Chapter 3

RT Level Modeling with C++

Zainalabedin Navabi

Slides prepared by: Hanieh Hashemi

RT Level Modeling with C/C++

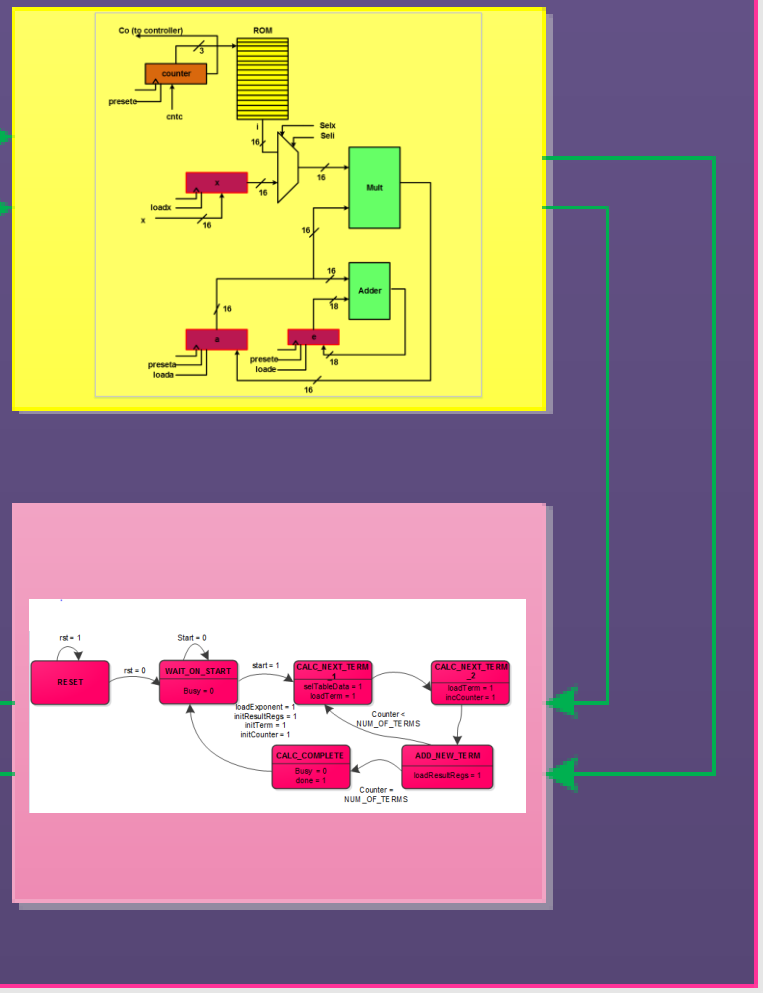
- RTL Principles
  - Elements of datapath
  - Elements of control unit
- Bus Communications
- Utility functions
- Bus operations
  - Array Attributes
  - Logical Operations
  - Adding Operations
  - Relational Operations
  - IO Operations
- Basic Elements of RTL
  - Combinational Elements
  - Registers and Counters
    - Functional register
    - dRegister
    - dRegisterE
    - dRegisterRaE
    - upCounter
    - upCounterRaE
  - Memory Structure
  - Controller FSM
  - Controller 11011
- RTL design example 1: LRU
  - LRU structure
  - LRU Modeling
    - Controller
    - Ordered instantiation
- RTL design example 2: Exponential Circuit
  - Exponential Circuit Datapath
  - Exponential Circuit Controller

RT Level Modeling with C/C++

- RTL Principles
  - Elements of datapath
  - Elements of control unit
- Bus Communications
- Utility functions
- Bus operations
  - Array Attributes
  - Logical Operations
  - Adding Operations
  - Relational Operations
  - IO Operations
- Basic Elements of RTL
  - Combinational Elements
  - Registers and Counters
    - Functional register
    - dRegister
    - dRegisterE
    - dRegisterRaE
    - upCounter
    - upCounterRaE
  - Memory Structure
  - Controller FSM
  - Controller 11011
- RTL design example1: LRU
  - LRU structure
  - LRU Modeling
    - Controller
    - Ordered instantiation
- RTL design example 2: Exponential Circuit
  - Exponential Circuit Datapath
  - Exponential Circuit Controller

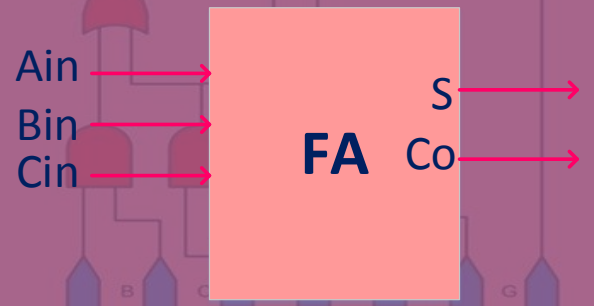
RTL Principles

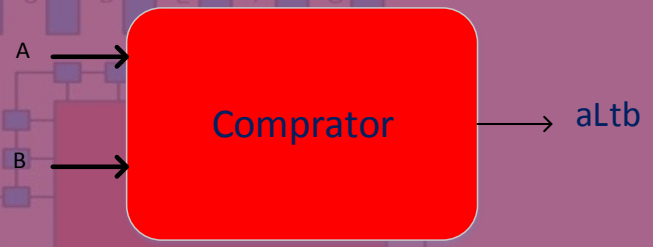
Elements of an RT level component



Elements of Datapath

- Combinational Components
  - Adders
  - Comparators
  - Multiplexers
  - ALUs

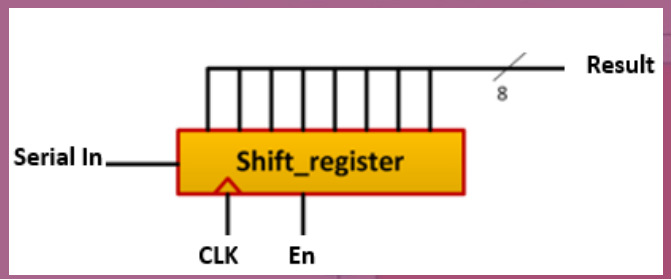


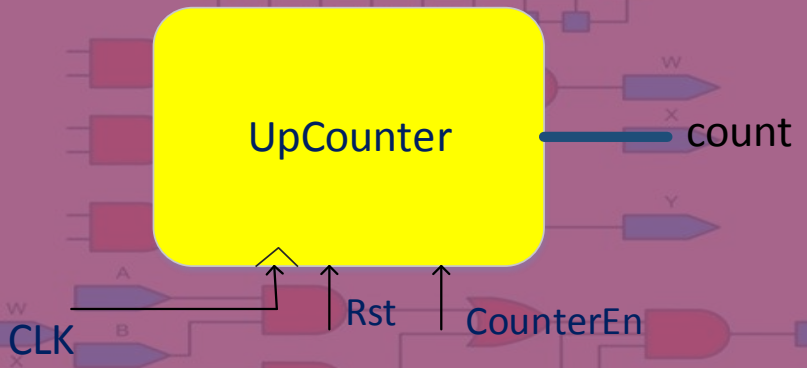


- Logical Operations
  - Vector based
  - Scalar operations
  - Mixed

Elements of Datapath

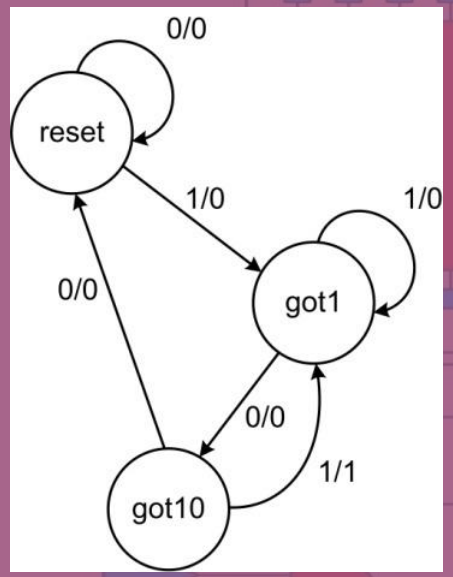
- RTL internal buses
  - Unconstrained
  - Represent multi-value logic system
- Sequential component
  - Registers
  - Registers with some functionality
  - Register files

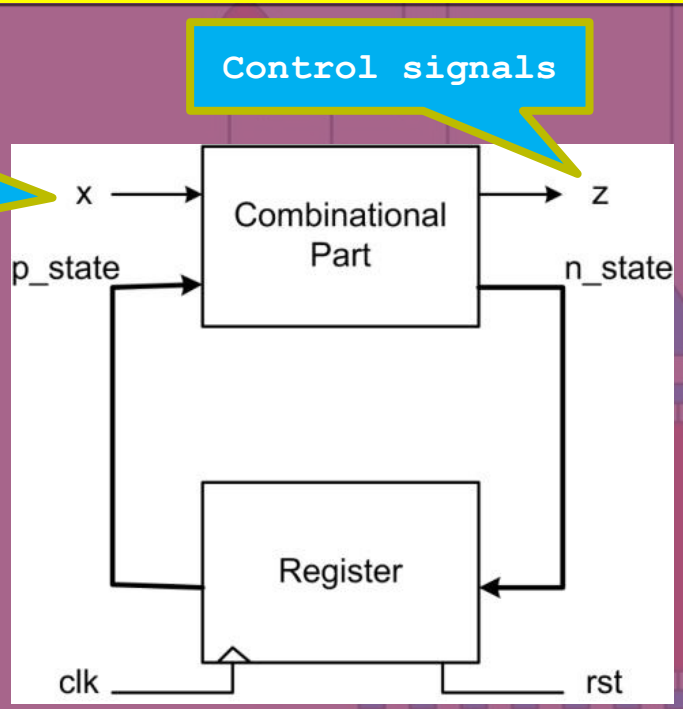




Elements of Control Unit

- One or more state machines
  - Huffman style



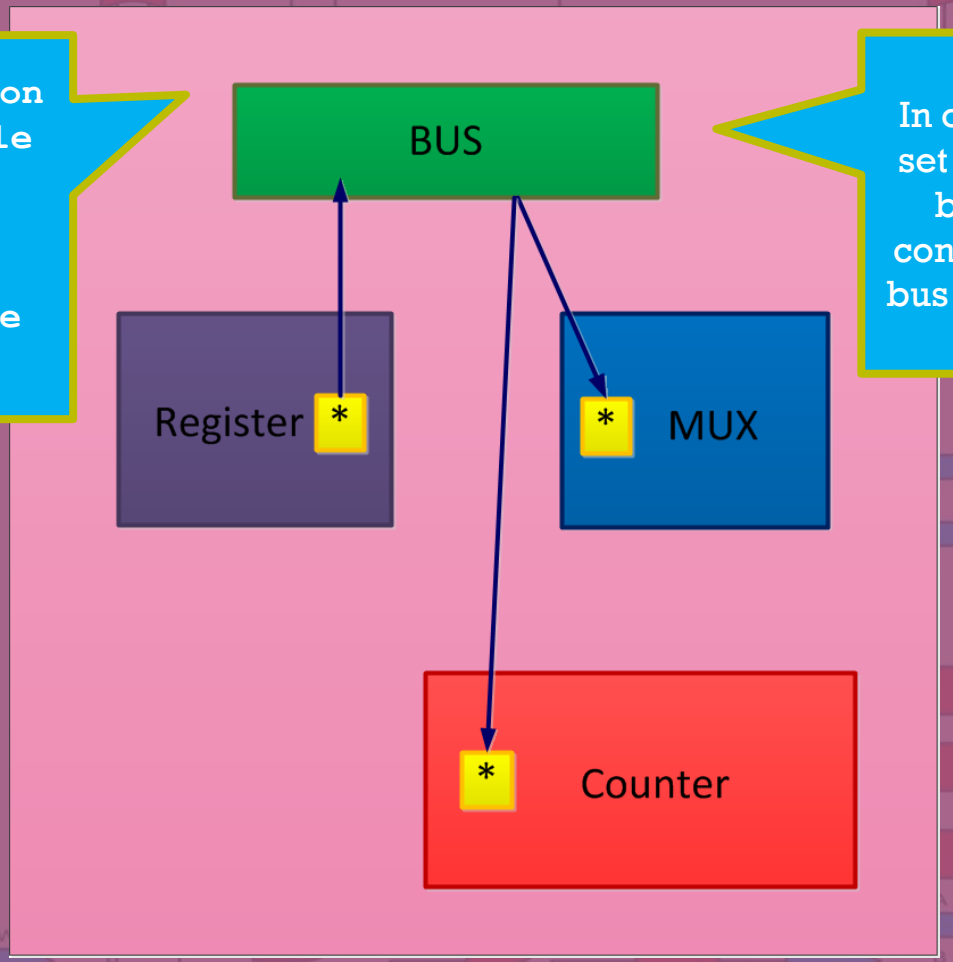


RT Level Modeling with C/C++

- RTL Principles
  - Elements of datapath
  - Elements of control unit
- Bus Communications
- Utility functions
- Bus operations
  - Array Attributes
  - Logical Operations
  - Adding Operations
  - Relational Operations
  - IO Operations
- Basic Elements of RTL
  - Combinational Elements
  - Registers and Counters
    - Functional register
    - dRegister
    - dRegisterE
    - dRegisterRaE
    - upCounter
    - upCounterRaE
  - Memory Structure
  - Controller FSM
  - Controller 11011
- RTL design example: LRU
  - LRU structure
  - LRU Modeling
    - Controller
    - Ordered instantiation
- RTL design example 2: Exponential Circuit
  - Exponential Circuit Datapath
  - Exponential Circuit Controller

Bus Communications

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



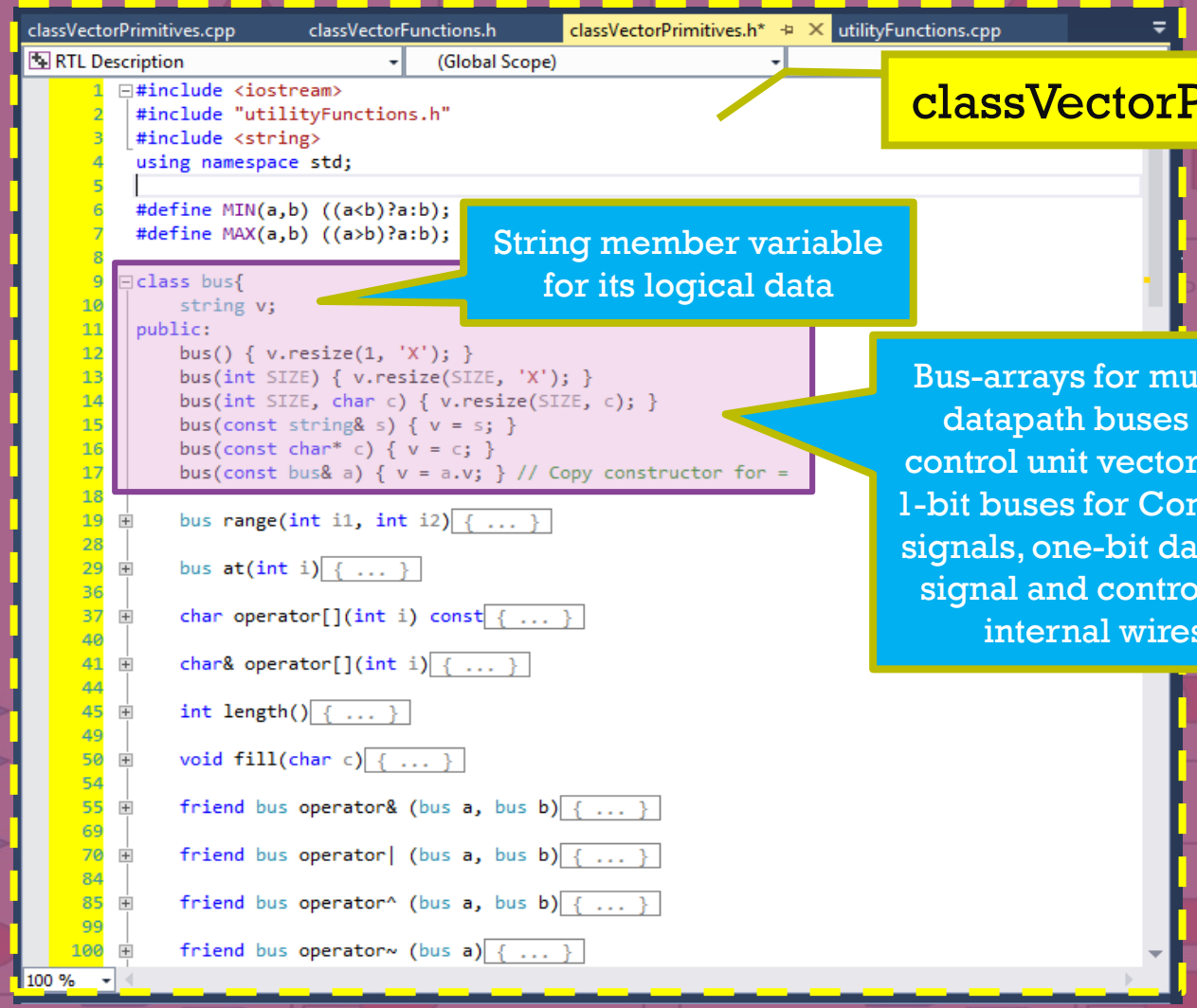
In order to have consistent set of communication lines between datapath and controller, the same type of bus should be used for both

