Chapter 3

RT Level Modeling with C++

Zainalabedin Navabi

Slides prepared by: Hanieh Hashemi

RT Level Modeling with C/C++

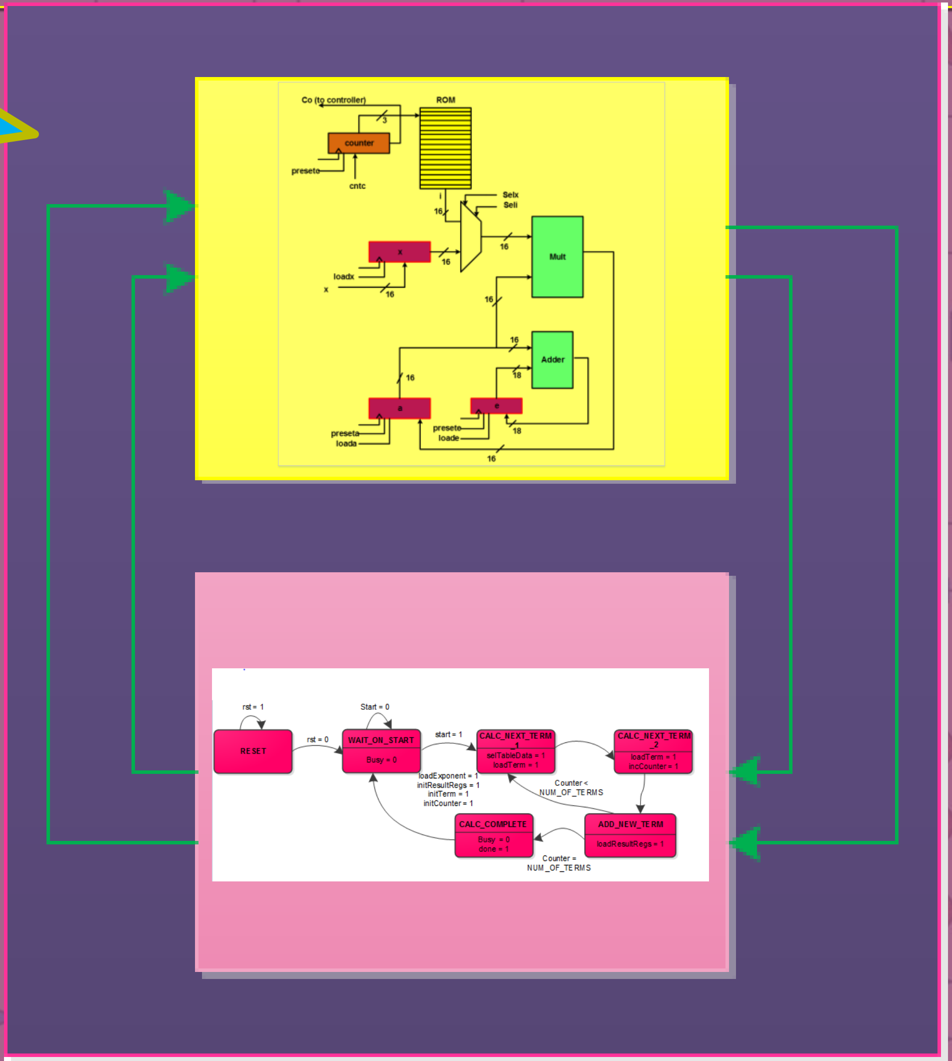
- RTL Principles
  - Elements of datapath
  - Elements of control unit
- Bus Communications
- Utility functions
- Bus operations
  - Array Attributes
  - Logical Operations
  - Adding Operations
  - Relational Operations
  - IO Operations
- Basic Elements of RTL
  - Combinational Elements
  - Registers 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

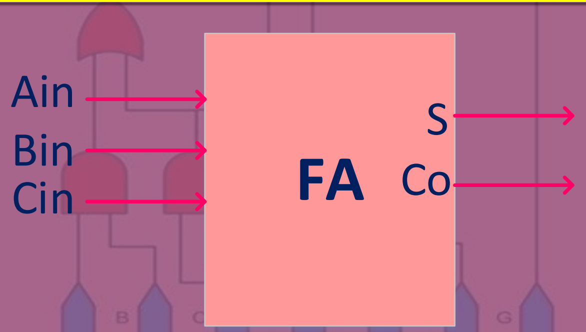
RTL Principles

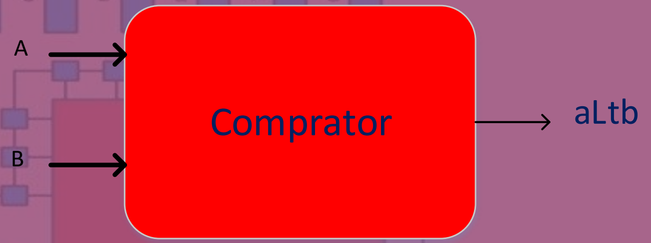
Elements of an RT level component



Elements of Datapath

- Combinational Components
  - Adders
  - Comparators
  - Multiplexers
  - ALUs



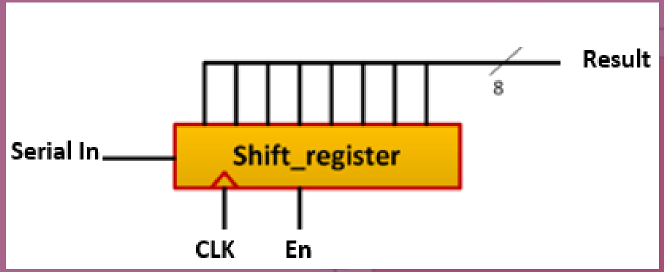


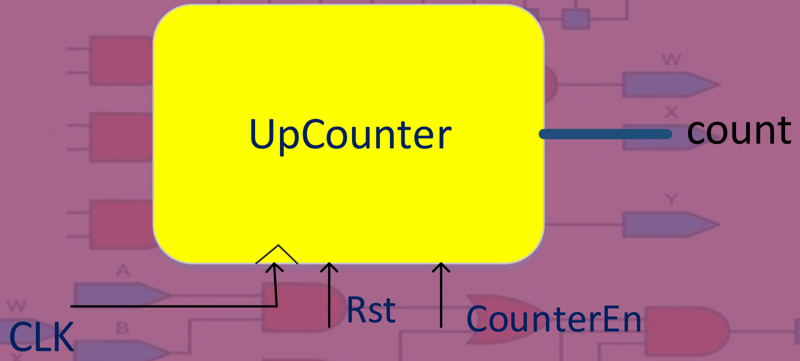
- Logical Operations
  - Vector based
  - Scalar operations
  - Mixed

Elements of Datapath

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

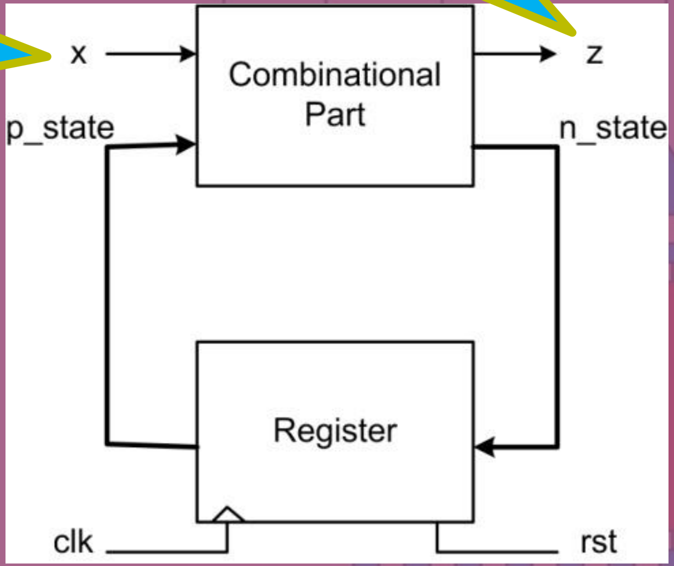
- Sequential component
  - Registers
  - Registers with some functionality
  - Register files





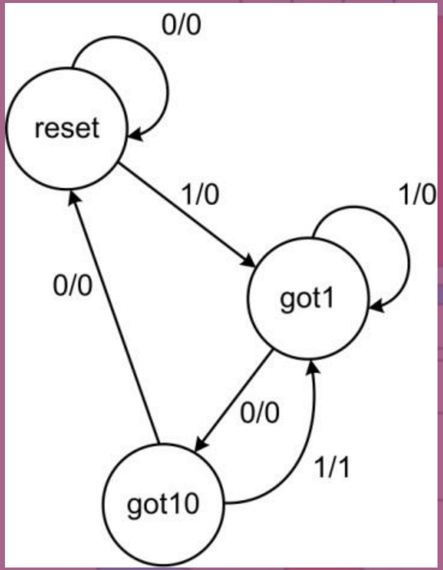
Elements of Control Unit

- One or more state machines
  - Huffman style



Control signals

Inputs from datapath or external

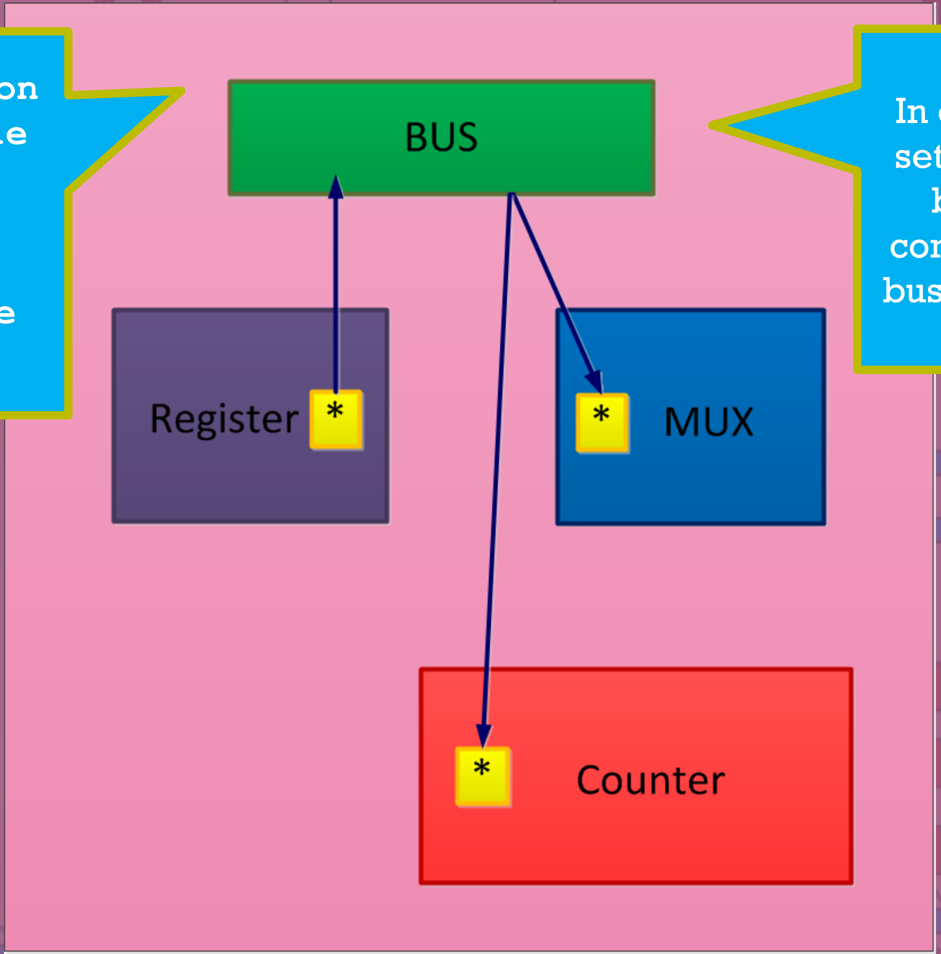


RT Level Modeling with C/C++

- 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



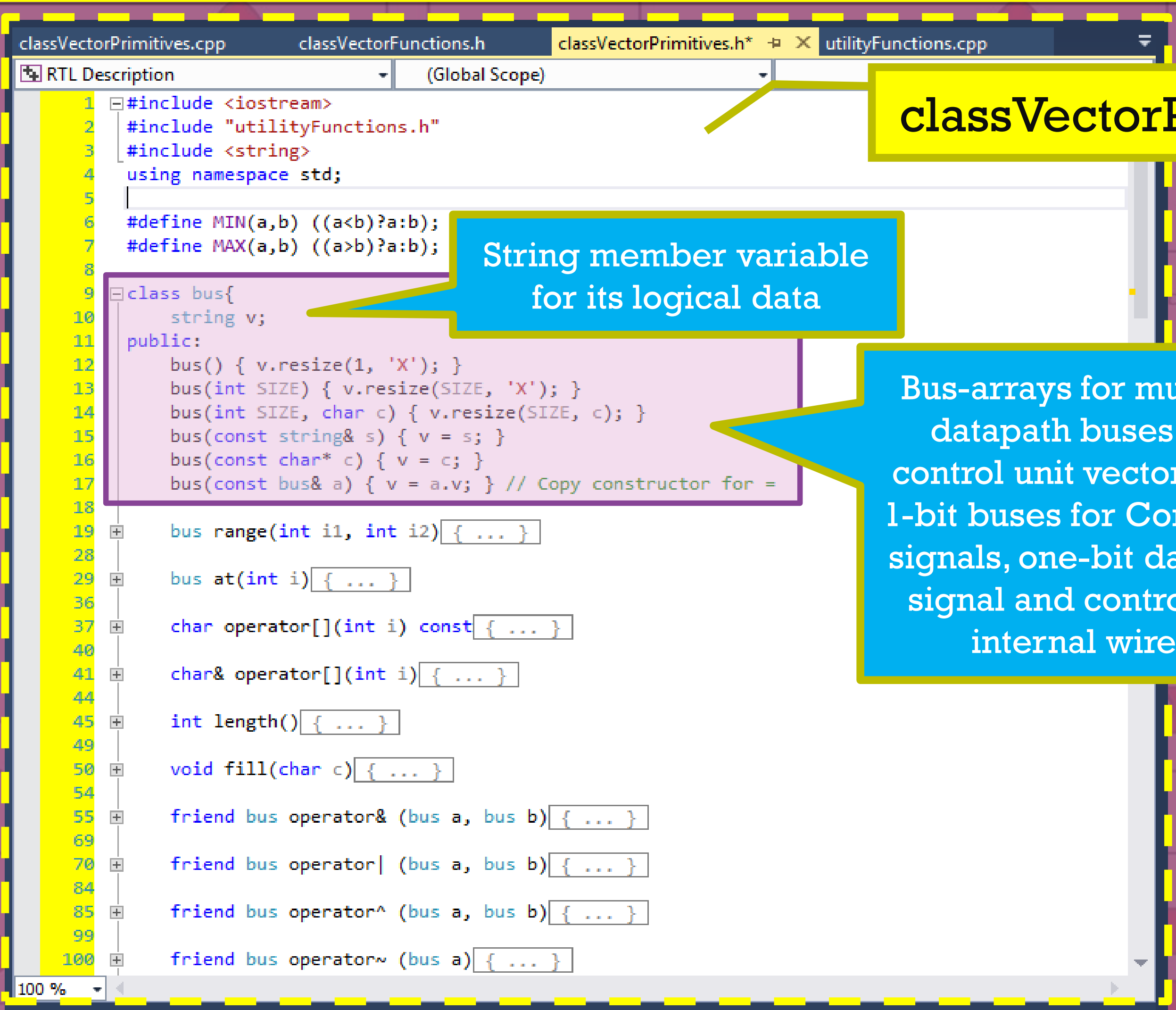
In order to have consistent set of communication lines between datapath and controller, the same type of bus should be used for both

Bus Communications

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

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

Bus Communications



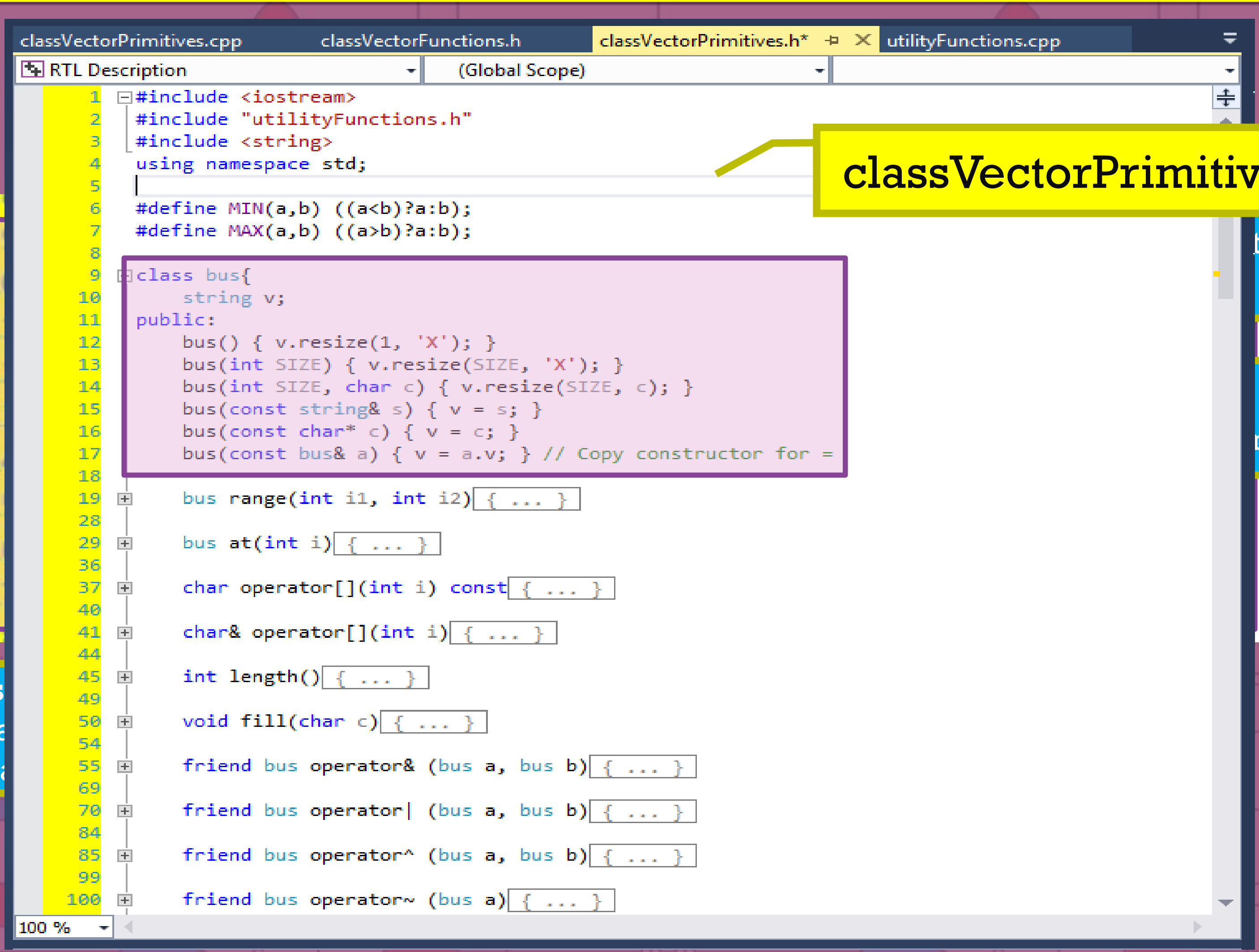
```
#include <iostream>
#include "utilityFunctions.h"
#include <string>
using namespace std;

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

class bus{
    string v;
public:
    bus() { v.resize(1, 'X'); }
    bus(int SIZE) { v.resize(SIZE, 'X'); }
    bus(int SIZE, char c) { v.resize(SIZE, c); }
    bus(const string& s) { v = s; }
    bus(const char* c) { v = c; }
    bus(const bus& a) { v = a.v; } // Copy constructor for =

    bus range(int i1, int i2){ ... }

    bus at(int i){ ... }

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

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

    int length(){ ... }

    void fill(char c){ ... }

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

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

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

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

classVectorPrimitives.h

String member variable for its logical data

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.

Bus Communications



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

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

class bus{
    string v;
public:
    bus() { v.resize(1, 'X'); }
    bus(int SIZE) { v.resize(SIZE, 'X'); }
    bus(int SIZE, char c) { v.resize(SIZE, c); }
    bus(const string& s) { v = s; }
    bus(const char* c) { v = c; }
    bus(const bus& a) { v = a.v; } // Copy constructor for =

    bus range(int i1, int i2){ ... }

    bus at(int i){ ... }

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

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

    int length(){ ... }

    void fill(char c){ ... }

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

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

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

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

classVectorPrimitives.h

th

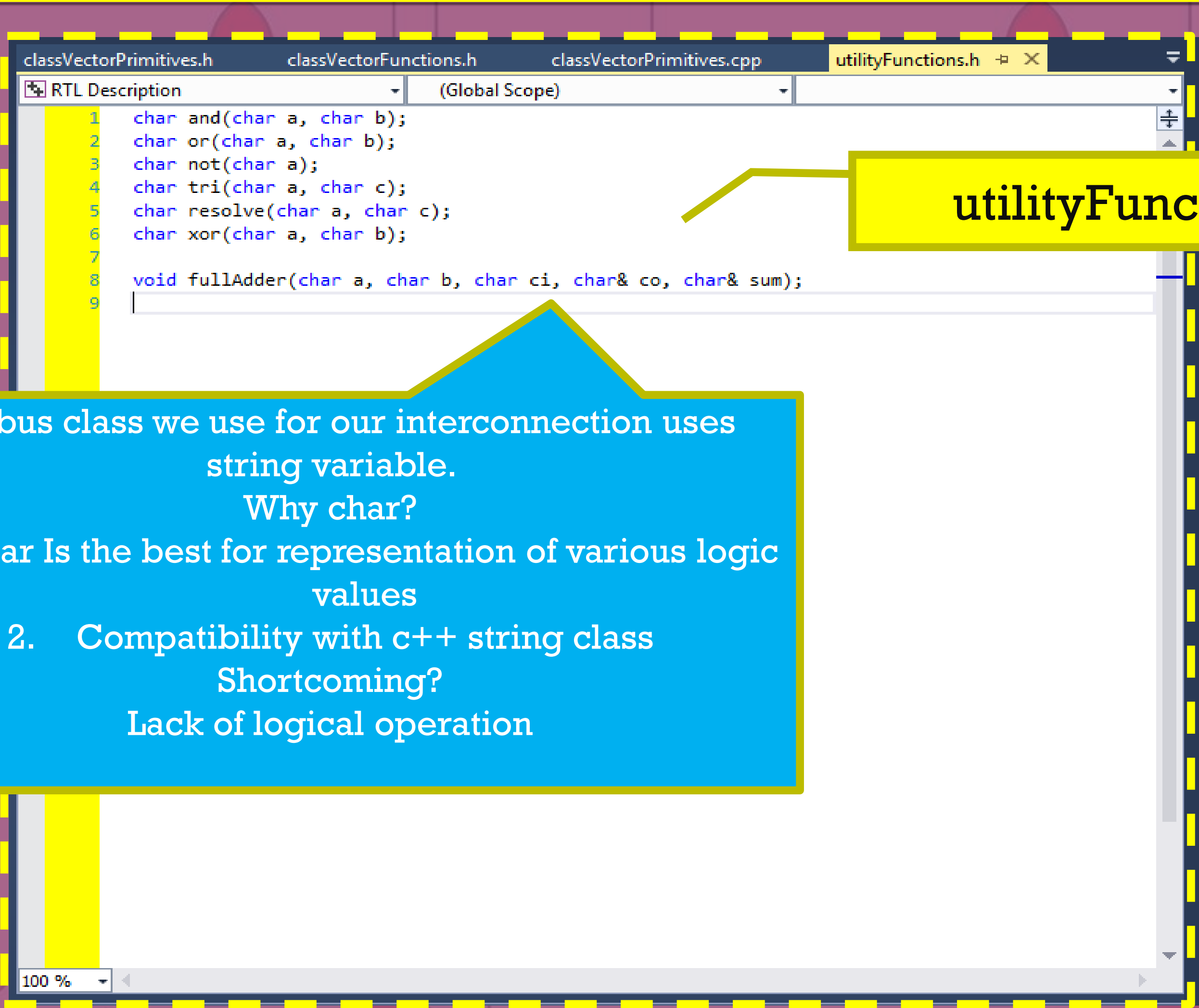
is given as
meter

Us
in a
cha

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



```
char and(char a, char b);
char or(char a, char b);
char not(char a);
char tri(char a, char c);
char resolve(char a, char c);
char xor(char a, char b);

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

utilityFunctions.h

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

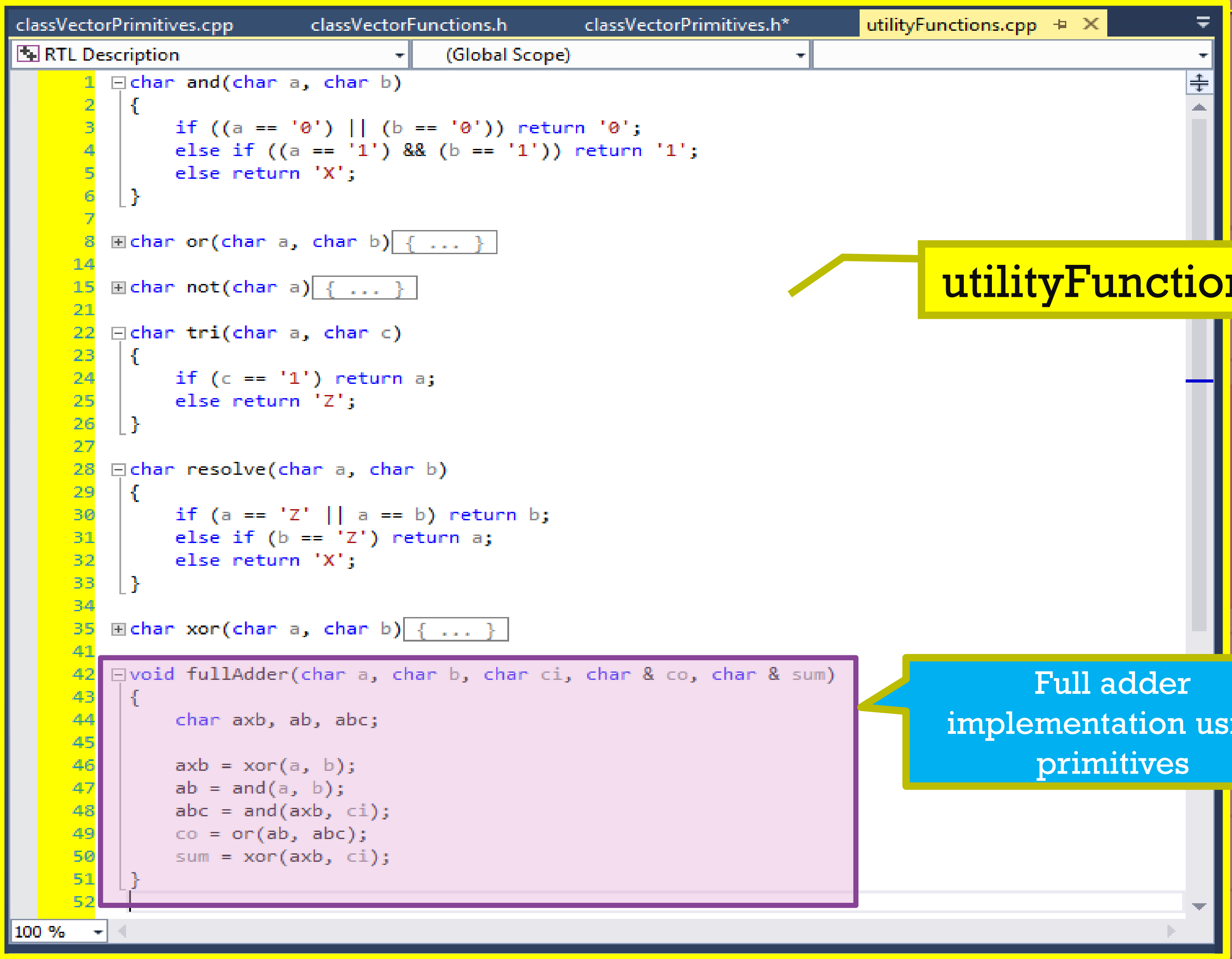
Why char?

