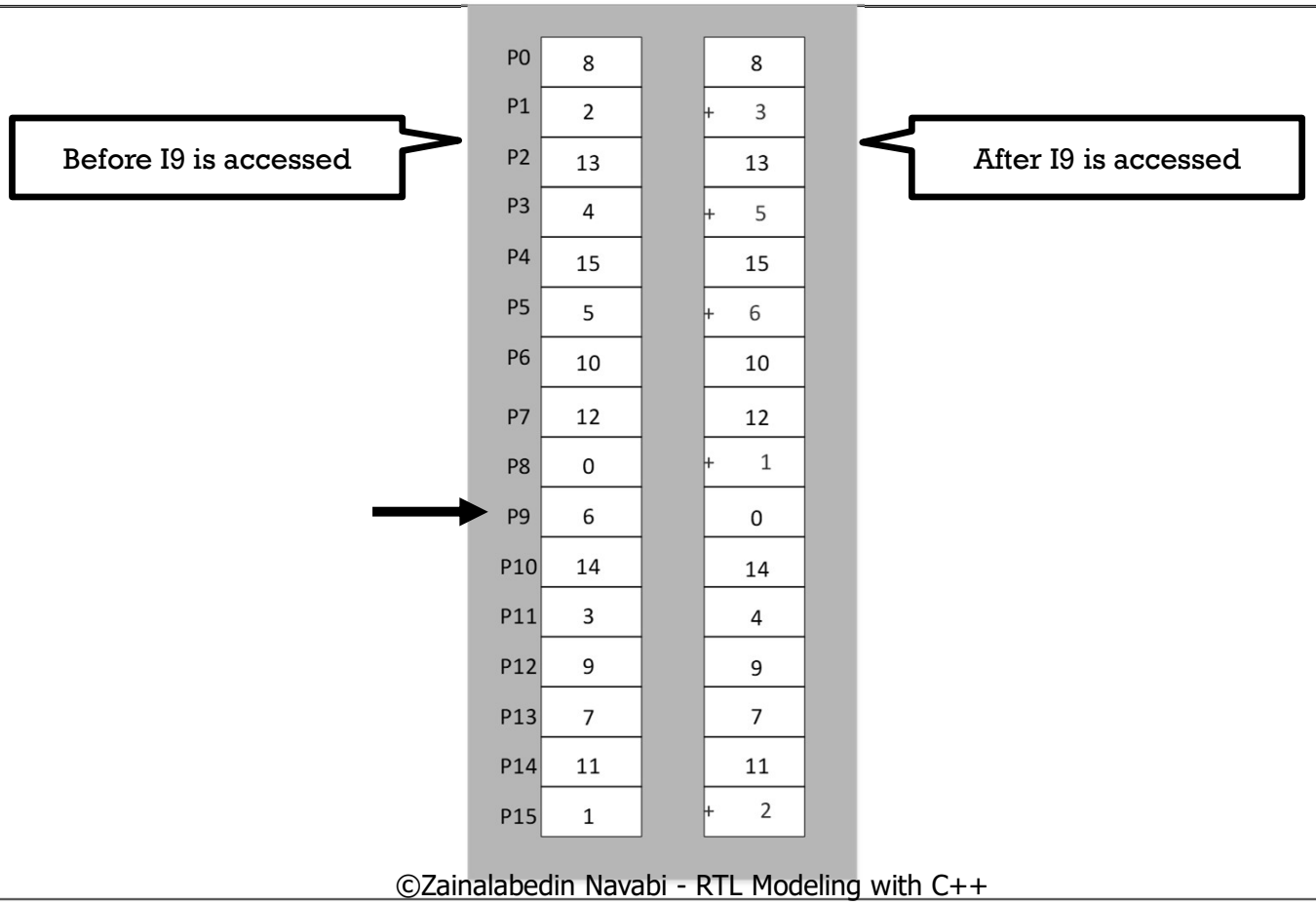


- Introduction
- \+ Bus Model and Operations
- \+ Basic Elements of RTL
- – RTL Design Examples
  - Memory Structure
  - Moore Sequence Detector (11011)
  - Least- Recently-Used (LRU) Circuit
  - Exponential Circuit