Bus Communications

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

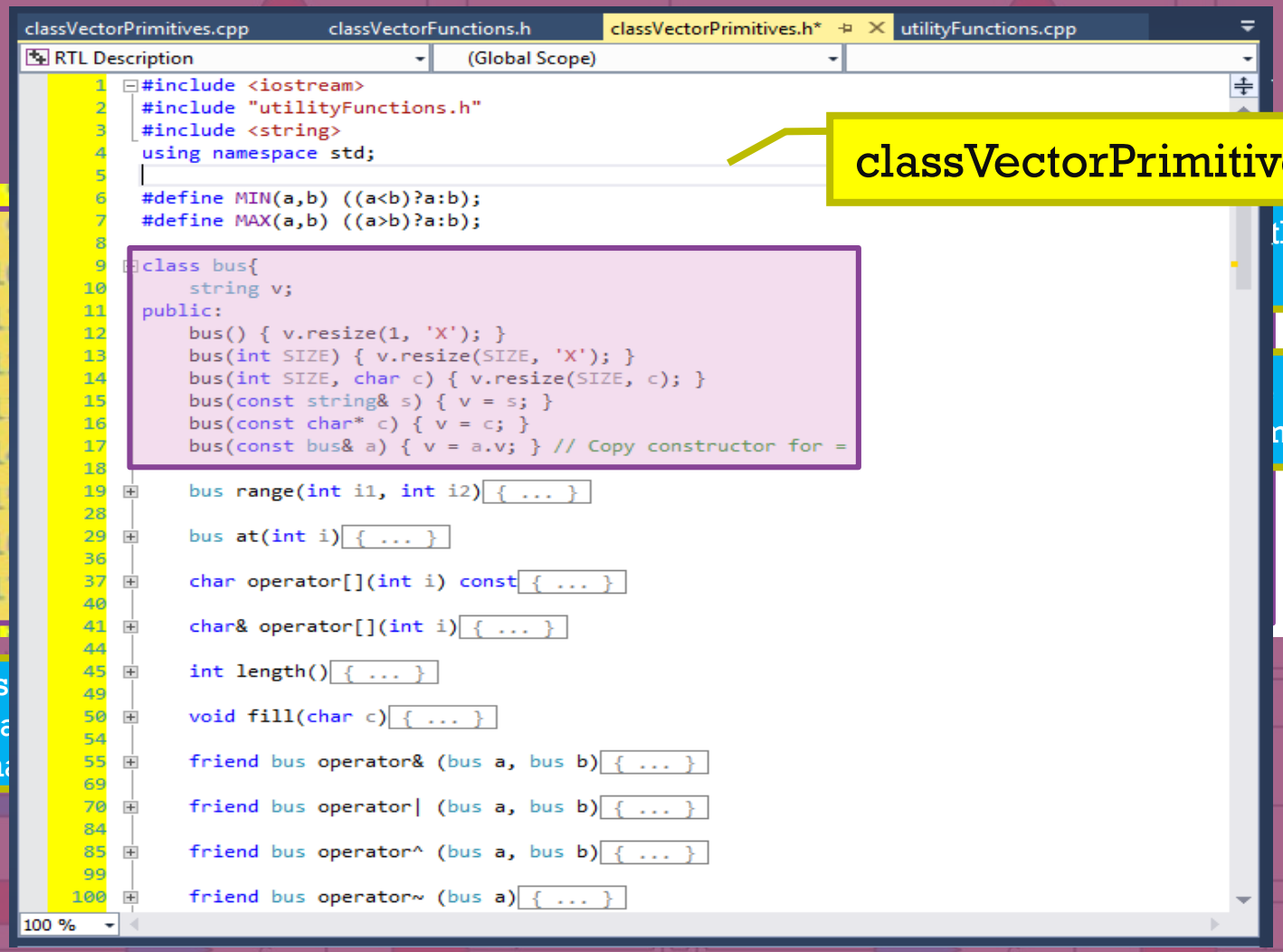
```
Abus = Rbus + Bbus
Abus = Rbus & Bbus
Abus = Rbus | Bbus
```

Bus Communications



Bus Communications

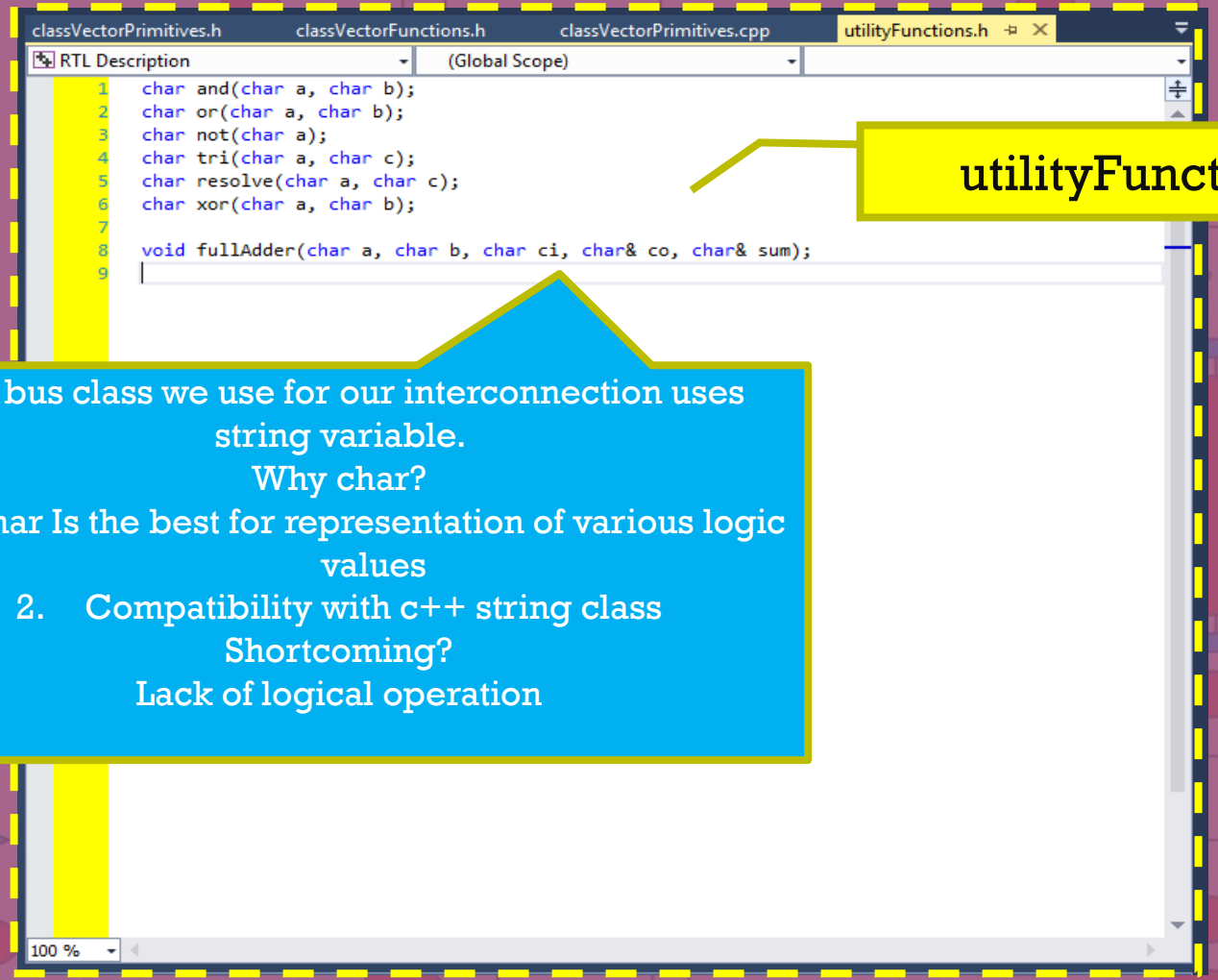
classVectorPrimitives.h



RT Level Modeling with C/C++

- RTL Principles
  - Elements of datapath
  - Elements of control unit
- Bus Communications
- Utility functions
- Bus operations
  - Array Attributes
  - Logical Operations
  - Adding Operations
  - Relational Operations
  - IO Operations
- Basic Elements of RTL
  - Combinational Elements
  - Registers and Counters
    - Functional register
    - dRegister
    - dRegisterE
    - dRegisterRaE
    - upCounter
    - upCounterRaE
  - Memory Structure
  - Controller FSM
  - Controller 11011
- RTL design example: LRU
  - LRU structure
  - LRU Modeling
    - Controller
    - Ordered instantiation
- RTL design example 2: Exponential Circuit
  - Exponential Circuit Datapath
  - Exponential Circuit Controller

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

classVectorPrimitives.cpp | classVectorFunctions.h | classVectorPrimitives.h* | utilityFunctions.cpp

RTL Description | (Global Scope)

```
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
```

100 %

utilityFunctions.cpp

Full adder implementation using primitives

RT Level Modeling with C/C++

- RTL Principles
  - Elements of datapath
  - Elements of control unit
- Bus Communications
- Utility functions
- Bus operations
  - Array Attributes
  - Logical Operations
  - Adding Operations
  - Relational Operations
  - IO Operations
- Basic Elements of RTL
  - Combinational Elements
  - Registers and Counters
    - Functional register
    - dRegister
    - dRegisterE
    - dRegisterRaE
    - upCounter
    - upCounterRaE
  - Memory Structure
  - Controller FSM
  - Controller 11011
- RTL design example: LRU
  - LRU structure
  - LRU Modeling
    - Controller
    - Ordered instantiation
- RTL design example 2: Exponential Circuit
  - Exponential Circuit Datapath
  - Exponential Circuit Controller

Array Attributes

classVectorPrimitives.h | 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 { ... }
40
41  char& operator[](int i) { ... }
44
45  int length() { ... }
49
50  void fill(char c) { ... }
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)
113 {
114     int aSize = a.v.length();
```

100 %

Bus slicing

classVectorPrimitives.h

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

Array Attributes

classVectorPrimitives.h | classVectorFunctions.h | utilityFunctions.h | utilityFunctions.cpp

RTL Description | (Global Scope)

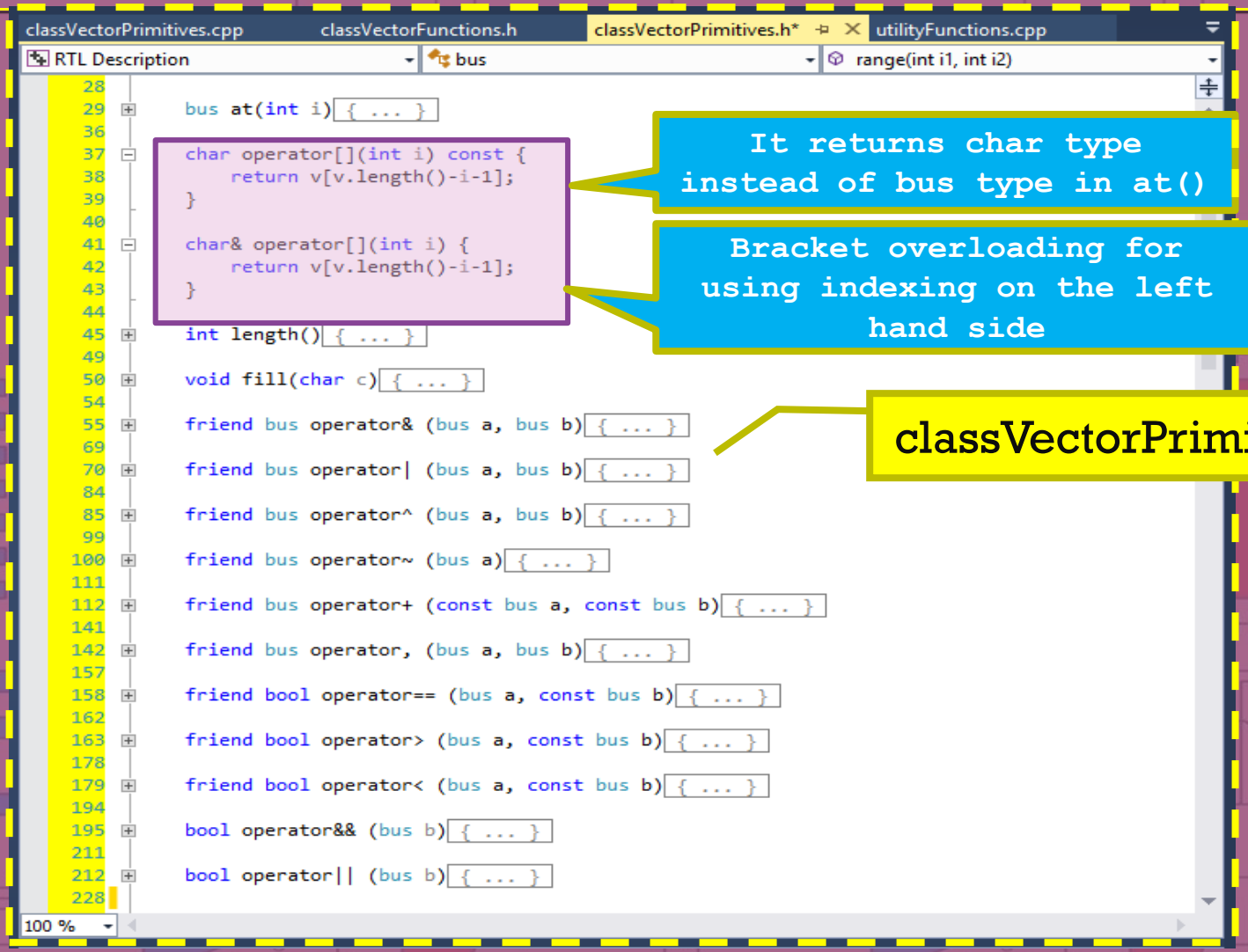
```
19   bus range(int i1, int i2)
20   {
21       int left = MAX(i1, i2);
22       int rite = MIN(i1, i2);
23       bus slice(left-rite, 'X');
24       int vSize = v.length();
25       slice.v = v.substr(vSize - left, vSize - rite);
26       return slice;
27   }
28
29   bus at(int i)
30   {
31       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 { ... }
40
41   char& operator[](int i) { ... }
44
45   int length() { ... }
49
50   void fill(char c) { ... }
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)
113  {
114      int aSize = a.v.length();
```

100 %

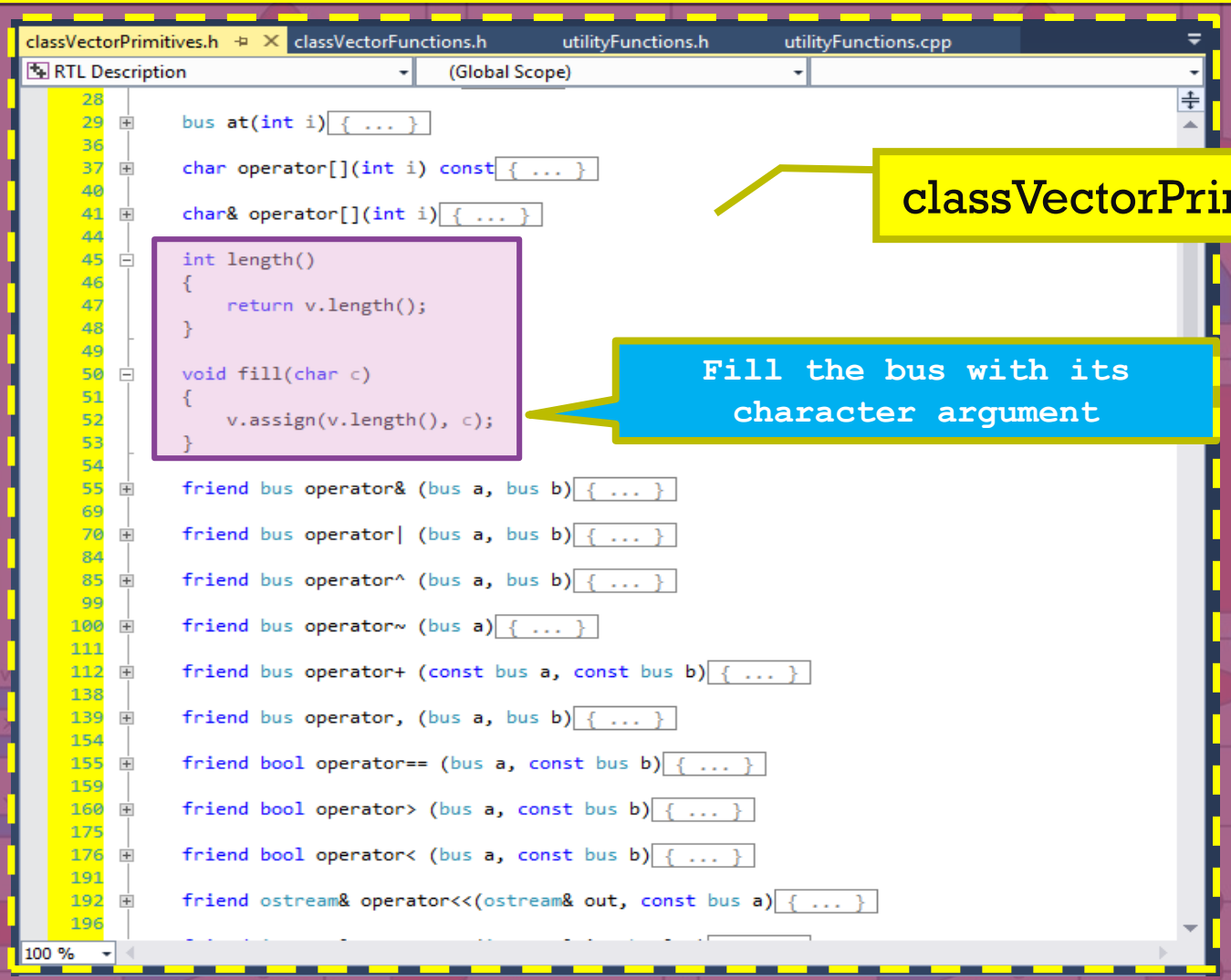
classVectorPrimitives.h

Bus indexing

Array Attributes



Array Attributes



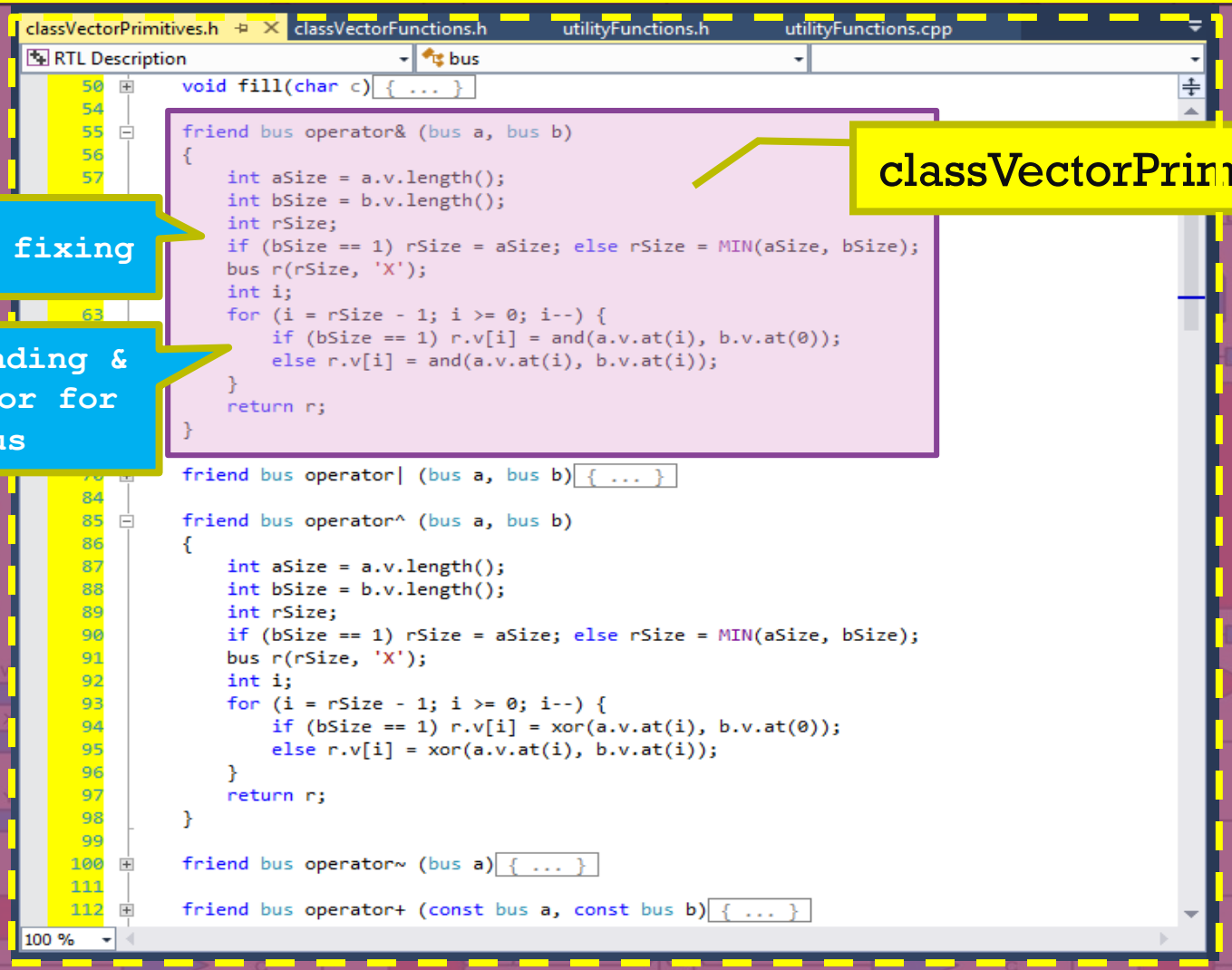
Logical Operations

Declared as friends inside the bus class

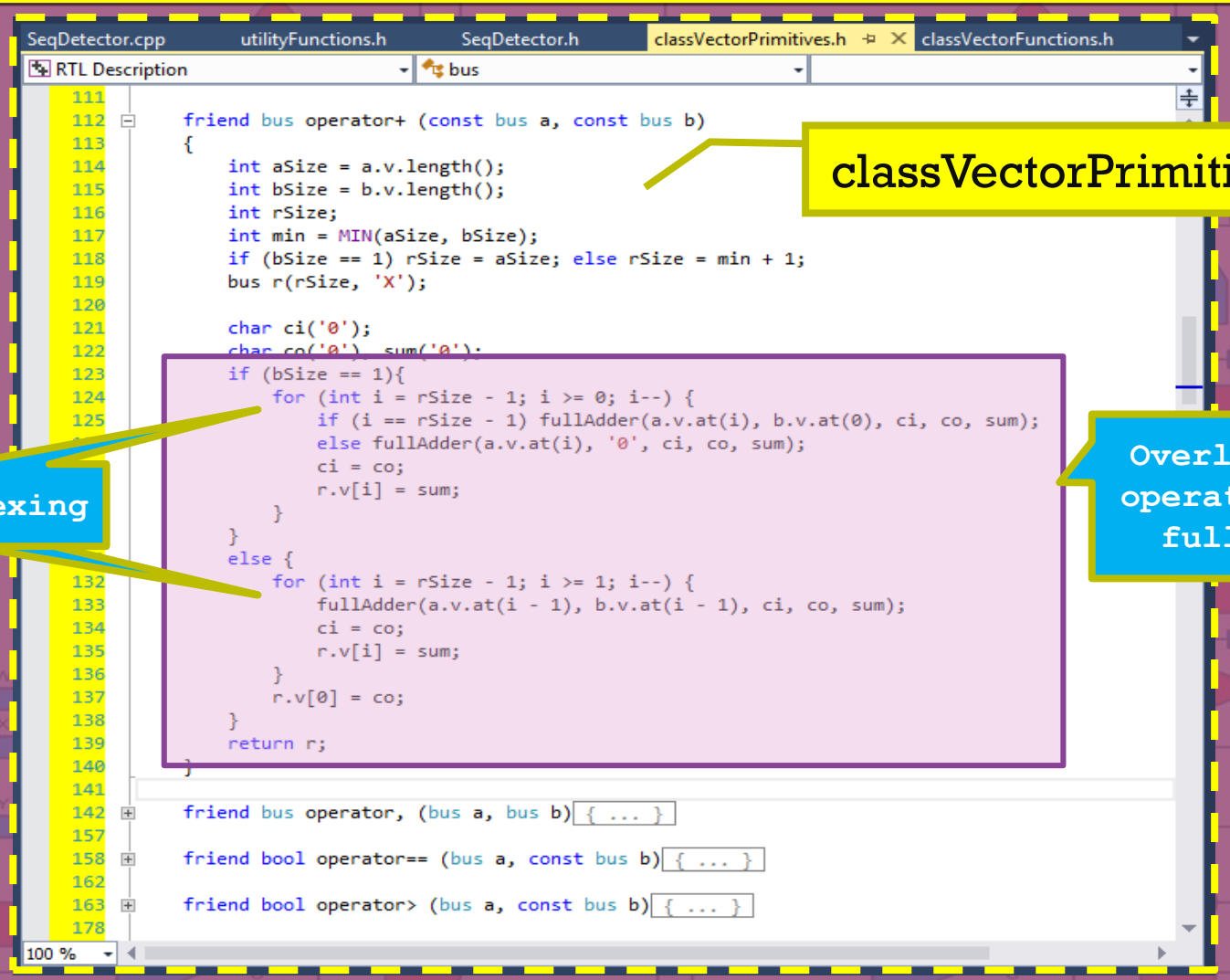
classVectorPrimitives.h