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

Shortcoming?

Lack of logical operation

Utility Functions



```
char and(char a, char b)
{
    if ((a == '0') || (b == '0')) return '0';
    else if ((a == '1') && (b == '1')) return '1';
    else return 'X';
}

char or(char a, char b){ ... }

char not(char a){ ... }

char tri(char a, char c)
{
    if (c == '1') return a;
    else return 'Z';
}

char resolve(char a, char b)
{
    if (a == 'Z' || a == b) return b;
    else if (b == 'Z') return a;
    else return 'X';
}

char xor(char a, char b){ ... }

void fullAdder(char a, char b, char ci, char & co, char & sum)
{
    char axb, ab, abc;

    axb = xor(a, b);
    ab = and(a, b);
    abc = and(axb, ci);
    co = or(ab, abc);
    sum = xor(axb, ci);
}

```

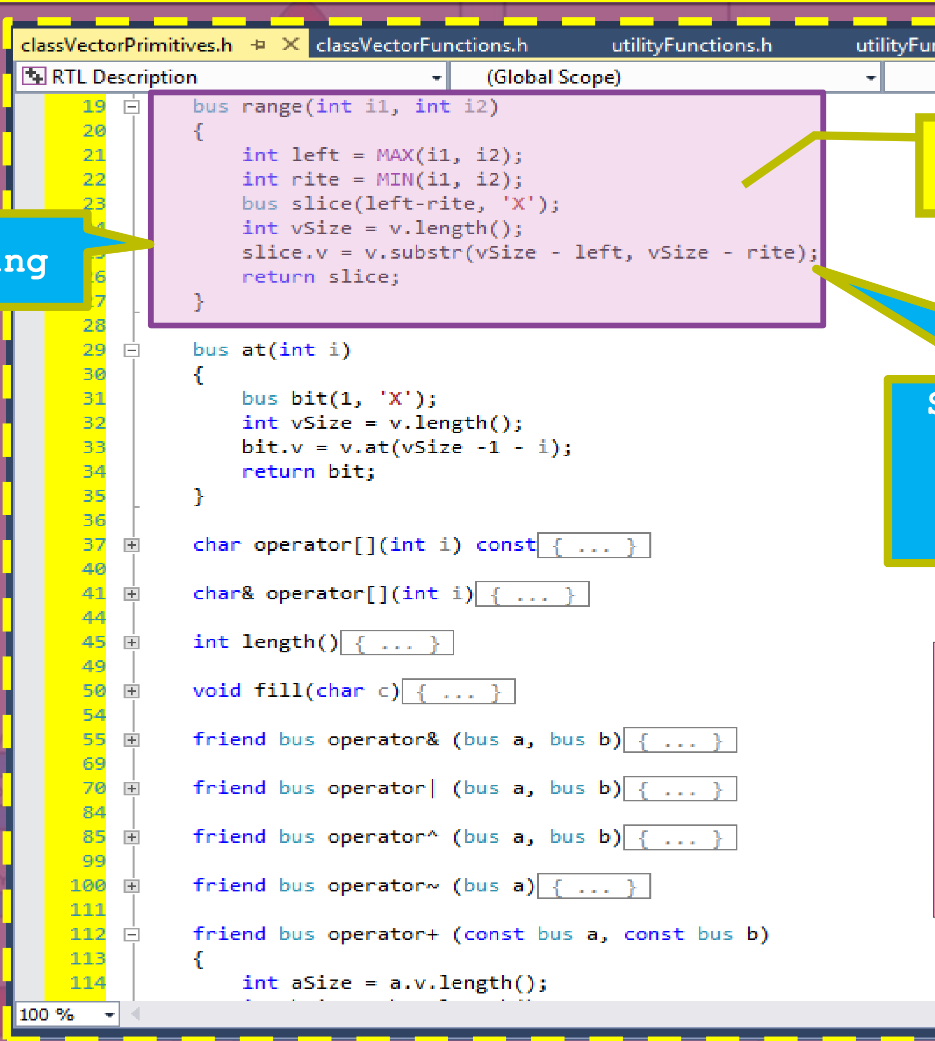
utilityFunctions.cpp

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

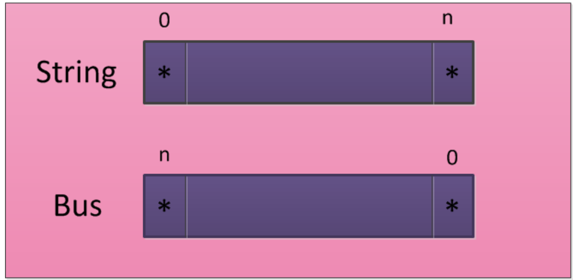
RTL Description | (Global Scope)

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

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

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

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

int length(){ ... }

void fill(char c){ ... }

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

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

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

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

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

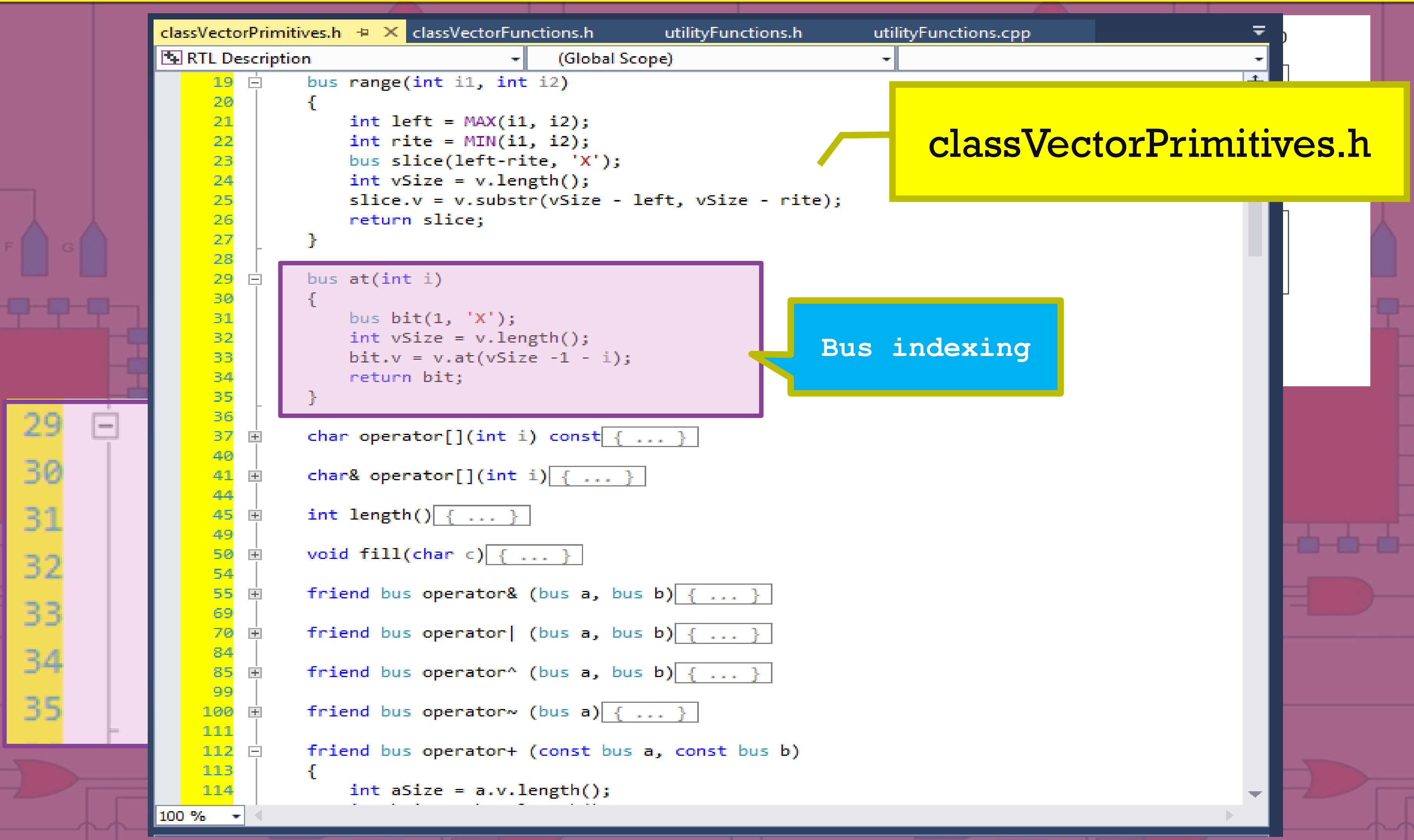
Bus slicing

classVectorPrimitives.h

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



Array Attributes

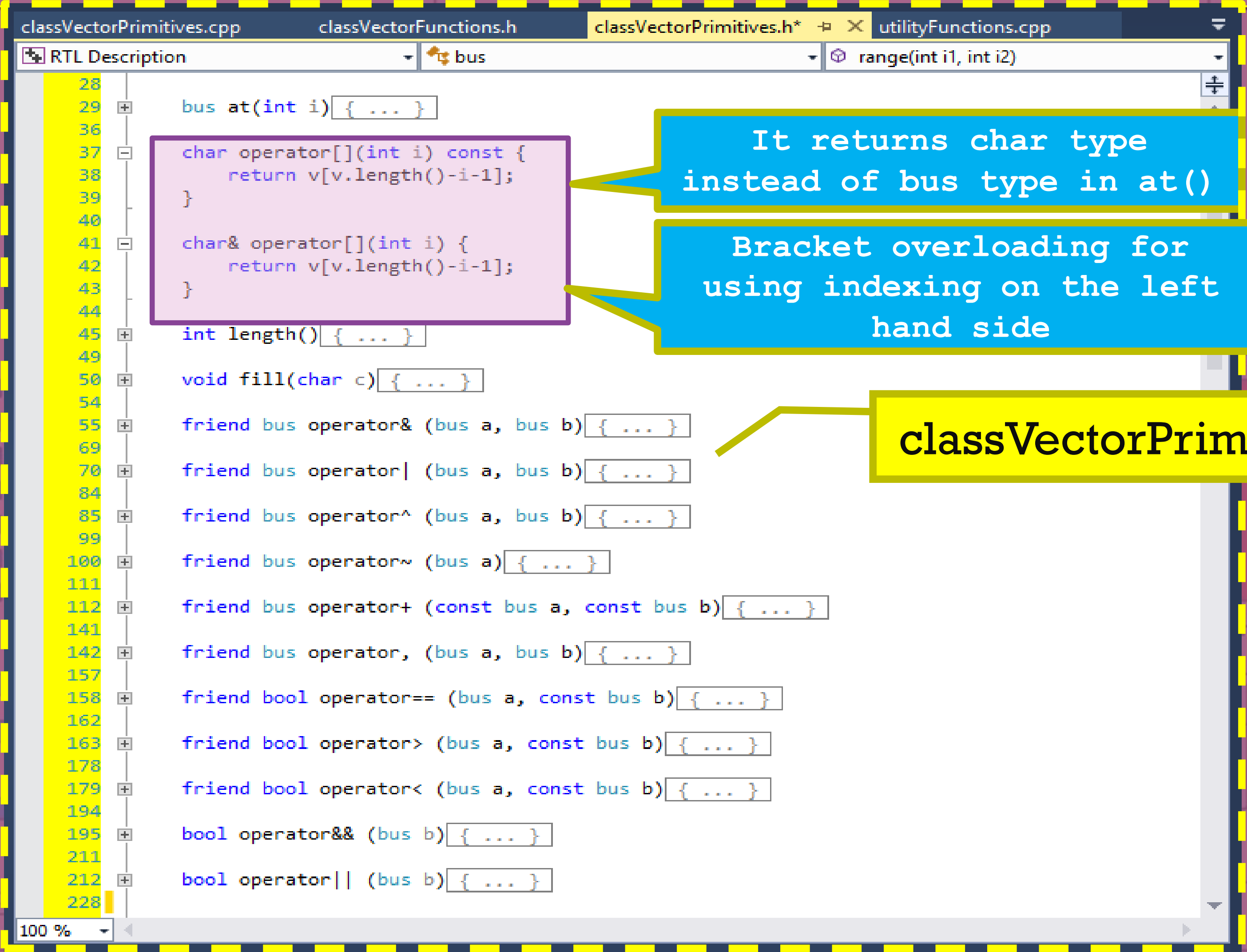


```
bus range(int i1, int i2)
{
    int left = MAX(i1, i2);
    int rite = MIN(i1, i2);
    bus slice(left-rite, 'X');
    int vSize = v.length();
    slice.v = v.substr(vSize - left, vSize - rite);
    return slice;
}

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

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

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

int length(){ ... }

void fill(char c){ ... }

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

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

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

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

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

classVectorPrimitives.h

Bus indexing

Array Attributes



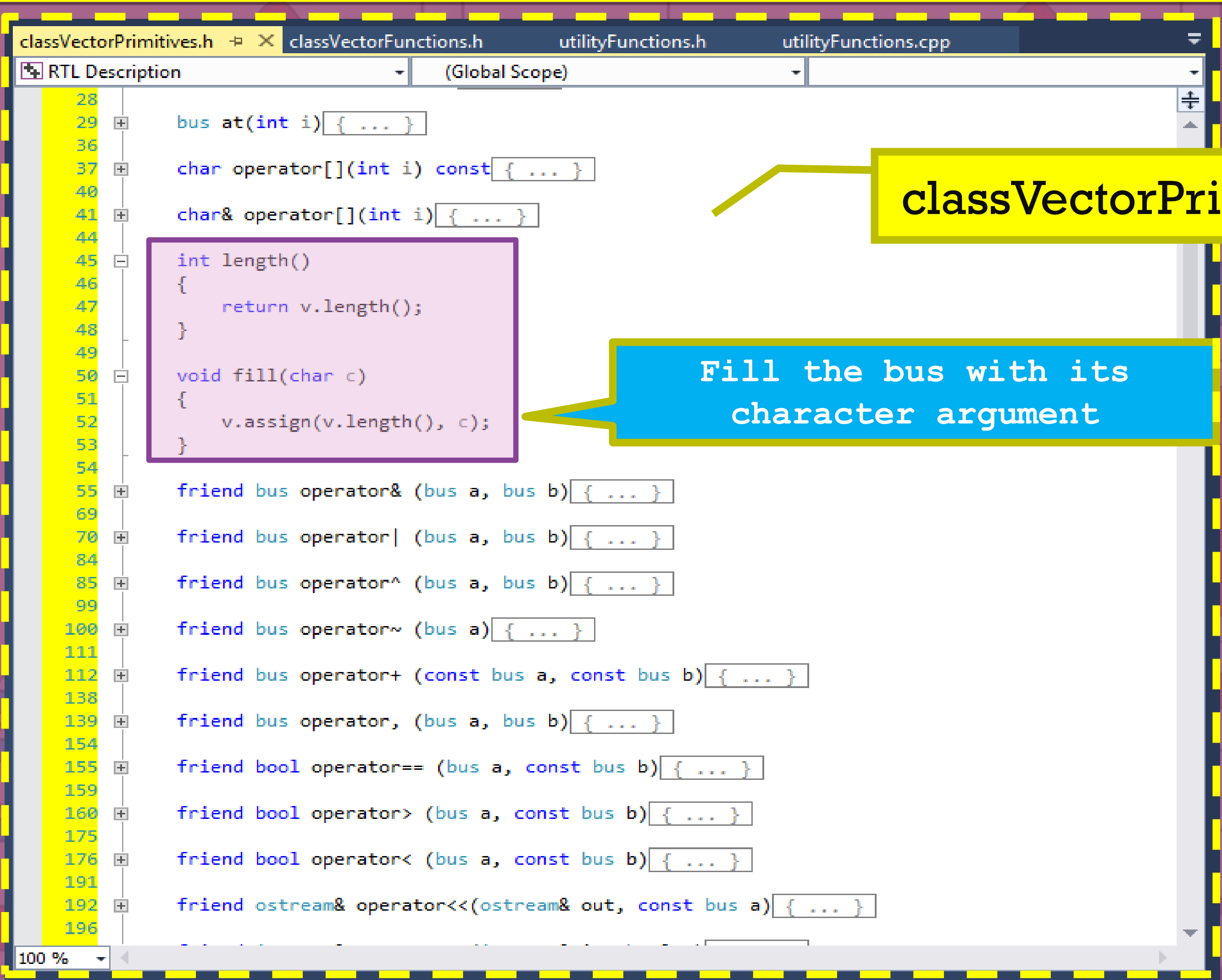
```
bus at(int i){ ... }

char operator[](int i) const {
    return v[v.length()-i-1];
}

char& operator[](int i) {
    return v[v.length()-i-1];
}

int length(){ ... }

void fill(char c){ ... }

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

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

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

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

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

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

friend bool operator== (bus a, const bus b){ ... }

friend bool operator> (bus a, const bus b){ ... }

friend bool operator< (bus a, const bus b){ ... }

bool operator&& (bus b){ ... }

bool operator|| (bus b){ ... }
```

It returns char type instead of bus type in at()

Bracket overloading for using indexing on the left hand side

classVectorPrimitives.h

Array Attributes

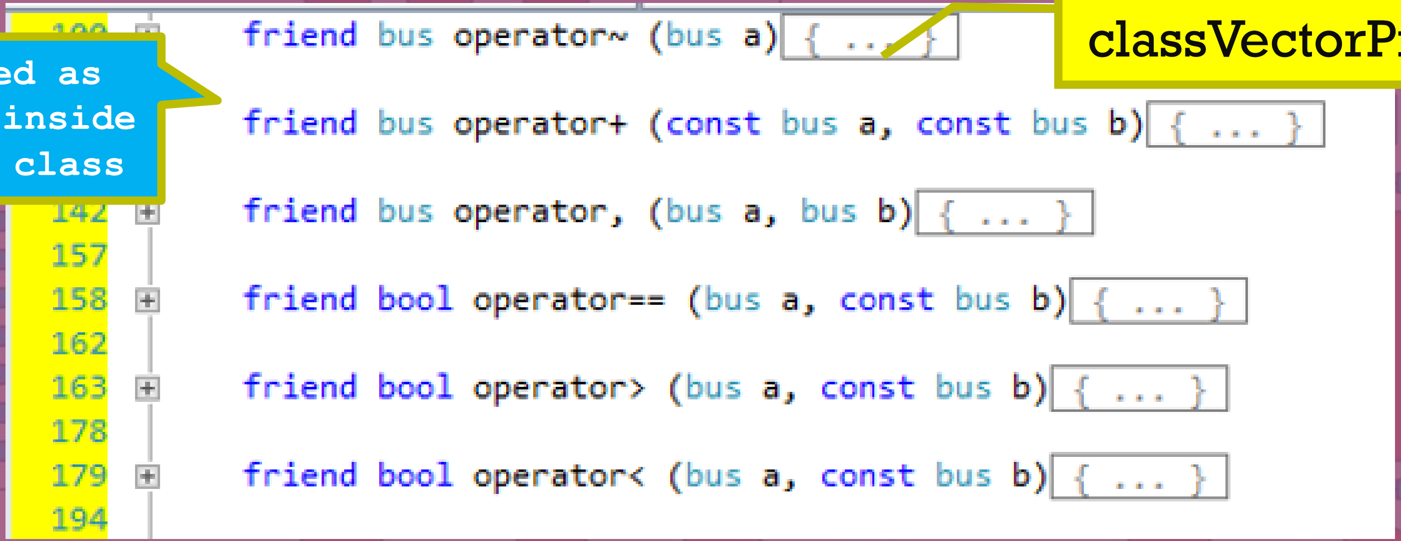


```
bus at(int i){ ... }

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

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

int length()
{
    return v.length();
}

void fill(char c)
{
    v.assign(v.length(), c);
}

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

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

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

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

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

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

friend bool operator== (bus a, const bus b){ ... }

friend bool operator> (bus a, const bus b){ ... }

friend bool operator< (bus a, const bus b){ ... }

friend ostream& operator<<(ostream& out, const bus a){ ... }
```

classVectorPrimitives.h

Fill the bus with its character argument

Logical Operations

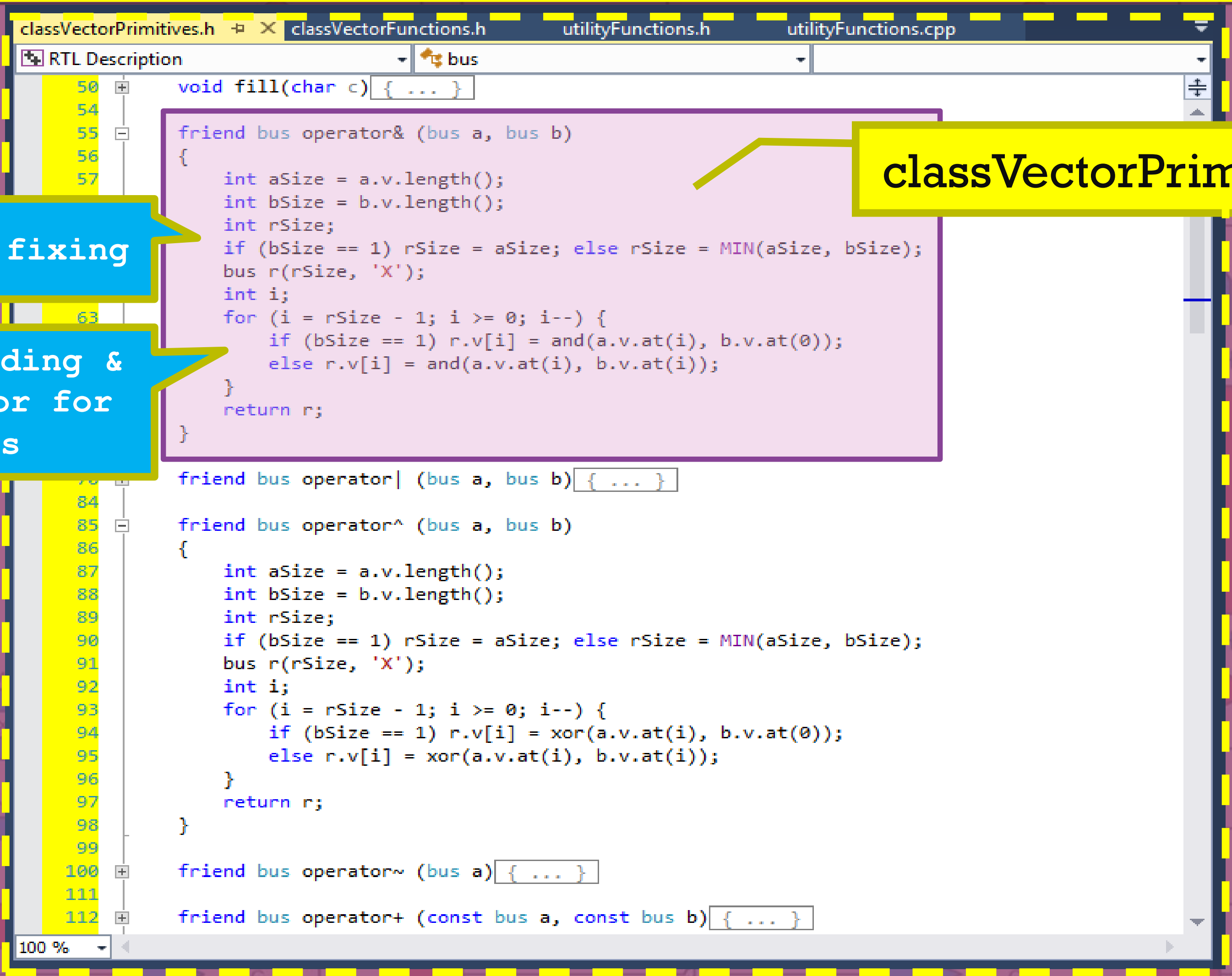


```
    friend bus operator~ (bus a) { ... }

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

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

    friend bool operator== (bus a, const bus b) { ... }

    friend bool operator> (bus a, const bus b) { ... }

    friend bool operator< (bus a, const bus b) { ... }
```

Declared as friends inside the bus class

classVectorPrimitives.h

Logical Operations



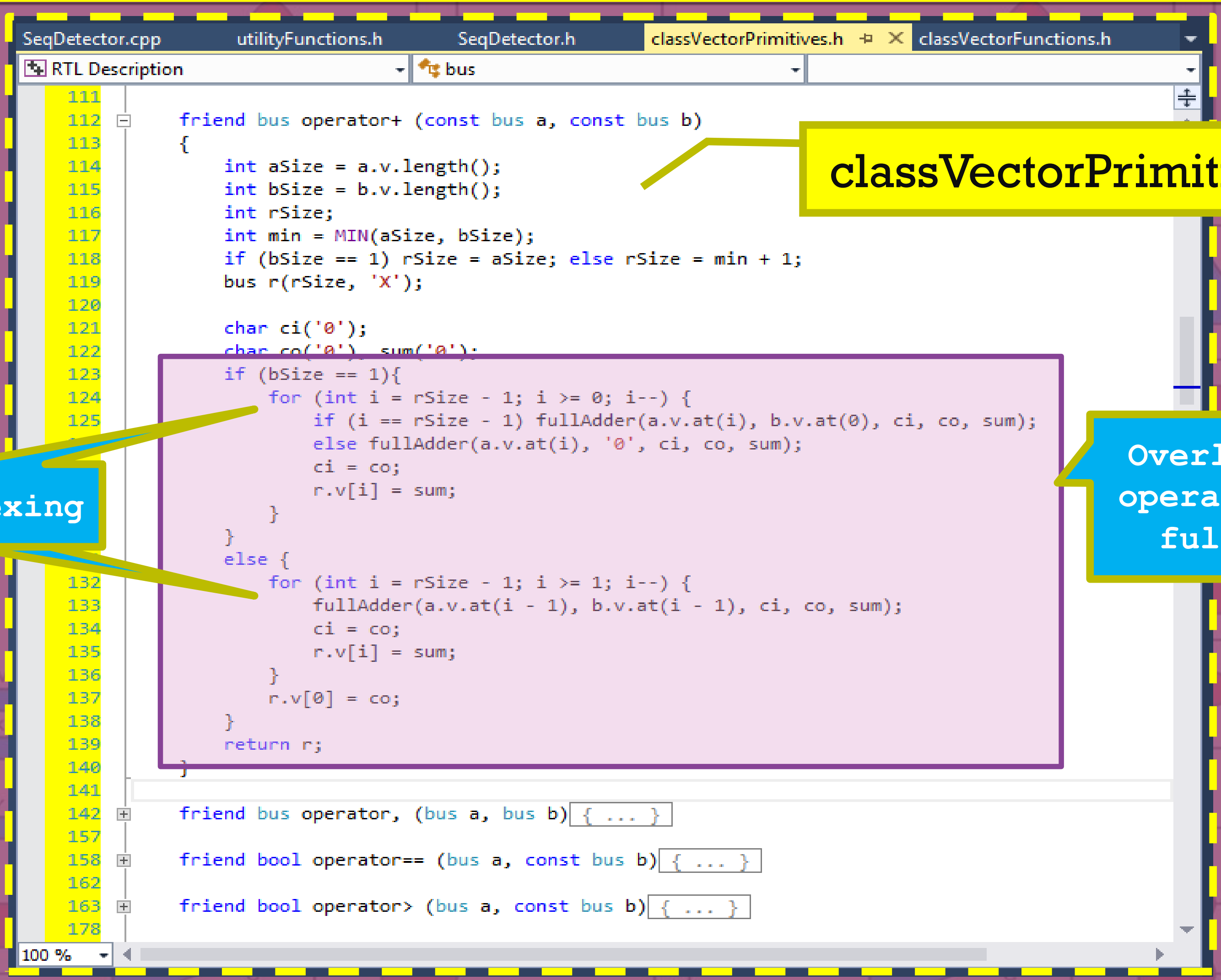
```
void fill(char c){ ... }