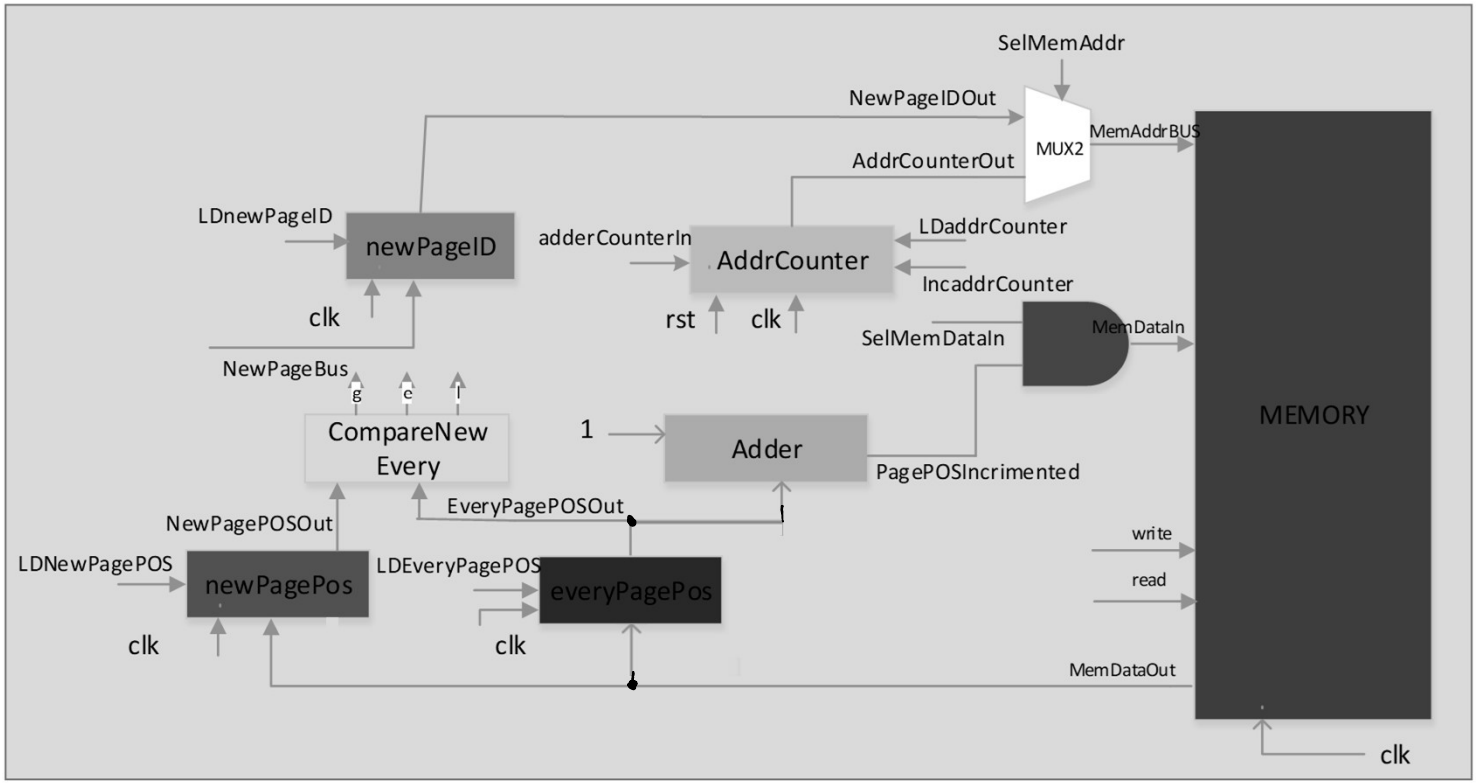
LRU Updater


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

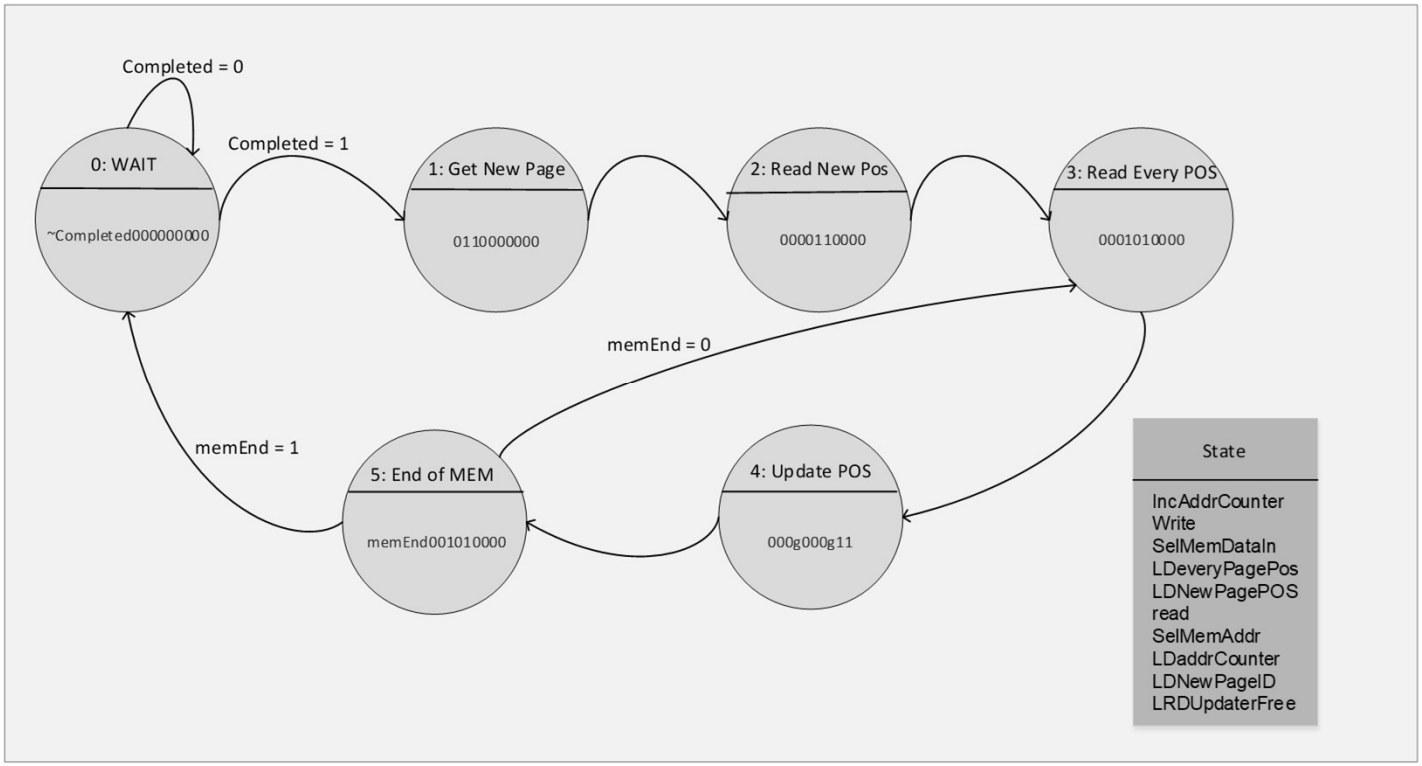
Queue Memory File



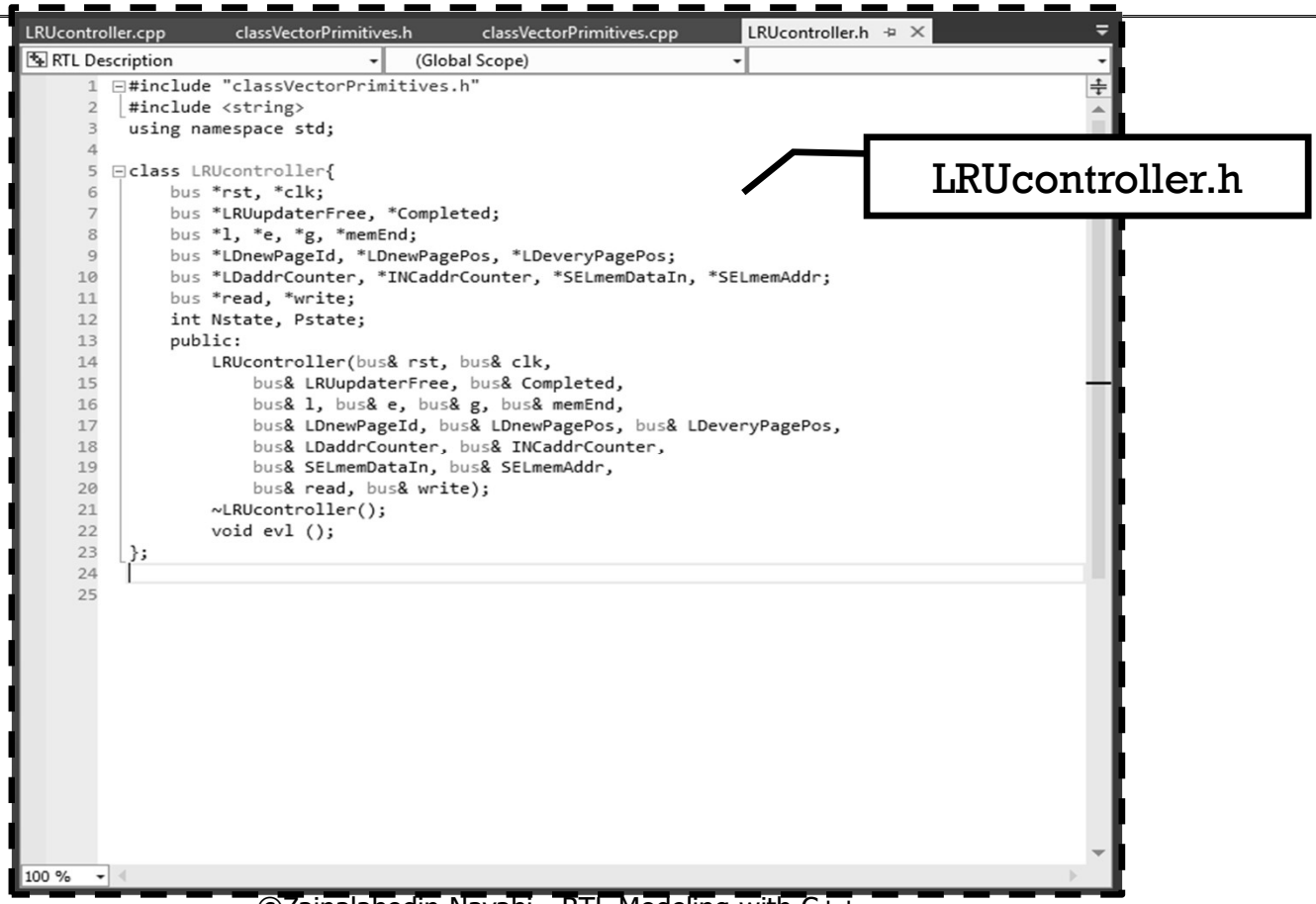
LRU Updater Datapath



LRU Controller



LRU Controller



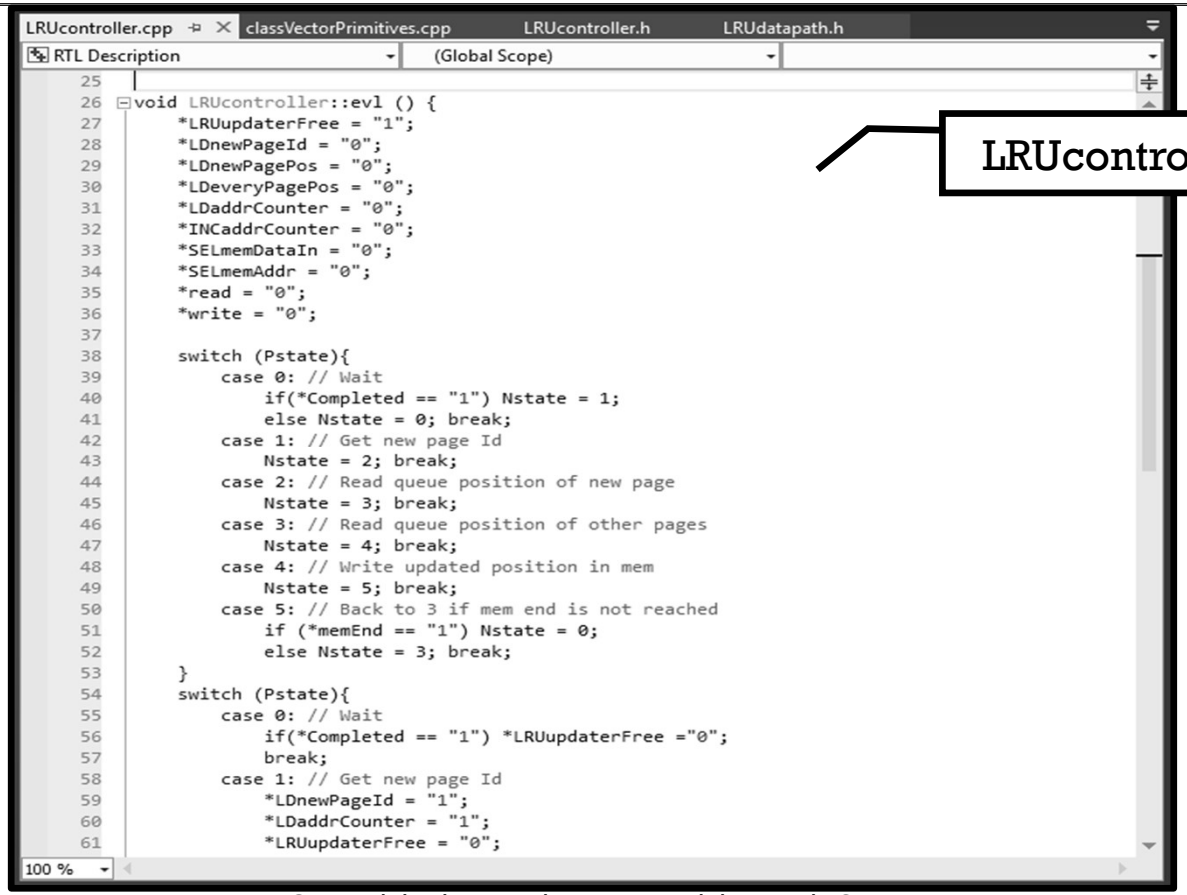

```
#include "classVectorPrimitives.h"
#include <string>
using namespace std;

class LRUcontroller{
     bus *rst, *clk;
     bus *LRUupdaterFree, *Completed;
     bus *l, *e, *g, *memEnd;
     bus *LDnewPageId, *LDnewPagePos, *LDeveryPagePos;
     bus *LDaddrCounter, *INCaddrCounter, *SELmemDataIn, *SELmemAddr;
     bus *read, *write;
     int Nstate, Pstate;
     public:
          LRUcontroller(bus& rst, bus& clk,
               bus& LRUupdaterFree, 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);
          ~LRUcontroller();
          void evl ();
};
```