```
friend bus operator~ (bus a) { ... }
friend bus operator+ (const bus a, const bus b) { ... }
142 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
```

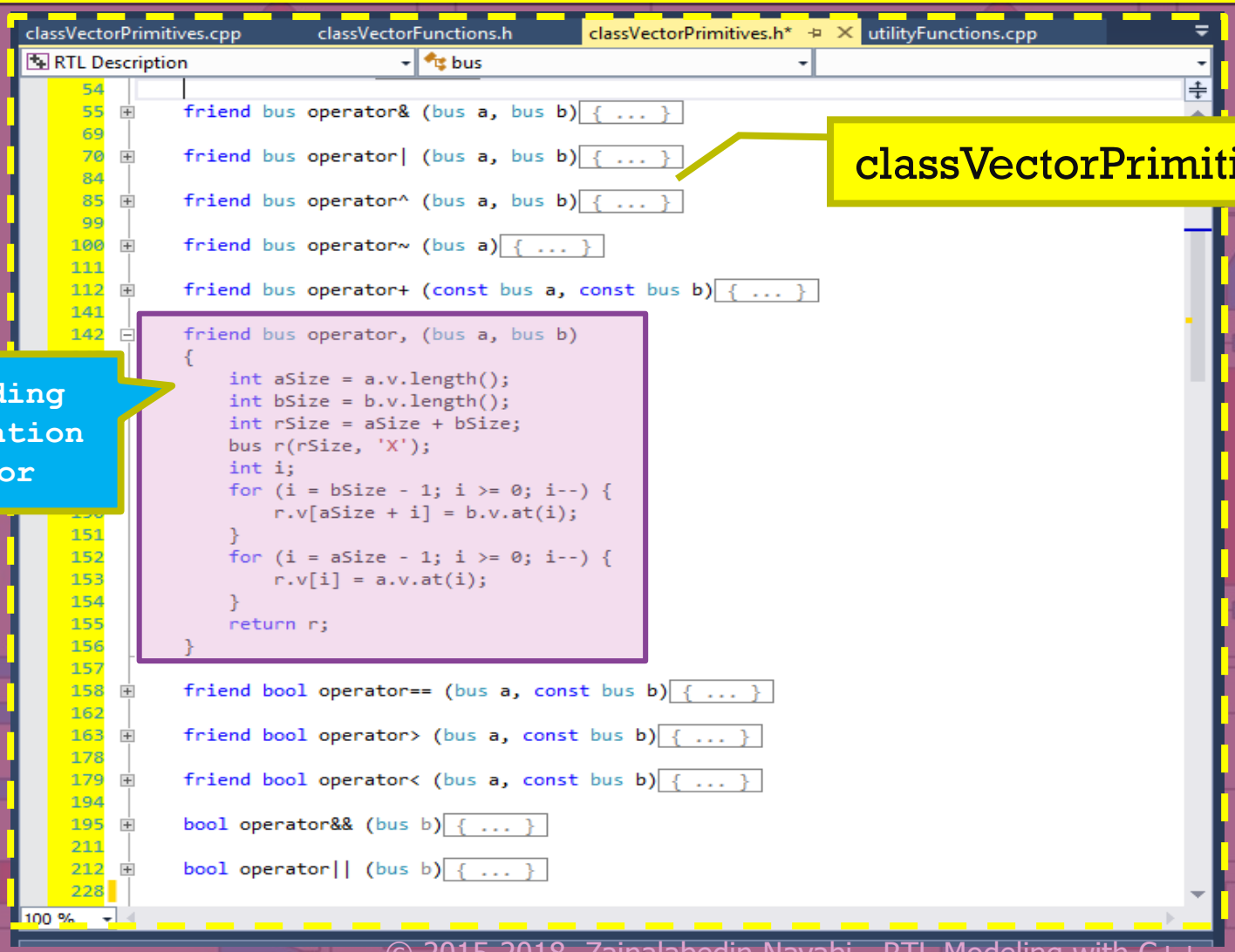
Logical Operations



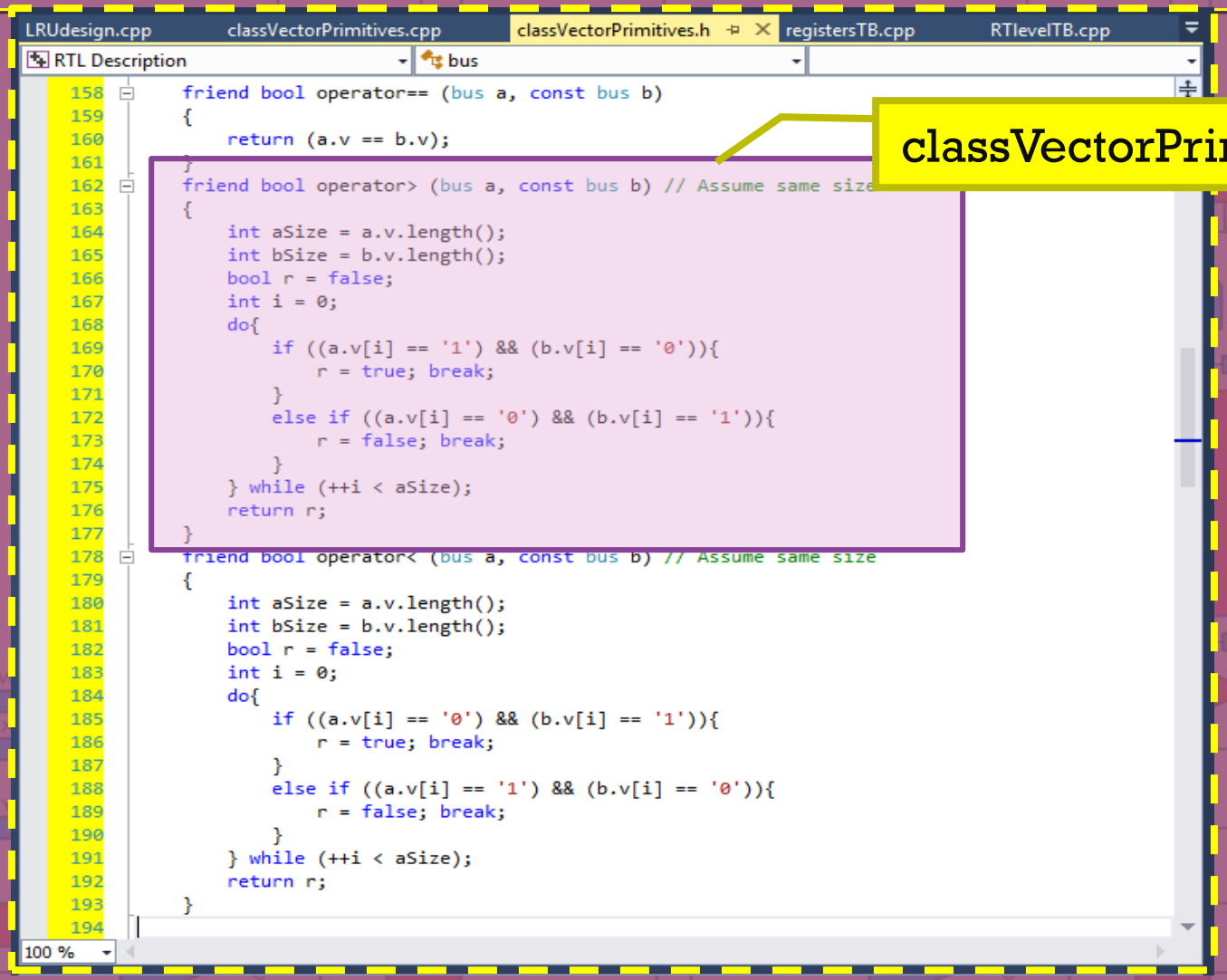
Adding Operations



Logical Operations



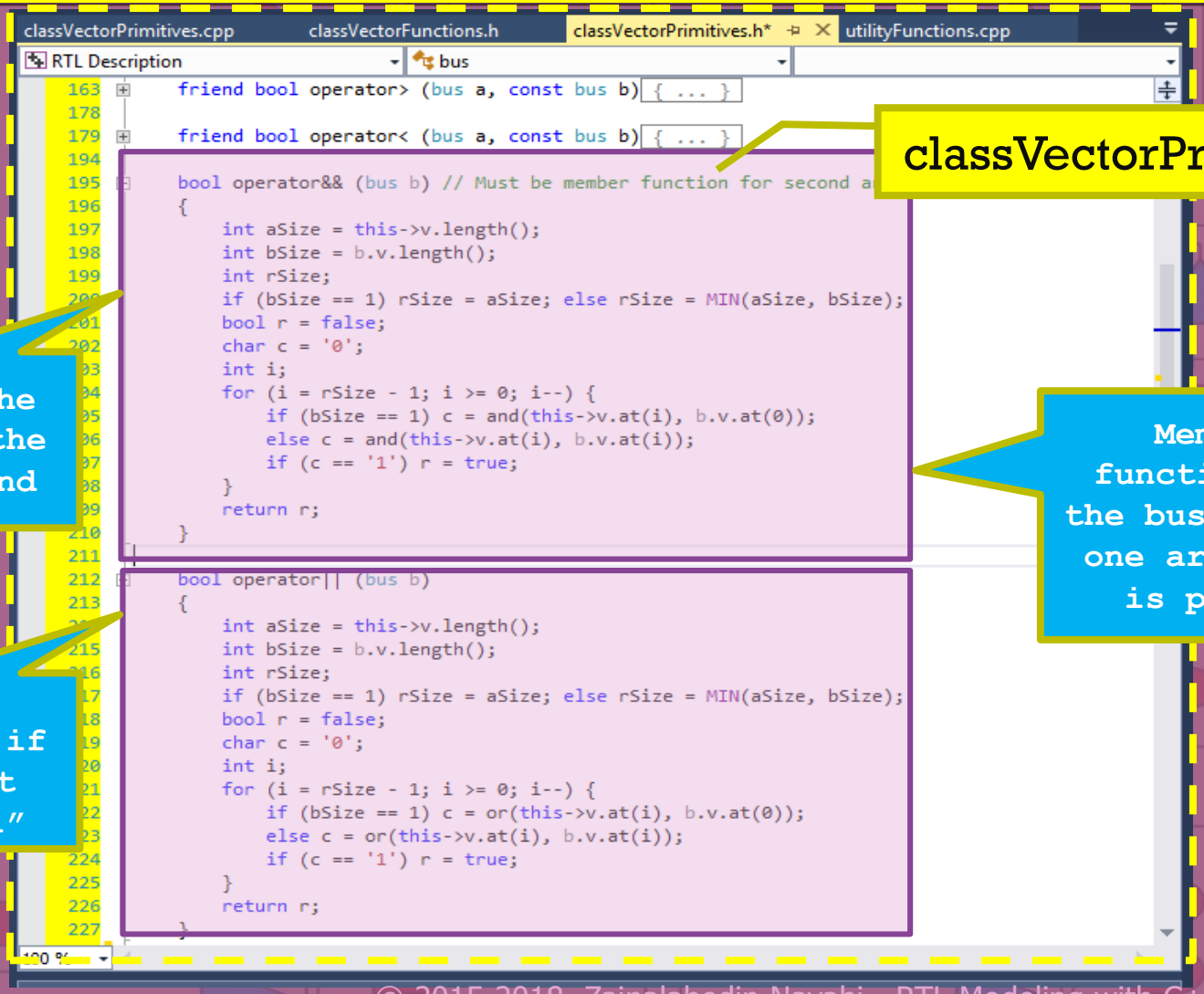
Rational Operations