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

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

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

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

classVectorPrimitives.h

Size fixing

Overloading & operator for bus

Adding Operations



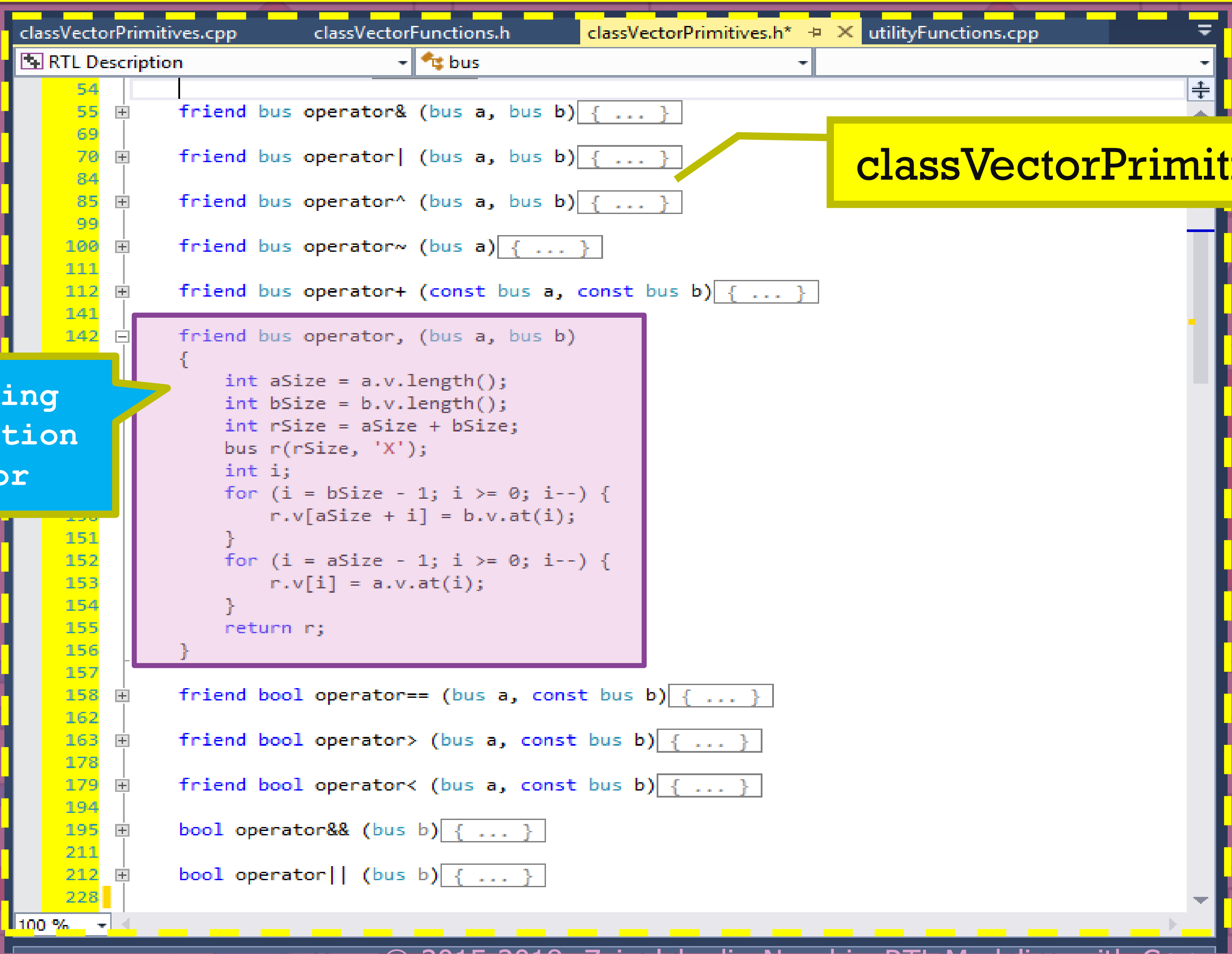
```
    friend bus operator+ (const bus a, const bus b)
    {
        int aSize = a.v.length();
        int bSize = b.v.length();
        int rSize;
        int min = MIN(aSize, bSize);
        if (bSize == 1) rSize = aSize; else rSize = min + 1;
        bus r(rSize, 'X');

        char ci('0');
        char co('0'), sum('0');
        if (bSize == 1){
            for (int i = rSize - 1; i >= 0; i--) {
                if (i == rSize - 1) fullAdder(a.v.at(i), b.v.at(0), ci, co, sum);
                else fullAdder(a.v.at(i), '0', ci, co, sum);
                ci = co;
                r.v[i] = sum;
            }
        }
        else {
            for (int i = rSize - 1; i >= 1; i--) {
                fullAdder(a.v.at(i - 1), b.v.at(i - 1), ci, co, sum);
                ci = co;
                r.v[i] = sum;
            }
            r.v[0] = co;
        }
        return r;
    }

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

    friend bool operator== (bus a, const bus b){ ... }

    friend bool operator> (bus a, const bus b){ ... }
```

classVectorPrimitives.h

String indexing

Overloading + operator using full adder

Logical Operations

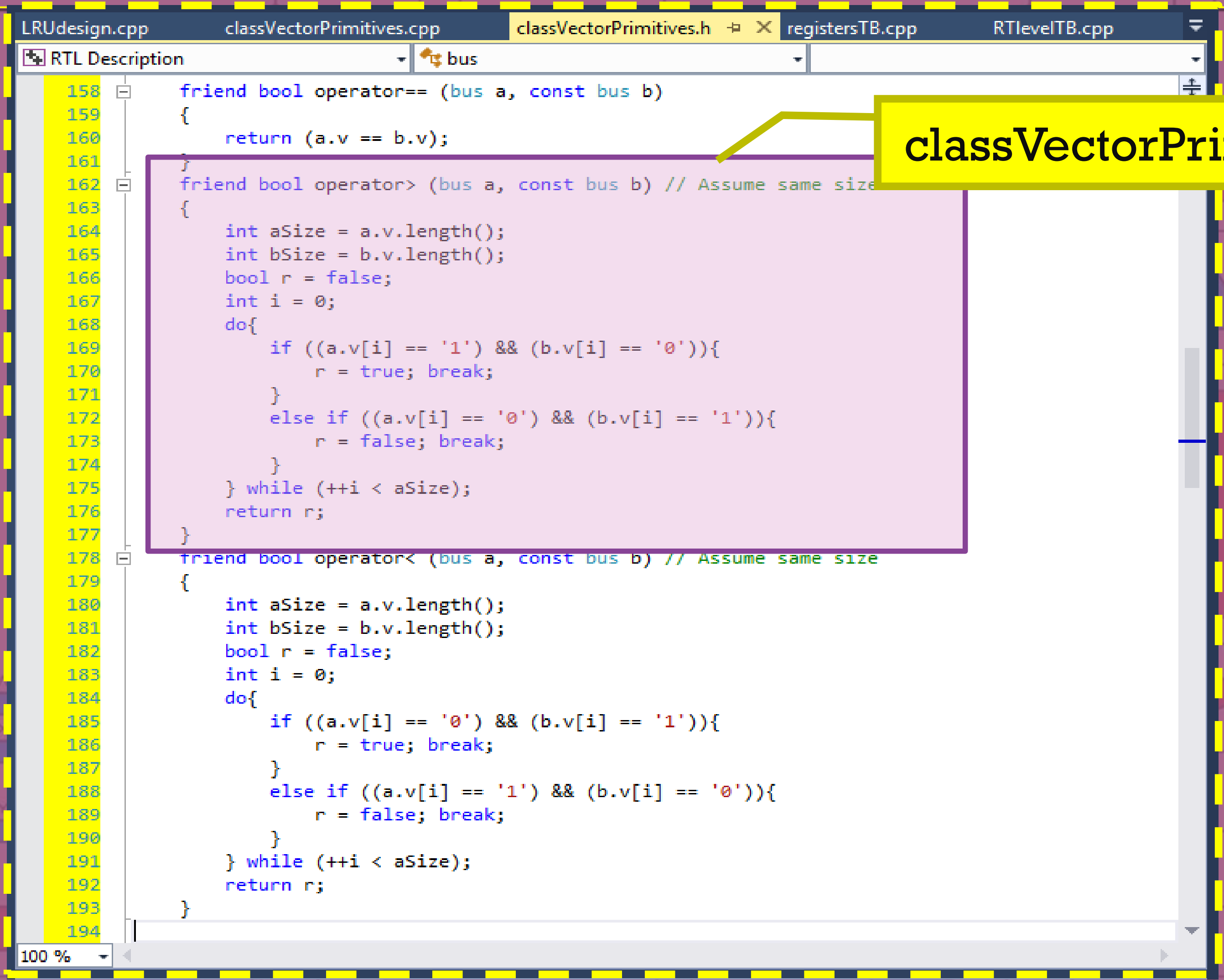


```
    friend bus operator& (bus a, bus b){ ... }

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

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

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

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

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

    friend bool operator== (bus a, const bus b){ ... }

    friend bool operator> (bus a, const bus b){ ... }

    friend bool operator< (bus a, const bus b){ ... }

    bool operator&& (bus b){ ... }

    bool operator|| (bus b){ ... }
```

classVectorPrimitives.h

Overloading concatenation operator

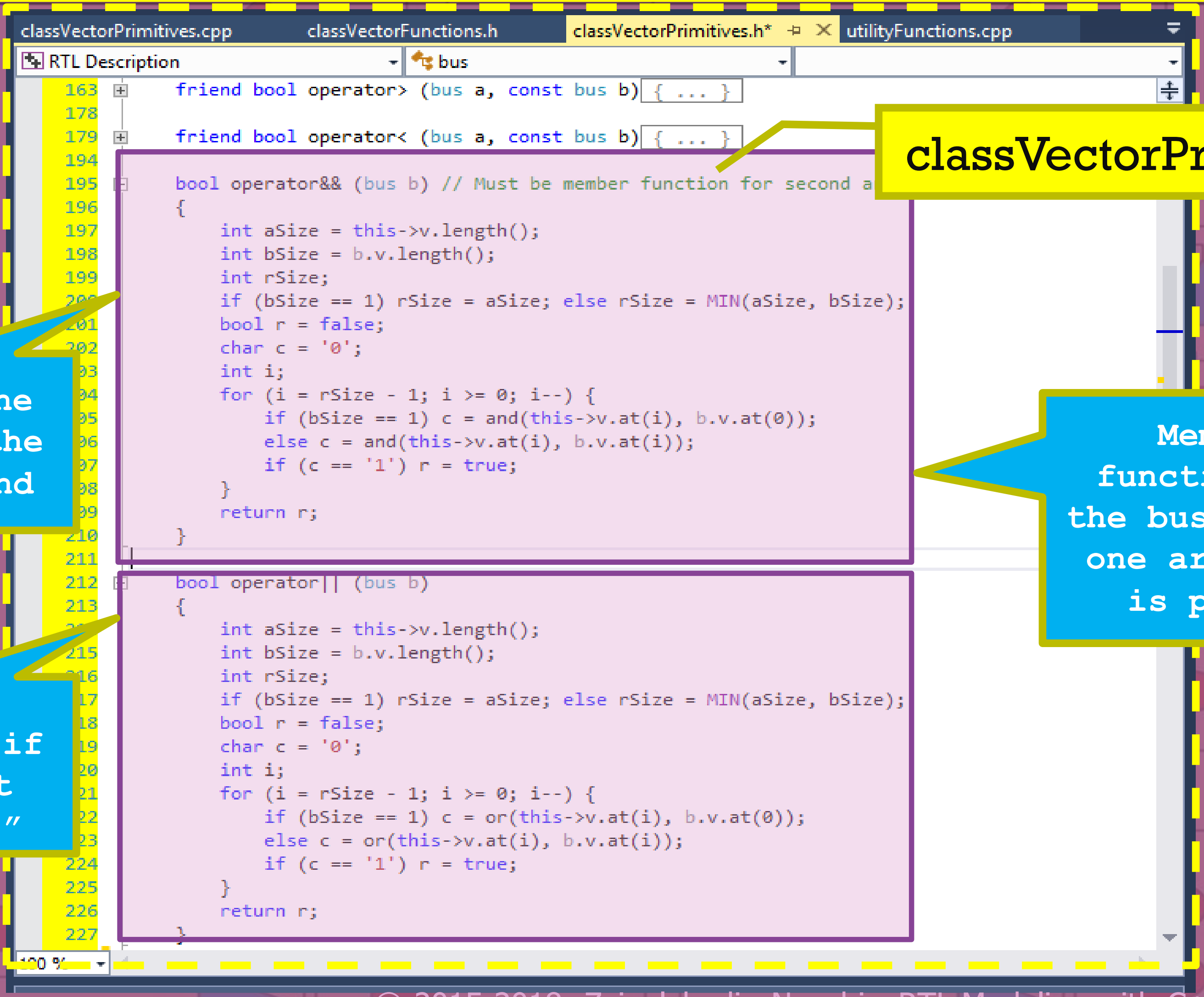
Rational Operations



```
    friend bool operator== (bus a, const bus b)
    {
        return (a.v == b.v);
    }
    friend bool operator> (bus a, const bus b) // Assume same size
    {
        int aSize = a.v.length();
        int bSize = b.v.length();
        bool r = false;
        int i = 0;
        do{
            if ((a.v[i] == '1') && (b.v[i] == '0')){
                r = true; break;
            }
            else if ((a.v[i] == '0') && (b.v[i] == '1')){
                r = false; break;
            }
        } while (++i < aSize);
        return r;
    }
    friend bool operator< (bus a, const bus b) // Assume same size
    {
        int aSize = a.v.length();
        int bSize = b.v.length();
        bool r = false;
        int i = 0;
        do{
            if ((a.v[i] == '0') && (b.v[i] == '1')){
                r = true; break;
            }
            else if ((a.v[i] == '1') && (b.v[i] == '0')){
                r = false; break;
            }
        } while (++i < aSize);
        return r;
    }
```

classVectorPrimitives.h

Logical Operations



```
    friend bool operator> (bus a, const bus b) { ... }

    friend bool operator< (bus a, const bus b) { ... }

    bool operator&& (bus b) // Must be member function for second a
    {
        int aSize = this->v.length();
        int bSize = b.v.length();
        int rSize;
        if (bSize == 1) rSize = aSize; else rSize = MIN(aSize, bSize);
        bool r = false;
        char c = '0';
        int i;
        for (i = rSize - 1; i >= 0; i--) {
            if (bSize == 1) c = and(this->v.at(i), b.v.at(0));
            else c = and(this->v.at(i), b.v.at(i));
            if (c == '1') r = true;
        }
        return r;
    }

    bool operator|| (bus b)
    {
        int aSize = this->v.length();
        int bSize = b.v.length();
        int rSize;
        if (bSize == 1) rSize = aSize; else rSize = MIN(aSize, bSize);
        bool r = false;
        char c = '0';
        int i;
        for (i = rSize - 1; i >= 0; i--) {
            if (bSize == 1) c = or(this->v.at(i), b.v.at(0));
            else c = or(this->v.at(i), b.v.at(i));
            if (c == '1') r = true;
        }
        return r;
    }
```

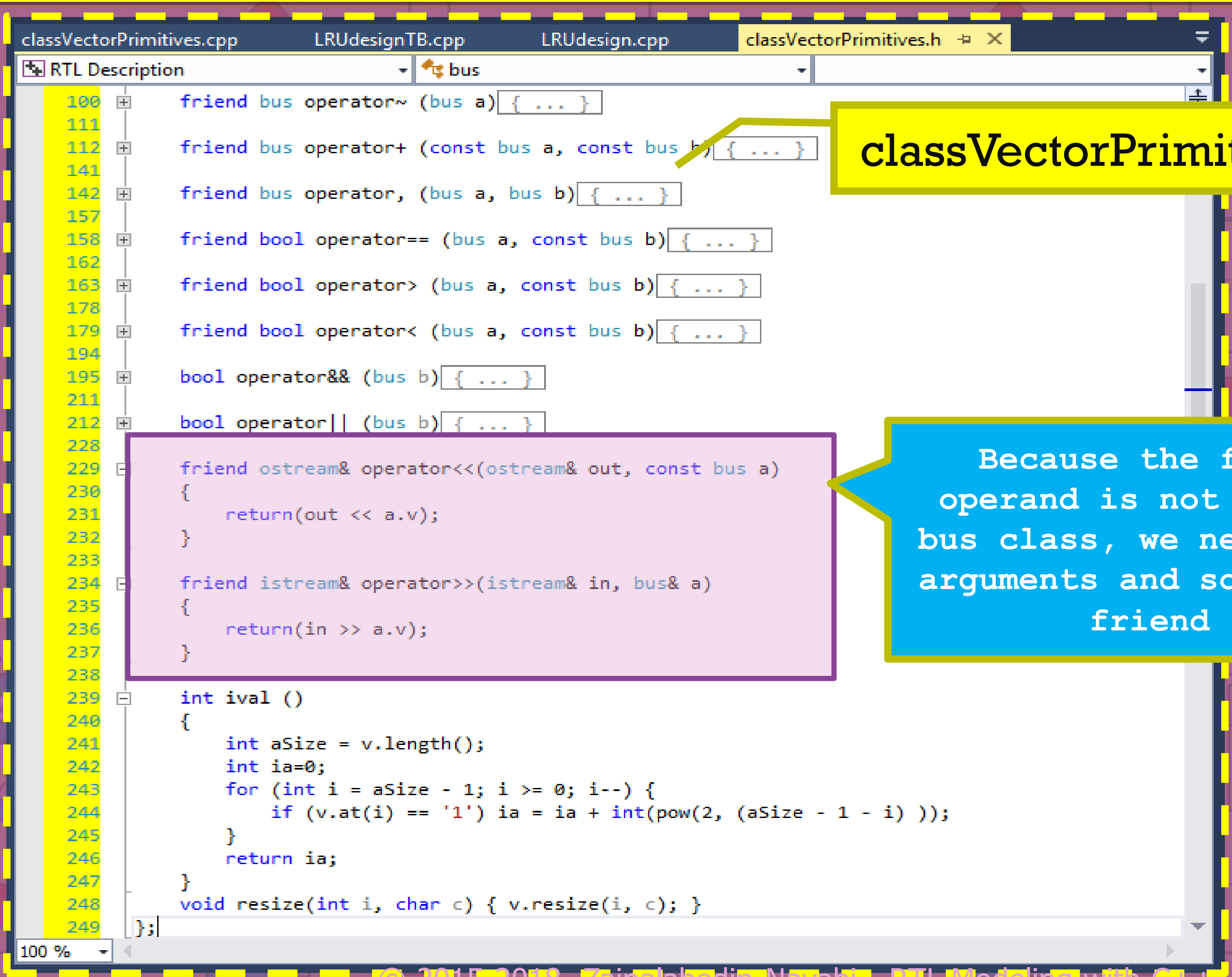
classVectorPrimitives.h

“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



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

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

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

    friend bool operator== (bus a, const bus b){ ... }

    friend bool operator> (bus a, const bus b){ ... }

    friend bool operator< (bus a, const bus b){ ... }

    bool operator&& (bus b){ ... }

    bool operator|| (bus b){ ... }

    friend ostream& operator<<(ostream& out, const bus a)
    {
        return(out << a.v);
    }

    friend istream& operator>>(istream& in, bus& a)
    {
        return(in >> a.v);
    }

    int ival ()
    {
        int aSize = v.length();
        int ia=0;
        for (int i = aSize - 1; i >= 0; i--) {
            if (v.at(i) == '1') ia = ia + int(pow(2, (aSize - 1 - i) ));
        }
        return ia;
    }
    void resize(int i, char c) { v.resize(i, c); }
};
```

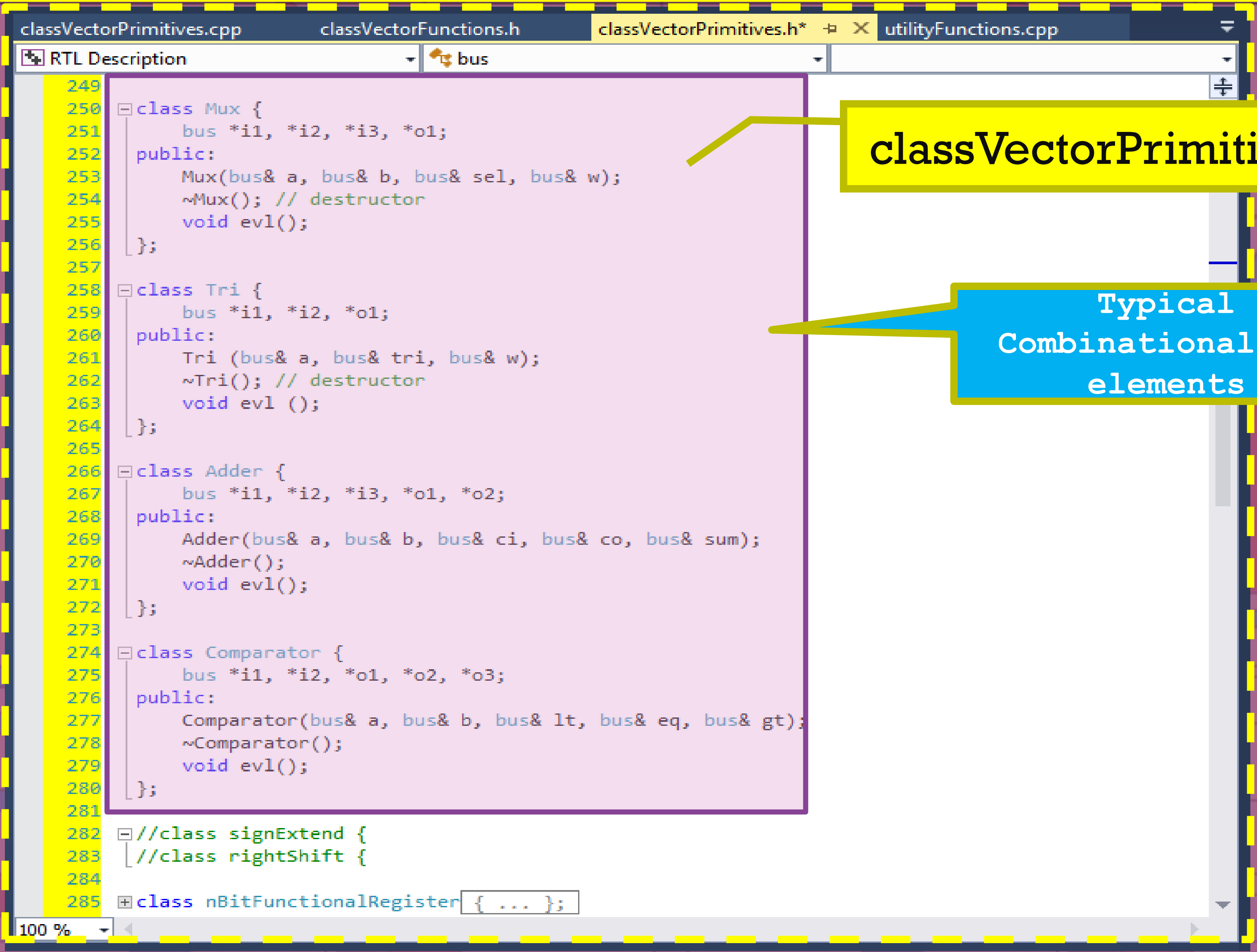
classVectorPrimitives.h

Because the first operand is not of the bus class, we need both arguments and so we use friend

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

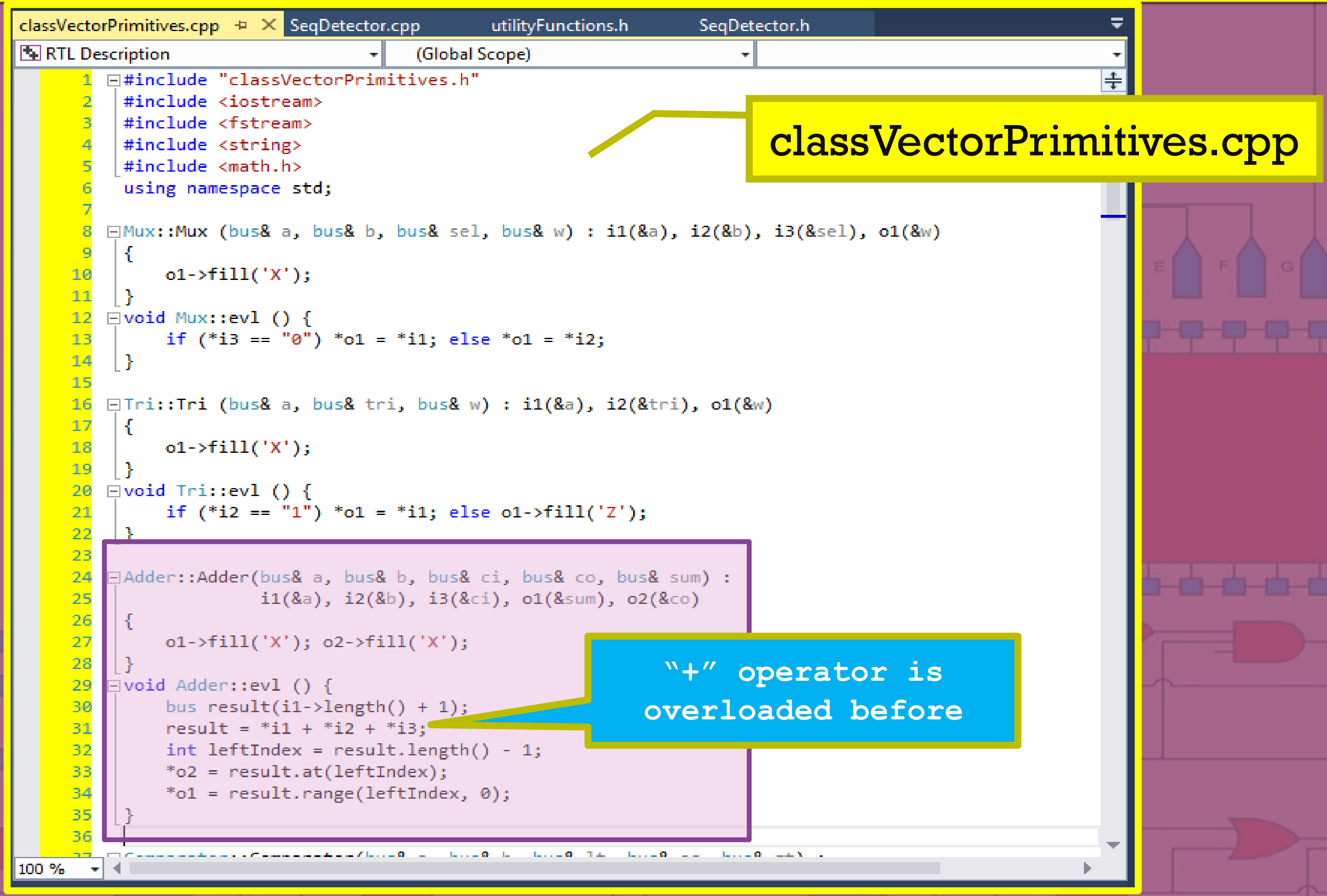


```
class Mux {
    bus *i1, *i2, *i3, *o1;
public:
    Mux(bus& a, bus& b, bus& sel, bus& w);
    ~Mux(); // destructor
    void evl();
};

class Tri {
    bus *i1, *i2, *o1;
public:
    Tri (bus& a, bus& tri, bus& w);
    ~Tri(); // destructor
    void evl ();
};

class Adder {
    bus *i1, *i2, *i3, *o1, *o2;
public:
    Adder(bus& a, bus& b, bus& ci, bus& co, bus& sum);
    ~Adder();
    void evl();
};

class Comparator {
    bus *i1, *i2, *o1, *o2, *o3;
public:
    Comparator(bus& a, bus& b, bus& lt, bus& eq, bus& gt);
    ~Comparator();
    void evl();
};