LRU Controller

LRUcontroller.cpp

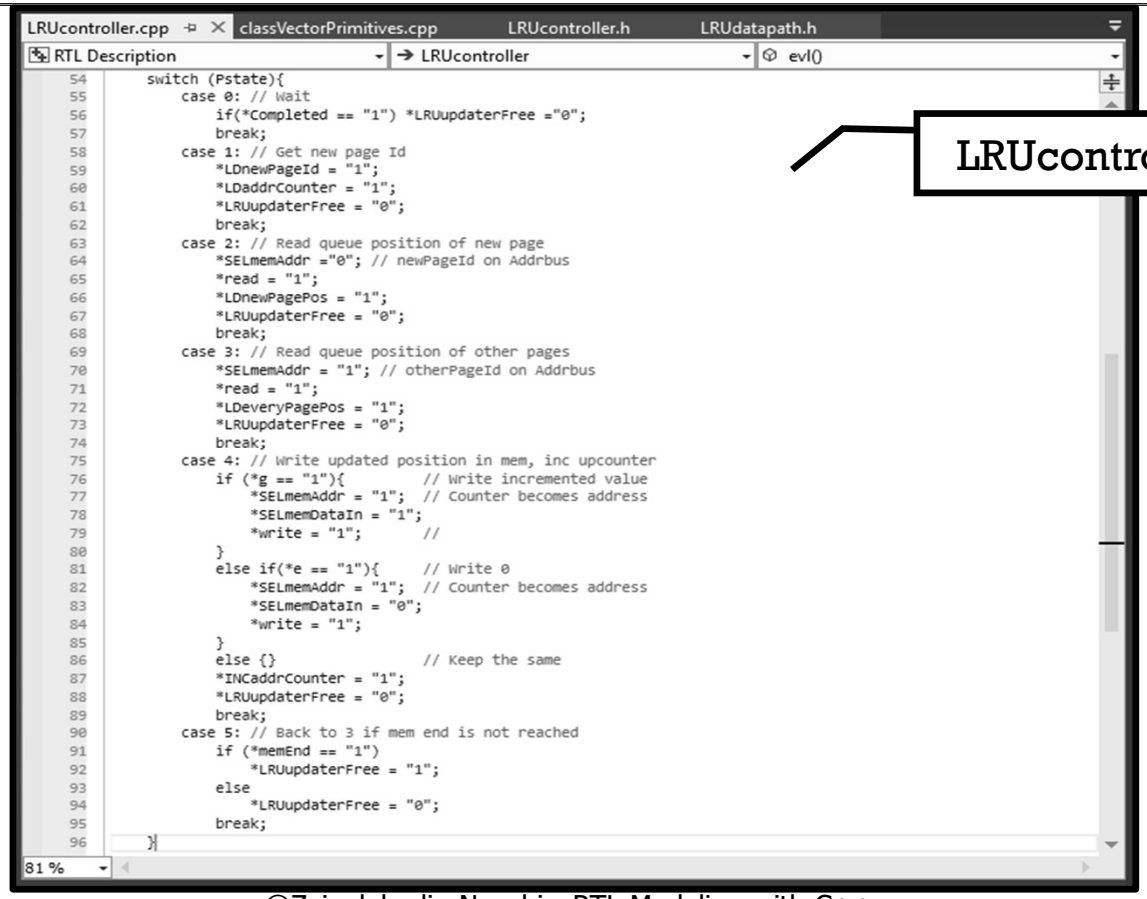
```
LRUcontroller.cpp | classVectorPrimitives.cpp | LRUcontroller.h | LRUdatapath.h
RTL Description | LRUcontroller | evl()
 1  #include "LRUcontroller.h"
 2
 3  LRUcontroller::LRUcontroller (bus& rst, bus& clk,
 4      bus& LRUupdaterFree, bus& Completed,
 5      bus& l, bus& e, bus& g, bus& memEnd,
 6      bus& LDnewPageId, bus& LDnewPagePos, bus& LDeveryPagePos,
 7      bus& LDaddrCounter, bus& INCaddrCounter,
 8      bus& SELmemDataIn, bus& SELmemAddr,
 9      bus& read, bus& write)      :
10
11      rst(&rst), clk(&clk),
12      LRUupdaterFree(&LRUupdaterFree), Completed(&Completed),
13      l(&l), e(&e), g(&g), memEnd(&memEnd),
14      LDnewPageId(&LDnewPageId), LDnewPagePos(&LDnewPagePos),
15      LDeveryPagePos(&LDeveryPagePos),
16      LDaddrCounter(&LDaddrCounter), INCaddrCounter(&INCaddrCounter),
17      SELmemDataIn(&SELmemDataIn), SELmemAddr(&SELmemAddr),
18      read(&read), write(&write)
19  {
20      this->e->fill('0');
21      this->g->fill('0');
22      Nstate = 0;
23      Pstate= 0;
24      }
25
26  void LRUcontroller::evl () {
27      *LRUupdaterFree = "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
100 %
```

LRU Controller

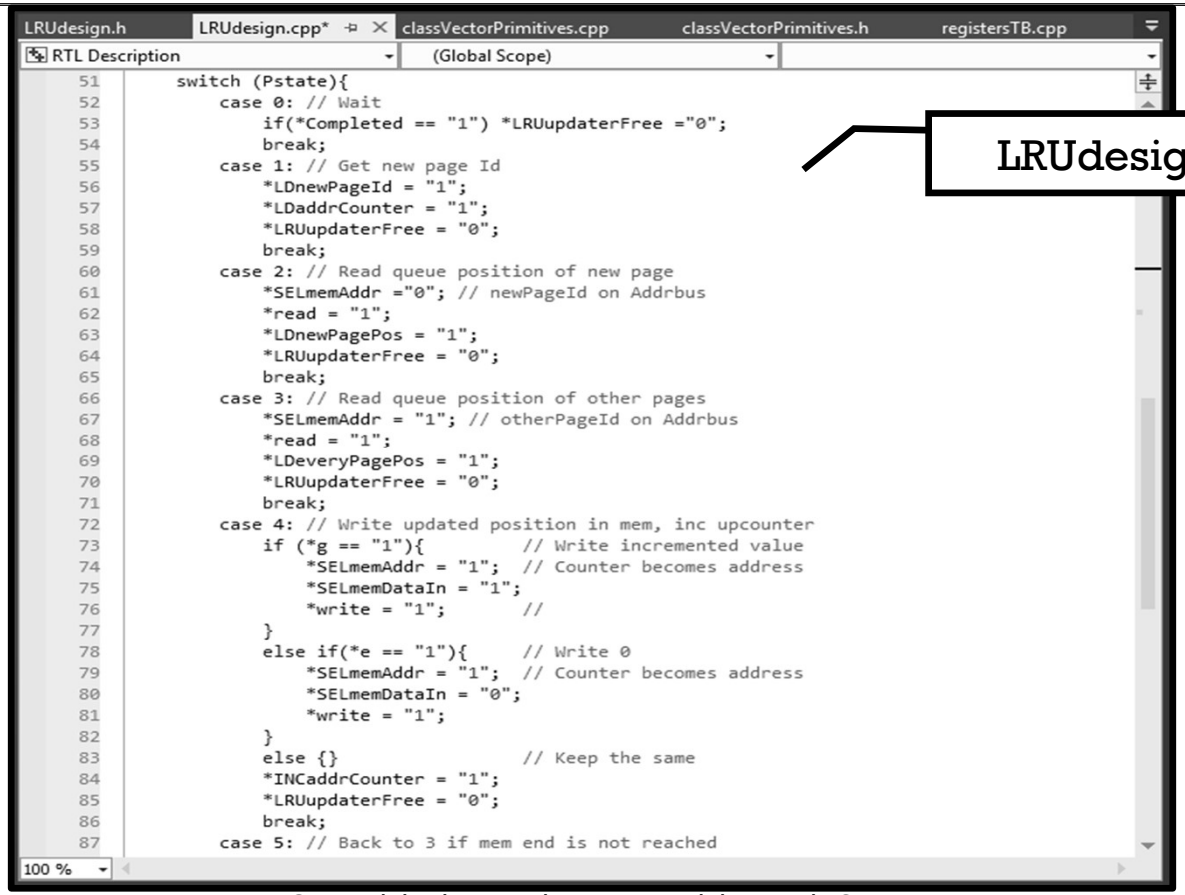



```
25
26  void LRUcontroller::evl () {
27      *LRUupdaterFree = "1";
28      *LDnewPageId = "0";
29      *LDnewPagePos = "0";
30      *LDeveryPagePos = "0";
31      *LDaddrCounter = "0";
32      *INCaddrCounter = "0";
33      *SELmemDataIn = "0";
34      *SELmemAddr = "0";
35      *read = "0";
36      *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") *LRUupdaterFree ="0";
57              break;
58          case 1: // Get new page Id
59              *LDnewPageId = "1";
60              *LDaddrCounter = "1";
61              *LRUupdaterFree = "0";
```