```
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
```

classVectorPrimitives.h

Logical Operations



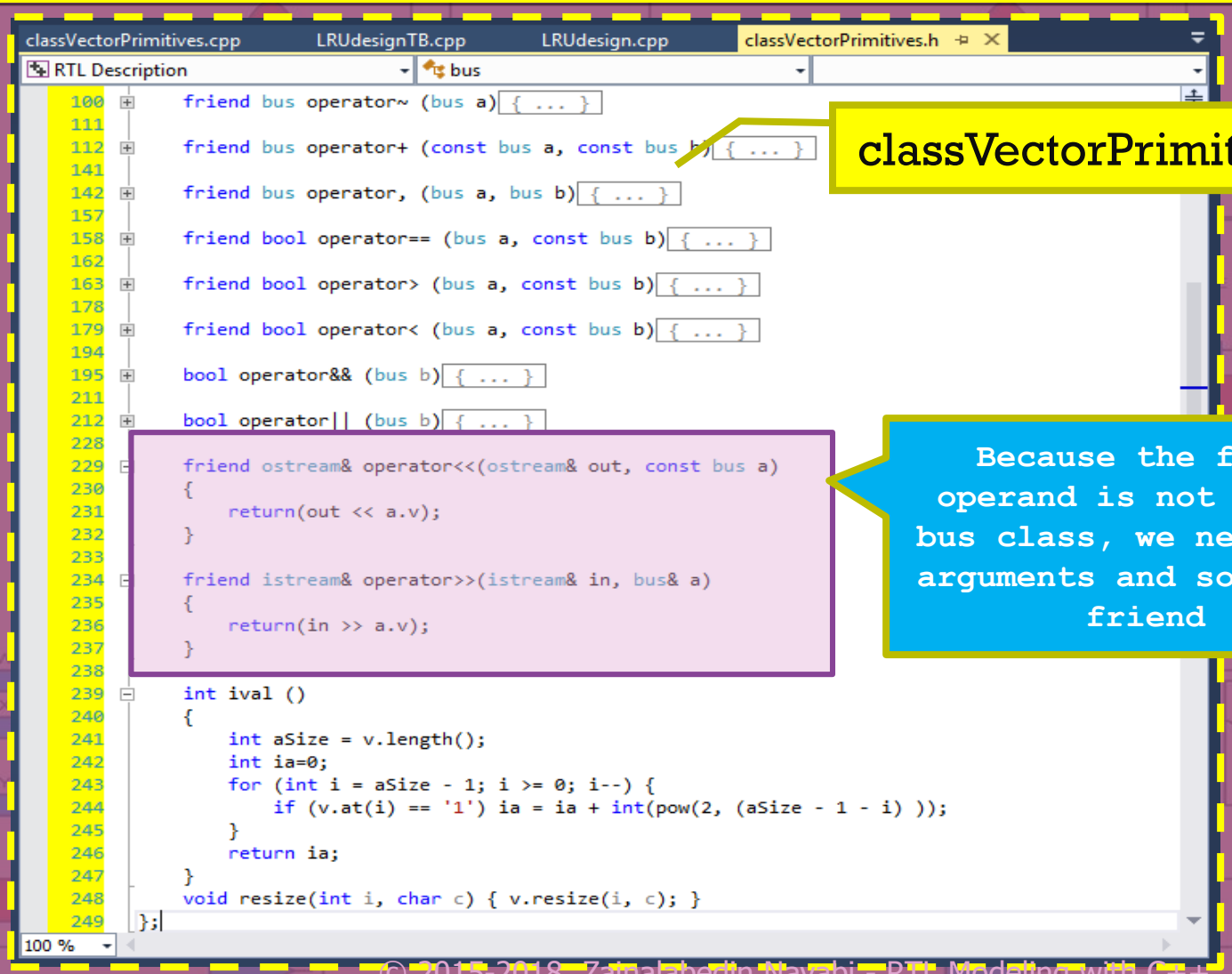
classVectorPrimitives.h

"this" is the pointer to the first operand

Member functions of the bus -> only one arguments is passed

Functions return true if contains at leas one "1"

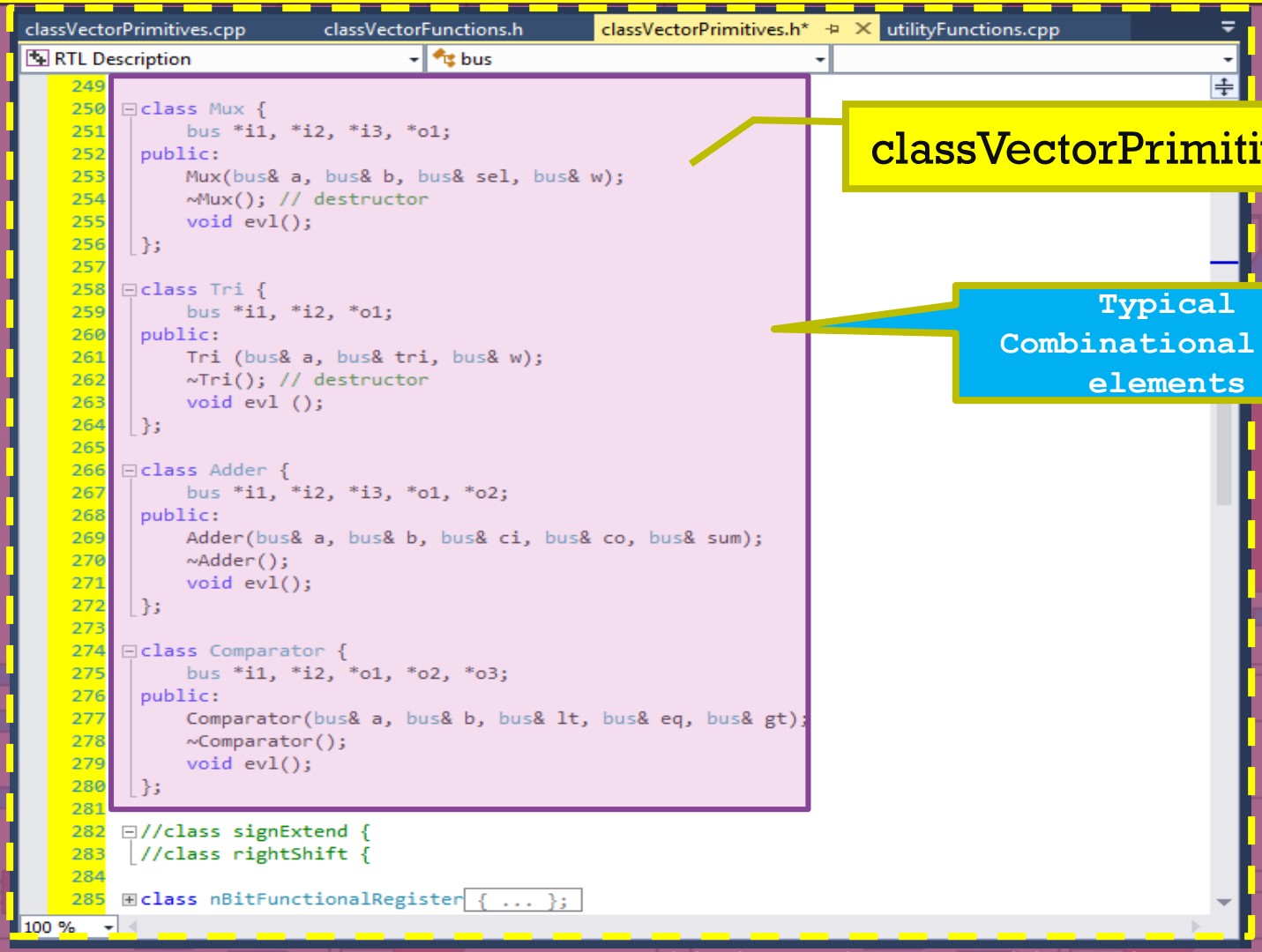
IO Operations



RT Level Modeling with C/C++

- RTL Principles
  - Elements of datapath
  - Elements of control unit
- Bus Communications
- Utility functions
- Bus operations
  - Array Attributes
  - Logical Operations
  - Adding Operations
  - Relational Operations
  - IO Operations
- Basic Elements of RTL
  - Combinational Elements
  - Registers and Counters
    - Functional register
    - dRegister
    - dRegisterE
    - dRegisterRaE
    - upCounter
    - upCounterRaE
  - Memory Structure
  - Controller FSM
  - Controller 11011
- RTL design example: LRU
  - LRU structure
  - LRU Modeling
    - Controller
    - Ordered instantiation
- RTL design example 2: Exponential Circuit
  - Exponential Circuit Datapath
  - Exponential Circuit Controller

Combinational Elements



classVectorPrimitives.h

Typical Combinational RTL elements

Combinational Elements

classVectorPrimitives.cpp

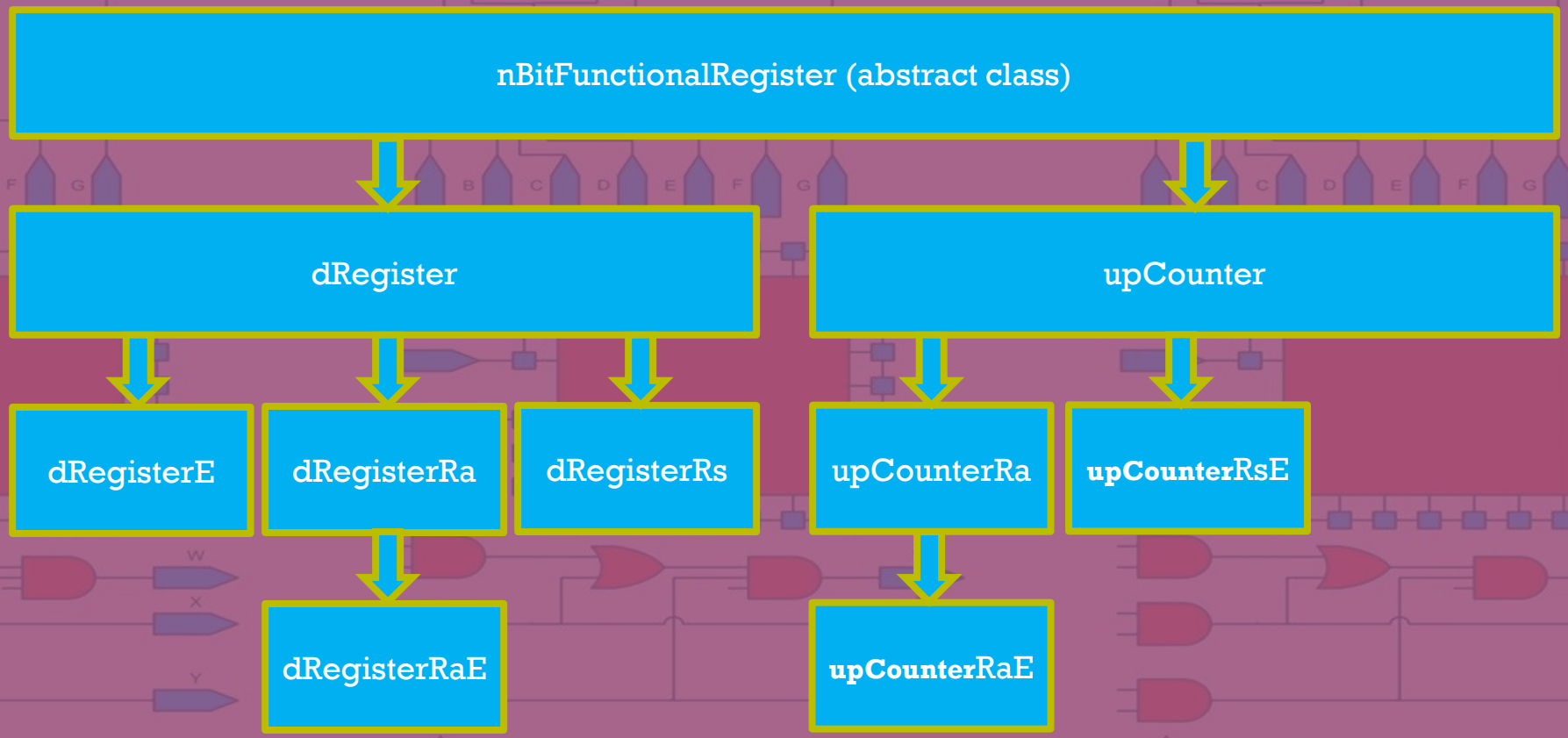
```
#include "classVectorPrimitives.h"
#include <iostream>
#include <fstream>
#include <string>
#include <math.h>
using namespace std;

Mux::Mux (bus& a, bus& b, bus& sel, bus& w) : i1(&a), i2(&b), i3(&sel), o1(&w)
{
    o1->fill('X');
}
void Mux::evl () {
    if (*i3 == "0") *o1 = *i1; else *o1 = *i2;
}

Tri::Tri (bus& a, bus& tri, bus& w) : i1(&a), i2(&tri), o1(&w)
{
    o1->fill('X');
}
void Tri::evl () {
    if (*i2 == "1") *o1 = *i1; else o1->fill('Z');
}