//class signExtend {
//class rightShift {

class nBitFunctionalRegister { ... };
```

classVectorPrimitives.h

Typical Combinational RTL elements

Combinational Elements



```
#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);
}

Comparator::Comparator(bus& a, bus& b, bus& lt, bus& eq, bus& gt) :
```

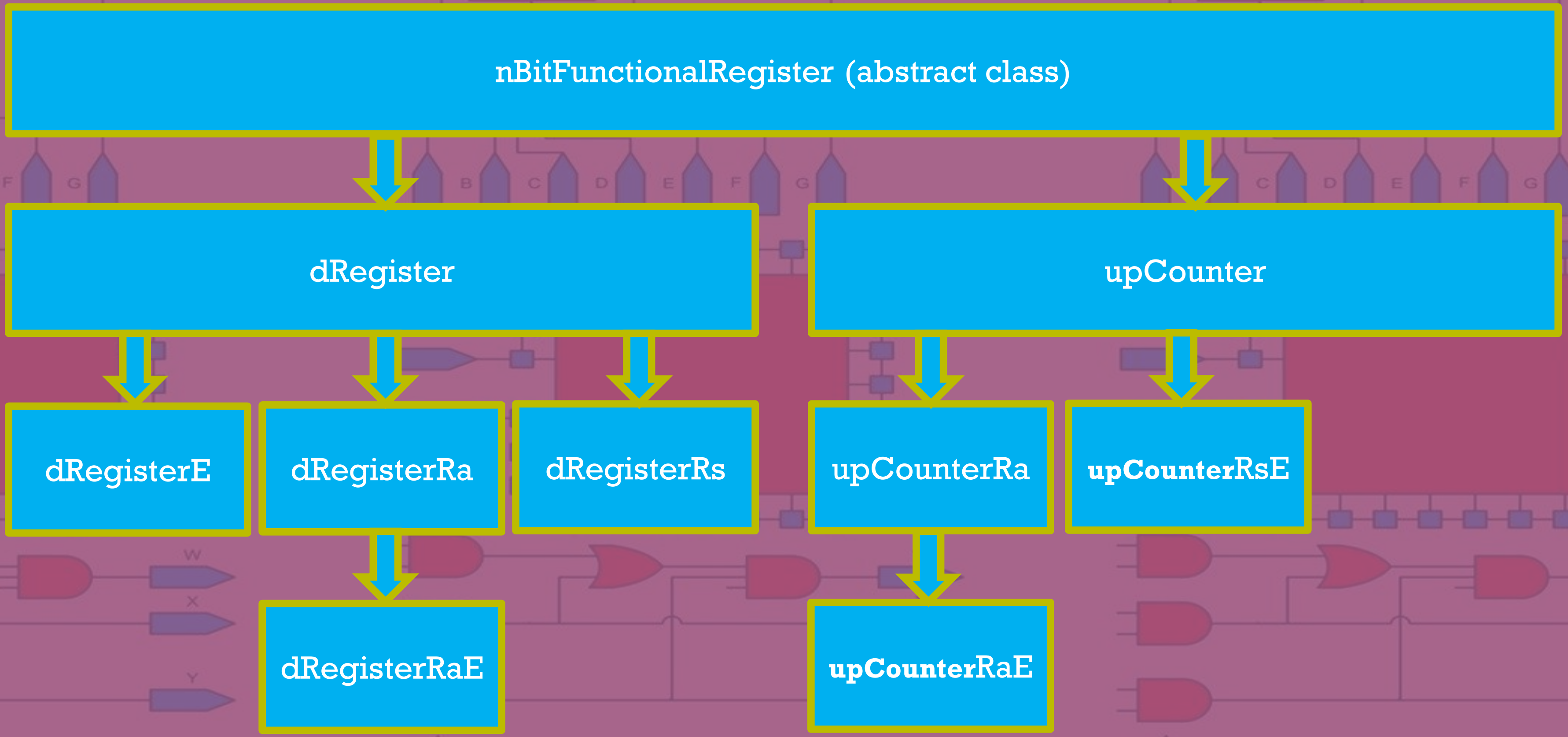
classVectorPrimitives.cpp

"+" 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



nBitFunctionalRegister (abstract class)

dRegister

upCounter

dRegisterE

dRegisterRa

dRegisterRs

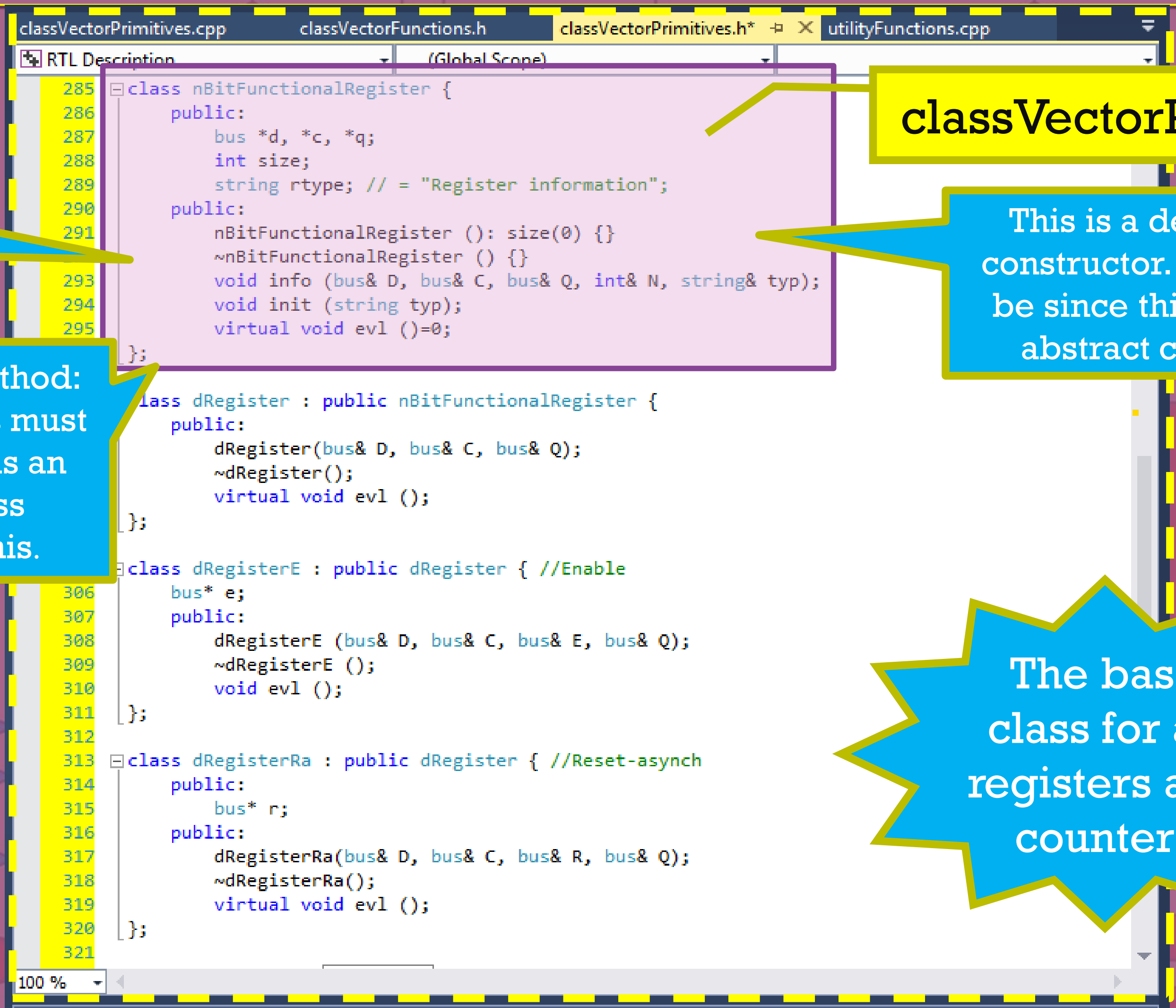
upCounterRa

upCounterRsE

dRegisterRaE

upCounterRaE

Functional Registers



```
class nBitFunctionalRegister {
    public:
        bus *d, *c, *q;
        int size;
        string rtype; // = "Register information";
    public:
        nBitFunctionalRegister (): size(0) {}
        ~nBitFunctionalRegister () {}
        void info (bus& D, bus& C, bus& Q, int& N, string& typ);
        void init (string typ);
        virtual void evl ()=0;
};

lass dRegister : public nBitFunctionalRegister {
    public:
        dRegister(bus& D, bus& C, bus& Q);
        ~dRegister();
        virtual void evl ();
};

class dRegisterE : public dRegister { //Enable
    bus* e;
    public:
        dRegisterE (bus& D, bus& C, bus& E, bus& Q);
        ~dRegisterE ();
        void evl ();
};

class dRegisterRa : public dRegister { //Reset-asynch
    public:
        bus* r;
    public:
        dRegisterRa(bus& D, bus& C, bus& R, bus& Q);
        ~dRegisterRa();
        virtual void evl ();
};
```

classVectorPrimitives.h

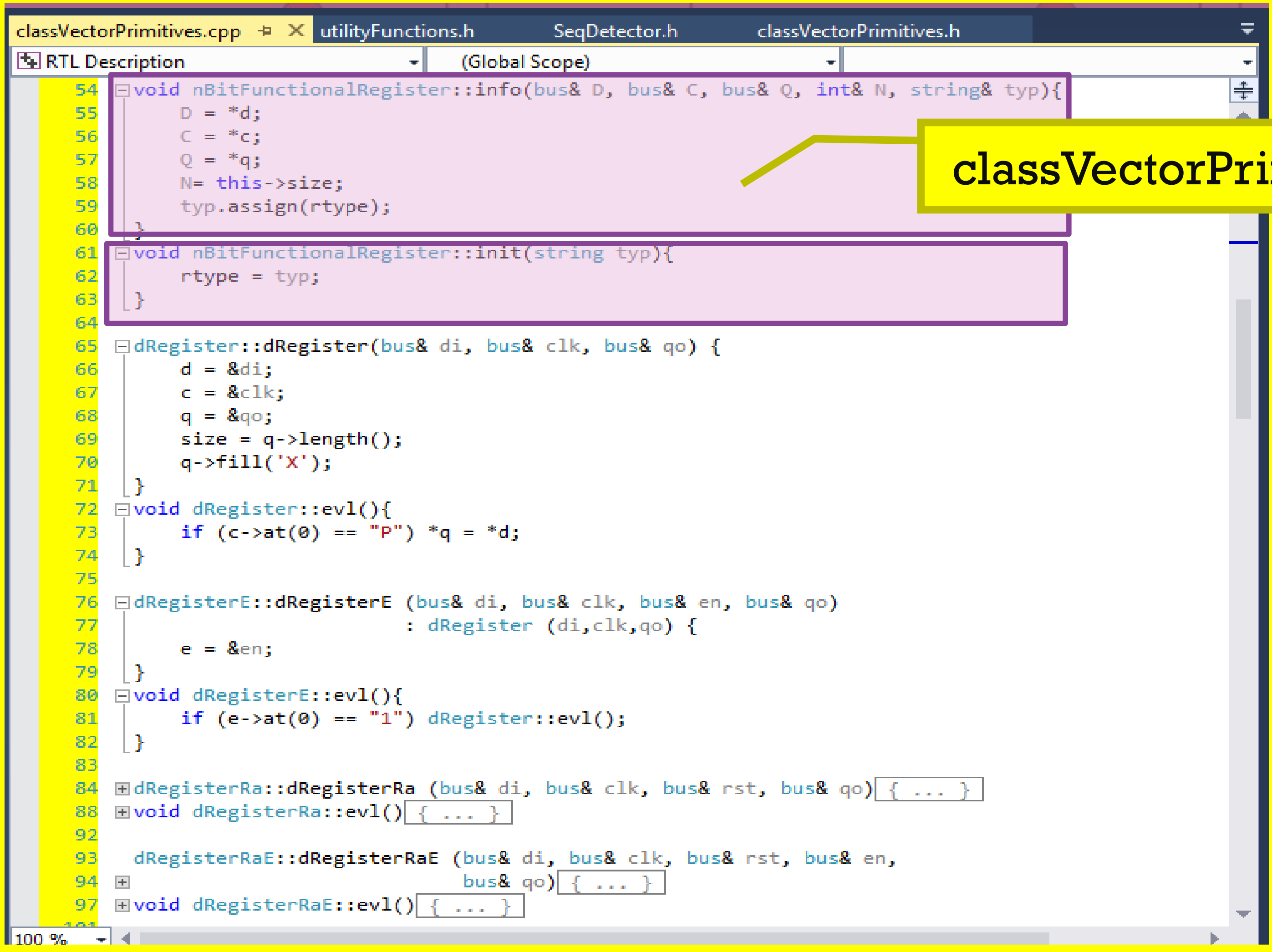
Initializing member variables.

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

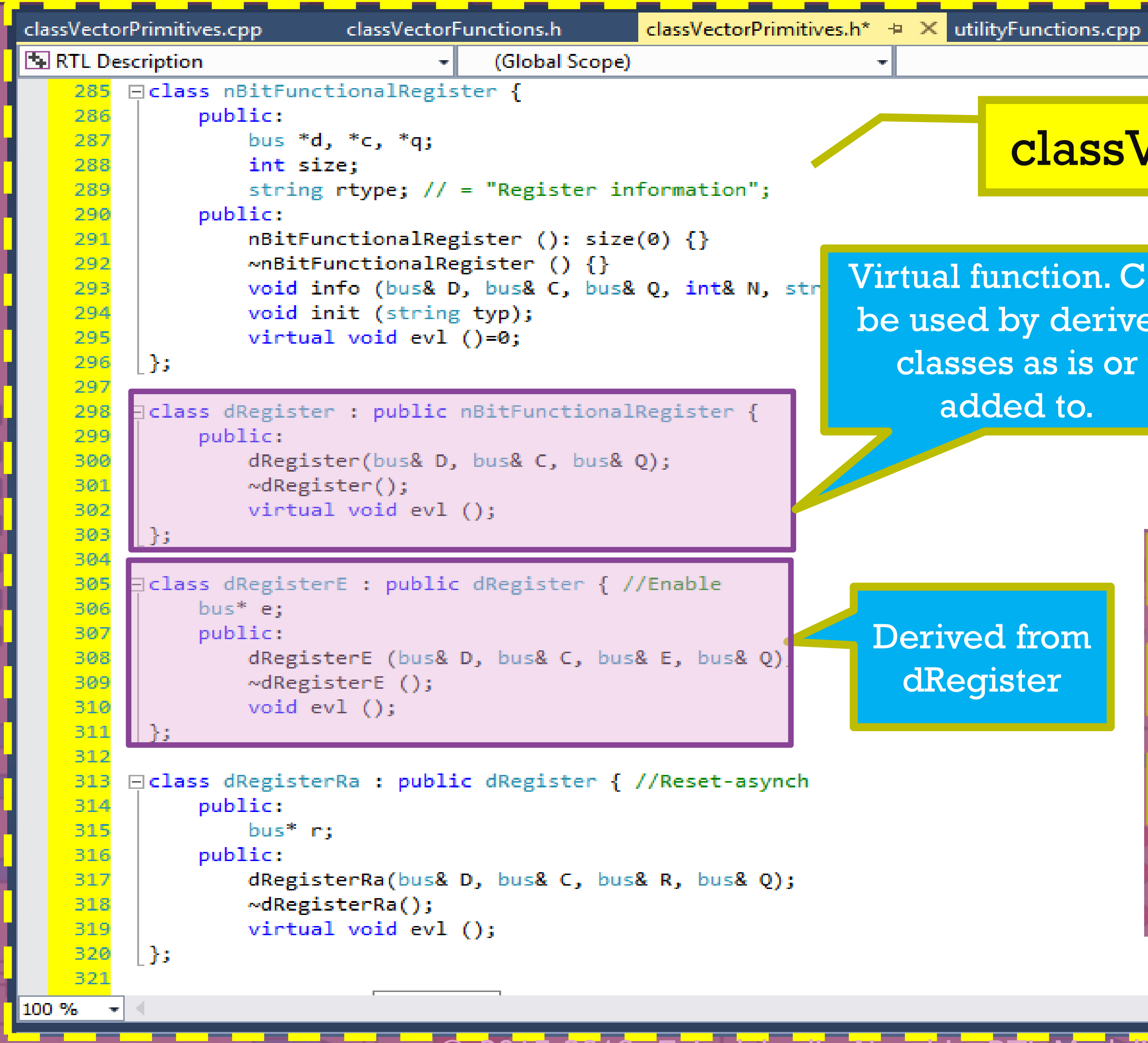
Functional Registers



```
void nBitFunctionalRegister::info(bus& D, bus& C, bus& Q, int& N, string& typ){
    D = *d;
    C = *c;
    Q = *q;
    N= this->size;
    typ.assign(rtype);
}
void nBitFunctionalRegister::init(string typ){
    rtype = typ;
}

dRegister::dRegister(bus& di, bus& clk, bus& qo) {
    d = &di;
    c = &clk;
    q = &qo;
    size = q->length();
    q->fill('X');
}
void dRegister::evl(){
    if (c->at(0) == "P") *q = *d;
}

dRegisterE::dRegisterE (bus& di, bus& clk, bus& en, bus& qo)
                        : dRegister (di,clk,qo) {
    e = &en;
}
void dRegisterE::evl(){
    if (e->at(0) == "1") dRegister::evl();
}

dRegisterRa::dRegisterRa (bus& di, bus& clk, bus& rst, bus& qo){ ... }
void dRegisterRa::evl(){ ... }

dRegisterRaE::dRegisterRaE (bus& di, bus& clk, bus& rst, bus& en,
                            bus& qo){ ... }
void dRegisterRaE::evl(){ ... }
```

classVectorPrimitives.cpp

DRegister

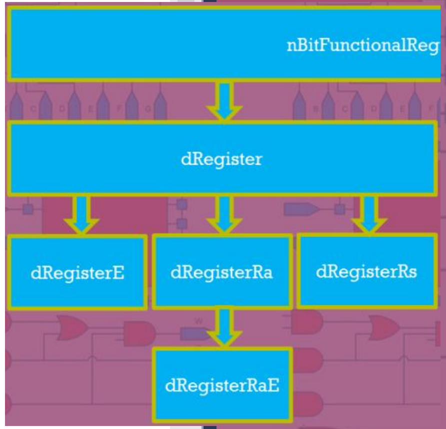


```
class nBitFunctionalRegister {
    public:
        bus *d, *c, *q;
        int size;
        string rtype; // = "Register information";
    public:
        nBitFunctionalRegister (): size(0) {}
        ~nBitFunctionalRegister () {}
        void info (bus& D, bus& C, bus& Q, int& N, str
        void init (string typ);
        virtual void evl ()=0;
};

class dRegister : public nBitFunctionalRegister {
    public:
        dRegister(bus& D, bus& C, bus& Q);
        ~dRegister();
        virtual void evl ();
};

class dRegisterE : public dRegister { //Enable
    bus* e;
    public:
        dRegisterE (bus& D, bus& C, bus& E, bus& Q)
        ~dRegisterE ();
        void evl ();
};

class dRegisterRa : public dRegister { //Reset-asynch
    public:
        bus* r;
    public:
        dRegisterRa(bus& D, bus& C, bus& R, bus& Q);
        ~dRegisterRa();
        virtual void evl ();
};
```

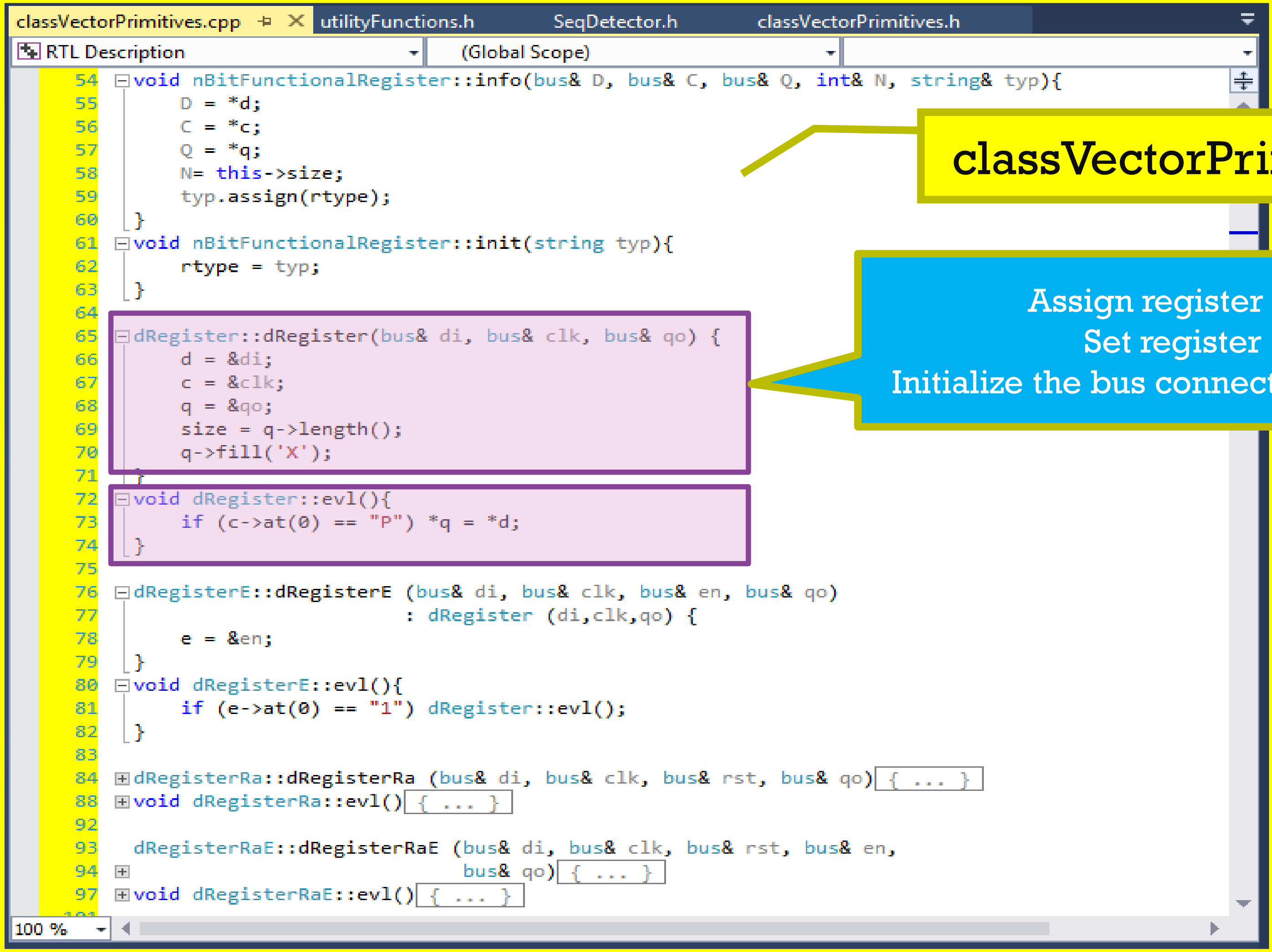
classVectorPrimitives.h

Virtual function. Can be used by derived classes as is or added to.

Derived from dRegister



DRegister

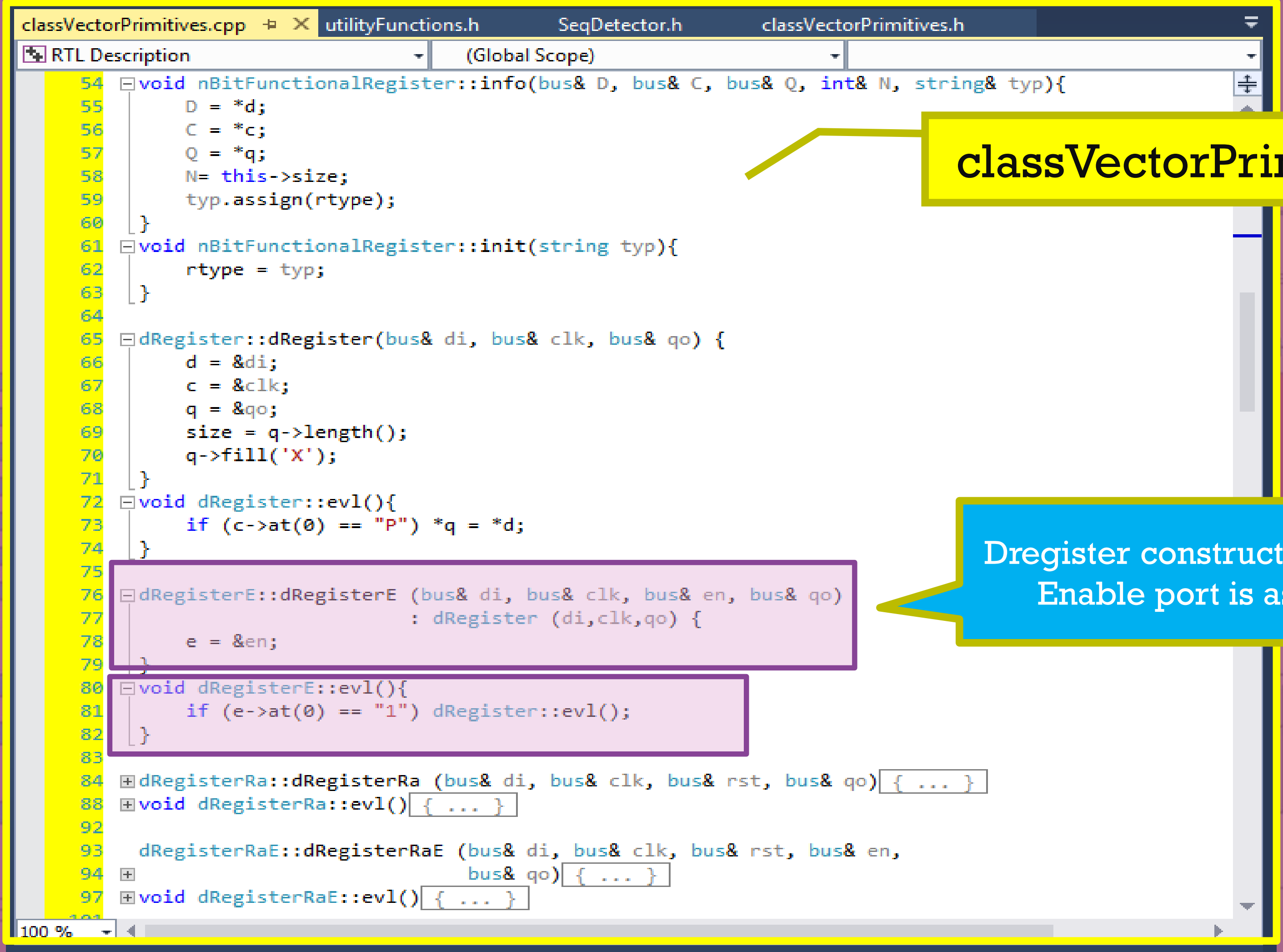