LRU Controller

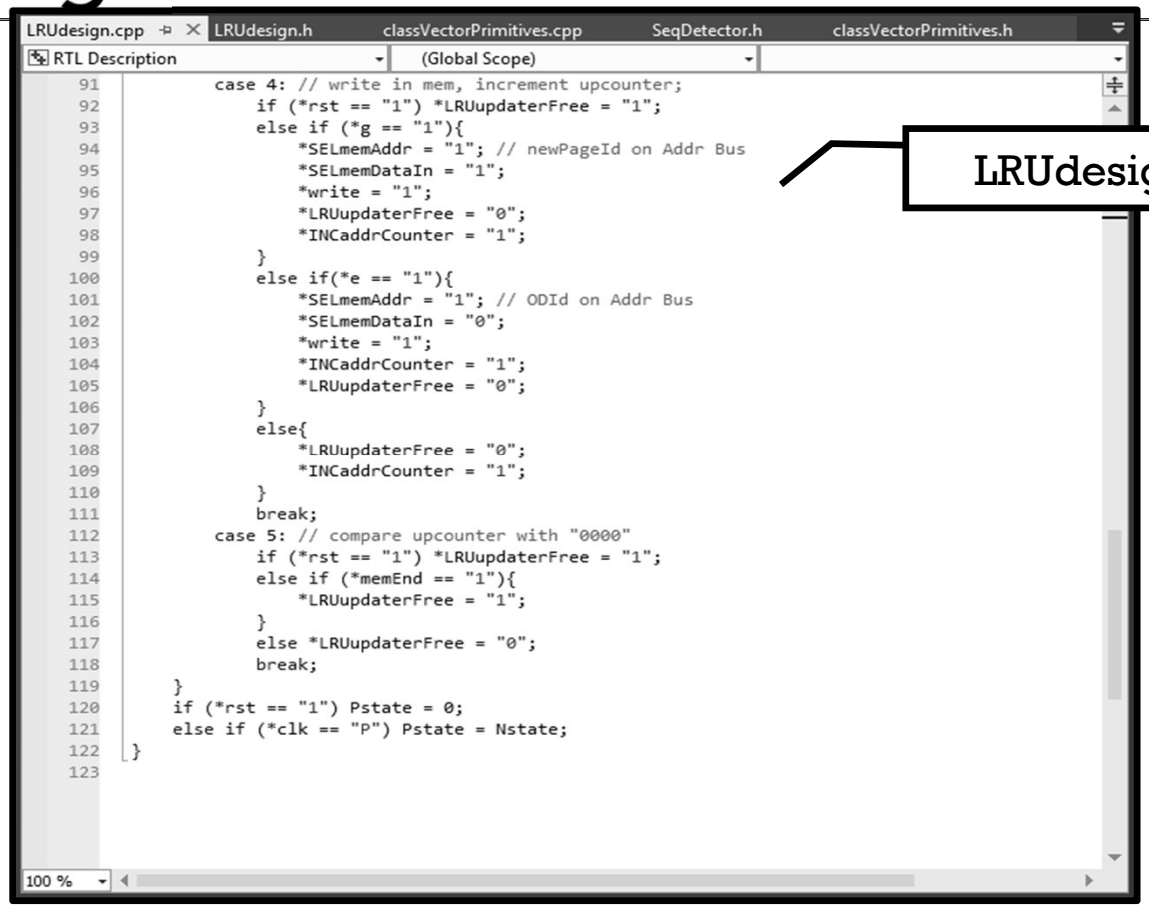


LRU Design



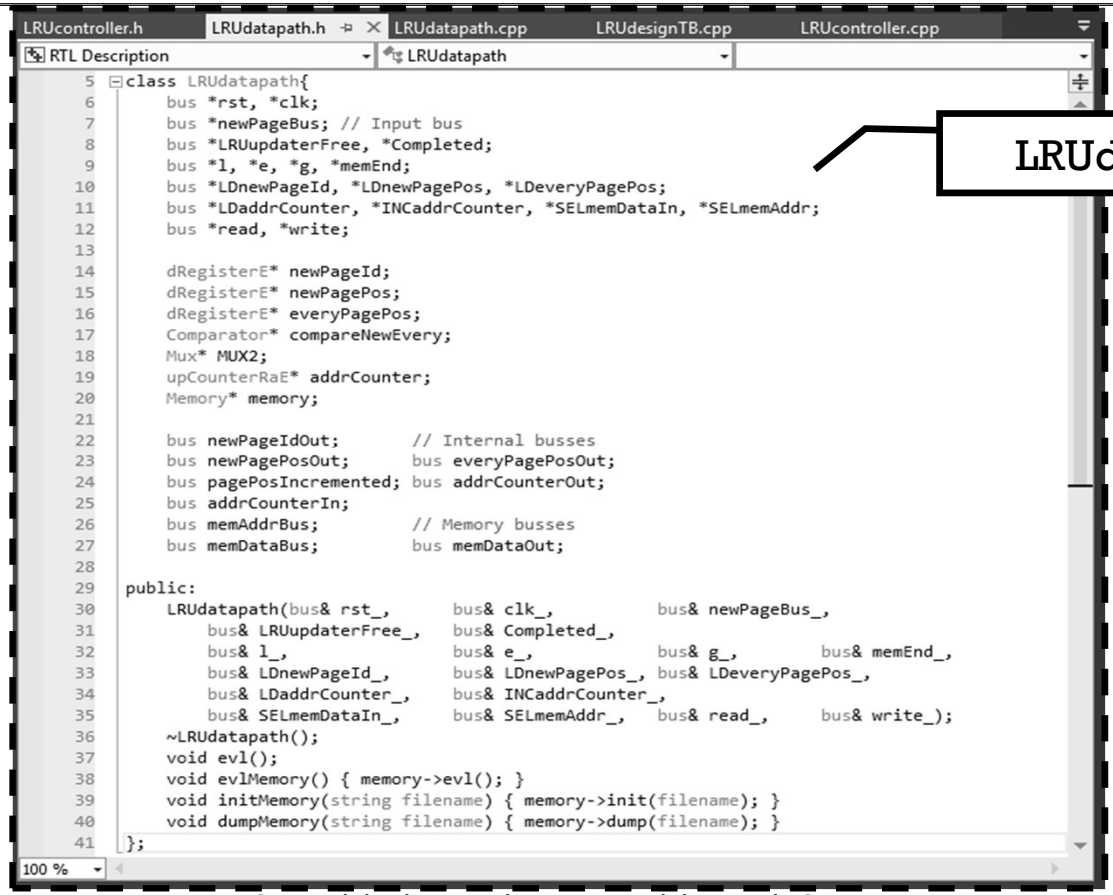

```
51      switch (Pstate){
52          case 0: // Wait
53              if(*Completed == "1") *LRUupdaterFree ="0";
54              break;
55          case 1: // Get new page Id
56              *LDnewPageId = "1";
57              *LDaddrCounter = "1";
58              *LRUupdaterFree = "0";
59              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"){          // Write incremented value
74                  *SELmemAddr = "1";  // Counter becomes address
75                  *SELmemDataIn = "1";
76                  *write = "1";       //
77              }
78              else if(*e == "1"){      // Write 0
79                  *SELmemAddr = "1";  // Counter becomes address
80                  *SELmemDataIn = "0";
81                  *write = "1";
82              }
83              else {}                  // Keep the same
84              *INCaddrCounter = "1";
85              *LRUupdaterFree = "0";
86              break;
87          case 5: // Back to 3 if mem end is not reached
```

LRU Design




```
91          case 4: // write in mem, increment upcounter;
92              if (*rst == "1") *LRUupdaterFree = "1";
93              else if (*g == "1"){
94                  *SELmemAddr = "1"; // newPageId on Addr Bus
95                  *SELmemDataIn = "1";
96                  *write = "1";
97                  *LRUupdaterFree = "0";
98                  *INCaddrCounter = "1";
99              }
100             else if(*e == "1"){
101                 *SELmemAddr = "1"; // ODId on Addr Bus
102                 *SELmemDataIn = "0";
103                 *write = "1";
104                 *INCaddrCounter = "1";
105                 *LRUupdaterFree = "0";
106             }
107             else{
108                 *LRUupdaterFree = "0";
109                 *INCaddrCounter = "1";
110             }
111             break;
112         case 5: // compare upcounter with "0000"
113             if (*rst == "1") *LRUupdaterFree = "1";
114             else if (*memEnd == "1"){
115                 *LRUupdaterFree = "1";
116             }
117             else *LRUupdaterFree = "0";
118             break;
119     }
120     if (*rst == "1") Pstate = 0;
121     else if (*clk == "P") Pstate = Nstate;
122 }
123
```

LRU Datapath

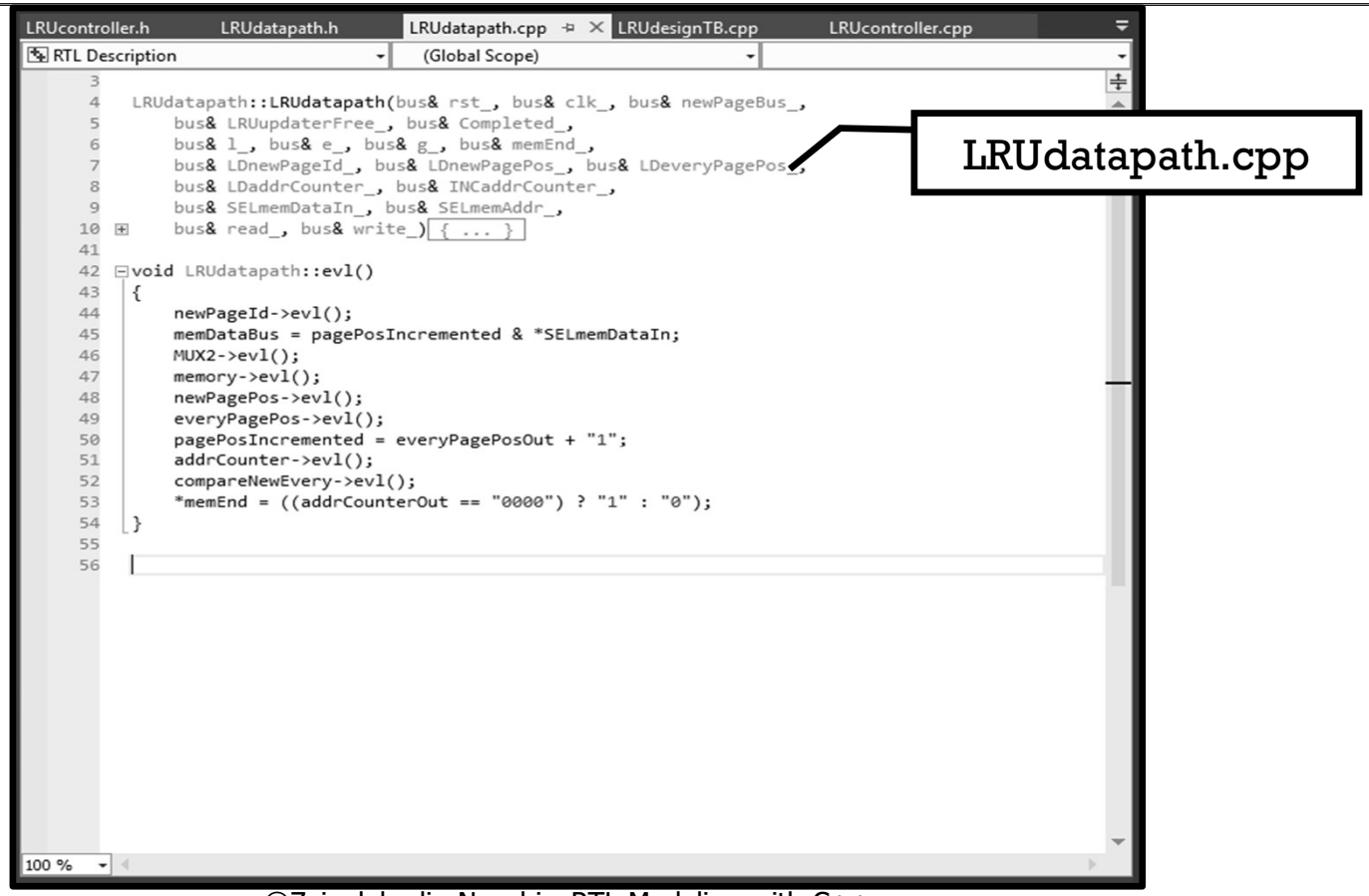



```
 5  class LRUdatapath{
 6      bus *rst, *clk;
 7      bus *newPageBus; // Input bus
 8      bus *LRUupdaterFree, *Completed;
 9      bus *l, *e, *g, *memEnd;
10      bus *LDnewPageId, *LDnewPagePos, *LDeveryPagePos;
11      bus *LDaddrCounter, *INCaddrCounter, *SELmemDataIn, *SELmemAddr;
12      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& LRUupdaterFree_,   bus& Completed_,
32          bus& l_,                bus& e_,            bus& g_,        bus& memEnd_,
33          bus& LDnewPageId_,      bus& LDnewPagePos_, bus& LDeveryPagePos_,
34          bus& LDaddrCounter_,    bus& INCaddrCounter_,
35          bus& SELmemDataIn_,     bus& SELmemAddr_,   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

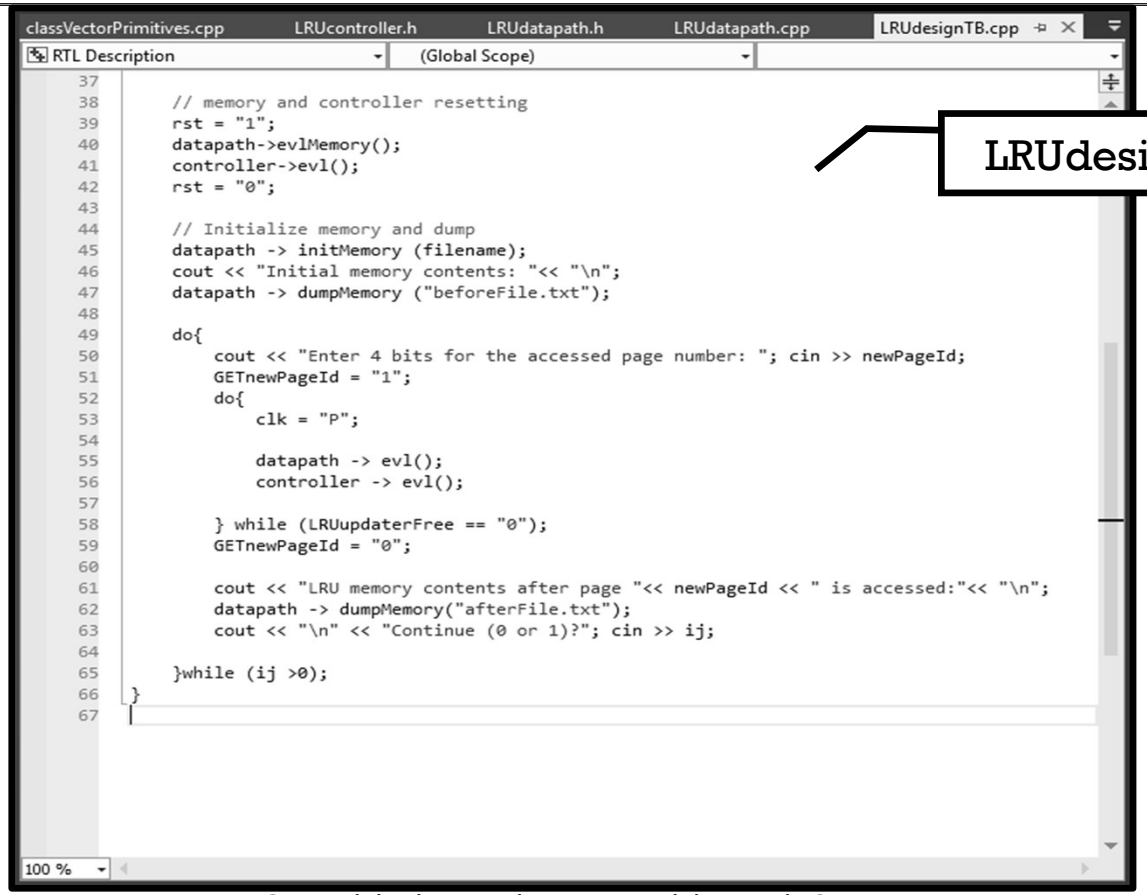