Adder::Adder(bus& a, bus& b, bus& ci, bus& co, bus& sum) :
             i1(&a), i2(&b), i3(&ci), o1(&sum), o2(&co)
{
    o1->fill('X'); o2->fill('X');
}
void Adder::evl () {
    bus result(i1->length() + 1);
    result = *i1 + *i2 + *i3;
    int leftIndex = result.length() - 1;
    *o2 = result.at(leftIndex);
    *o1 = result.range(leftIndex, 0);
}
```

"+" operator is overloaded before

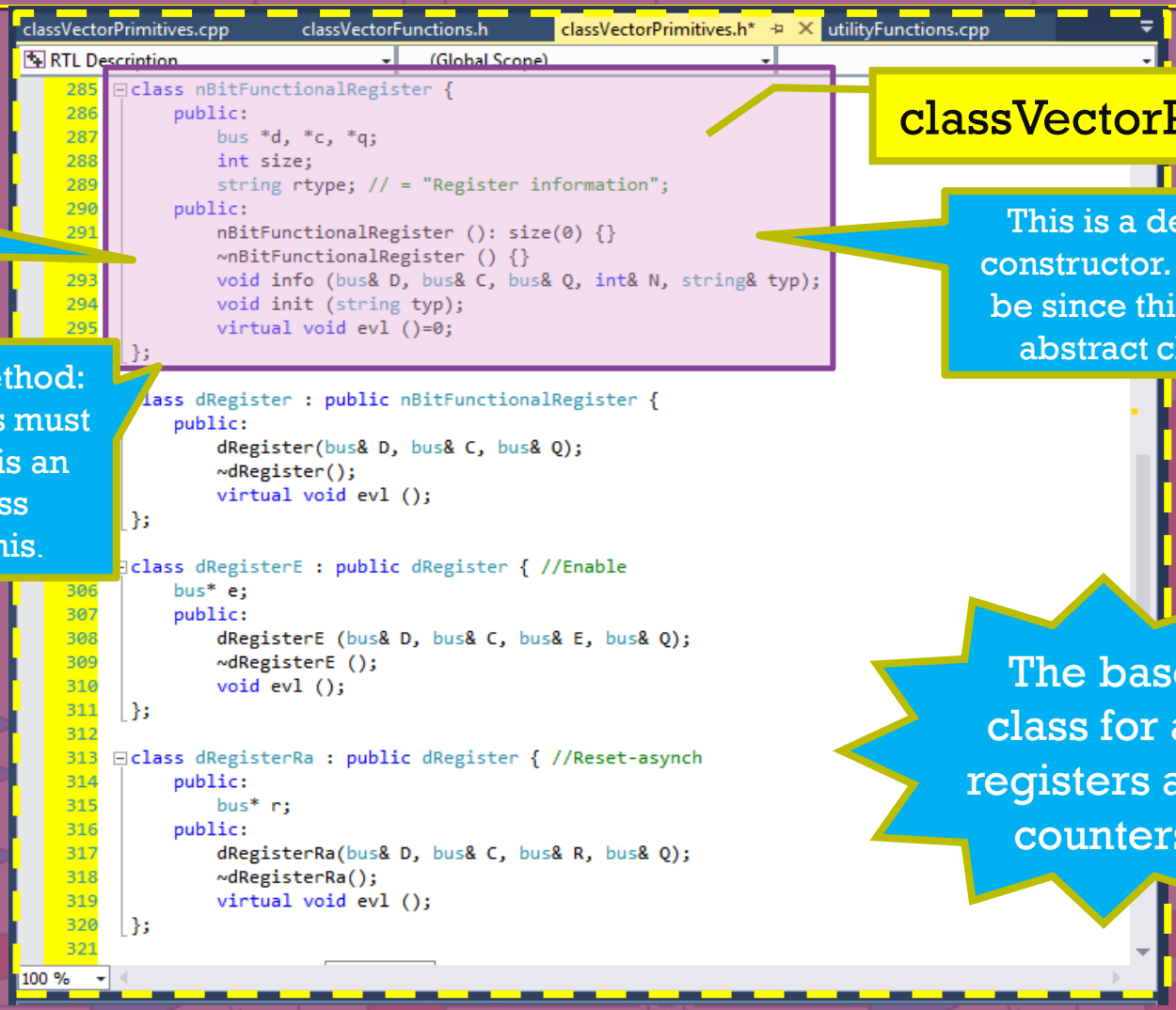
Registers and Counters

- Basic registers and counters are modeled in C/ C++
- Various functionalities
- Various clocking schemes
- Various resetting mechanisms
- Inheritance and Polymorphism is shown here

Registers and Counters



Functional Registers



classVectorPrimitives.h

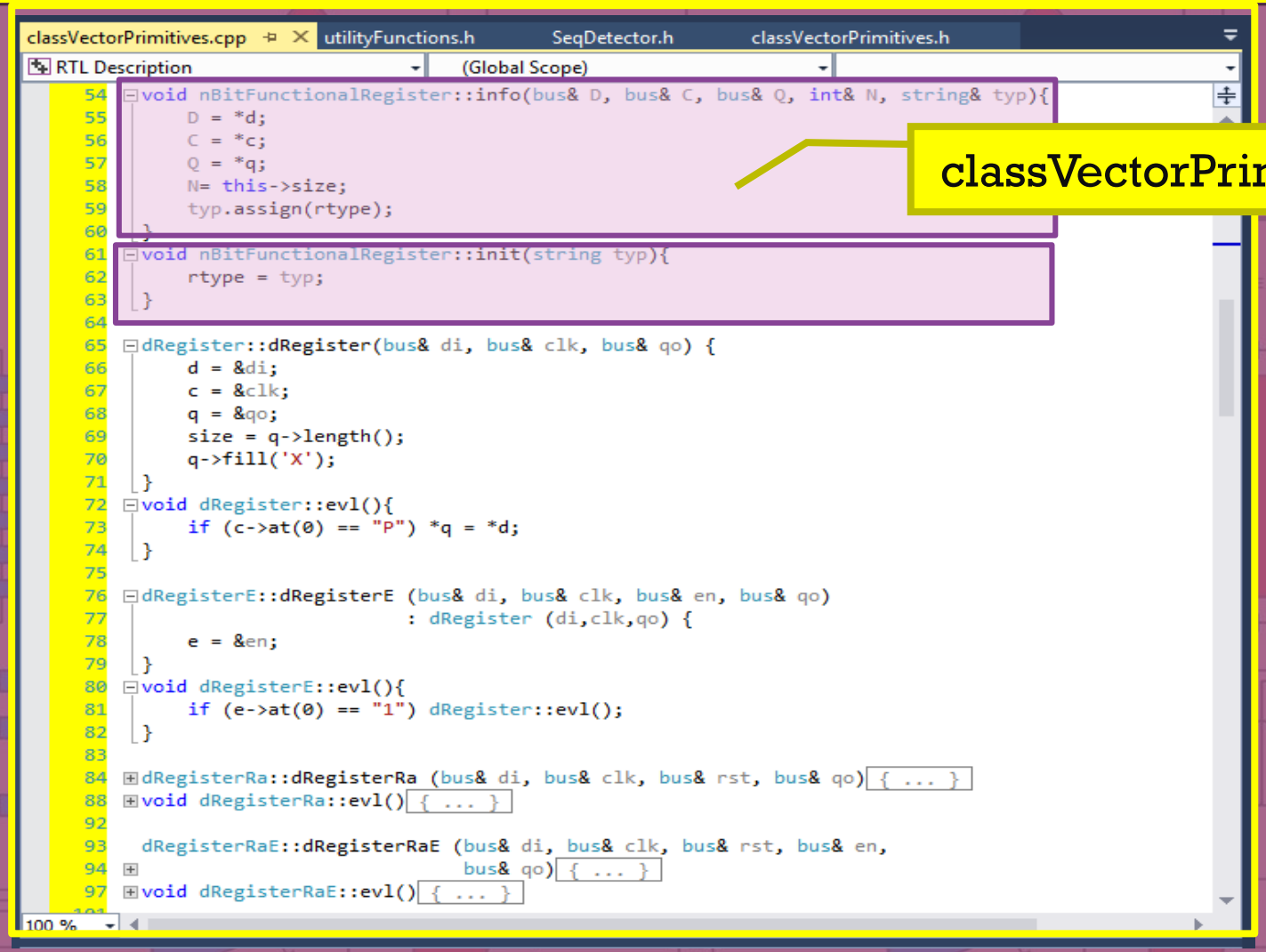
Initializing member variables.

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)

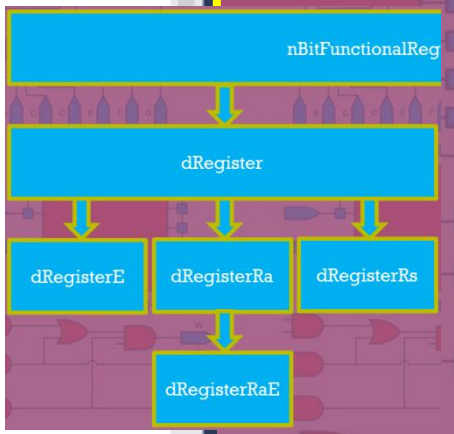
```
285 class nBitFunctionalRegister {
286     public:
287         bus *d, *c, *q;
288         int size;
289         string rtype; // = "Register information";
290     public:
291         nBitFunctionalRegister (): size(0) {}
292         ~nBitFunctionalRegister () {}
293         void info (bus& D, bus& 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.

Derived from dRegister



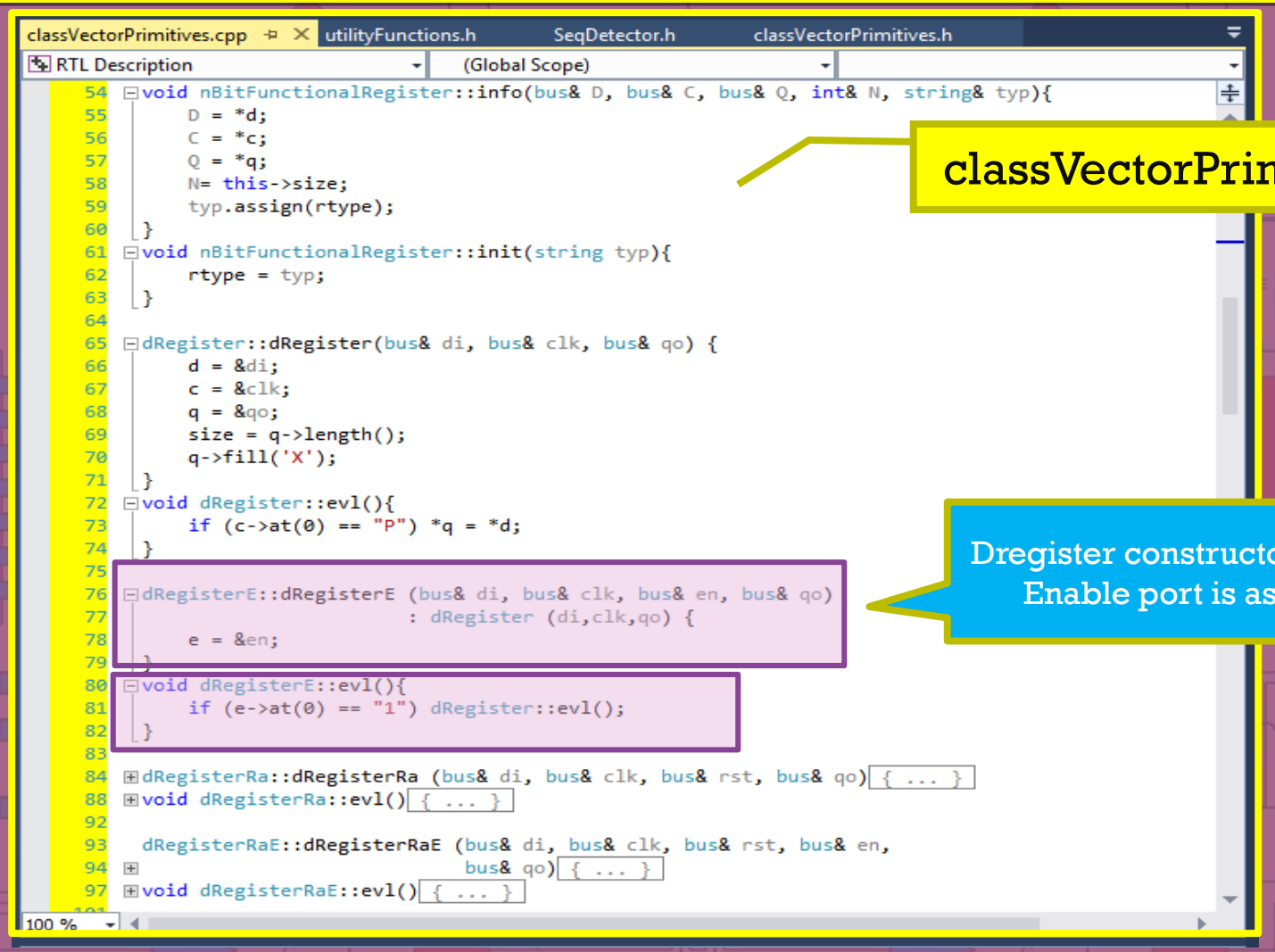
DRegister

classVectorPrimitives.cpp

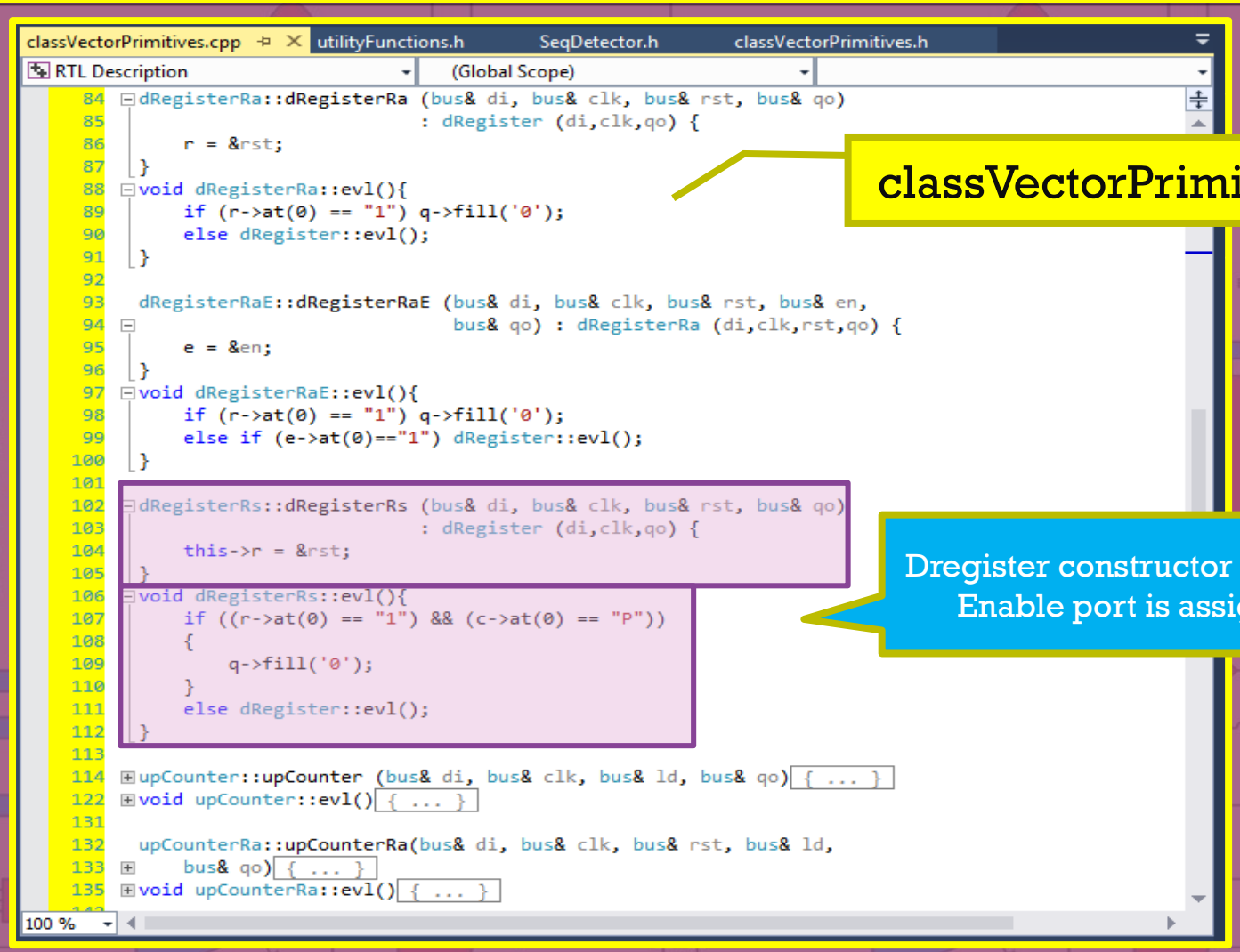
Assign register pointer
Set register size
Initialize the bus connected to the output

```
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(){ ... }
```

DRegisterE



DRegisterRS

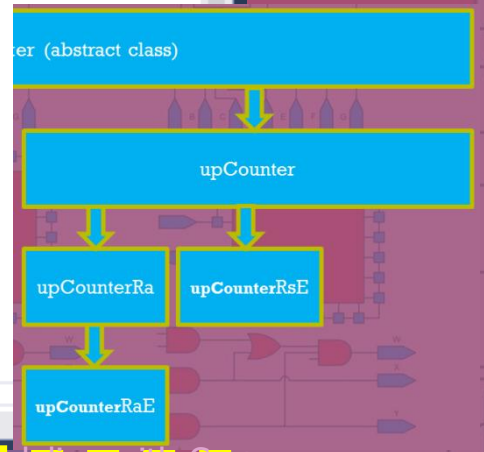


UpCounter

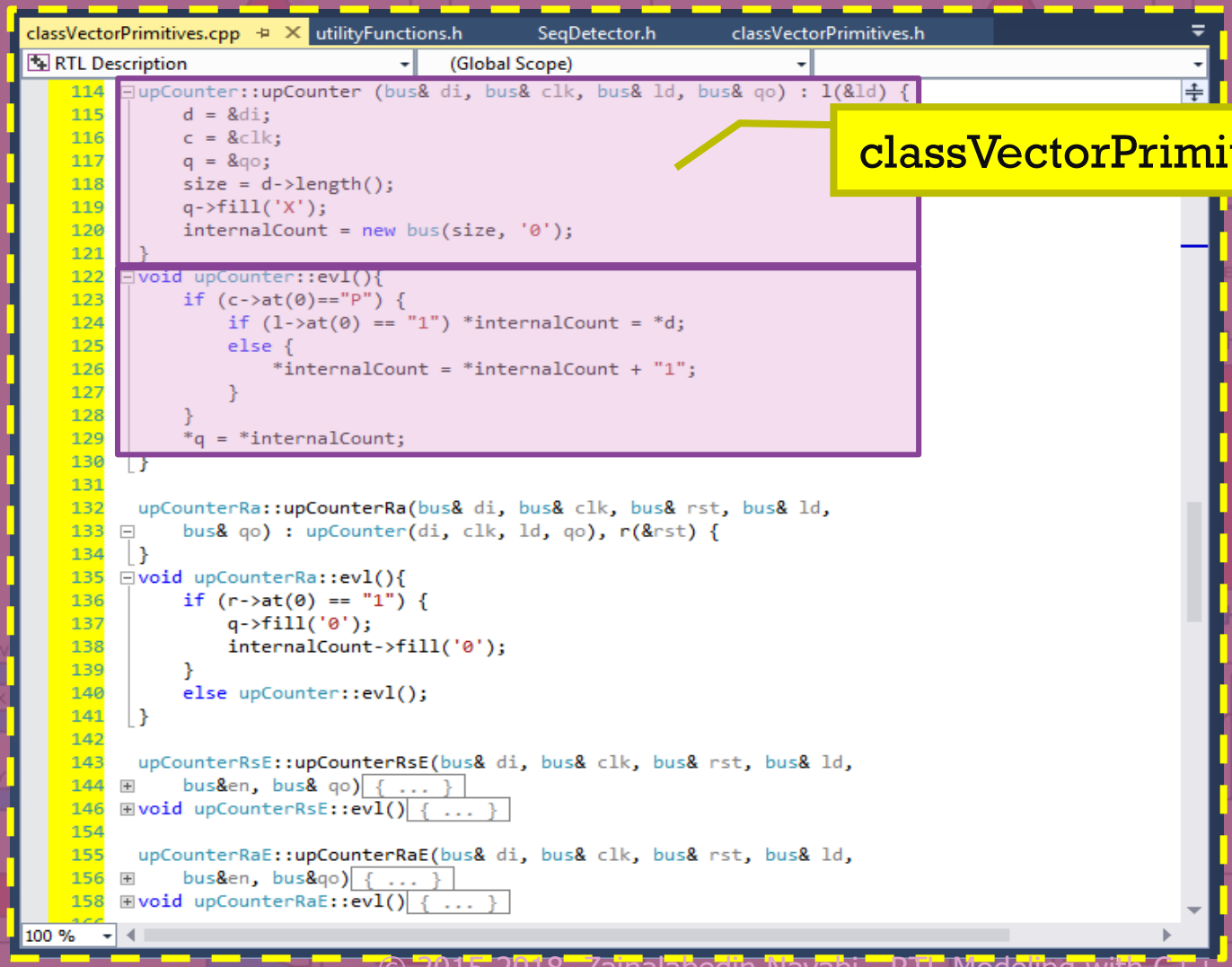
classVectorPrimitives.h

```
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:
343         upCounter (bus& D, bus& C, bus& L, bus& Q);
344         ~upCounter();
345         virtual void evl ();
346 };
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
```

Counters add load input.