```
void nBitFunctionalRegister::info(bus& D, bus& C, bus& Q, int& N, string& typ){
    D = *d;
    C = *c;
    Q = *q;
    N= this->size;
    typ.assign(rtype);
}
void nBitFunctionalRegister::init(string typ){
    rtype = typ;
}

dRegister::dRegister(bus& di, bus& clk, bus& qo) {
    d = &di;
    c = &clk;
    q = &qo;
    size = q->length();
    q->fill('X');
}
void dRegister::evl(){
    if (c->at(0) == "P") *q = *d;
}

dRegisterE::dRegisterE (bus& di, bus& clk, bus& en, bus& qo)
                      : dRegister (di,clk,qo) {
    e = &en;
}
void dRegisterE::evl(){
    if (e->at(0) == "1") dRegister::evl();
}

dRegisterRa::dRegisterRa (bus& di, bus& clk, bus& rst, bus& qo){ ... }
void dRegisterRa::evl(){ ... }

dRegisterRaE::dRegisterRaE (bus& di, bus& clk, bus& rst, bus& en,
                            bus& qo){ ... }
void dRegisterRaE::evl(){ ... }
```

classVectorPrimitives.cpp

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

DRegisterE

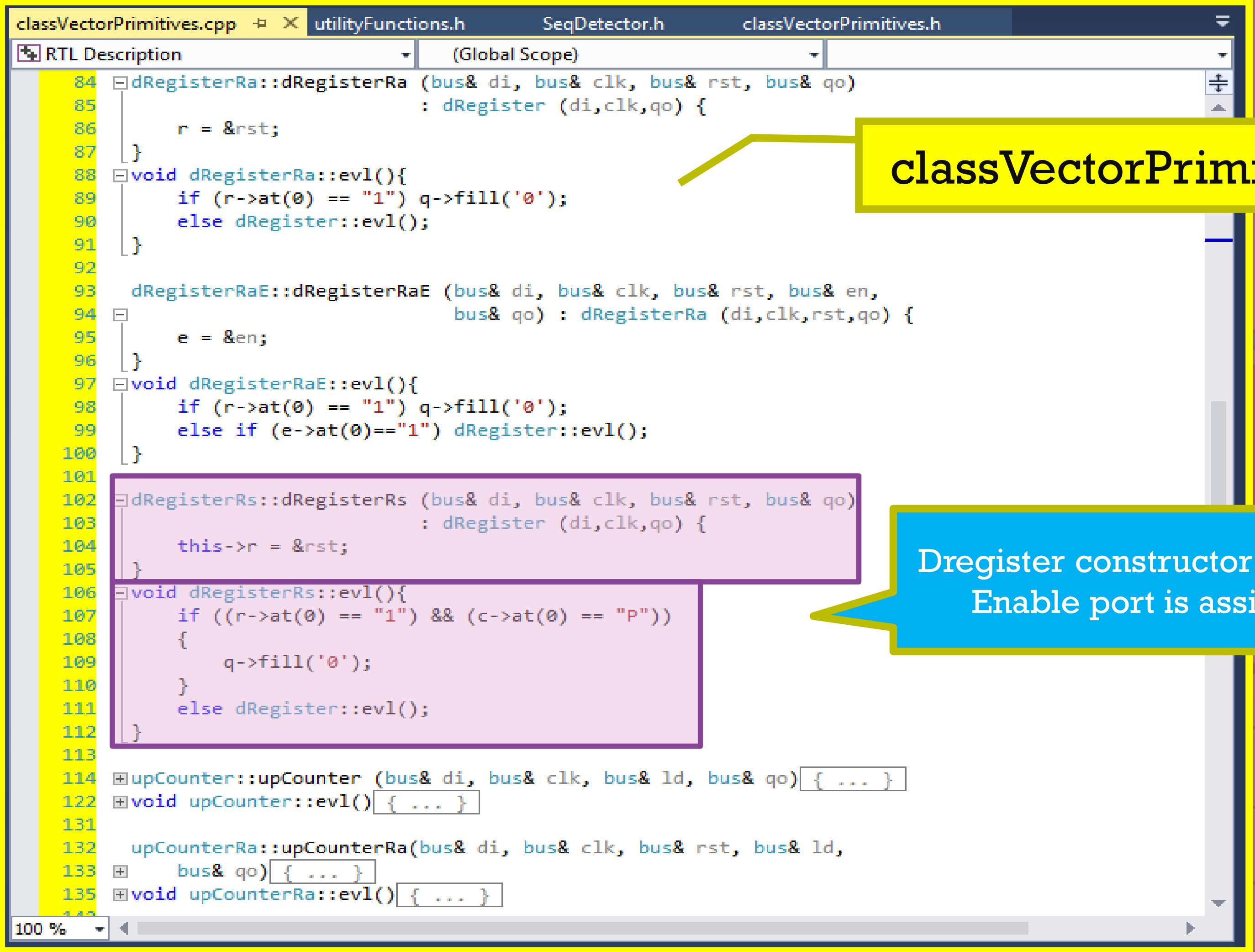


```
void nBitFunctionalRegister::info(bus& D, bus& C, bus& Q, int& N, string& typ){
    D = *d;
    C = *c;
    Q = *q;
    N= this->size;
    typ.assign(rtype);
}
void nBitFunctionalRegister::init(string typ){
    rtype = typ;
}

dRegister::dRegister(bus& di, bus& clk, bus& qo) {
    d = &di;
    c = &clk;
    q = &qo;
    size = q->length();
    q->fill('X');
}
void dRegister::evl(){
    if (c->at(0) == "P") *q = *d;
}

dRegisterE::dRegisterE (bus& di, bus& clk, bus& en, bus& qo)
                    : dRegister (di,clk,qo) {
    e = &en;
}
void dRegisterE::evl(){
    if (e->at(0) == "1") dRegister::evl();
}

dRegisterRa::dRegisterRa (bus& di, bus& clk, bus& rst, bus& qo){ ... }
void dRegisterRa::evl(){ ... }

dRegisterRaE::dRegisterRaE (bus& di, bus& clk, bus& rst, bus& en,
                            bus& qo){ ... }
void dRegisterRaE::evl(){ ... }
```

classVectorPrimitives.cpp

Dregister constructor is called
Enable port is assigned

DRegisterRS



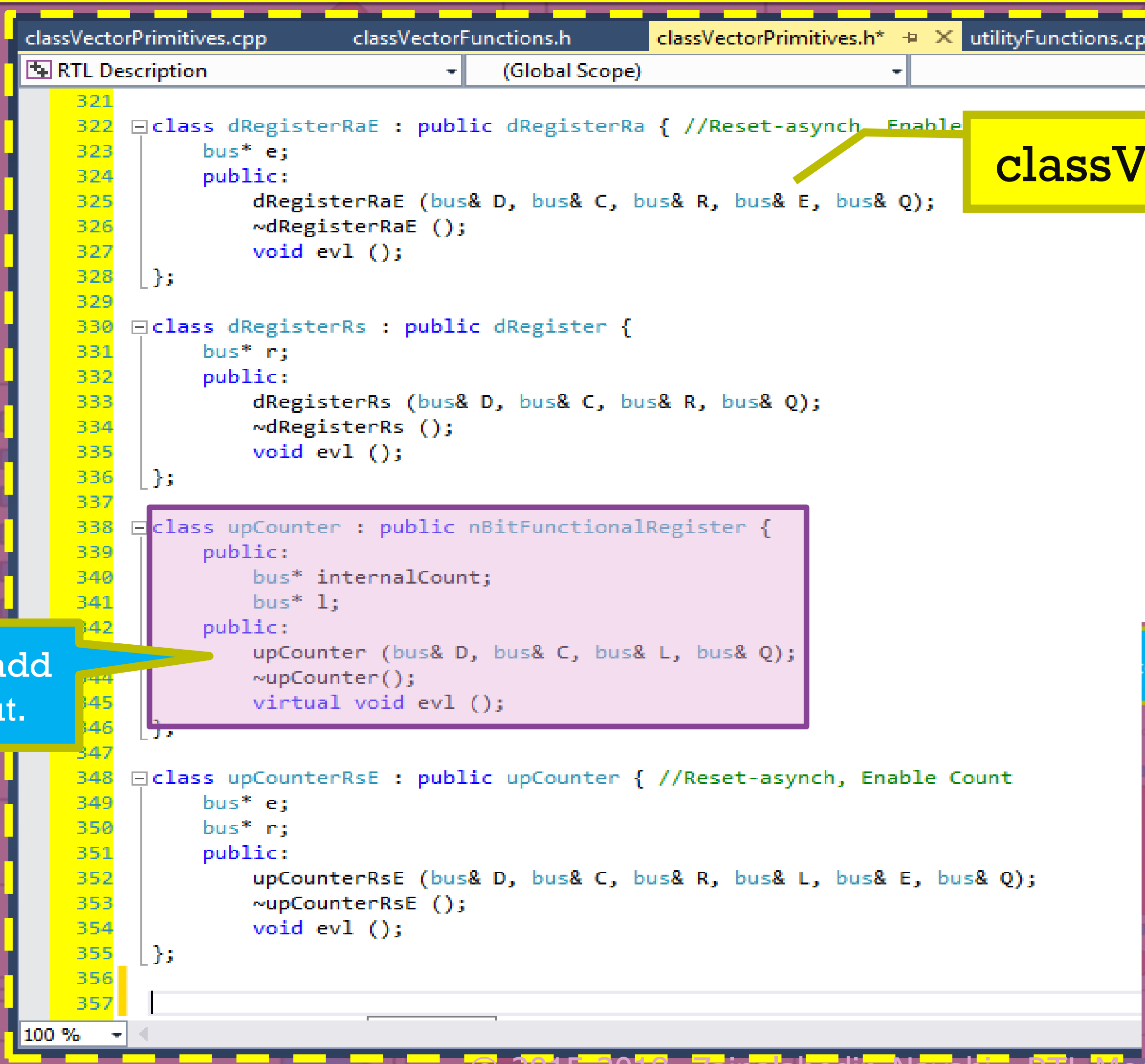
```
dRegisterRa::dRegisterRa (bus& di, bus& clk, bus& rst, bus& qo)
                          : dRegister (di,clk,qo) {
    r = &rst;
}
void dRegisterRa::evl(){
    if (r->at(0) == "1") q->fill('0');
    else dRegister::evl();
}

dRegisterRaE::dRegisterRaE (bus& di, bus& clk, bus& rst, bus& en,
                            bus& qo) : dRegisterRa (di,clk,rst,qo) {
    e = &en;
}
void dRegisterRaE::evl(){
    if (r->at(0) == "1") q->fill('0');
    else if (e->at(0)=="1") dRegister::evl();
}

dRegisterRs::dRegisterRs (bus& di, bus& clk, bus& rst, bus& qo)
                          : dRegister (di,clk,qo) {
    this->r = &rst;
}
void dRegisterRs::evl(){
    if ((r->at(0) == "1") && (c->at(0) == "P"))
    {
        q->fill('0');
    }
    else dRegister::evl();
}

upCounter::upCounter (bus& di, bus& clk, bus& ld, bus& qo){ ... }
void upCounter::evl(){ ... }

upCounterRa::upCounterRa(bus& di, bus& clk, bus& rst, bus& ld,
    bus& qo){ ... }
void upCounterRa::evl(){ ... }
```

classVectorPrimitives.cpp

Dregister constructor is called
Enable port is assigned

© 2015-2018, Zainalabedin Navabi - RTL Modeling with C++

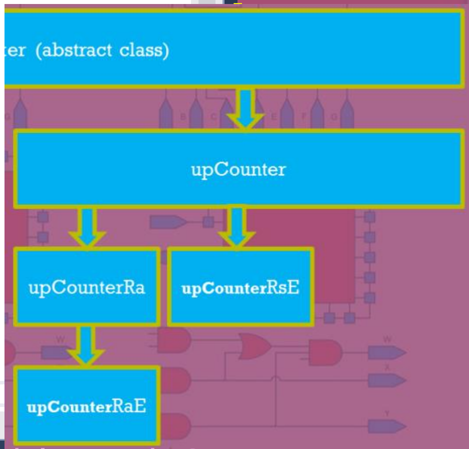
UpCounter



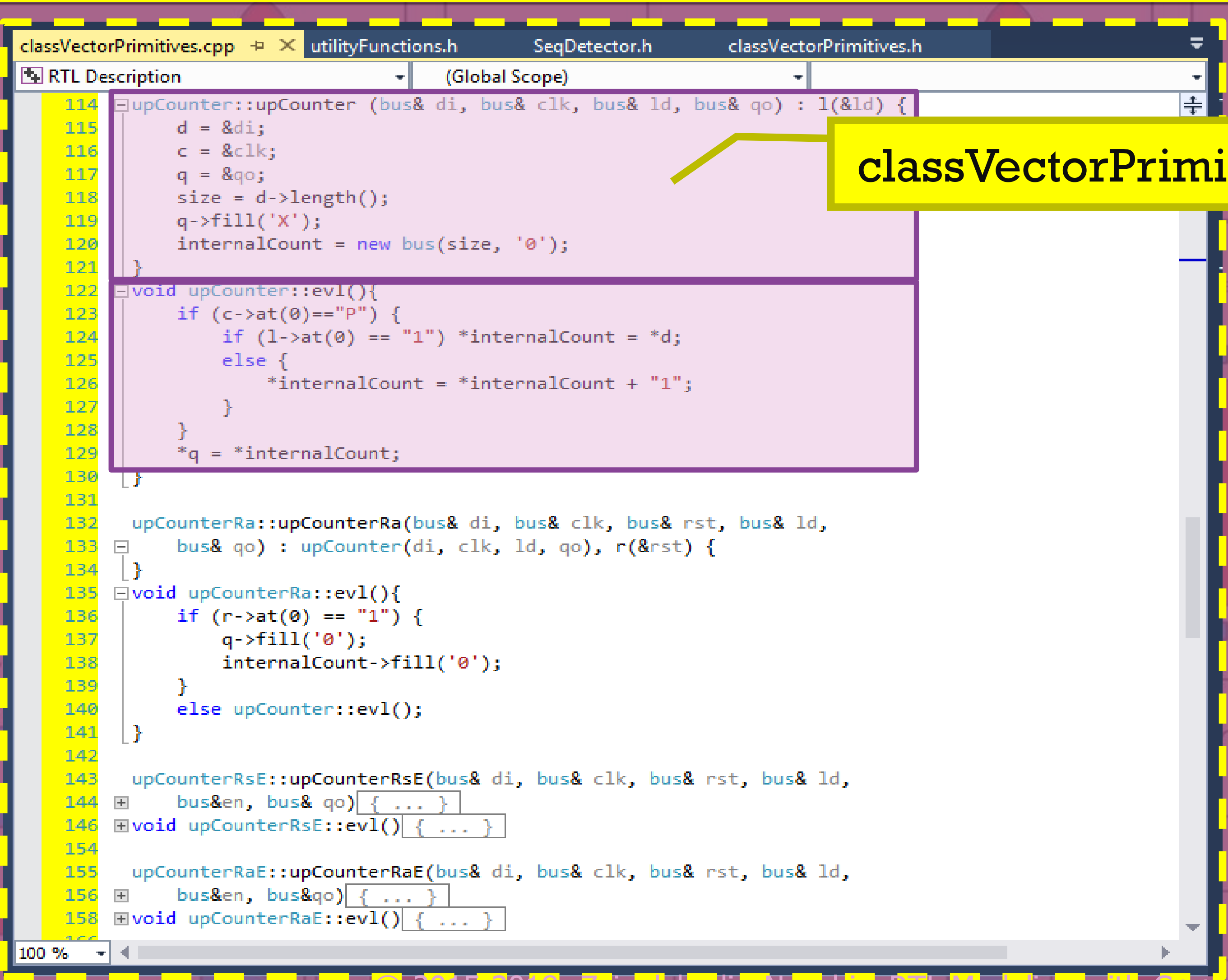
```
class dRegisterRaE : public dRegisterRa { //Reset-asynch, Enable
    bus* e;
    public:
        dRegisterRaE (bus& D, bus& C, bus& R, bus& E, bus& Q);
        ~dRegisterRaE ();
        void evl ();
};

class dRegisterRs : public dRegister {
    bus* r;
    public:
        dRegisterRs (bus& D, bus& C, bus& R, bus& Q);
        ~dRegisterRs ();
        void evl ();
};

class upCounter : public nBitFunctionalRegister {
    public:
        bus* internalCount;
        bus* l;
    public:
        upCounter (bus& D, bus& C, bus& L, bus& Q);
        ~upCounter();
        virtual void evl ();
};

class upCounterRsE : public upCounter { //Reset-asynch, Enable Count
    bus* e;
    bus* r;
    public:
        upCounterRsE (bus& D, bus& C, bus& R, bus& L, bus& E, bus& Q);
        ~upCounterRsE ();
        void evl ();
};
```

classVectorPrimitives.h

Counters add load input.



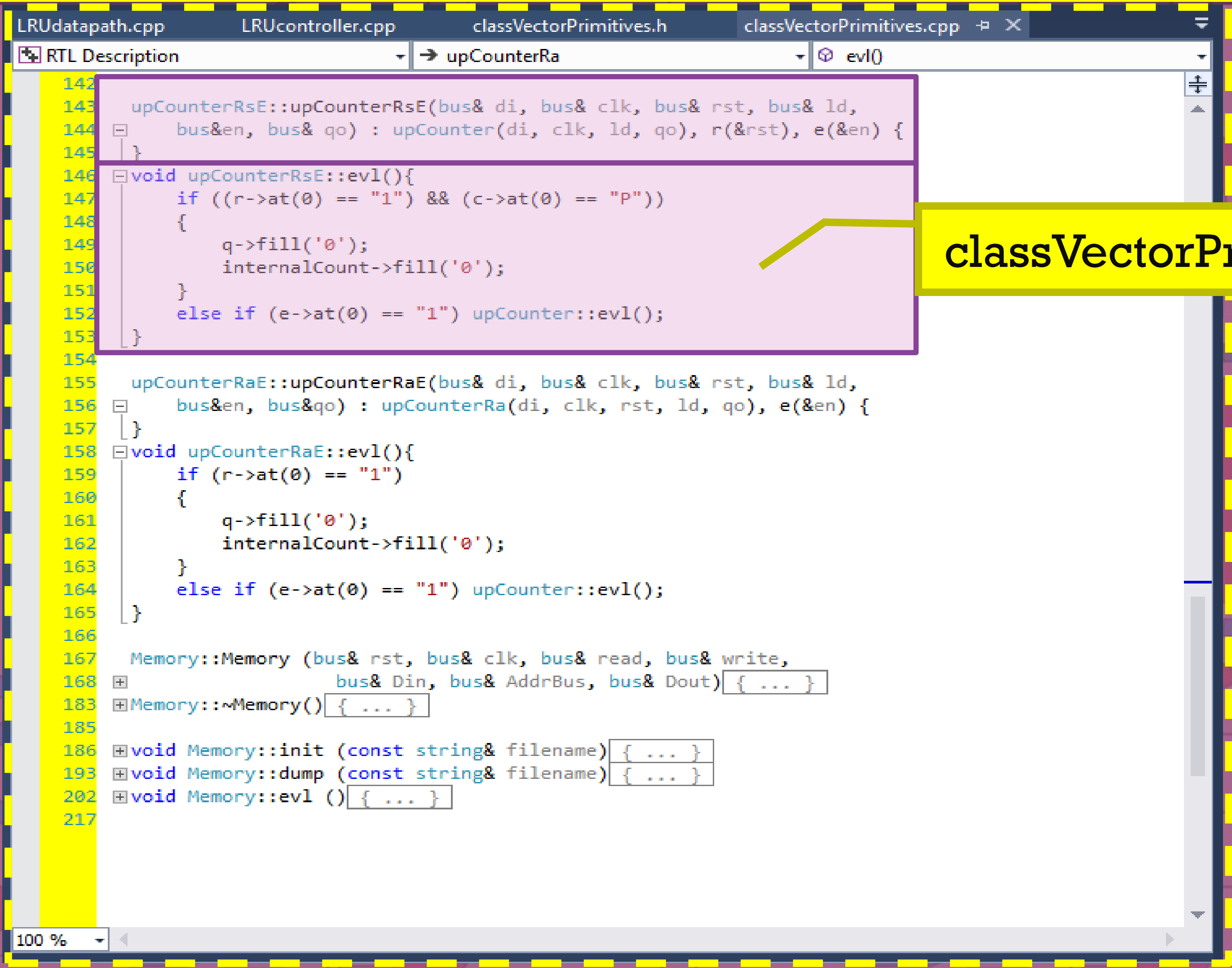
UpCounter



```cpp
upCounter::upCounter (bus& di, bus& clk, bus& ld, bus& qo) : l(&ld) {
    d = &di;
    c = &clk;
    q = &qo;
    size = d->length();
    q->fill('X');
    internalCount = new bus(size, '0');
}
void upCounter::evl(){
    if (c->at(0)=="P") {
        if (l->at(0) == "1") *internalCount = *d;
        else {
            *internalCount = *internalCount + "1";
        }
    }
    *q = *internalCount;
}

upCounterRa::upCounterRa(bus& di, bus& clk, bus& rst, bus& ld,
    bus& qo) : upCounter(di, clk, ld, qo), r(&rst) {
}
void upCounterRa::evl(){
    if (r->at(0) == "1") {
        q->fill('0');
        internalCount->fill('0');
    }
    else upCounter::evl();
}

upCounterRsE::upCounterRsE(bus& di, bus& clk, bus& rst, bus& ld,
    bus&en, bus& qo) { ... }
void upCounterRsE::evl() { ... }

upCounterRaE::upCounterRaE(bus& di, bus& clk, bus& rst, bus& ld,
    bus&en, bus&qo) { ... }
void upCounterRaE::evl() { ... }
```

classVectorPrimitives.h

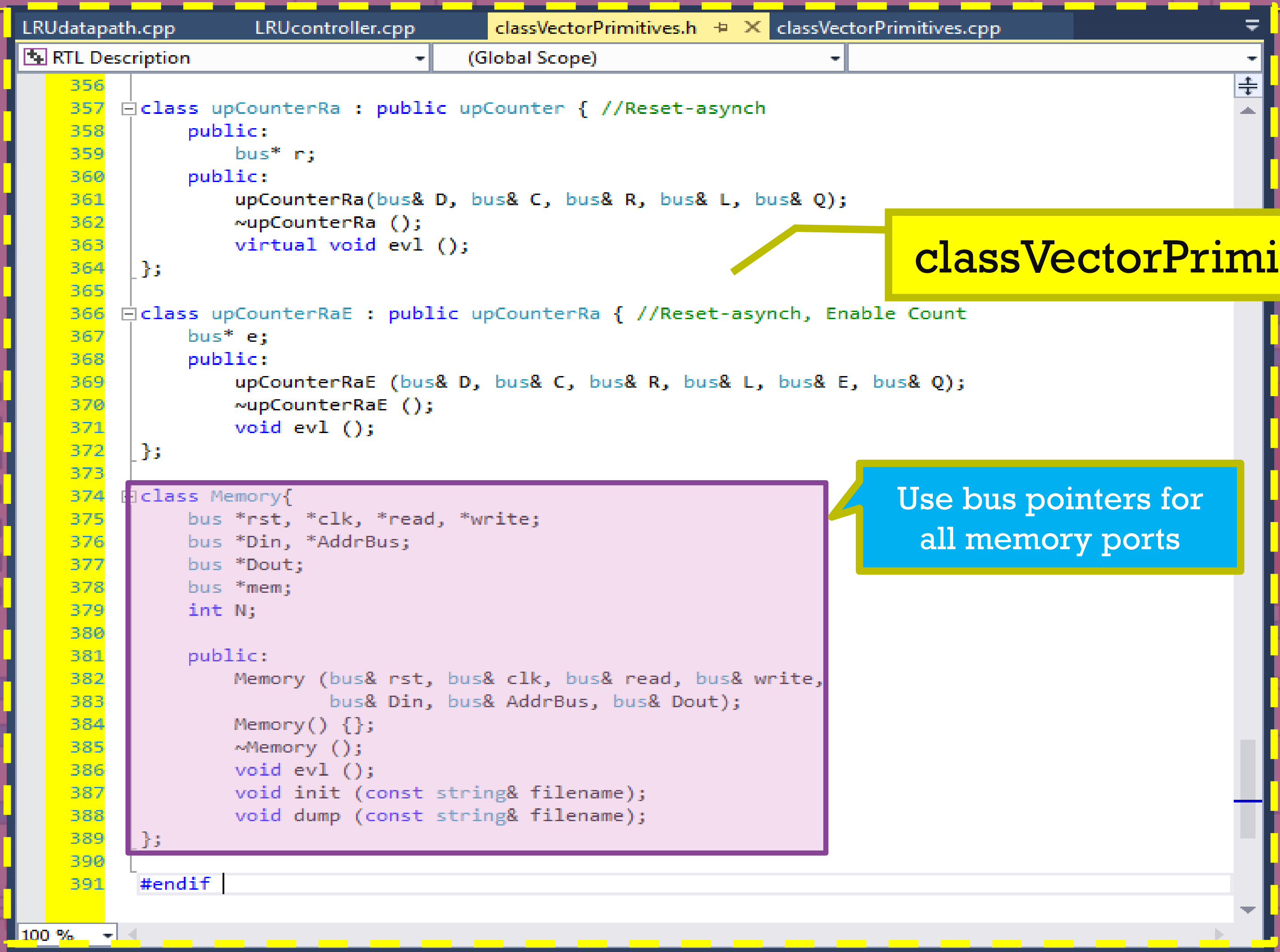
UpCounterRsE



```
upCounterRsE::upCounterRsE(bus& di, bus& clk, bus& rst, bus& ld,
    bus&en, bus& qo) : upCounter(di, clk, ld, qo), r(&rst), e(&en) {
}
void upCounterRsE::evl(){
    if ((r->at(0) == "1") && (c->at(0) == "P"))
    {
        q->fill('0');
        internalCount->fill('0');
    }
    else if (e->at(0) == "1") upCounter::evl();
}

upCounterRaE::upCounterRaE(bus& di, bus& clk, bus& rst, bus& ld,
    bus&en, bus&qo) : upCounterRa(di, clk, rst, ld, qo), e(&en) {
}
void upCounterRaE::evl(){
    if (r->at(0) == "1")
    {
        q->fill('0');
        internalCount->fill('0');
    }
    else if (e->at(0) == "1") upCounter::evl();
}

Memory::Memory (bus& rst, bus& clk, bus& read, bus& write,
                  bus& Din, bus& AddrBus, bus& Dout){ ... }
Memory::~Memory(){ ... }

void Memory::init (const string& filename){ ... }
void Memory::dump (const string& filename){ ... }
void Memory::evl (){ ... }
```

classVectorPrimitives.h

Memory Structure

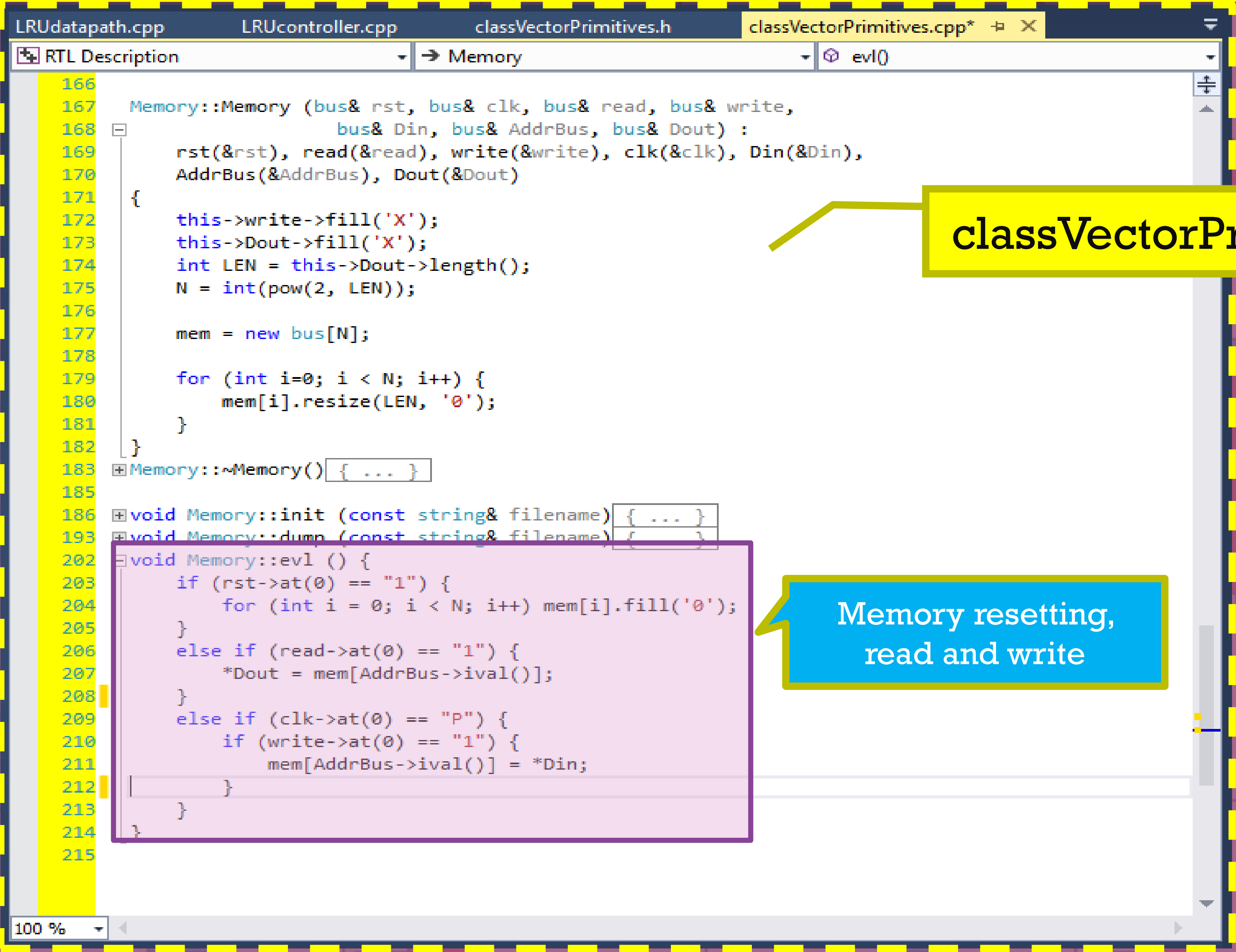