```
4   LRUdatapath::LRUdatapath(bus& rst_, bus& clk_, bus& newPageBus_,
5       bus& LRUupdaterFree_, bus& Completed_,
6       bus& l_, bus& e_, bus& g_, bus& memEnd_,
7       bus& LDnewPageId_, bus& LDnewPagePos_, bus& LDeveryPagePos_,
8       bus& LDaddrCounter_, bus& INCaddrCounter_,
9       bus& 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, memAddrBus, memDataOut);
40  }
```

LRU Datapath



LRU Testbench

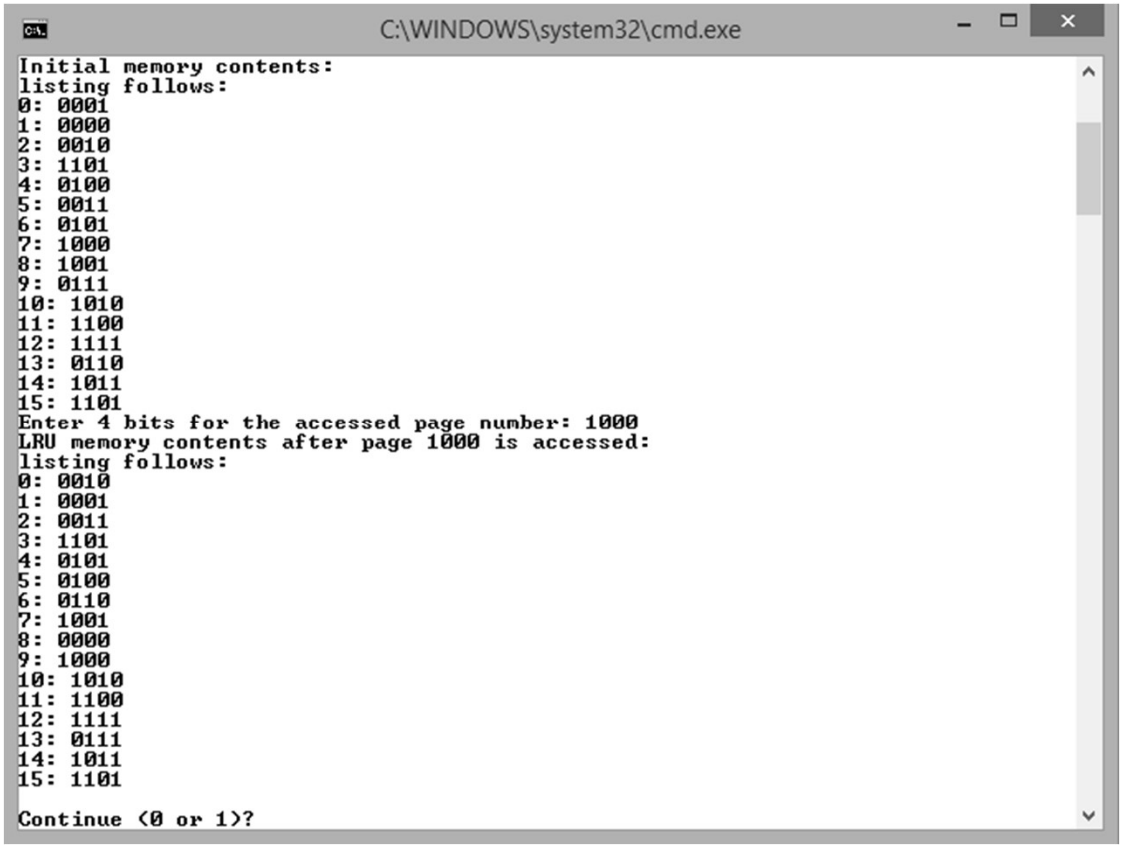
LRU Testbench



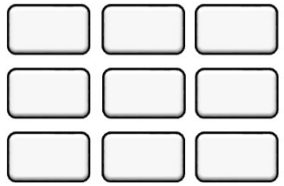

```
37
38      // memory and controller resetting
39      rst = "1";
40      datapath->evlMemory();
41      controller->evl();
42      rst = "0";
43
44      // Initialize memory and dump
45      datapath -> initMemory (filename);
46      cout << "Initial memory contents: "<< "\n";
47      datapath ->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

LRUdesignTB.cpp

LRU Output

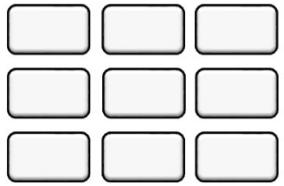

C:\WINDOWS\system32\cmd.exe
Initial memory contents:
listing follows:
0: 0001
1: 0000
2: 0010
3: 1101
4: 0100
5: 0011
6: 0101
7: 1000
8: 1001
9: 0111
10: 1010
11: 1100
12: 1111
13: 0110
14: 1011
15: 1101
Enter 4 bits for the accessed page number: 1000
LRU memory contents after page 1000 is accessed:
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

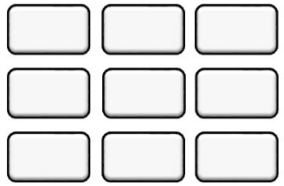


- Taylor Series

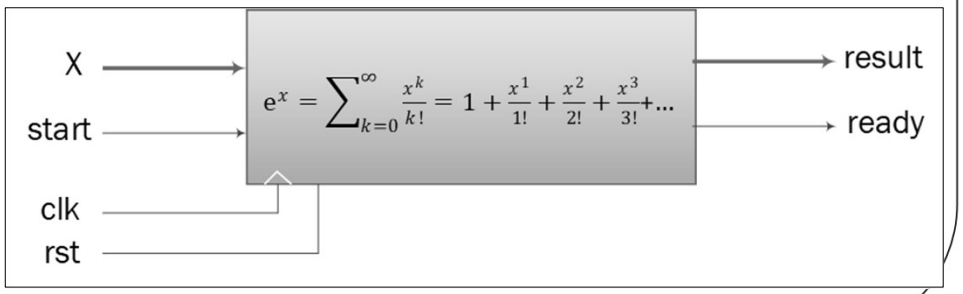
$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