UpCounter



classVectorPrimitives.h

UpCounterRsE

LRUdatapath.cpp | LRUcontroller.cpp | classVectorPrimitives.h | classVectorPrimitives.cpp

RTL Description | upCounterRa | evl()

```
142
143   upCounterRsE::upCounterRsE(bus& di, bus& clk, bus& rst, bus& ld,
144       bus&en, bus& qo) : upCounter(di, clk, ld, qo), r(&rst), e(&en) {
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) { ... }
183   Memory::~Memory() { ... }
185
186   void Memory::init (const string& filename) { ... }
193   void Memory::dump (const string& filename) { ... }
202   void Memory::evl () { ... }
217
```

100 %

classVectorPrimitives.h

Memory Structure

LRUdatapath.cpp | LRUcontroller.cpp | classVectorPrimitives.h | 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
```

100 %

classVectorPrimitives.h

Use bus pointers for all memory ports

Memory Structure

LRUdatapath.cpp | LRUcontroller.cpp | classVectorPrimitives.h | classVectorPrimitives.cpp*

RTL Description → Memory → evl()

```
166
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() { ... }
185
186  void Memory::init (const string& filename) { ... }
193  void Memory::dump (const string& filename) { ... }
202  void Memory::evl () {
203      if (rst->at(0) == "1") {
204          for (int i = 0; i < N; i++) mem[i].fill('0');
205      }
206      else if (read->at(0) == "1") {
207          *Dout = mem[AddrBus->ival()];
208      }
209      else if (clk->at(0) == "P") {
210          if (write->at(0) == "1") {
211              mem[AddrBus->ival()] = *Din;
212          }
213      }
214  }
215
```

100 %

classVectorPrimitives.h

Memory resetting, read and write

Memory Structure

LRUdatapath.cpp | LRUcontroller.cpp | classVectorPrimitives.h | classVectorPrimitives.cpp

RTL Description → upCounterRa → evl()

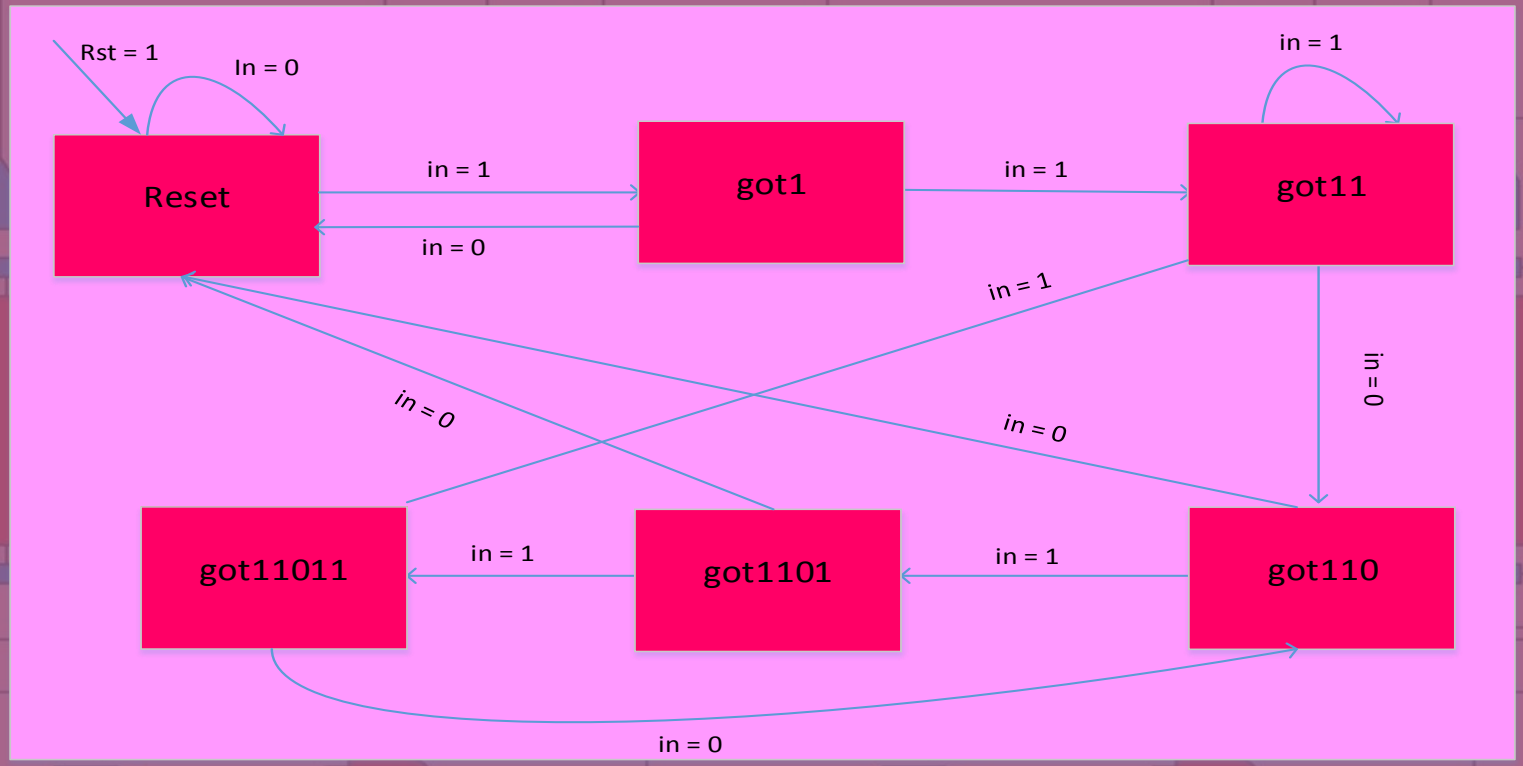
```
167  Memory::Memory (bus& rst, bus& clk, bus& read, bus& write,
168                  bus& Din, bus& AddrBus, bus& Dout) :
169      rst(&rst), read(&read), write(&write), clk(&clk), Din(&Din),
170      AddrBus(&AddrBus), Dout(&Dout)
171  {
172      this->write->fill('X');
173      this->Dout->fill('X');
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 () { ... }
217
```

100 %

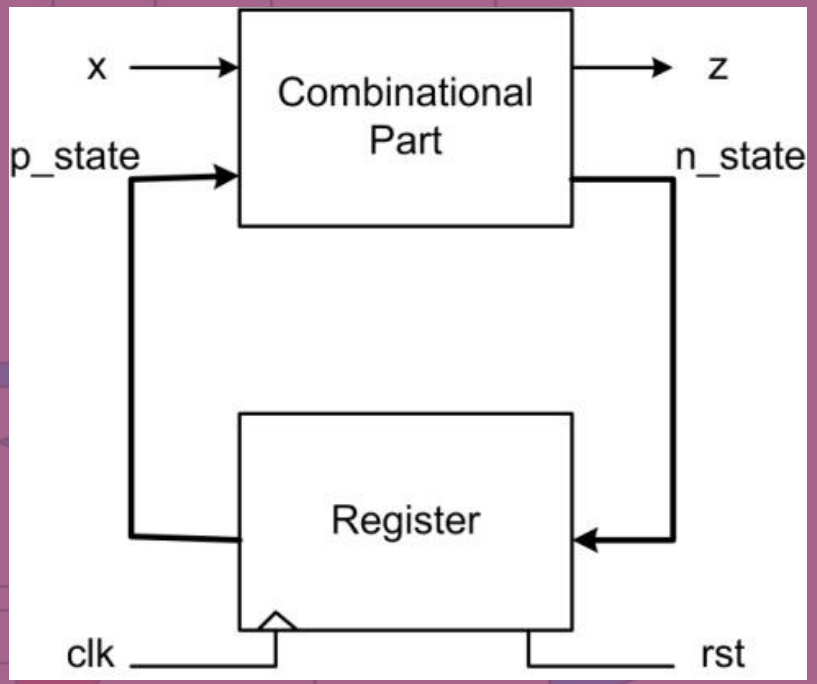
classVectorPrimitives.h

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

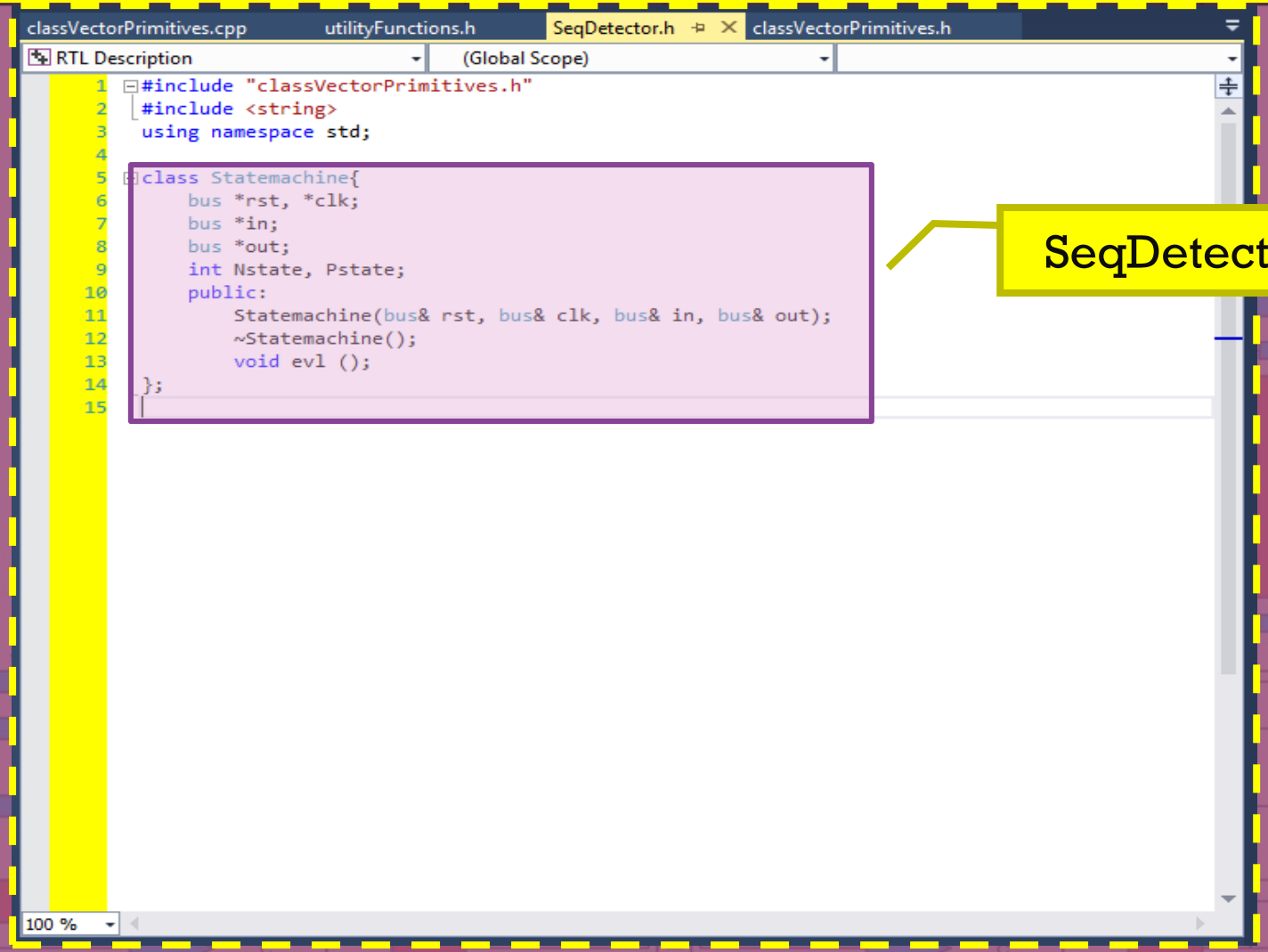
Moore Sequence Detector (11011)



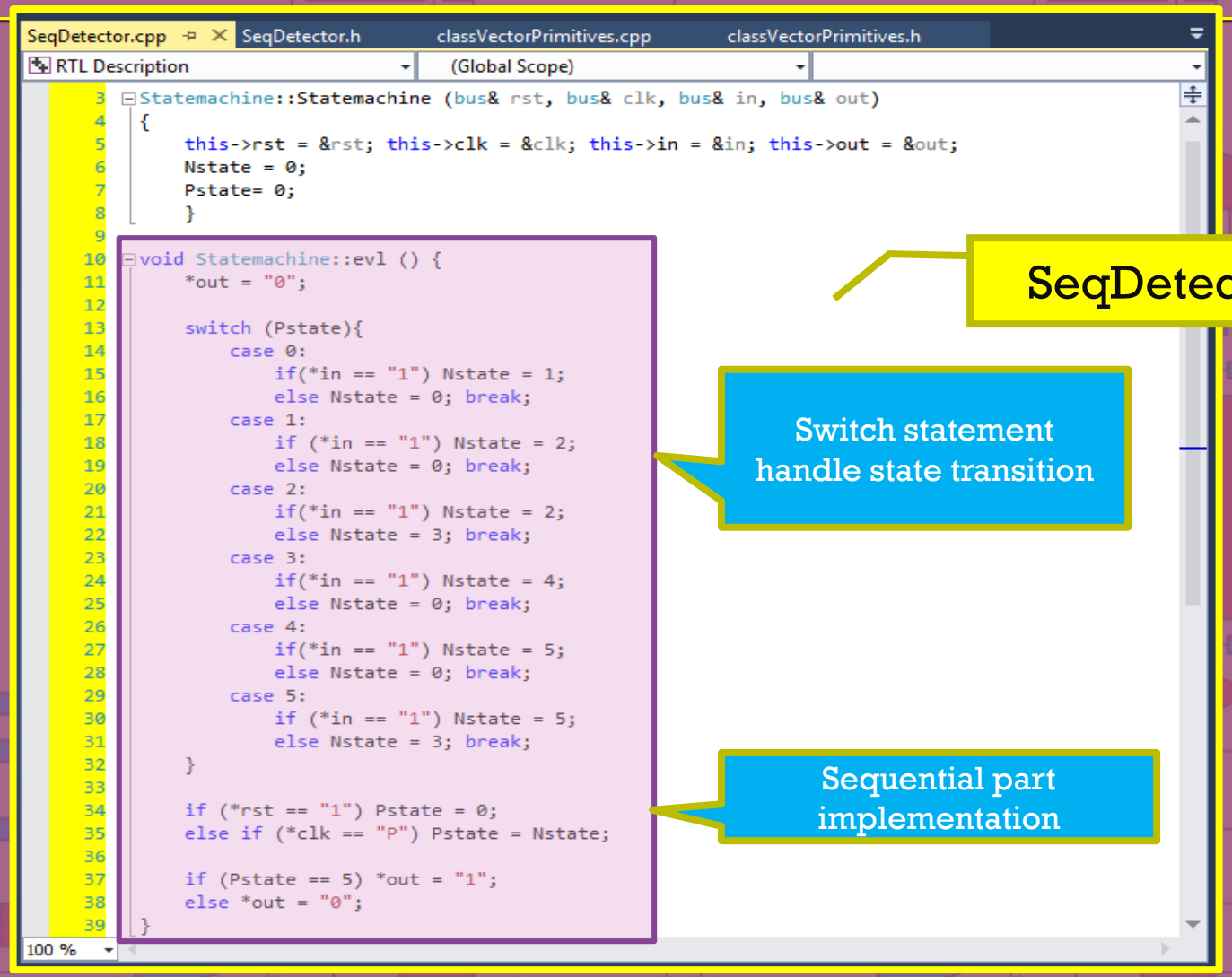
Moore Sequence Detector (11011)



Moore Sequence Detector (11011)



Moore Sequence Detector (11011)



Moore Sequence Detector (11011)

SeqDetectorTB.cpp