```
class upCounterRa : public upCounter { //Reset-asynch
    public:
        bus* r;
    public:
        upCounterRa(bus& D, bus& C, bus& R, bus& L, bus& Q);
        ~upCounterRa ();
        virtual void evl ();
};

class upCounterRaE : public upCounterRa { //Reset-asynch, Enable Count
    bus* e;
    public:
        upCounterRaE (bus& D, bus& C, bus& R, bus& L, bus& E, bus& Q);
        ~upCounterRaE ();
        void evl ();
};

class Memory{
    bus *rst, *clk, *read, *write;
    bus *Din, *AddrBus;
    bus *Dout;
    bus *mem;
    int N;

    public:
        Memory (bus& rst, bus& clk, bus& read, bus& write,
                bus& Din, bus& AddrBus, bus& Dout);
        Memory() {};
        ~Memory ();
        void evl ();
        void init (const string& filename);
        void dump (const string& filename);
};

#endif
```

classVectorPrimitives.h

Use bus pointers for all memory ports

Memory Structure

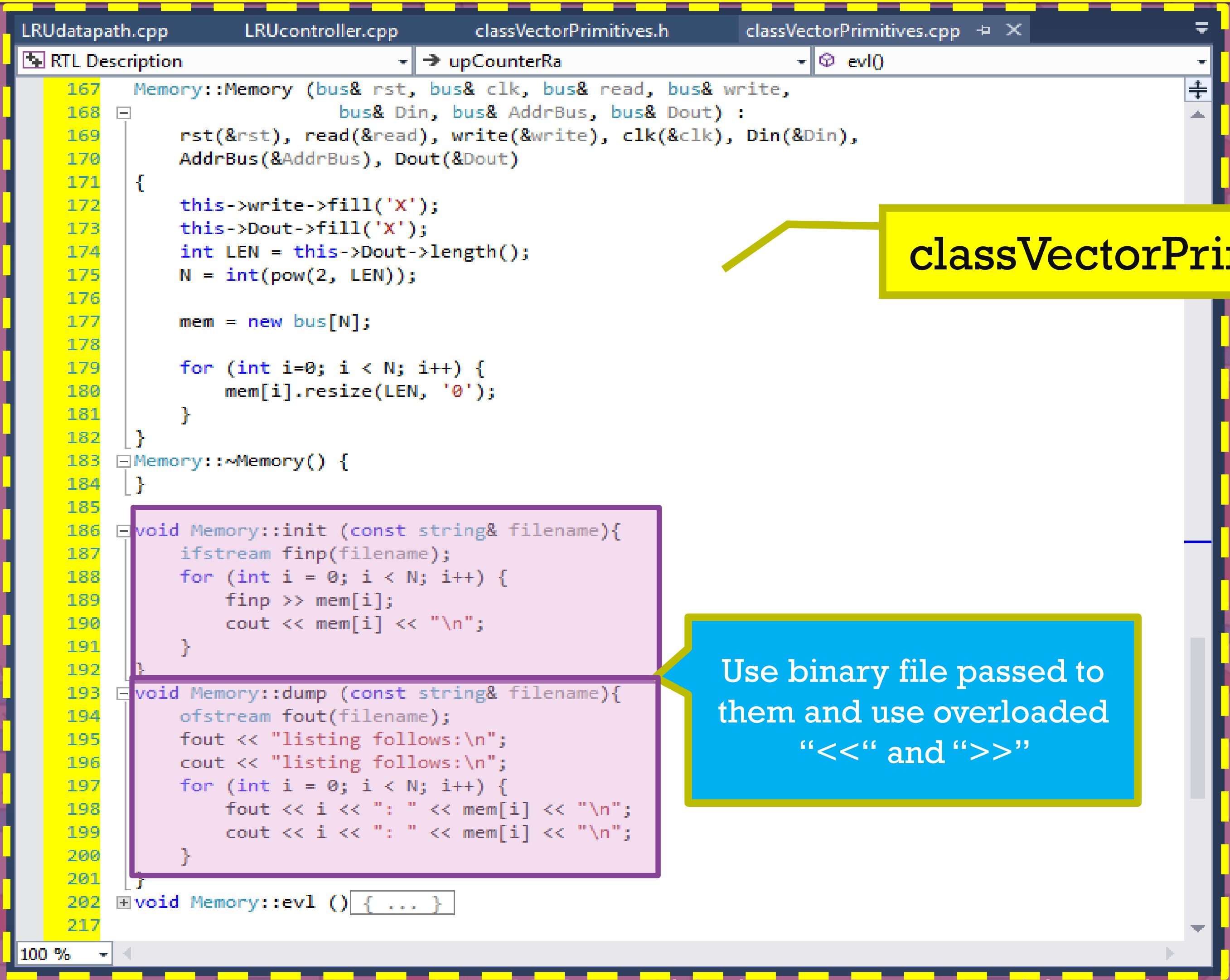


```
Memory::Memory (bus& rst, bus& clk, bus& read, bus& write,
                  bus& Din, bus& AddrBus, bus& Dout) :
    rst(&rst), read(&read), write(&write), clk(&clk), Din(&Din),
    AddrBus(&AddrBus), Dout(&Dout)
{
    this->write->fill('X');
    this->Dout->fill('X');
    int LEN = this->Dout->length();
    N = int(pow(2, LEN));

    mem = new bus[N];

    for (int i=0; i < N; i++) {
        mem[i].resize(LEN, '0');
    }
}
Memory::~Memory() { ... }

void Memory::init (const string& filename) { ... }
void Memory::dump (const string& filename) { ... }
void Memory::evl () {
    if (rst->at(0) == "1") {
        for (int i = 0; i < N; i++) mem[i].fill('0');
    }
    else if (read->at(0) == "1") {
        *Dout = mem[AddrBus->ival()];
    }
    else if (clk->at(0) == "P") {
        if (write->at(0) == "1") {
            mem[AddrBus->ival()] = *Din;
        }
    }
}
```

classVectorPrimitives.h

Memory resetting, read and write

Memory Structure

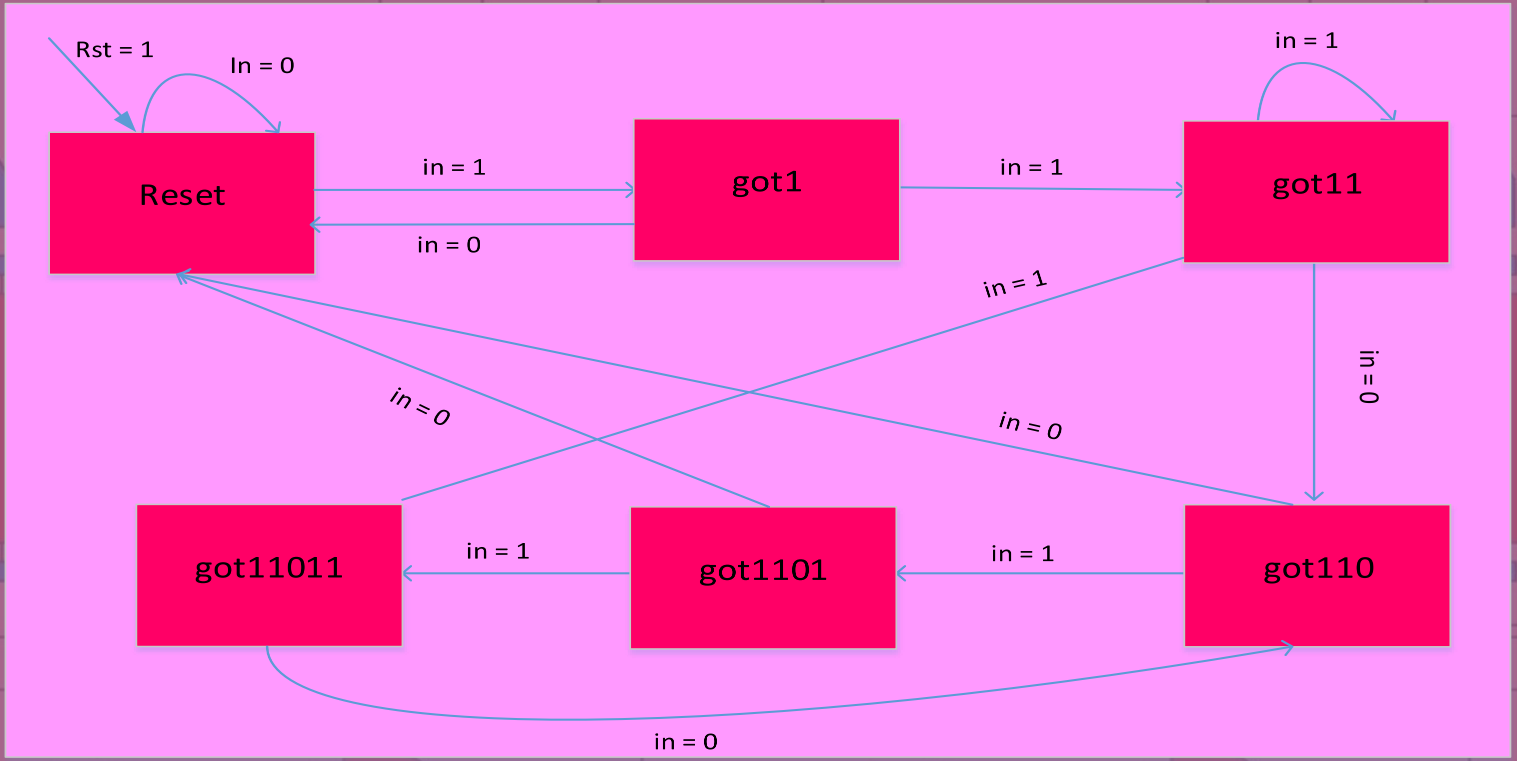


```cpp
Memory::Memory (bus& rst, bus& clk, bus& read, bus& write,
                bus& Din, bus& AddrBus, bus& Dout) :
    rst(&rst), read(&read), write(&write), clk(&clk), Din(&Din),
    AddrBus(&AddrBus), Dout(&Dout)
{
    this->write->fill('X');
    this->Dout->fill('X');
    int LEN = this->Dout->length();
    N = int(pow(2, LEN));

    mem = new bus[N];

    for (int i=0; i < N; i++) {
        mem[i].resize(LEN, '0');
    }
}
Memory::~Memory() {
}

void Memory::init (const string& filename){
    ifstream finp(filename);
    for (int i = 0; i < N; i++) {
        finp >> mem[i];
        cout << mem[i] << "\n";
    }
}
void Memory::dump (const string& filename){
    ofstream fout(filename);
    fout << "listing follows:\n";
    cout << "listing follows:\n";
    for (int i = 0; i < N; i++) {
        fout << i << ": " << mem[i] << "\n";
        cout << i << ": " << mem[i] << "\n";
    }
}
void Memory::evl () { ... }
```

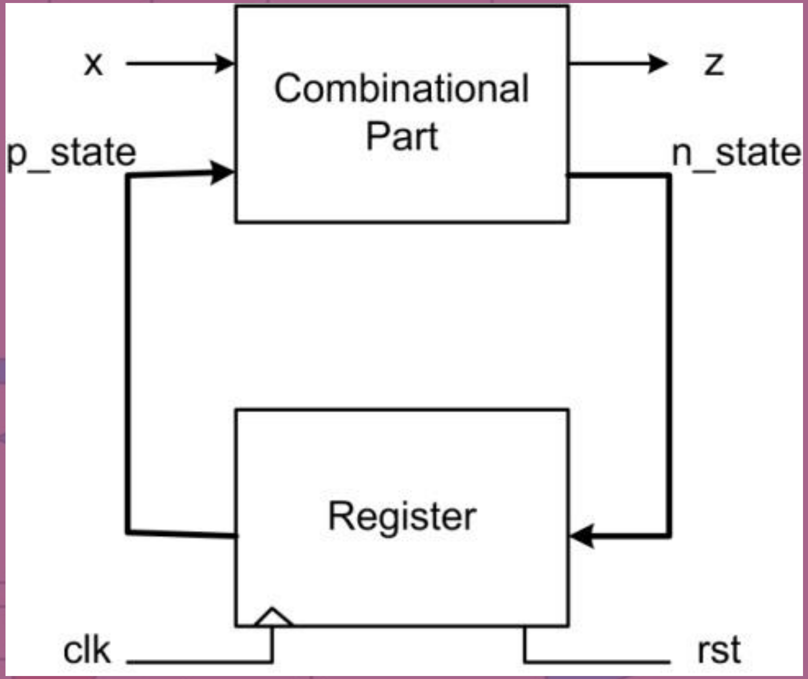
classVectorPrimitives.h

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

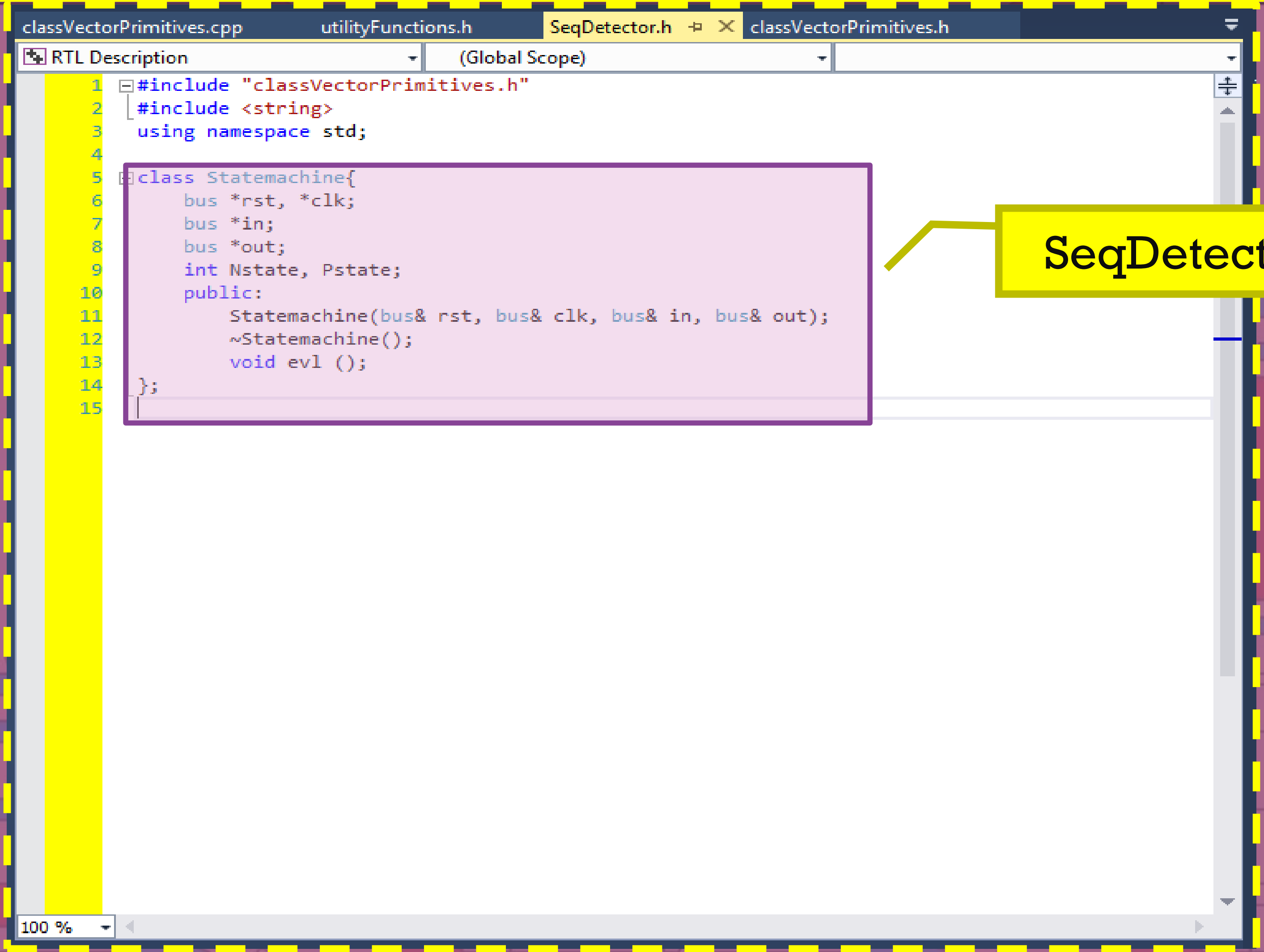
Moore Sequence Detector (11011)



Moore Sequence Detector (11011)



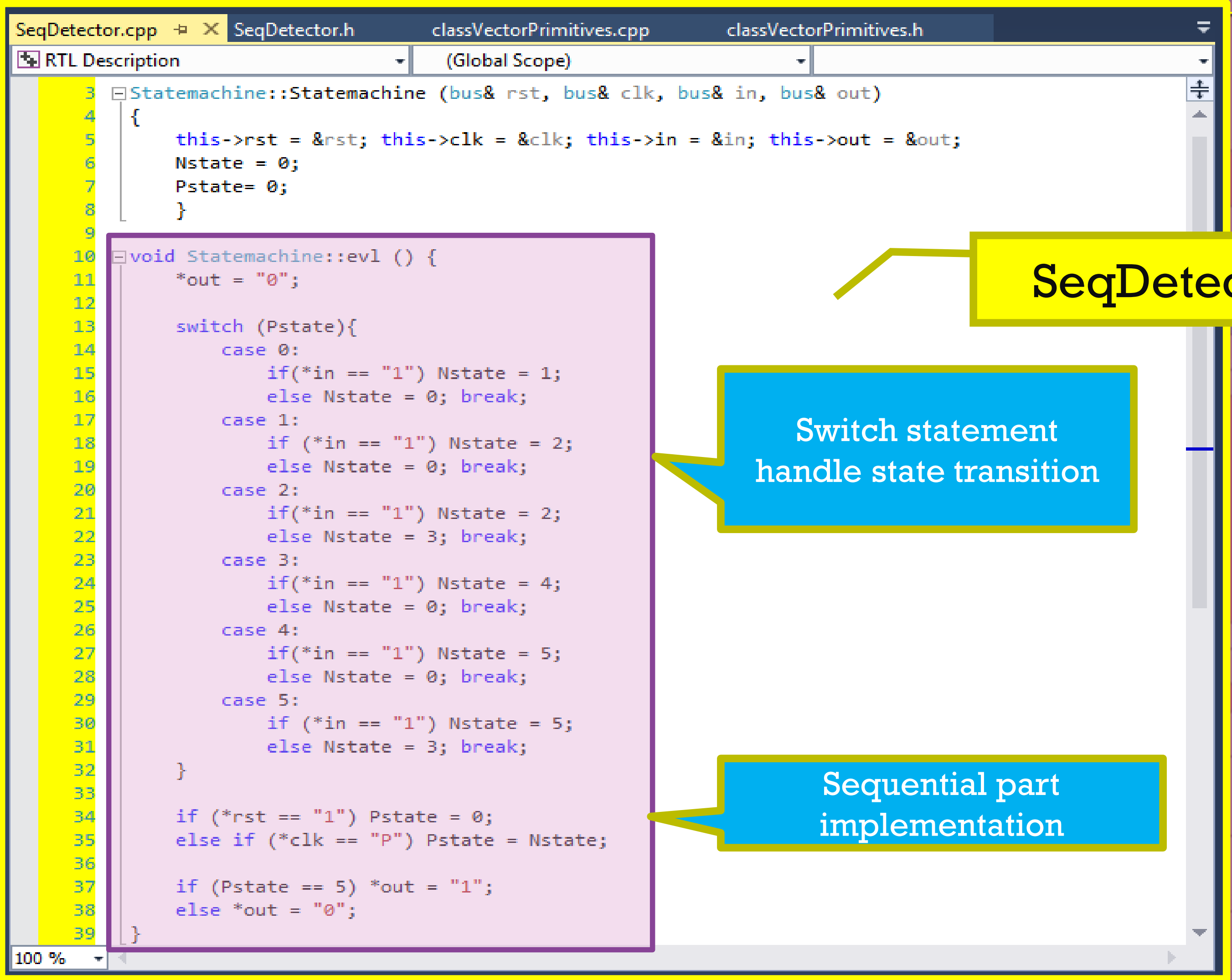
Moore Sequence Detector (11011)



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

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

```

SeqDetector.h

Moore Sequence Detector (11011)



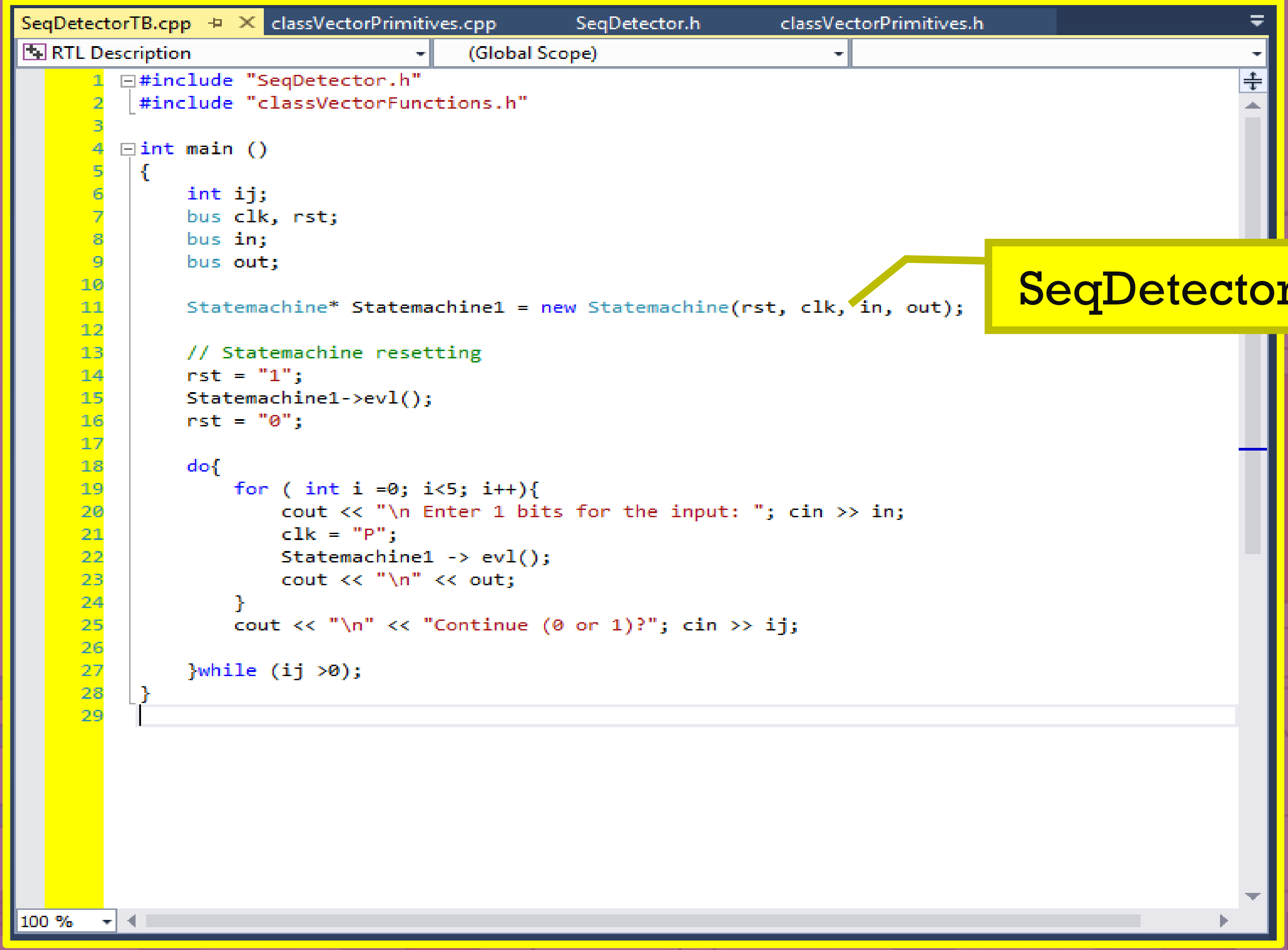
```
Statemachine::Statemachine (bus& rst, bus& clk, bus& in, bus& out)
{
    this->rst = &rst; this->clk = &clk; this->in = &in; this->out = &out;
    Nstate = 0;
    Pstate= 0;
    }

void Statemachine::evl () {
    *out = "0";

    switch (Pstate){
        case 0:
            if(*in == "1") Nstate = 1;
            else Nstate = 0; break;
        case 1:
            if (*in == "1") Nstate = 2;
            else Nstate = 0; break;
        case 2:
            if(*in == "1") Nstate = 2;
            else Nstate = 3; break;
        case 3:
            if(*in == "1") Nstate = 4;
            else Nstate = 0; break;
        case 4:
            if(*in == "1") Nstate = 5;
            else Nstate = 0; break;
        case 5:
            if (*in == "1") Nstate = 5;
            else Nstate = 3; break;
    }

    if (*rst == "1") Pstate = 0;
    else if (*clk == "P") Pstate = Nstate;

    if (Pstate == 5) *out = "1";
    else *out = "0";
}
```

SeqDetector.cpp

Switch statement handle state transition

Sequential part implementation

Moore Sequence Detector (11011)