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

Exponential Circuit



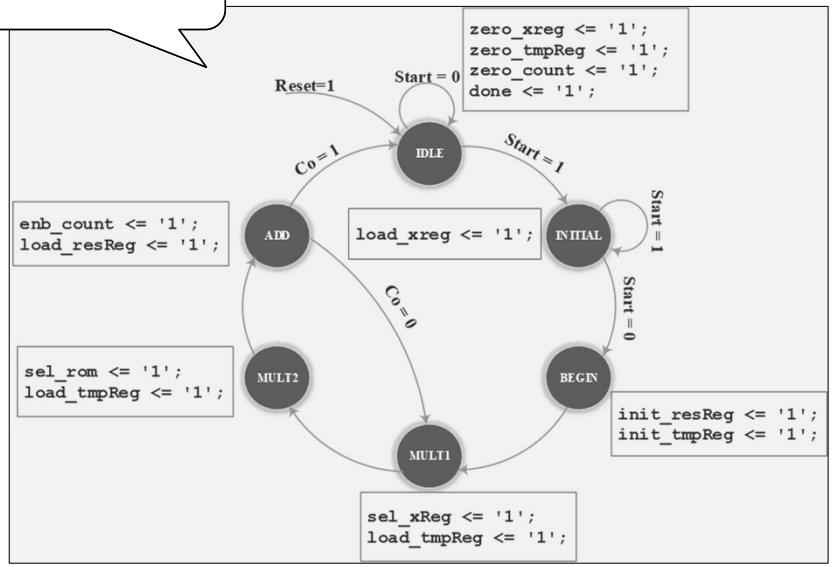
- The circuit calculates e^x using Taylor expansion.
- The input is an 8-bit fixed-point number.
- The output is a 10-bit fixed-point number including 2 integer bits and 8 fractional bits.
- The circuit receives *x* as the input with the pulse on the start signal.
- The calculation continues for 8 iterations.
  - 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



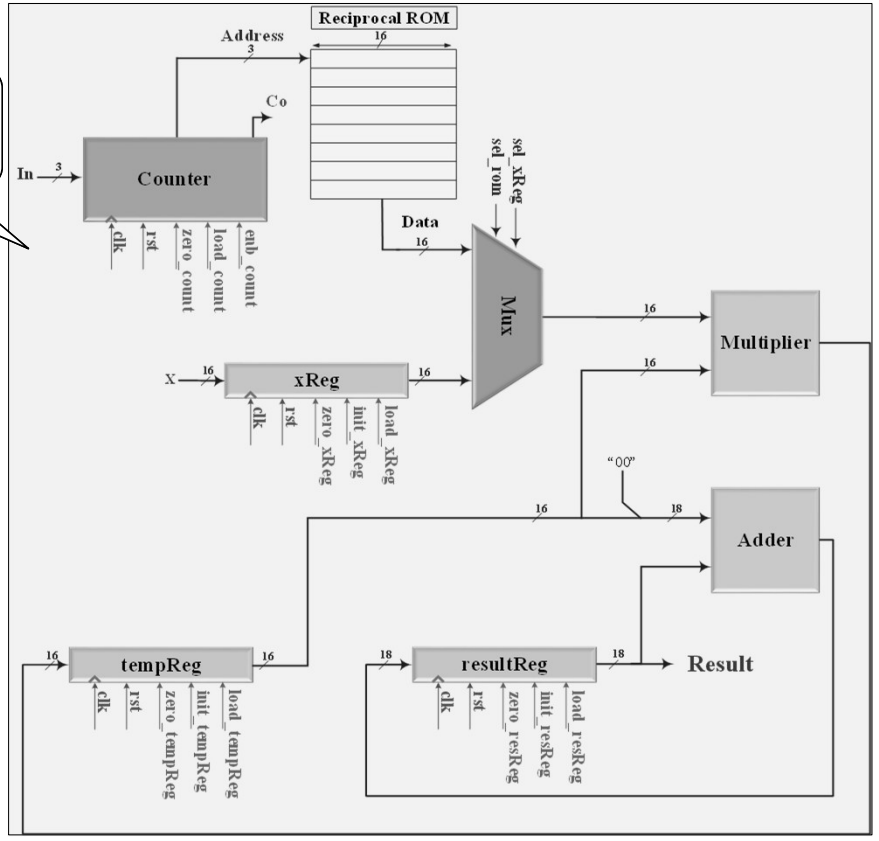
Exponential Circuit

- RTL design

Controller



Datapath



Exponential Circuit – Datapath

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

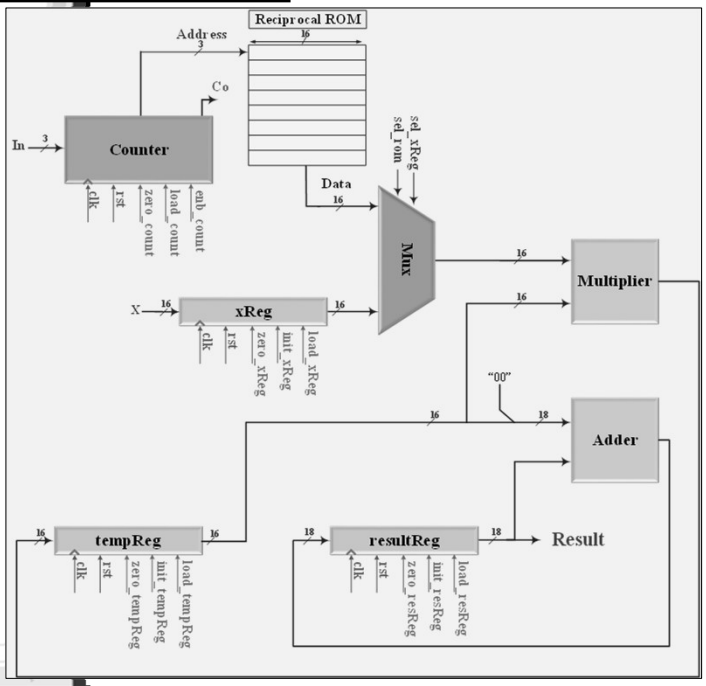
Exponential | expDatapath

```
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 memDatabus;
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
```