```
#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);
}
```

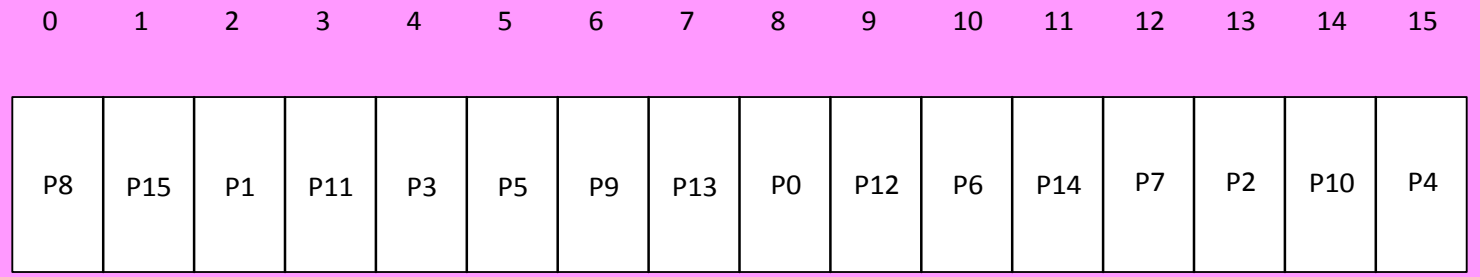
RT Level Modeling with C/C++

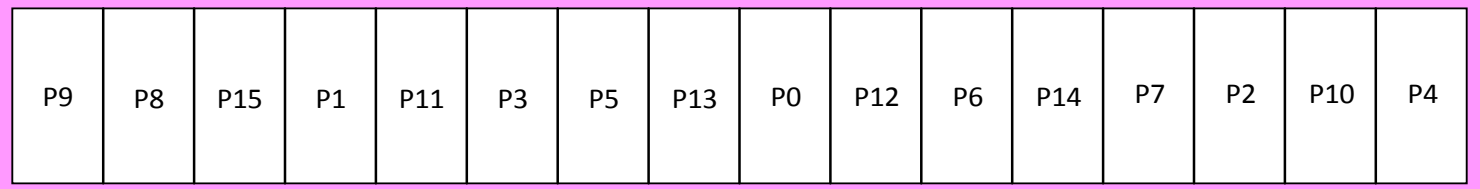
- RTL Principles
  - Elements of datapath
  - Elements of control unit
- Bus Communications
- Utility functions
- Bus operations
  - Array Attributes
  - Logical Operations
  - Adding Operations
  - Relational Operations
  - IO Operations
- Basic Elements of RTL
  - Combinational Elements
  - Registers and Counters
    - Functional register
    - dRegister
    - dRegisterE
    - dRegisterRaE
    - upCounter
    - upCounterRaE
  - Memory Structure
  - Controller FSM
  - Controller 11011
- RTL design example: LRU
  - LRU structure
  - LRU Modeling
    - Controller
    - Ordered instantiation
- RTL design example 2: Exponential Circuit
  - Exponential Circuit Datapath
  - Exponential Circuit Controller

LRU Updater

Assign queue positions to items based on frequency of their use

Before I9 is accessed

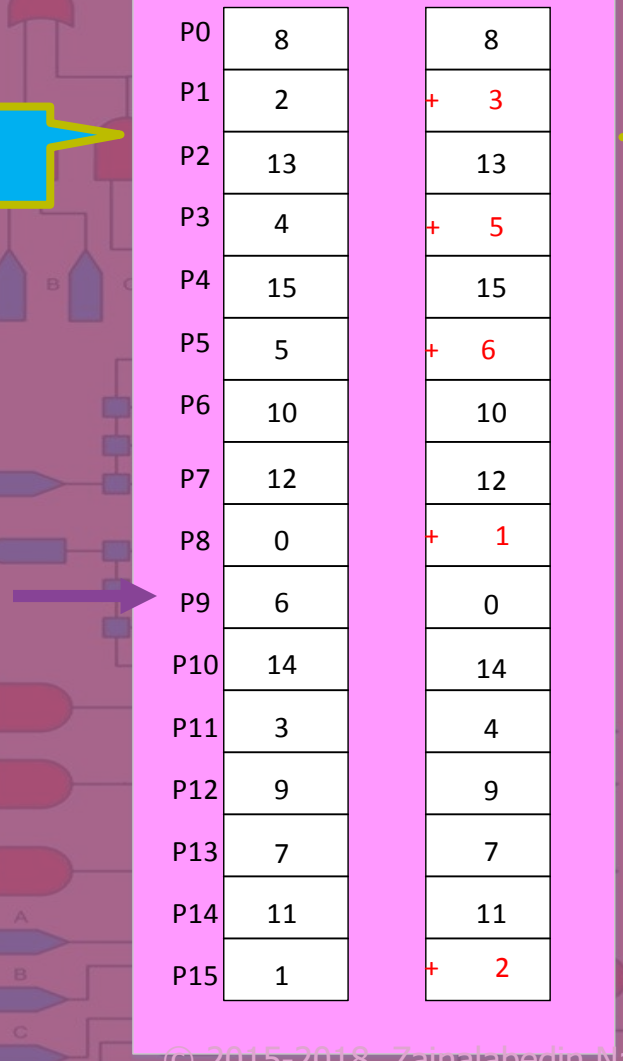




After I9 is accessed

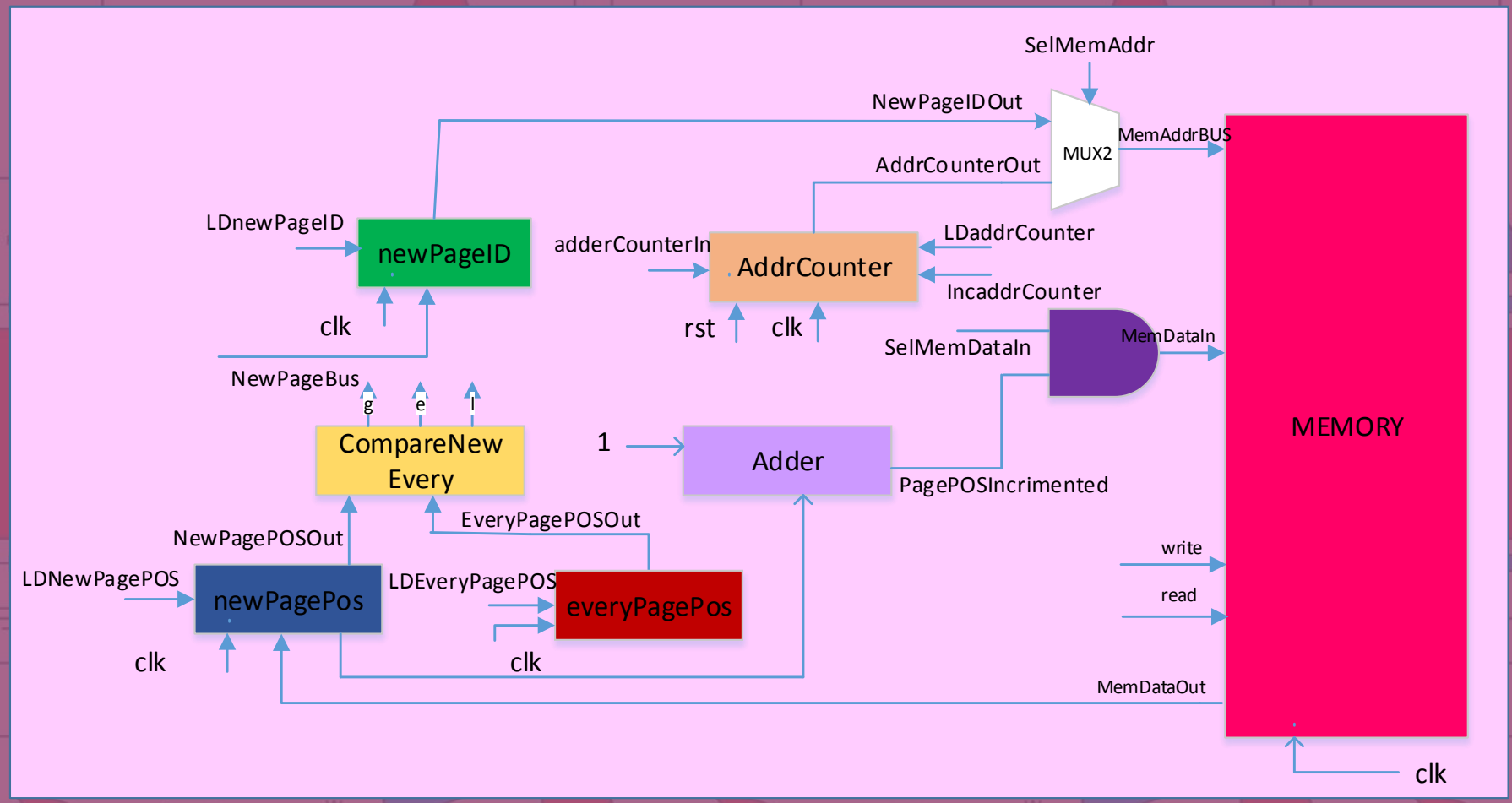
Queue Memory File

Before I9 is accessed

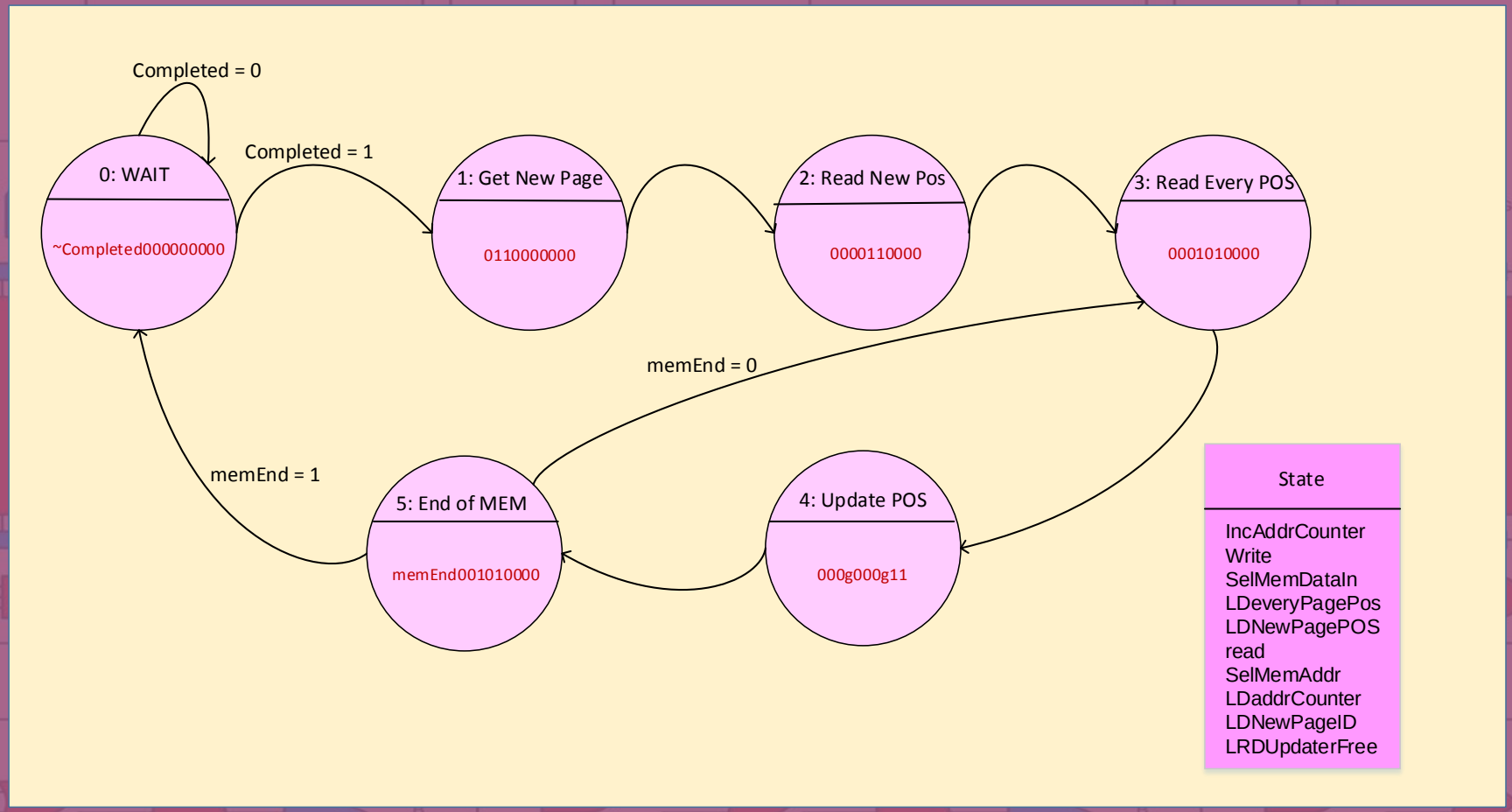


After I9 is accessed

LRU Updater Datapath



LRU Controller



LRU Controller

LRUcontroller.cpp | classVectorPrimitives.h | classVectorPrimitives.cpp | LRUcontroller.h

RTL Description | (Global Scope)

```
1  #include "classVectorPrimitives.h"
2  #include <string>
3  using namespace std;
4
5  class LRUcontroller{
6      bus *rst, *clk;
7      bus *LRUupdaterFree, *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     public:
14         LRUcontroller(bus& rst, bus& clk,
15             bus& LRUupdaterFree, 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
```

100 %

LRUcontroller.h

LRU Controller

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 %

LRUcontroller.cpp

LRU Controller

LRUcontroller.cpp | classVectorPrimitives.cpp | LRUcontroller.h | LRUdatapath.h

RTL Description | (Global Scope)

```
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";
```

100 %

LRUcontroller.cpp

LRU Controller

LRUcontroller.cpp | classVectorPrimitives.cpp | LRUcontroller.h | LRUdatapath.h

RTL Description → LRUcontroller → evl()

```
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";
62              break;
63          case 2: // Read queue position of new page
64              *SELmemAddr ="0"; // newPageId on Addrbus
65              *read = "1";
66              *LDnewPagePos = "1";
67              *LRUupdaterFree = "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              *LRUupdaterFree = "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              *LRUupdaterFree = "0";
89              break;
90          case 5: // Back to 3 if mem end is not reached
91              if (*memEnd == "1")
92                  *LRUupdaterFree = "1";
93              else
94                  *LRUupdaterFree = "0";
95              break;
96      }
```

81 %

LRUcontroller.cpp

LRU Design

LRUdesign.h | LRUdesign.cpp* | 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              *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
```

100 %

LRUdesign.cpp

LRU Design

LRUdesign.cpp | LRUdesign.h | classVectorPrimitives.cpp | SeqDetector.h | classVectorPrimitives.h

RTL Description | (Global Scope)

```
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
```

100 %

LRU Datapath

LRUdatapath.h

```
class LRUdatapath{
    bus *rst, *clk;
    bus *newPageBus; // Input bus
    bus *LRUupdaterFree, *Completed;
    bus *l, *e, *g, *memEnd;
    bus *LDnewPageId, *LDnewPagePos, *LDeveryPagePos;
    bus *LDaddrCounter, *INCaddrCounter, *SELmemDataIn, *SELmemAddr;
    bus *read, *write;

    dRegisterE* newPageId;
    dRegisterE* newPagePos;
    dRegisterE* everyPagePos;
    Comparator* compareNewEvery;
    Mux* MUX2;
    upCounterRaE* addrCounter;
    Memory* memory;

    bus newPageIdOut;       // Internal busses
    bus newPagePosOut;      bus everyPagePosOut;
    bus pagePosIncremented; bus addrCounterOut;
    bus addrCounterIn;
    bus memAddrBus;         // Memory busses
    bus memDataBus;         bus memDataOut;

public:
    LRUdatapath(bus& rst_,      bus& clk_,          bus& newPageBus_,
        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_);
    ~LRUdatapath();
    void evl();
    void evlMemory() { memory->evl(); }
    void initMemory(string filename) { memory->init(filename); }
    void dumpMemory(string filename) { memory->dump(filename); }
};
```

LRU Datapath

LRUcontroller.h | LRUdatapath.h | LRUdatapath.cpp | LRUdesignTB.cpp | LRUcontroller.cpp

RTL Description | (Global Scope)

```
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  }
```

100 %

LRUdatapath.cpp

LRU Datapath

LRUcontroller.h | LRUdatapath.h | LRUdatapath.cpp | LRUdesignTB.cpp | LRUcontroller.cpp

RTL Description | (Global Scope)

```
3
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_) { ... }
41
42  void LRUdatapath::evl()
43  {
44      newPageId->evl();
45      memDataBus = pagePosIncremented & *SELmemDataIn;
46      MUX2->evl();
47      memory->evl();
48      newPagePos->evl();
49      everyPagePos->evl();
50      pagePosIncremented = everyPagePosOut + "1";
51      addrCounter->evl();
52      compareNewEvery->evl();
53      *memEnd = ((addrCounterOut == "0000") ? "1" : "0");
54  }
55
56
```

100 %

LRUdatapath.cpp

LRU Testbench

LRUcontroller.cpp | LRUdatapath.cpp | LRUdesignTB.cpp | utilityFunctions.cpp

RTL Description | (Global Scope) | main()

```
1  #include "LRUcontroller.h"
2  #include "LRUdatapath.h"
3
4
5  int main ()
6  {
7      int ij;
8      string filename("priority.txt");
9
10     bus clk, rst;
11     bus newPageId(4);
12     bus LRUupdaterFree, GETnewPageId;
13     bus l, e, g;
14     bus memEnd;
15     bus LDnewPageId;
16     bus LDnewPagePos, LDeveryPagePos;
17     bus LDaddrCounter, INCaddrCounter;
18     bus SELmemDataIn, SELmemAddr;
19     bus read, write;
20     bus co;
21     bus ci(1, '0');
22
23     LRUdatapath* datapath = new LRUdatapath(rst, clk, newPageId,
24         LRUupdaterFree, GETnewPageId,
25         l, e, g, memEnd,
26         LDnewPageId, LDnewPagePos, LDeveryPagePos,
27         LDaddrCounter, INCaddrCounter,
28         SELmemDataIn, SELmemAddr,
29         read, write);
30     LRUcontroller* controller = new LRUcontroller(rst, clk,
31         LRUupdaterFree, GETnewPageId,
32         l, e, g, memEnd,
33         LDnewPageId, LDnewPagePos, LDeveryPagePos,
34         LDaddrCounter, INCaddrCounter,
35         SELmemDataIn, SELmemAddr,
36         read, write);
37
```

100 %

LRUdesignTB.cpp

LRU Testbench

classVectorPrimitives.cpp | LRUcontroller.h | LRUdatapath.h | LRUdatapath.cpp | LRUdesignTB.cpp

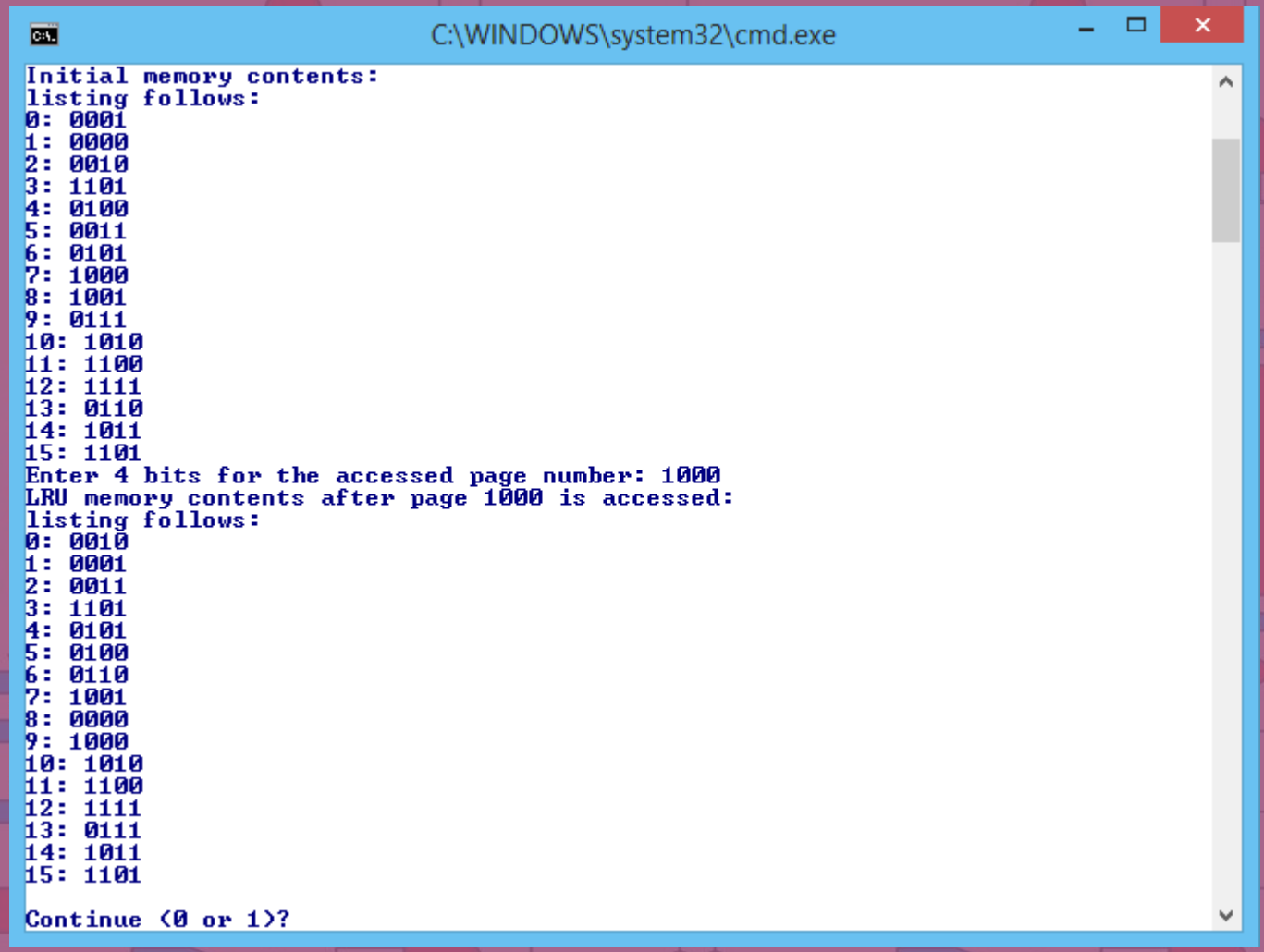
RTL Description | (Global Scope)

```
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
```

100 %

LRUdesignTB.cpp

LRU Output



RT Level Modeling with C/C++

- RTL Principles
  - Elements of datapath
  - Elements of control unit
- Bus Communications
- Utility functions
- Bus operations
  - Array Attributes
  - Logical Operations
  - Adding Operations
  - Relational Operations
  - IO Operations
- Basic Elements of RTL
  - Combinational Elements
  - Registers and Counters
    - Functional register
    - dRegister
    - dRegisterE
    - dRegisterRaE
    - upCounter
    - upCounterRaE
  - Memory Structure
  - Controller FSM
  - Controller 11011
- RTL design example1: LRU
  - LRU structure
  - LRU Modeling
    - Controller
    - Ordered instantiation
- RTL design example 2: Exponential Circuit
  - Exponential Circuit Datapath
  - Exponential Circuit Controller

RTL Example 2: 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.
- When the result becomes ready, done signal will be issued.