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

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

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

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

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

    }while (ij >0);
}
```

SeqDetectorTB.cpp

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

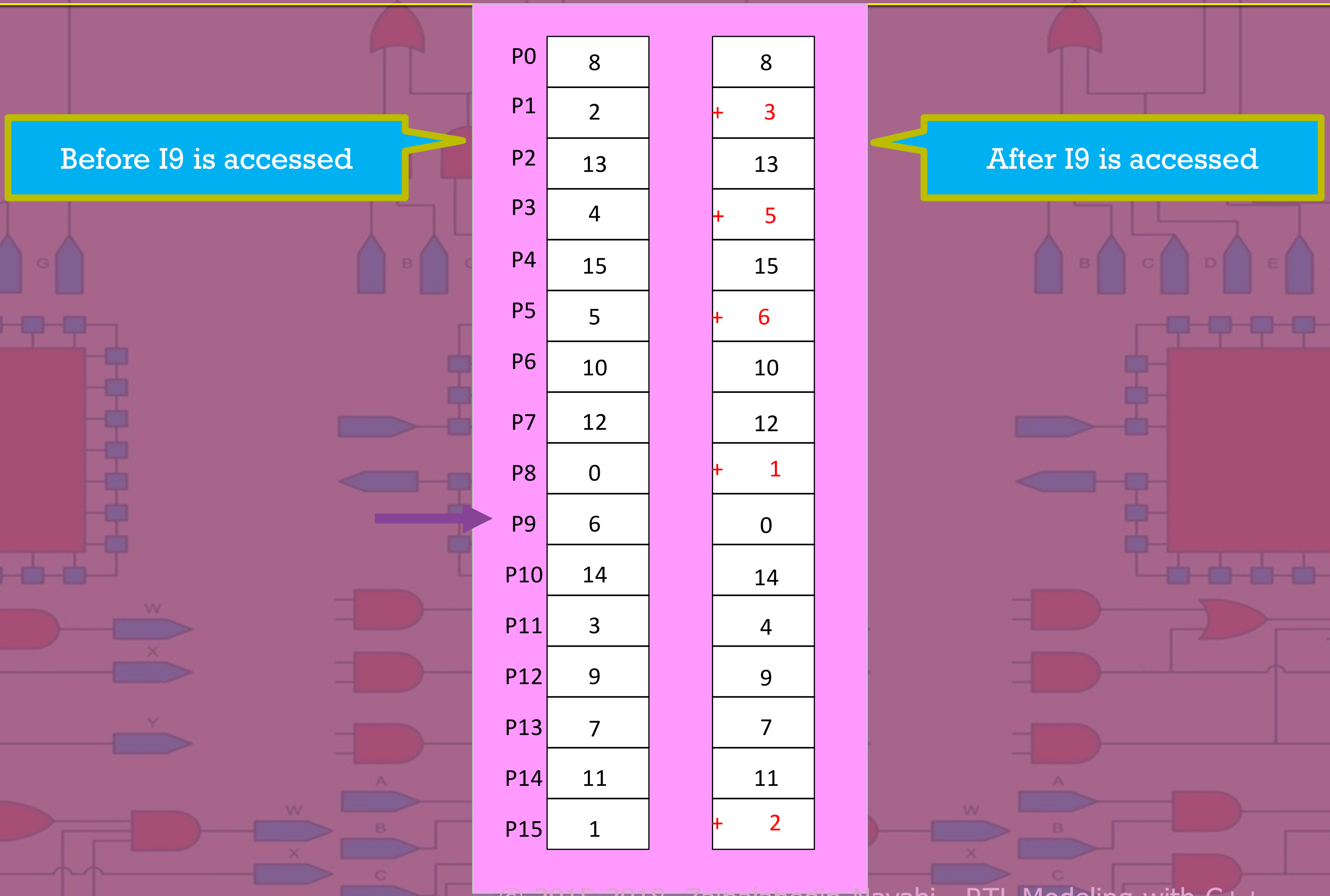
Assign queue positions to items based on frequency of their use

Before I9 is accessed

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

After I9 is accessed

Queue Memory File

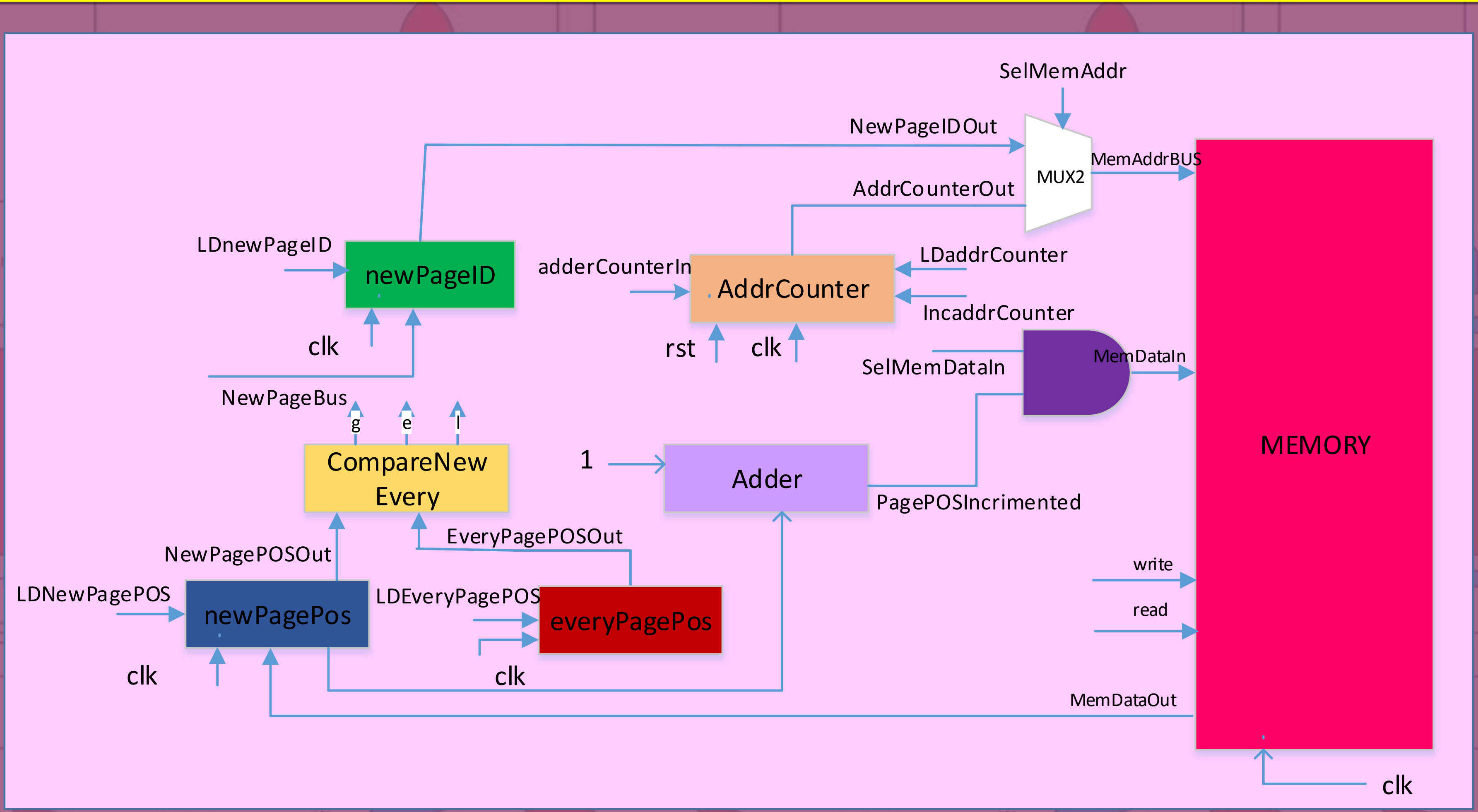


Before I9 is accessed

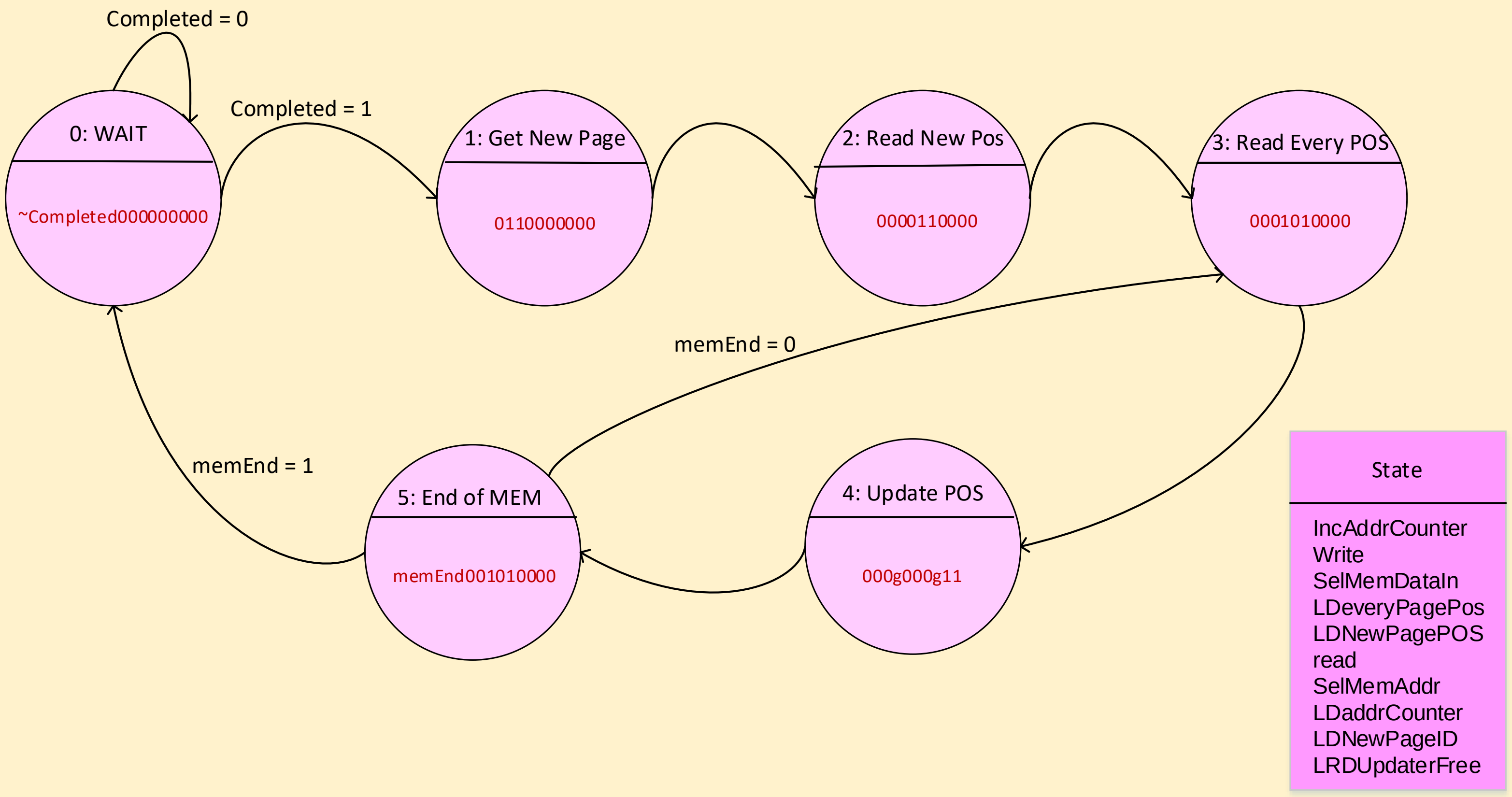
After I9 is accessed

	Before I9 is accessed	After I9 is accessed
P0	8	8
P1	2	+ 3
P2	13	13
P3	4	+ 5
P4	15	15
P5	5	+ 6
P6	10	10
P7	12	12
P8	0	+ 1
P9	6	0
P10	14	14
P11	3	4
P12	9	9
P13	7	7
P14	11	11
P15	1	+ 2

LRU Updater Datapath



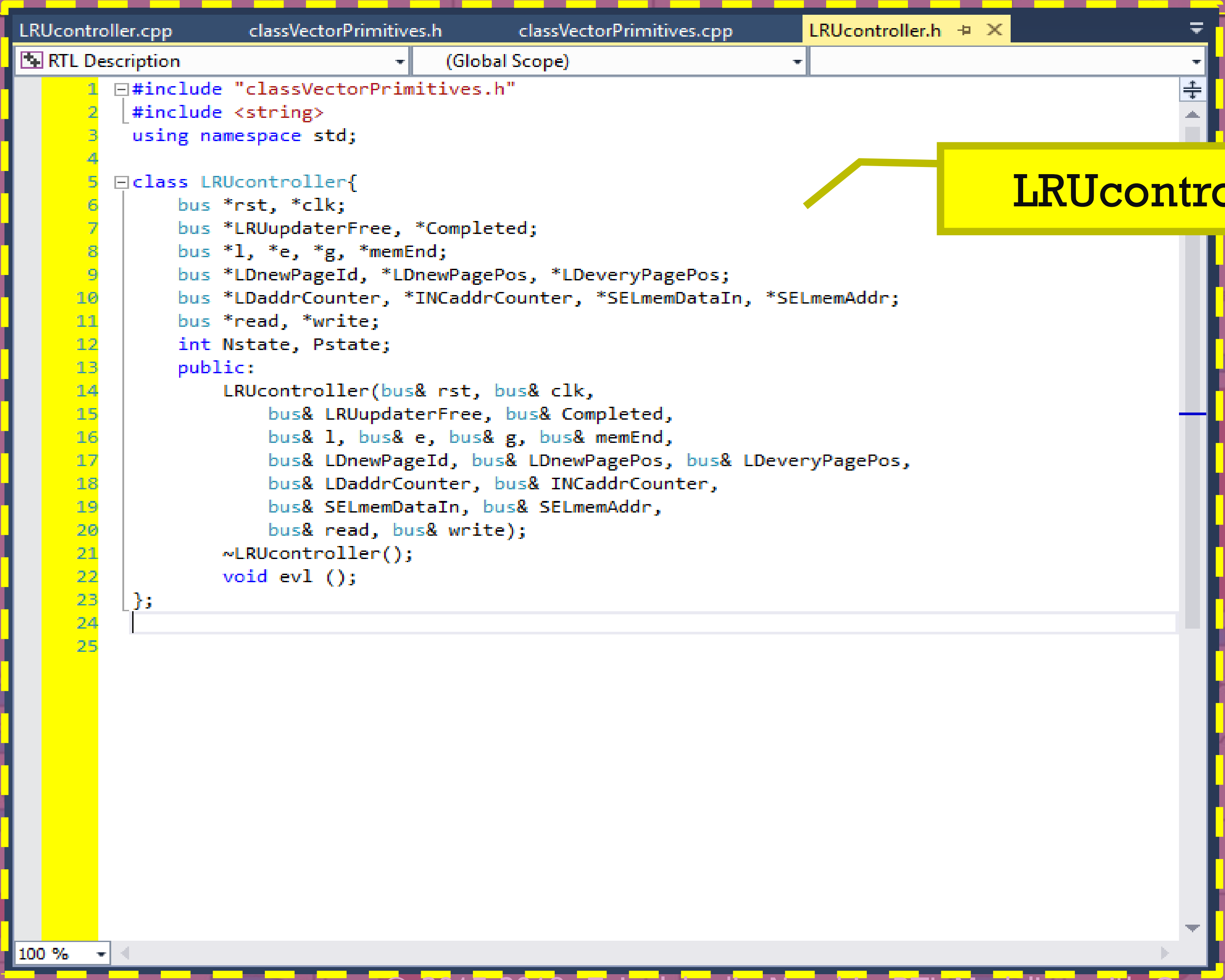
LRU Controller



State

IncAddrCounter
Write
SelMemDataIn
LDeveryPagePos
LDNewPagePOS
read
SelMemAddr
LDaddrCounter
LDNewPageID
LRDUpdaterFree

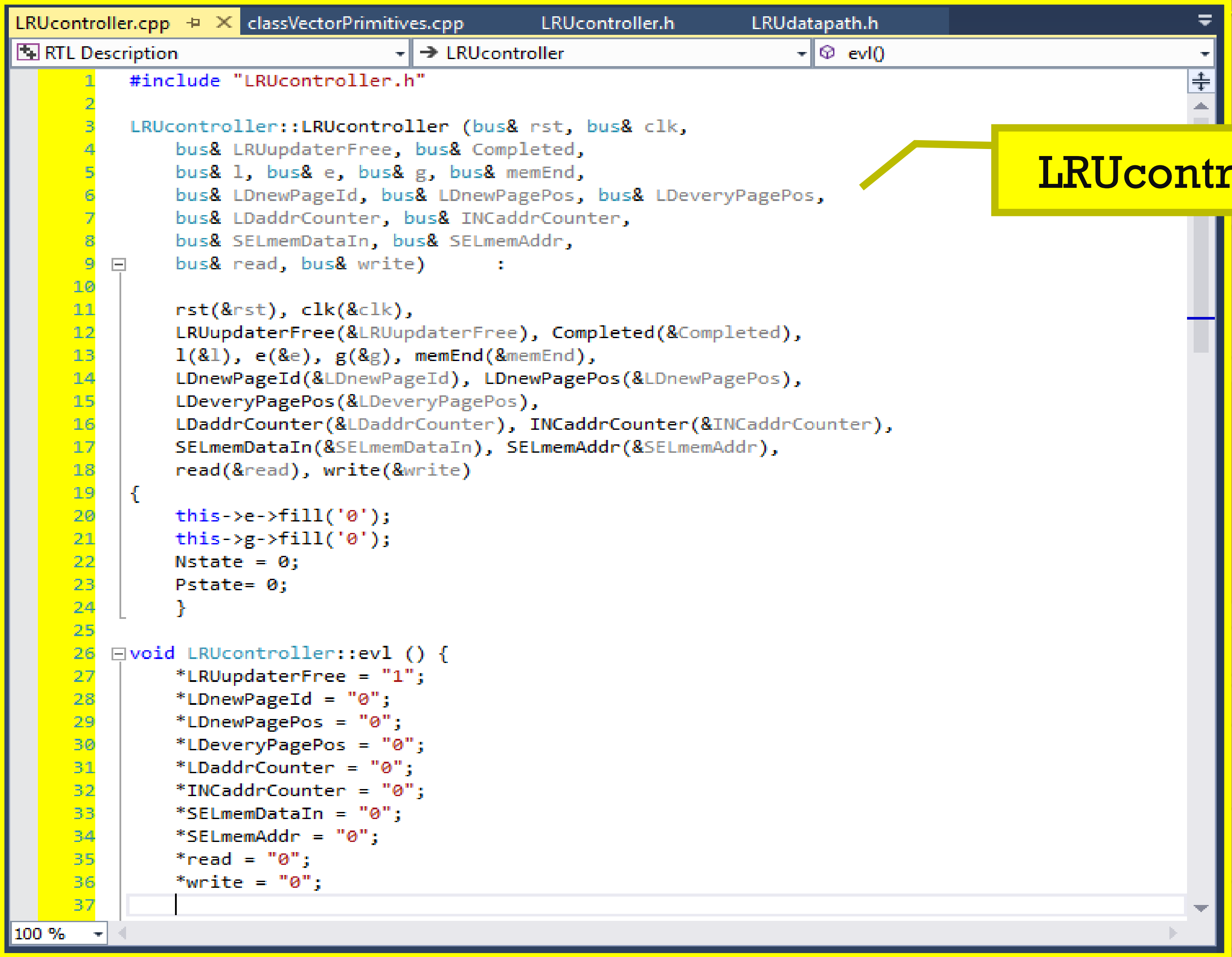
LRU Controller



```
#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 ();
};
```

LRUcontroller.h

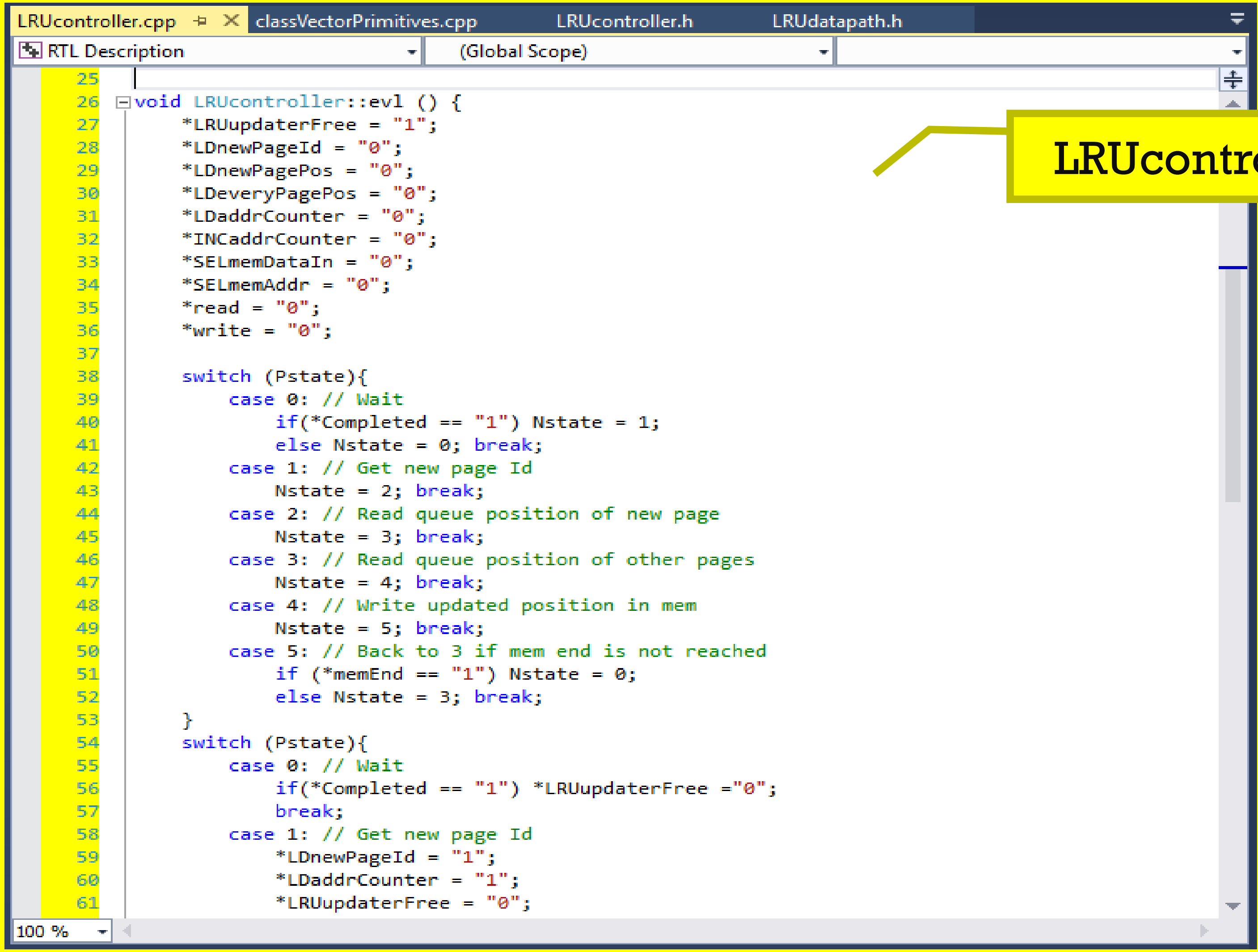
LRU Controller



```cpp
#include "LRUcontroller.h"

LRUcontroller::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)      :

    rst(&rst), clk(&clk),
    LRUupdaterFree(&LRUupdaterFree), Completed(&Completed),
    l(&l), e(&e), g(&g), memEnd(&memEnd),
    LDnewPageId(&LDnewPageId), LDnewPagePos(&LDnewPagePos),
    LDeveryPagePos(&LDeveryPagePos),
    LDaddrCounter(&LDaddrCounter), INCaddrCounter(&INCaddrCounter),
    SELmemDataIn(&SELmemDataIn), SELmemAddr(&SELmemAddr),
    read(&read), write(&write)
{
    this->e->fill('0');
    this->g->fill('0');
    Nstate = 0;
    Pstate= 0;
    }

void LRUcontroller::evl () {
    *LRUupdaterFree = "1";
    *LDnewPageId = "0";
    *LDnewPagePos = "0";
    *LDeveryPagePos = "0";
    *LDaddrCounter = "0";
    *INCaddrCounter = "0";
    *SELmemDataIn = "0";
    *SELmemAddr = "0";
    *read = "0";
    *write = "0";
```

LRUcontroller.cpp

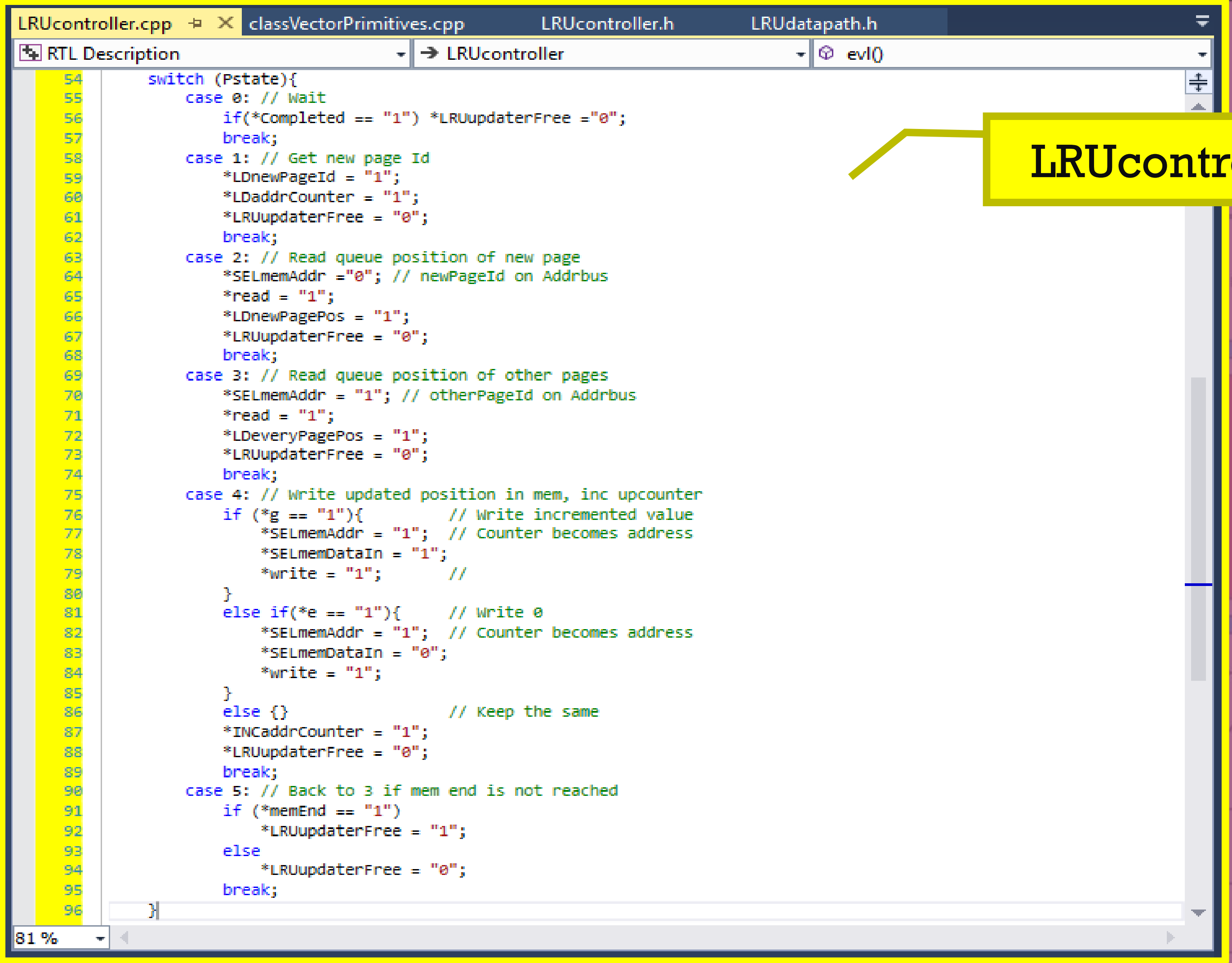
LRU Controller