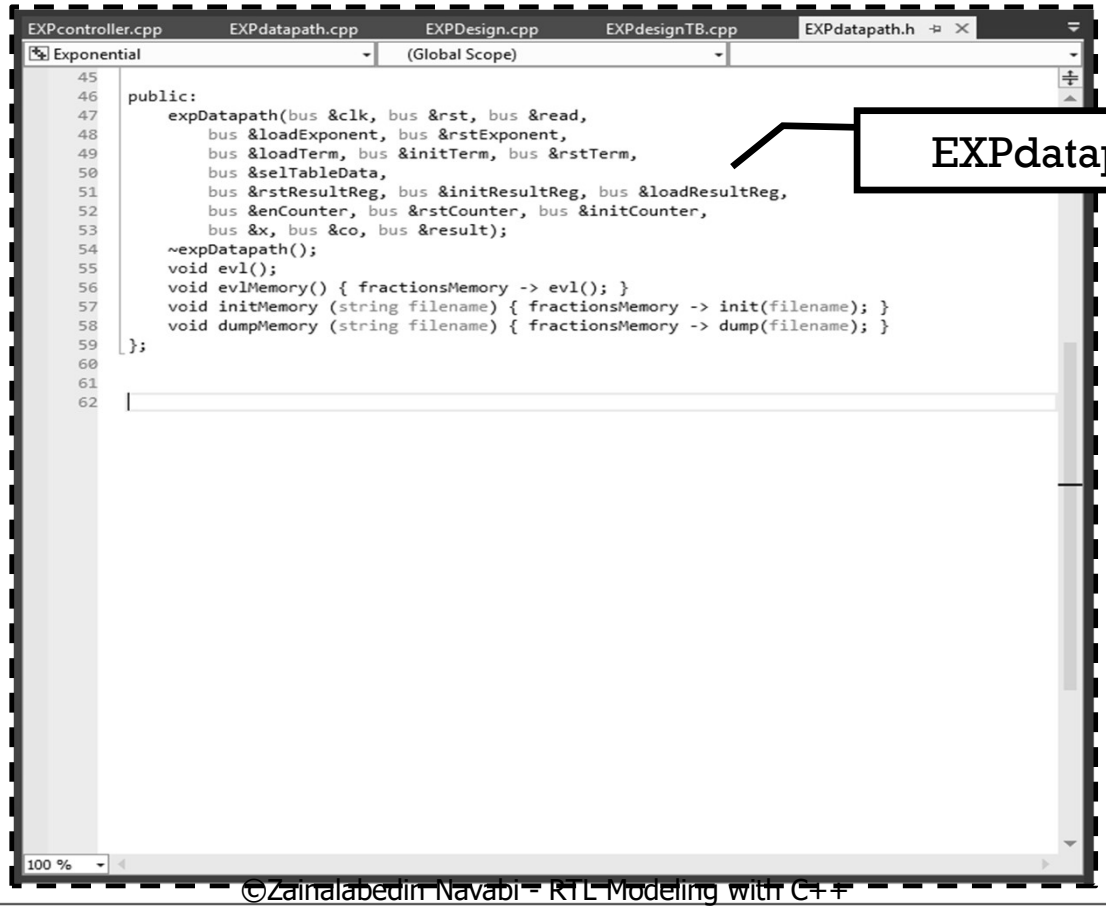
100 %

EXPdatapath.h

Register for saving partial results



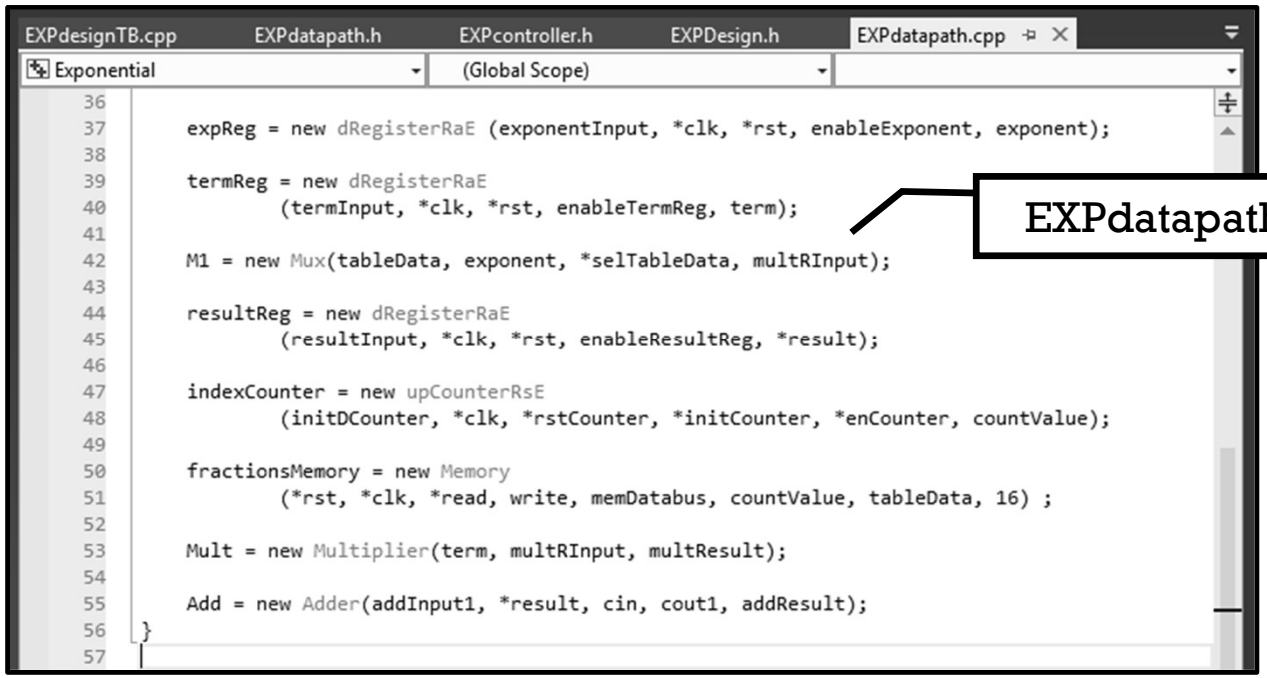
Exponential Circuit – Datapath



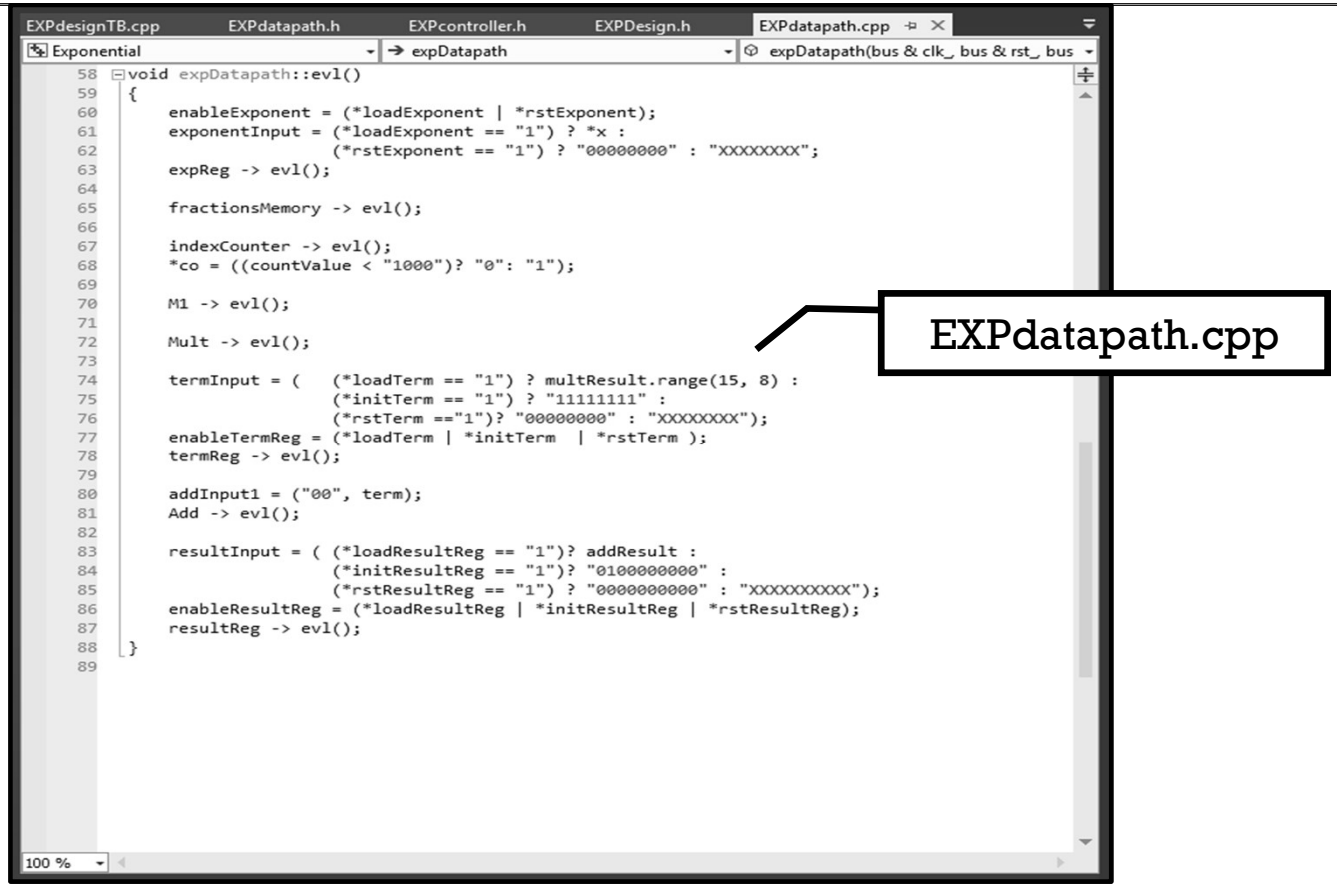
Exponential Circuit – Datapath


```
#include "expDatapath.h"

expDatapath::expDatapath(bus &clk_, bus &rst_, bus &read_,
    bus &loadExponent_, bus &rstExponent_,
    bus &loadTerm_, bus &initTerm_, bus &rstTerm_,
    bus &selTableData_,
    bus &rstResultReg_, bus &initResultReg_, bus &loadResultReg_,
    bus &enCounter_, bus &rstCounter_, bus &initCounter_,
    bus &x_, bus &co_, bus &result_)
                        :
    clk(&clk_), rst(&rst_), read(&read_),
    loadExponent(&loadExponent_), rstExponent(&rstExponent_),
    loadTerm(&loadTerm_), initTerm(&initTerm_), rstTerm(&rstTerm_),
    selTableData(&selTableData_),
    rstResultReg(&rstResultReg_), initResultReg(&initResultReg_),
    loadResultReg(&loadResultReg_),
    enCounter(&enCounter_), rstCounter(&rstCounter_), initCounter(&initCounter_),
    x(&x_), co(&co_), result(&result_)
{
    // internal buses:
    exponent.assign("XXXXXXXX");
    term.assign("XXXXXXXX");
    termInput.assign("XXXXXXXX");
    countValue.assign("XXXX");
    initDCounter.assign("0001");
    multRInput.assign("XXXXXXXX");
    addResult.assign("XXXXXXXXX");
    resultInput.assign("XXXXXXXXX");
    multResult.assign("XXXXXXXXXXXXXXX");
    tableData.assign("XXXXXXXX");
    memDatabus.assign("XXXXXXXX");
    addInput1.assign("XXXXXXXXX");
    exponentInput.assign("XXXXXXXX");

    cin = "0";
```

Exponential Circuit – Datapath



Exponential Circuit – Datapath




```
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" : "XXXXXXXXXX");
86      enableResultReg = (*loadResultReg | *initResultReg | *rstResultReg);
87      resultReg -> evl();
88  }
89
```

Exponential Circuit – Controller

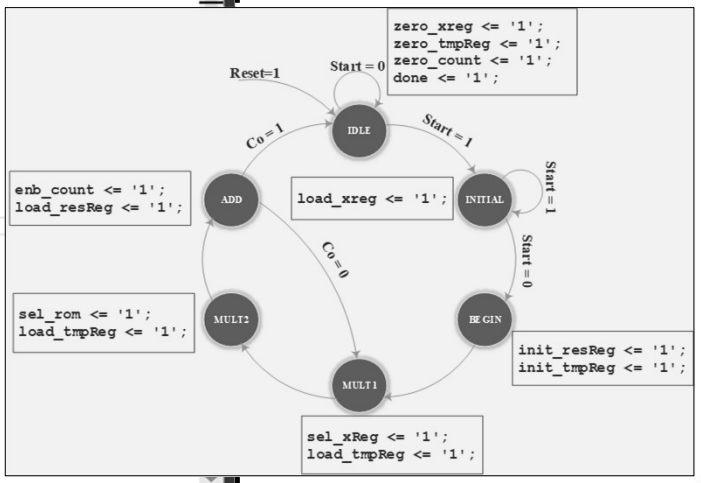
EXPdatapath.cpp | EXPDesign.cpp | EXPdesignTB.cpp | EXPdatapath.h | EXPcontroller.h

Exponential | (Global Scope)