Input-output Range

- With 8 bit input size, x is in the range between 0.00000000 for the smallest and 0.11111111 for the largest value.
- Smallest output = 1 when e^0
- Largest output = 10.1010111 (2.68357) when e^1

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 x x ( 1 / i );
    e = e + a;
}
```

Exponential Circuit Datapath

EXPdatapath.cpp | EXPcontroller.cpp | EXPdesignTB.cpp | EXPdatapath.h | EXPcontroller.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      Adder* Add;
20      Multiplier* Mult;
21      Mux* M1;
22      Memory* fractionsMemory;
23      upCounterRsE* indexCounter;
24
25      // internal busses
26      bus exponent;
27      bus exponentInput;
28      bus term;
29      bus termInput;
30      bus resultInput;
31      bus addUpperInput;
32      bus addResult;
33      bus multUpperInput;
34      bus multResult;
35      bus tableData;
36      bus memDatabus;
37      bus countValue;
38      bus initialCount;
39
40      bus addCi, addCo, write;
41      bus enableExponent;
42      bus enableTermReg;
43      bus enableResultReg;
44
45  public:
```

100 %

EXPdatapath.h

Third register for saving partial result

1-a = a* x
2- multresult= a*1/i

ROM for 1/i

Exponential Circuit Datapath

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

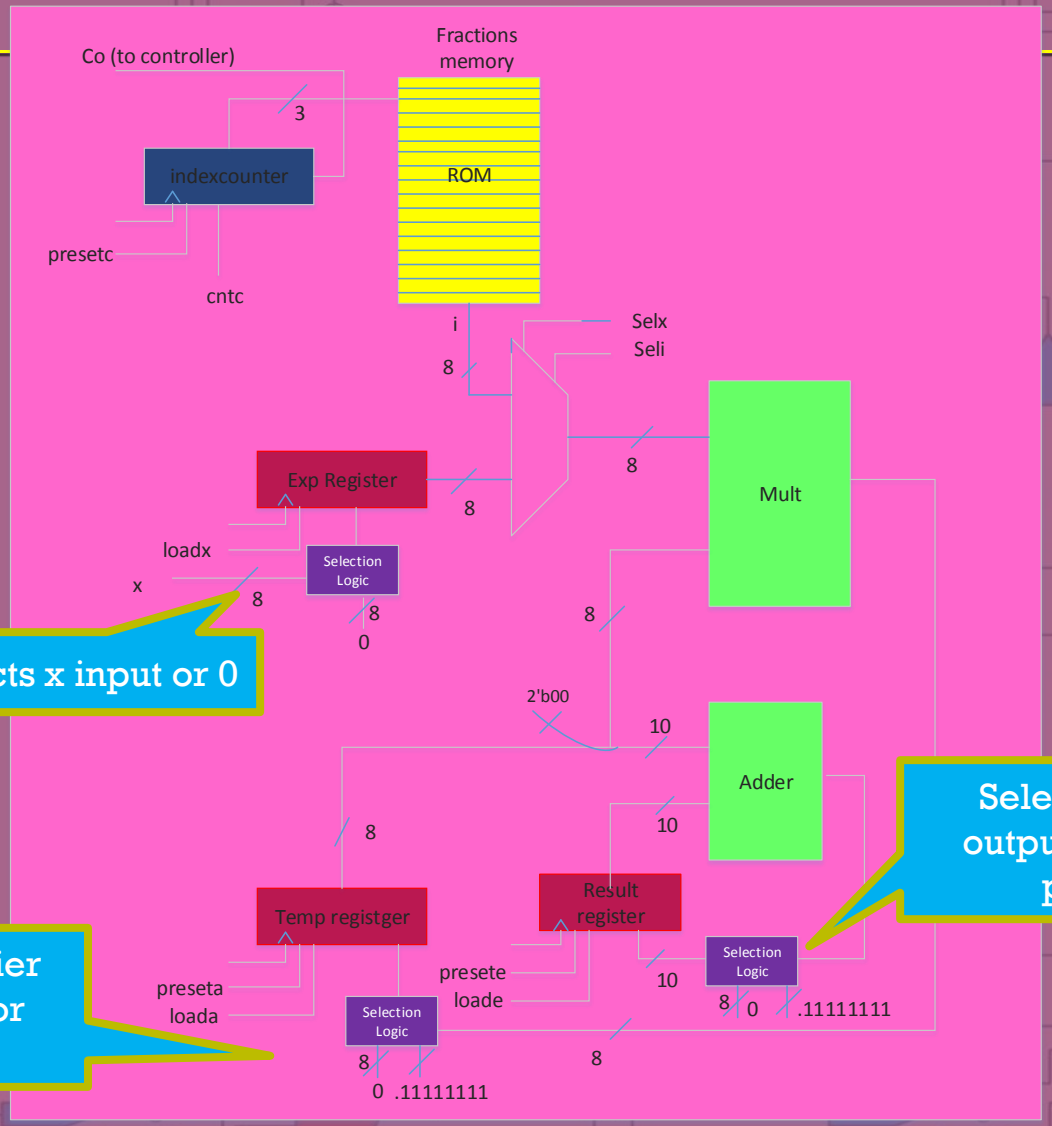
Exponential | (Global Scope)

```
44
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
```

EXPdatapath.h

100 %

Exponential Circuit Datapath



Selects x input or 0

Selects adder output, Reset or preset

Selects multiplier output, Reset or preset

Exponential Circuit Datapath

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

Exponential | (Global Scope)

```
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");
    exponentInput.assign     ("XXXXXXXX");
    term.assign              ("XXXXXXXX");
    termInput.assign         ("XXXXXXXX");
    resultInput.assign       ("XXXXXXXXXX");
    addResult.assign         ("XXXXXXXXXX");
    addUpperInput.assign     ("XXXXXXXXXX");
    multResult.assign        ("XXXXXXXXXXXXXXXX");
    multUpperInput.assign    ("XXXXXXXX");
    tableData.assign         ("XXXXXXXX");
    memDatabus.assign        ("XXXXXXXX");
    countValue.assign        ("XXXX");
    initialCount.assign      ("0001");

    addCi = "0";

    expReg = new dRegisterRaE (exponentInput, *clk, *rst, enableExponent, exponent);

    termReg = new dRegisterRaE
            (termInput, *clk, *rst, enableTermReg, term);

    M1 = new Mux(tableData, exponent, *selTableData, multUpperInput);
```

100 %

EXPdatapath.cpp

Expand to their required size

Exponential Circuit Datapath

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

Exponential | (Global Scope)

```
34
35      addCi = "0";
36
37      expReg = new dRegisterRaE (exponentInput, *clk, *rst, enableExponent, exponent);
38
39      termReg = new dRegisterRaE
40              (termInput, *clk, *rst, enableTermReg, term);
41
42      M1 = new Mux(tableData, exponent, *selTableData, multUpperInput);
43
44      resultReg = new dRegisterRaE
45              (resultInput, *clk, *rst, enableResultReg, *result);
46
47      indexCounter = new upCounterRsE
48              (initialCount, *clk, *rstCounter, *initCounter, *enCounter, count
49
50      fractionsMemory = new Memory
51              (*rst, *clk, *read, write, memDatabus, countValue, tableData, 16)
52
53      Mult = new Multiplier(term, multUpperInput, multResult);
54
55      Add = new Adder(addUpperInput, *result, addCi, addCo, addResult);
56  }
57
```

EXPdatapath.cpp

Some control signals are direct from controller, some is formed by oring several signals from the controller

Exponential Circuit Datapath

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

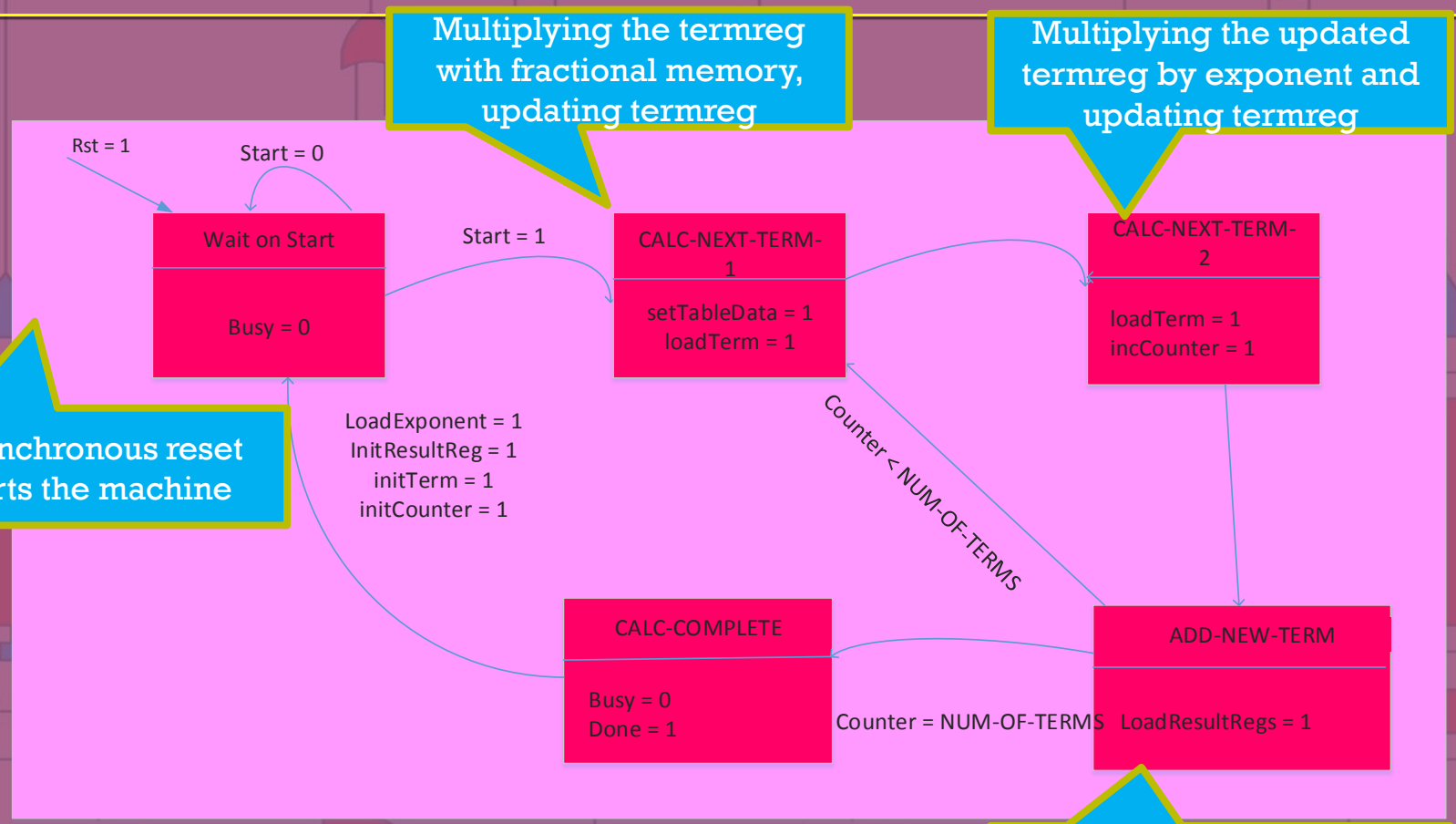
Exponential | expDatapath | expDatapath(bus & clk_, bus & rst_, bus

```
57
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      addUpperInput = ("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
```

EXPdatapath.cpp

The ordering of functions is important

Exponential Circuit Controller



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
```

EXPcontroller.h

100 %

Exponential Circuit Controller

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

Exponential | expController | 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
```

100 %

EXPcontroller.cpp

Control signals are set to inactive values

Exponential Circuit Controller

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

Exponential | expController | 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 %

EXPcontroller.cpp

Switch Case for setting Nstate

Exponential Circuit Controller

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

Exponential → expController → evl()

```
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
```

100 %

EXPcontroller.cpp

Switch for issuing control signals

Getting pstate ready for the next clock cycle

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
```

EXPDesign.h

Controller and datapath class pointers

100 %

Exponential Circuit Design

EXPdatapath.cpp | EXPdesignTB.cpp | EXPdatapath.h | EXPcontroller.h | EXPDesign.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     CT = new expController (*rst, *clk, *start, co, read,
15                 loadExponent, rstExponent, loadTerm, initTerm, rstTerm,
16                 selTableData,
17                 rstResultReg, initResultReg, loadResultReg,
18                 enCounter, rstCounter, initCounter,
19                 *busy, *done);
20 }
21 void expDesign::evl()
22 {
23     DP -> evl();
24     CT -> evl();
25 }
26 void expDesign::initialize(const string& filename)
27 {
28     //resetting
29     *rst = "1";
30     DP -> evlMemory();
31     DP -> evl();
32     CT -> evl();
33     *rst = "0";
34     //Initialize fractionsMemory and dump
35     DP -> initMemory (filename);
36     DP -> dumpMemory ("beforeFile.txt");
37 }
38
39
```

EXPcontroller.cpp

Resets the memory and initializes datapath and controller units

Exponential Circuit Design

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

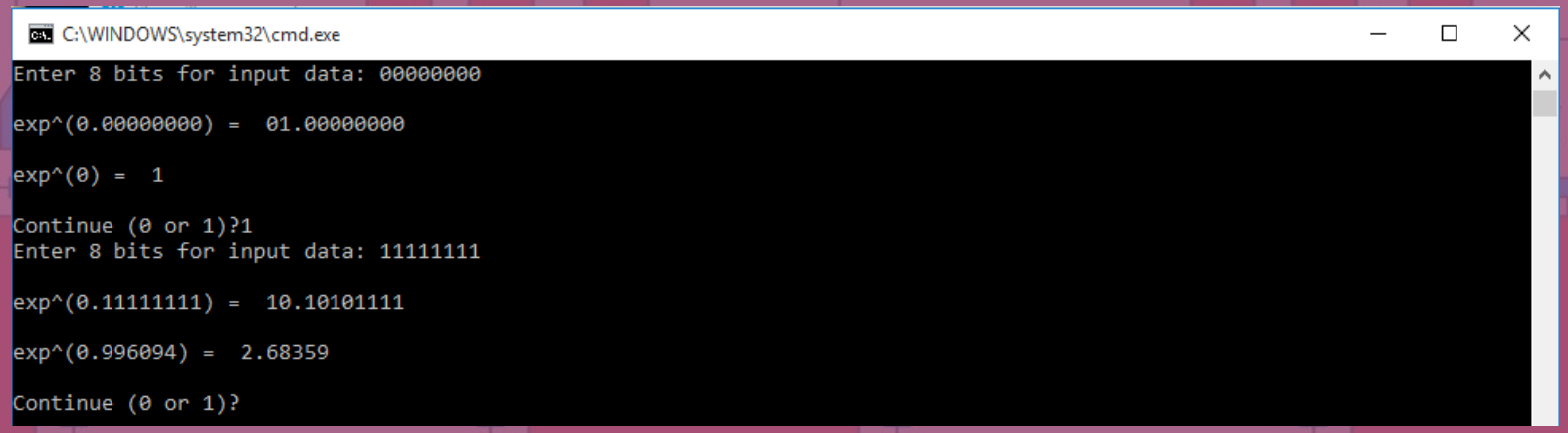
Exponential | (Global Scope)

```
#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);
}
```

100 %

EXPdesignTB.cpp

Exponential Circuit Design



Summary

- Developing utilities for bus classes
- Utilities are used for developing components for design of circuit at the RTL
- Ordering the operations is a key point to correct simulation
- Complete RTL Design: LRU
- Complete RTL Design: Exponential Circuit