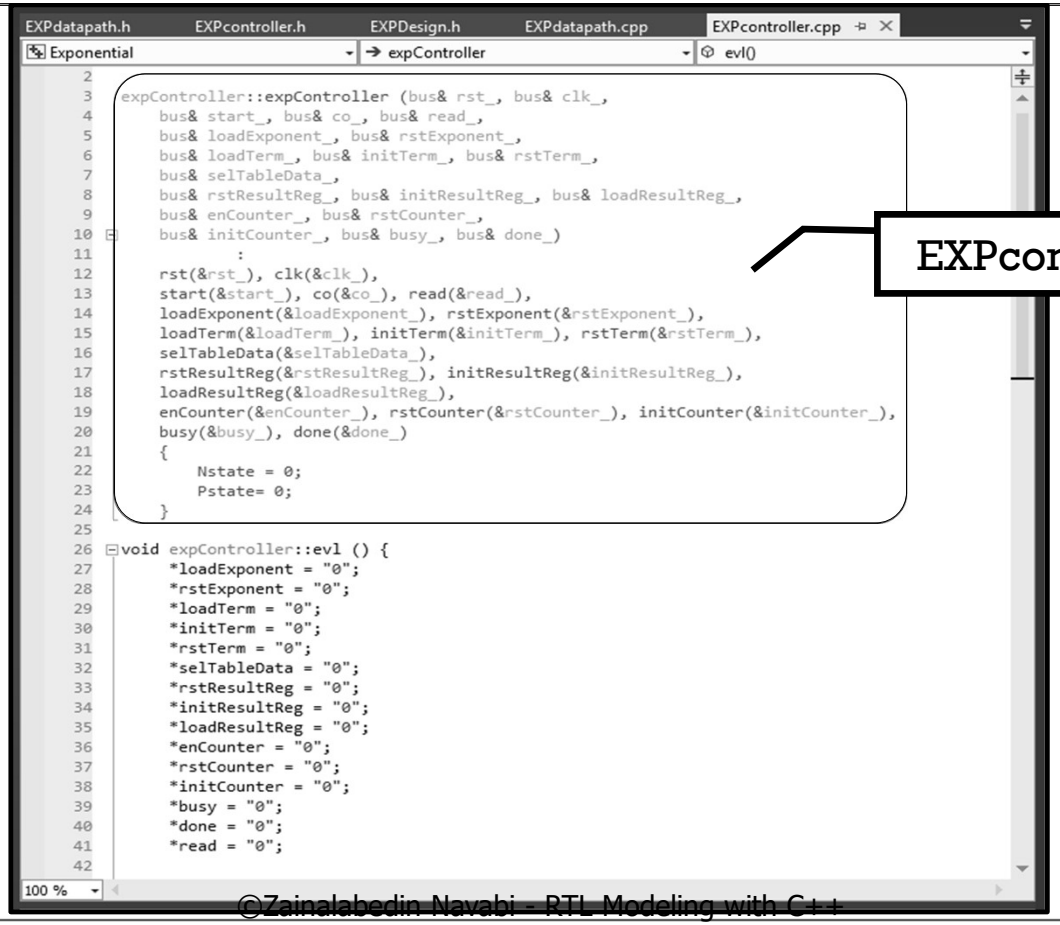
```
1  #include "classVectorPrimitives.h"
2  #include <string>
3  using namespace std;
4
5  class expController{
6       bus *rst, *clk;
7       bus *start, *co, *read;
8       bus *loadExponent, *rstExponent, *loadTerm;
9       bus *initTerm, *rstTerm;
10      bus *selTableData;
11      bus *rstResultReg, *initResultReg, *loadResultReg;
12      bus *enCounter, *rstCounter, *initCounter;
13      bus *busy, *done;
14      int Nstate, Pstate;
15      public:
16          expController(bus& rst, bus& clk,
17              bus& start, bus& co, bus &read,
18              bus& loadExponent, bus& rstExponent,
19              bus& loadTerm, bus& initTerm, bus& rstTerm,
20              bus& selTableData,
21              bus& rstResultReg, bus& initResultReg, bus& loadResultReg,
22              bus& enCounter, bus& rstCounter, bus& initCounter,
23              bus& busy, bus& done);
24          ~expController();
25          void evl ();
26 };
27
28
```

100 %

EXPcontroller.h

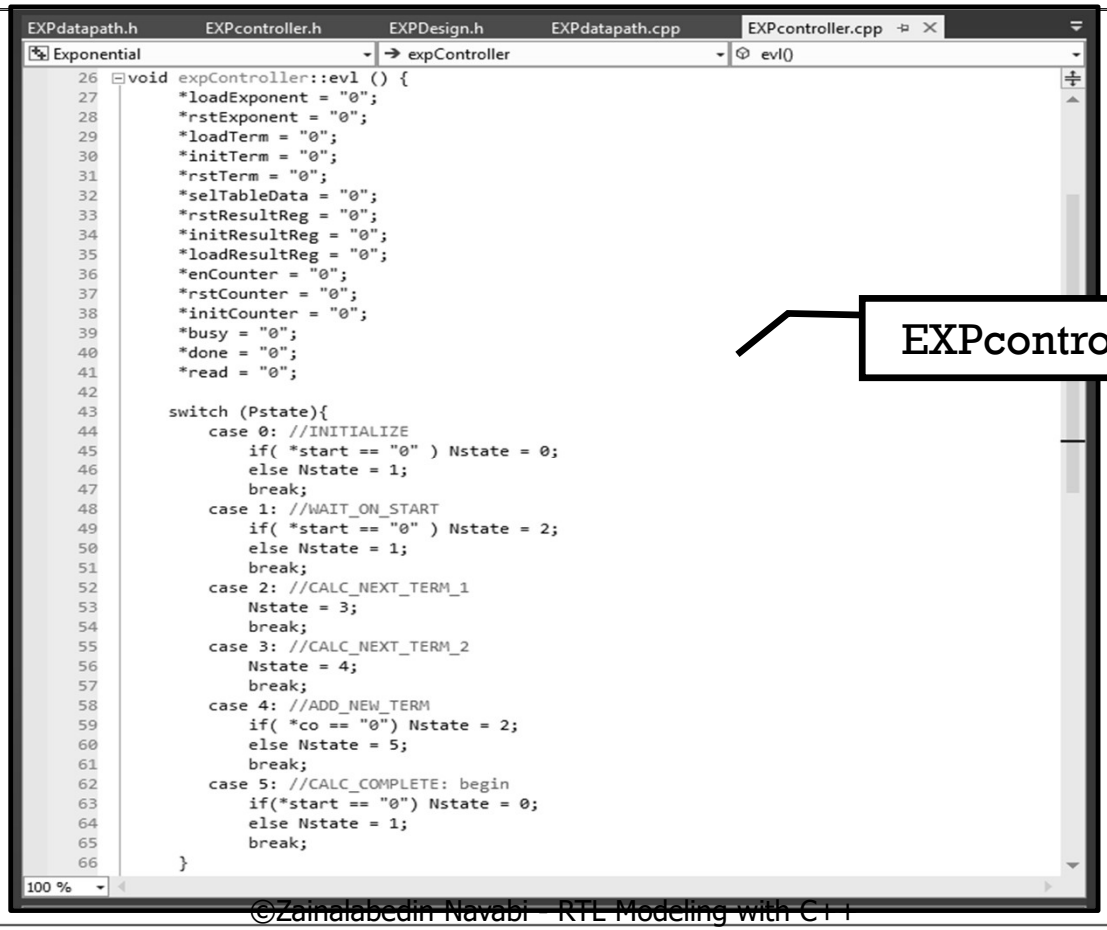


Exponential Circuit – Controller




```
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
```

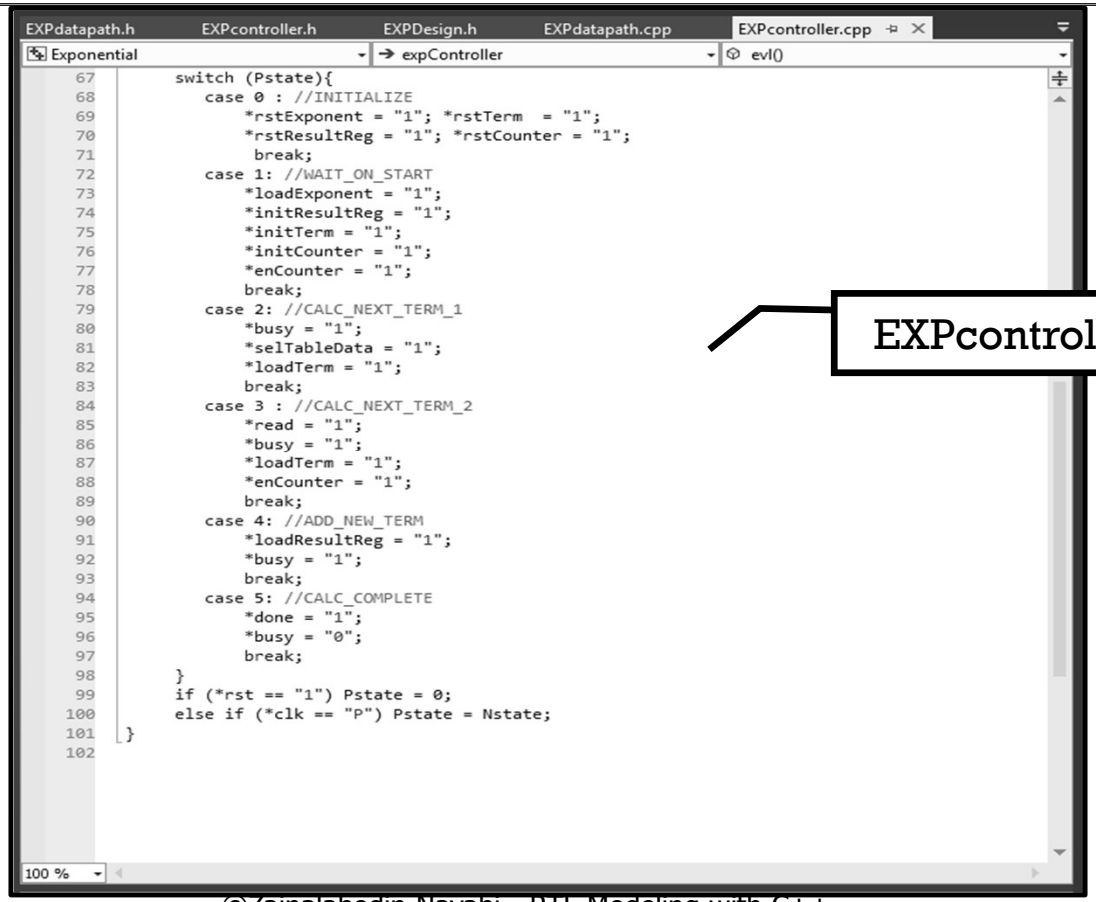
Exponential Circuit – Controller




```
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       }
```

EXPcontroller.cpp

Exponential Circuit – Controller



EXPcontroller.cpp

Exponential Circuit – Design

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

Exponential | (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     public:
23         expDesign ( bus &clk, bus &rst, bus &start, bus &x,
24                     bus &result, bus &busy, bus &done);
25         ~expDesign();
26         void evl();
27         void initialize (const string& filename);
28 };
29
```

100 %

EXPDesign.h

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
```

100 %

EXPDesign.cpp

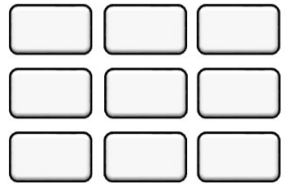
Exponential Circuit – Testbench

EXPdesignTB.cpp

```
#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^(0." << 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

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