```
void LRUcontroller::evl () {
    *LRUupdaterFree = "1";
    *LDnewPageId = "0";
    *LDnewPagePos = "0";
    *LDeveryPagePos = "0";
    *LDaddrCounter = "0";
    *INCaddrCounter = "0";
    *SELmemDataIn = "0";
    *SELmemAddr = "0";
    *read = "0";
    *write = "0";

    switch (Pstate){
        case 0: // Wait
            if(*Completed == "1") Nstate = 1;
            else Nstate = 0; break;
        case 1: // Get new page Id
            Nstate = 2; break;
        case 2: // Read queue position of new page
            Nstate = 3; break;
        case 3: // Read queue position of other pages
            Nstate = 4; break;
        case 4: // Write updated position in mem
            Nstate = 5; break;
        case 5: // Back to 3 if mem end is not reached
            if (*memEnd == "1") Nstate = 0;
            else Nstate = 3; break;
    }
    switch (Pstate){
        case 0: // Wait
            if(*Completed == "1") *LRUupdaterFree ="0";
            break;
        case 1: // Get new page Id
            *LDnewPageId = "1";
            *LDaddrCounter = "1";
            *LRUupdaterFree = "0";
```

LRUcontroller.cpp

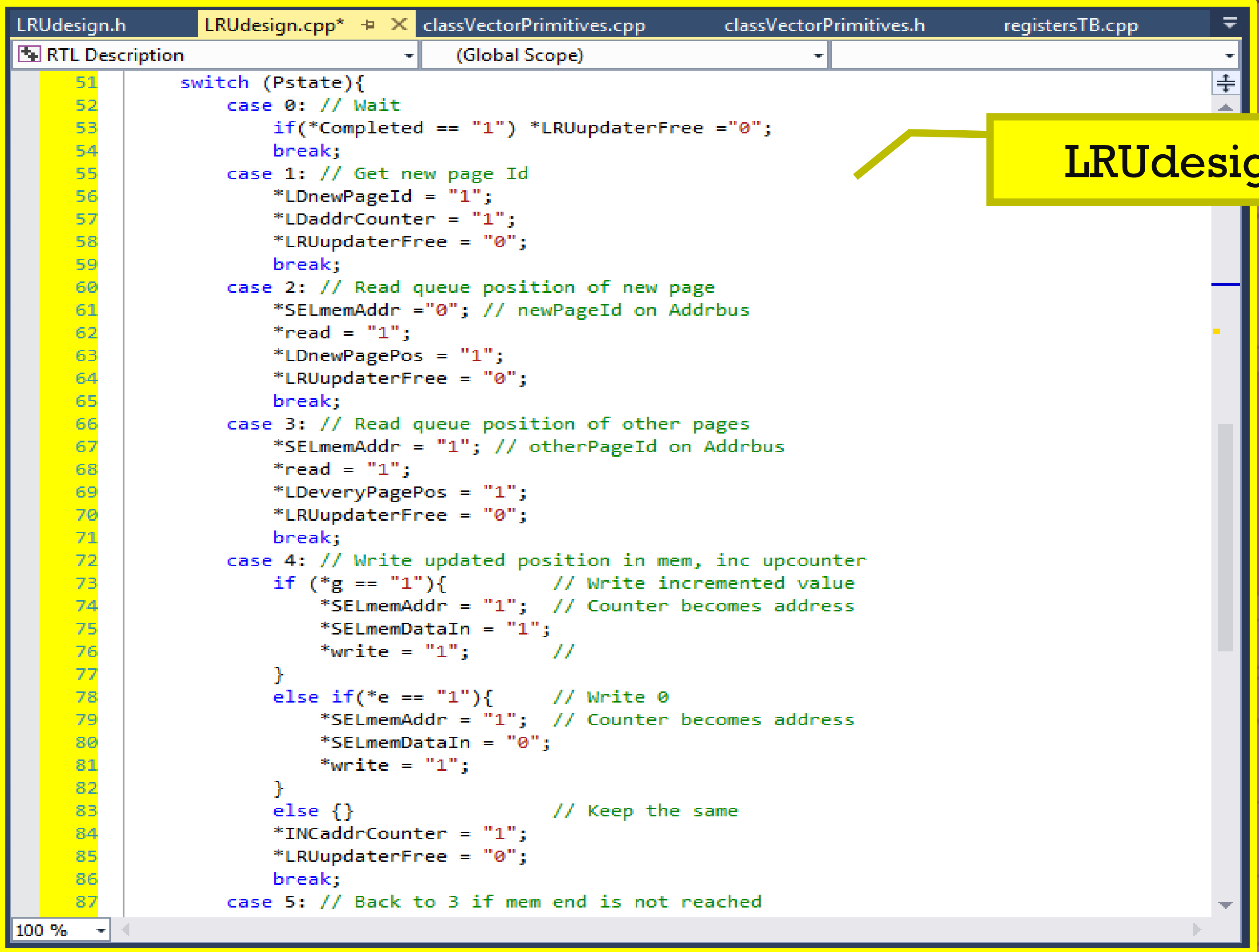
LRU Controller



```
        switch (Pstate){
            case 0: // Wait
                if(*Completed == "1") *LRUupdaterFree ="0";
                break;
            case 1: // Get new page Id
                *LDnewPageId = "1";
                *LDaddrCounter = "1";
                *LRUupdaterFree = "0";
                break;
            case 2: // Read queue position of new page
                *SELmemAddr ="0"; // newPageId on Addrbus
                *read = "1";
                *LDnewPagePos = "1";
                *LRUupdaterFree = "0";
                break;
            case 3: // Read queue position of other pages
                *SELmemAddr = "1"; // otherPageId on Addrbus
                *read = "1";
                *LDeveryPagePos = "1";
                *LRUupdaterFree = "0";
                break;
            case 4: // Write updated position in mem, inc upcounter
                if (*g == "1"){         // Write incremented value
                    *SELmemAddr = "1";  // Counter becomes address
                    *SELmemDataIn = "1";
                    *write = "1";       //
                }
                else if(*e == "1"){     // Write 0
                    *SELmemAddr = "1";  // Counter becomes address
                    *SELmemDataIn = "0";
                    *write = "1";
                }
                else {}                 // Keep the same
                *INCaddrCounter = "1";
                *LRUupdaterFree = "0";
                break;
            case 5: // Back to 3 if mem end is not reached
                if (*memEnd == "1")
                    *LRUupdaterFree = "1";
                else
                    *LRUupdaterFree = "0";
                break;
        }
```

LRUcontroller.cpp

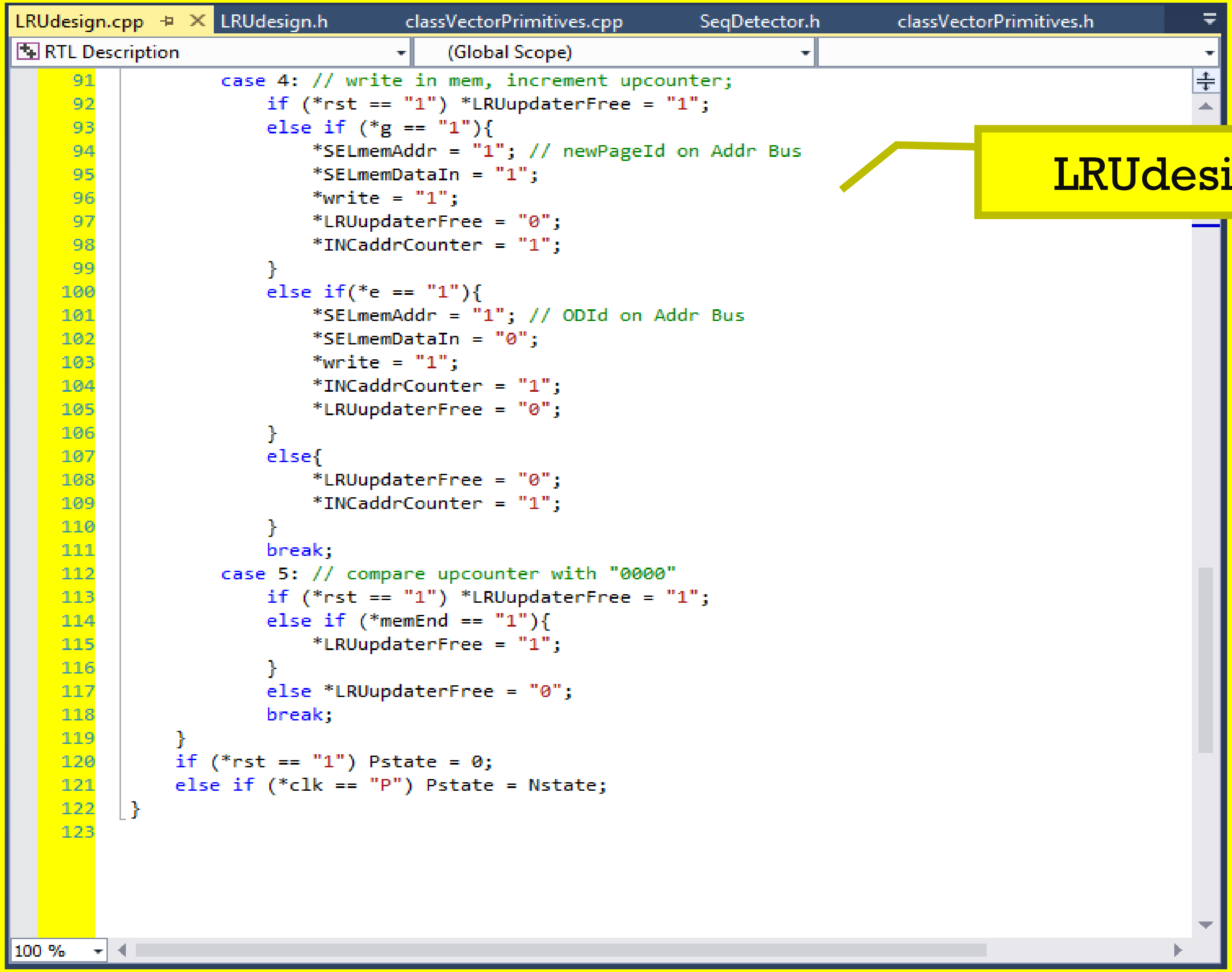
LRU Design



```
    switch (Pstate){
        case 0: // Wait
            if(*Completed == "1") *LRUupdaterFree ="0";
            break;
        case 1: // Get new page Id
            *LDnewPageId = "1";
            *LDaddrCounter = "1";
            *LRUupdaterFree = "0";
            break;
        case 2: // Read queue position of new page
            *SELmemAddr ="0"; // newPageId on Addrbus
            *read = "1";
            *LDnewPagePos = "1";
            *LRUupdaterFree = "0";
            break;
        case 3: // Read queue position of other pages
            *SELmemAddr = "1"; // otherPageId on Addrbus
            *read = "1";
            *LDeveryPagePos = "1";
            *LRUupdaterFree = "0";
            break;
        case 4: // Write updated position in mem, inc upcounter
            if (*g == "1"){         // Write incremented value
                *SELmemAddr = "1";  // Counter becomes address
                *SELmemDataIn = "1";
                *write = "1";       //
            }
            else if(*e == "1"){     // Write 0
                *SELmemAddr = "1";  // Counter becomes address
                *SELmemDataIn = "0";
                *write = "1";
            }
            else {}                 // Keep the same
            *INCaddrCounter = "1";
            *LRUupdaterFree = "0";
            break;
        case 5: // Back to 3 if mem end is not reached
```

LRUdesign.cpp

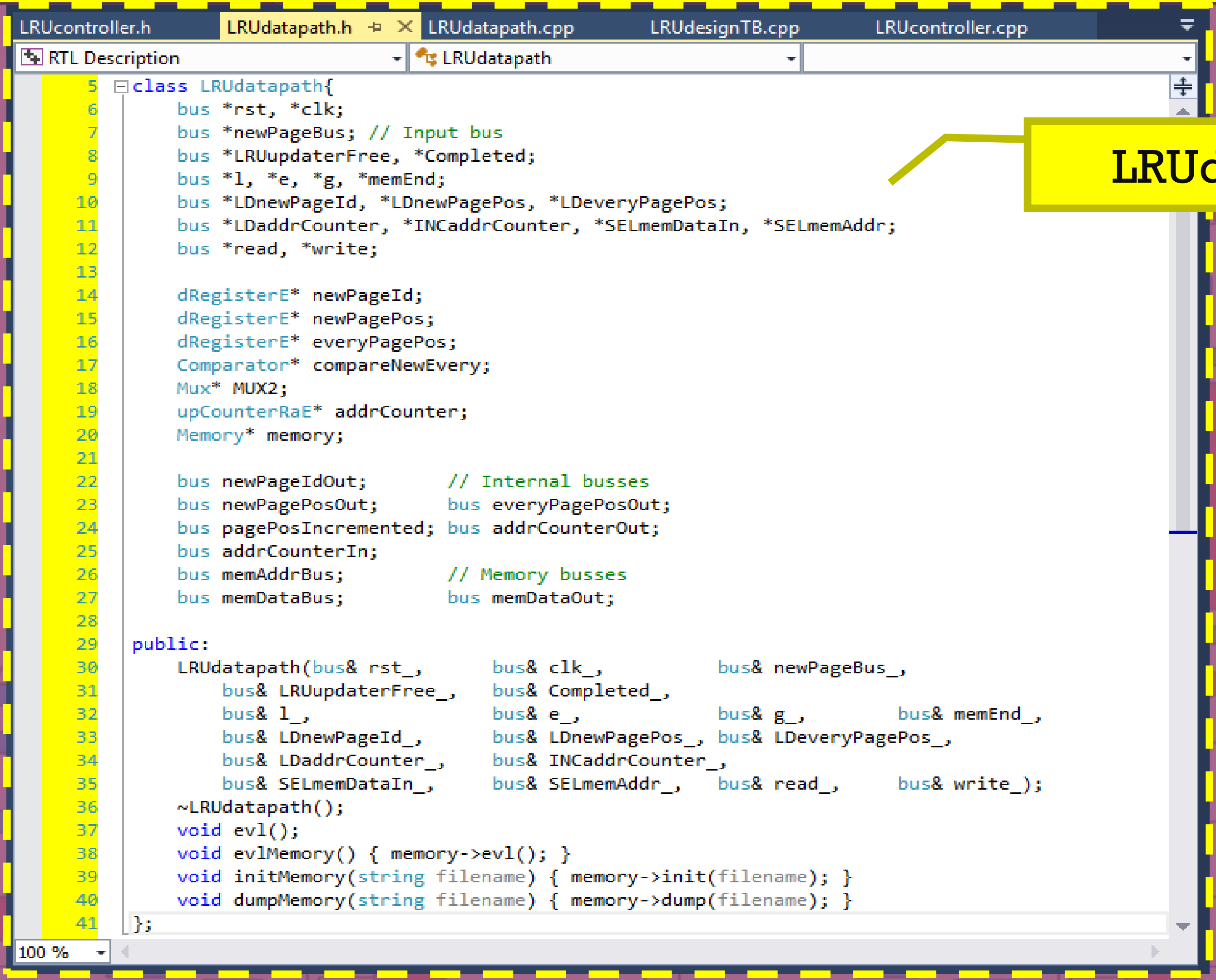
LRU Design



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

LRUdesign.cpp

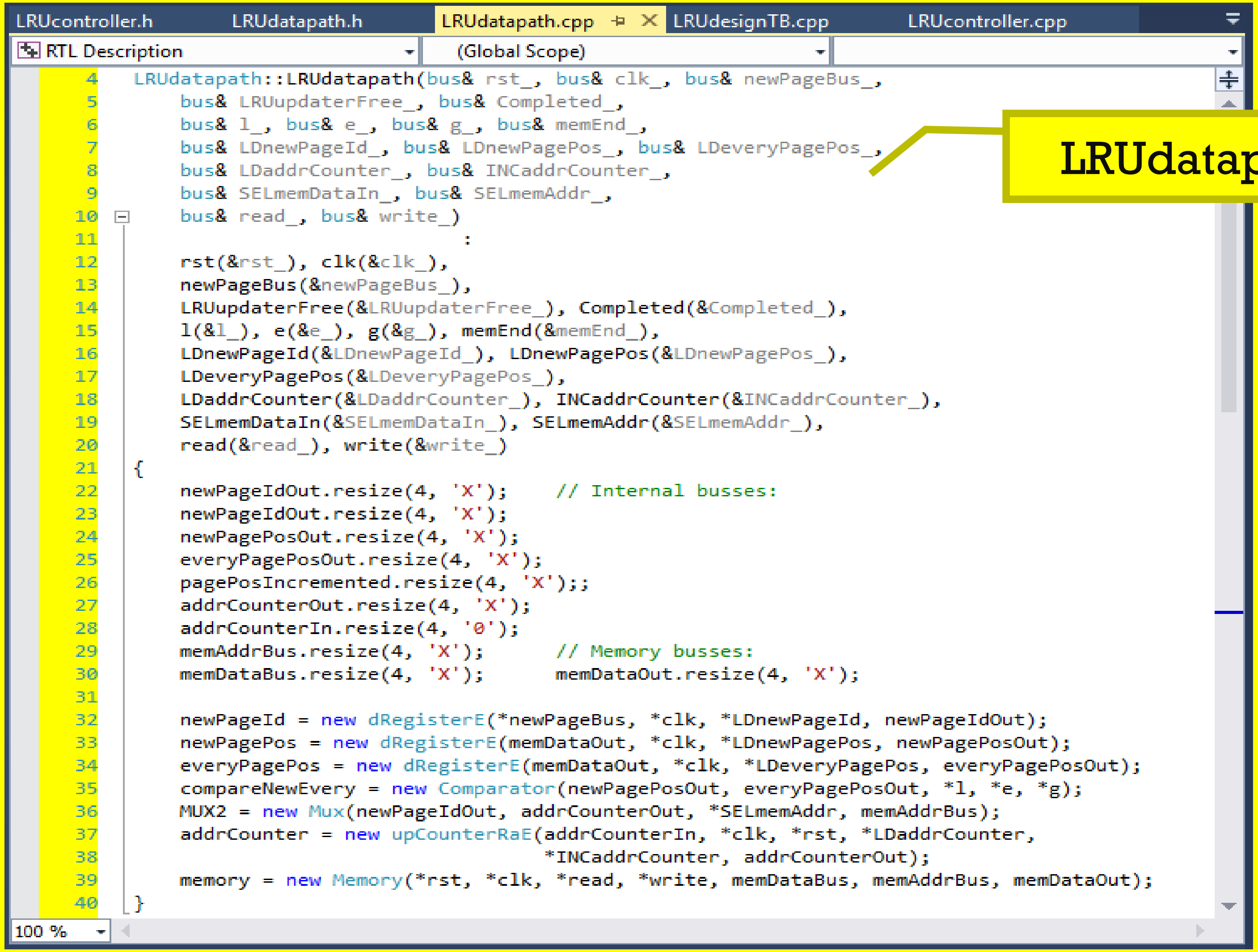
LRU Datapath



```
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); }
};
```

LRUdatapath.h

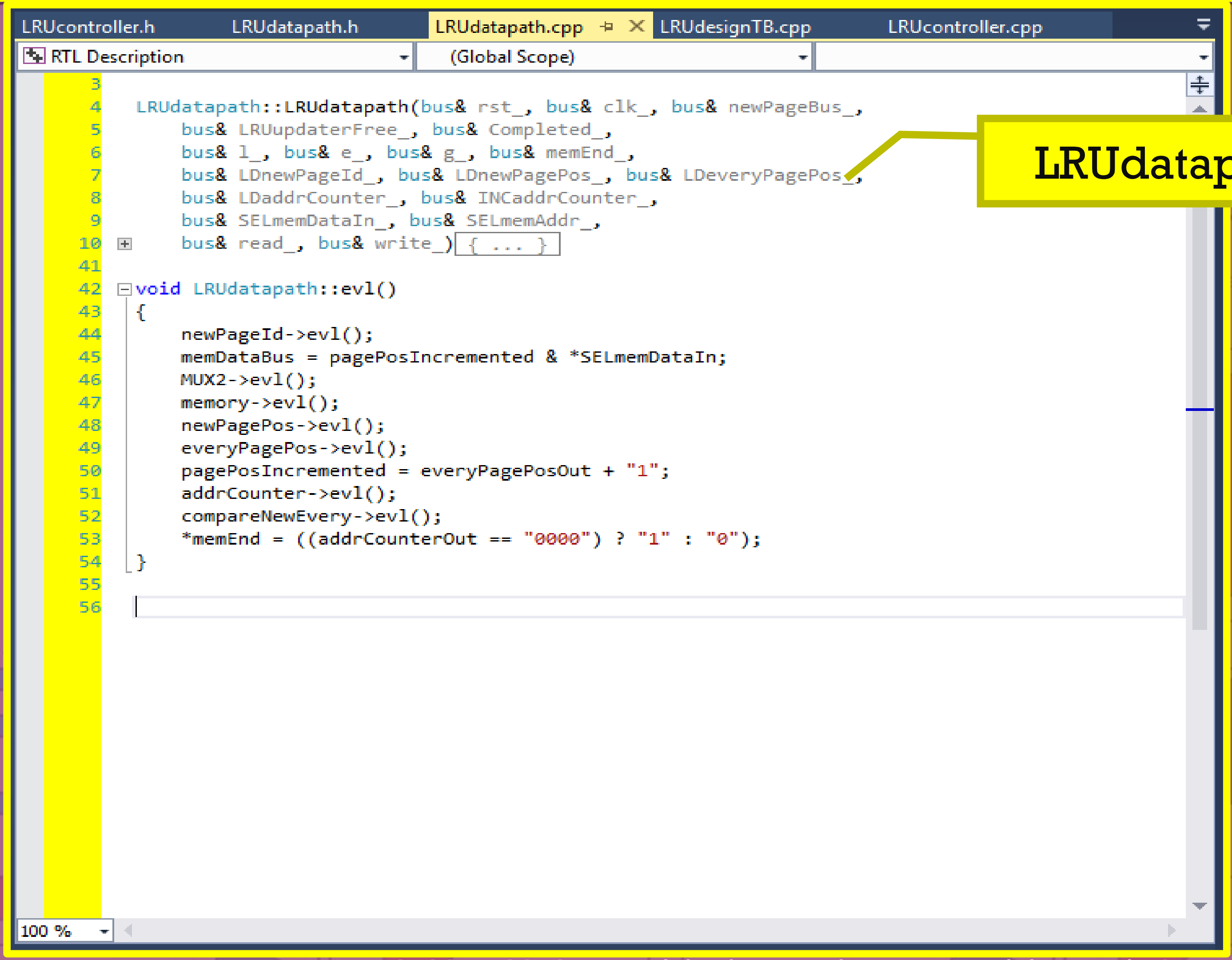
LRU Datapath



```
LRUdatapath::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_)
                          :
    rst(&rst_), clk(&clk_),
    newPageBus(&newPageBus_),
    LRUupdaterFree(&LRUupdaterFree_), Completed(&Completed_),
    l(&l_), e(&e_), g(&g_), memEnd(&memEnd_),
    LDnewPageId(&LDnewPageId_), LDnewPagePos(&LDnewPagePos_),
    LDeveryPagePos(&LDeveryPagePos_),
    LDaddrCounter(&LDaddrCounter_), INCaddrCounter(&INCaddrCounter_),
    SELmemDataIn(&SELmemDataIn_), SELmemAddr(&SELmemAddr_),
    read(&read_), write(&write_)
{
    newPageIdOut.resize(4, 'X');    // Internal busses:
    newPageIdOut.resize(4, 'X');
    newPagePosOut.resize(4, 'X');
    everyPagePosOut.resize(4, 'X');
    pagePosIncremented.resize(4, 'X');;
    addrCounterOut.resize(4, 'X');
    addrCounterIn.resize(4, '0');
    memAddrBus.resize(4, 'X');      // Memory busses:
    memDataBus.resize(4, 'X');      memDataOut.resize(4, 'X');

    newPageId = new dRegisterE(*newPageBus, *clk, *LDnewPageId, newPageIdOut);
    newPagePos = new dRegisterE(memDataOut, *clk, *LDnewPagePos, newPagePosOut);
    everyPagePos = new dRegisterE(memDataOut, *clk, *LDeveryPagePos, everyPagePosOut);
    compareNewEvery = new Comparator(newPagePosOut, everyPagePosOut, *l, *e, *g);
    MUX2 = new Mux(newPageIdOut, addrCounterOut, *SELmemAddr, memAddrBus);
    addrCounter = new upCounterRaE(addrCounterIn, *clk, *rst, *LDaddrCounter,
                               *INCaddrCounter, addrCounterOut);
    memory = new Memory(*rst, *clk, *read, *write, memDataBus, memAddrBus, memDataOut);
}
```

LRUdatapath.cpp

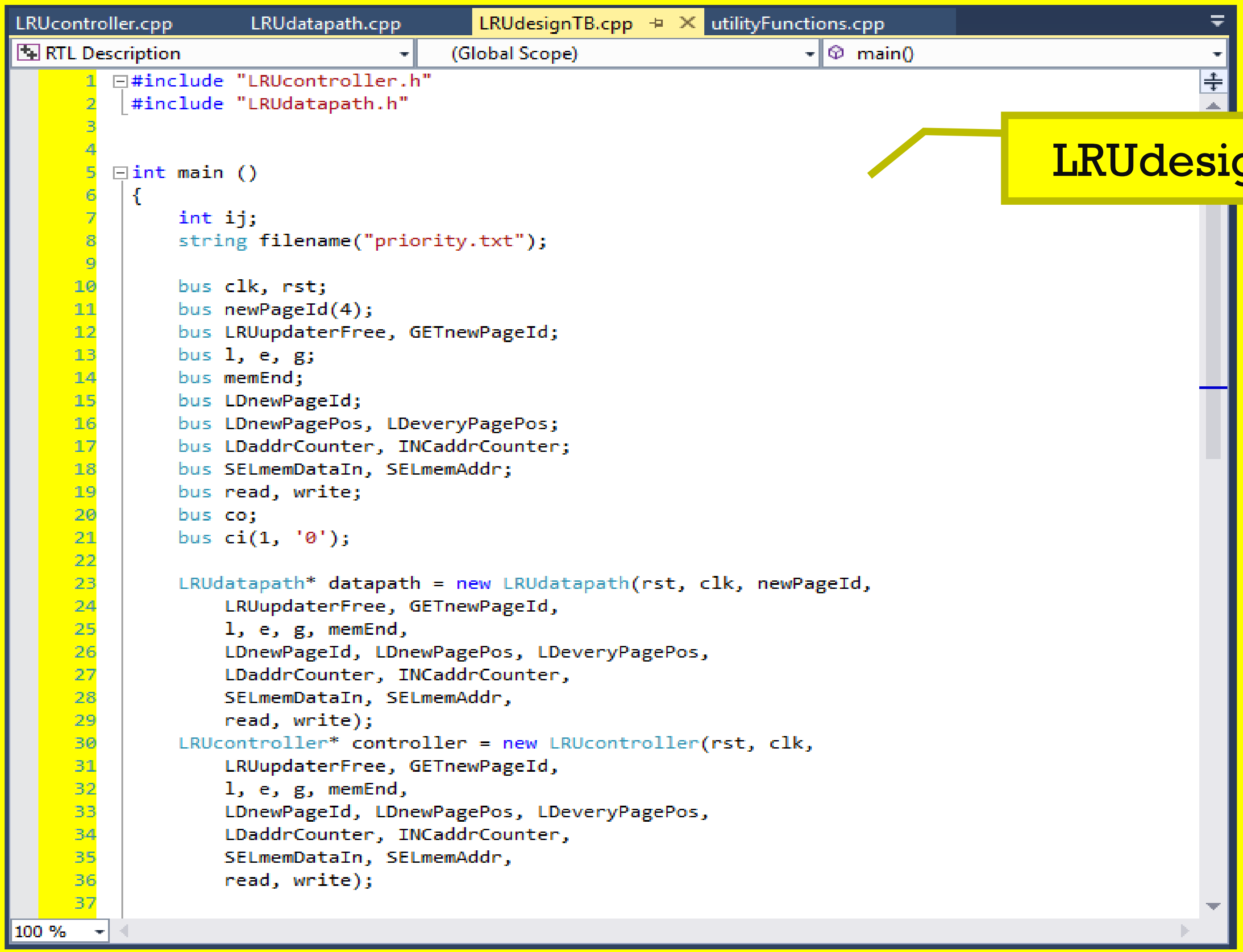
LRU Datapath



```
LRUdatapath::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_){ ... }

void LRUdatapath::evl()
{
    newPageId->evl();
    memDataBus = pagePosIncremented & *SELmemDataIn;
    MUX2->evl();
    memory->evl();
    newPagePos->evl();
    everyPagePos->evl();
    pagePosIncremented = everyPagePosOut + "1";
    addrCounter->evl();
    compareNewEvery->evl();
    *memEnd = ((addrCounterOut == "0000") ? "1" : "0");
}

```

LRUdatapath.cpp

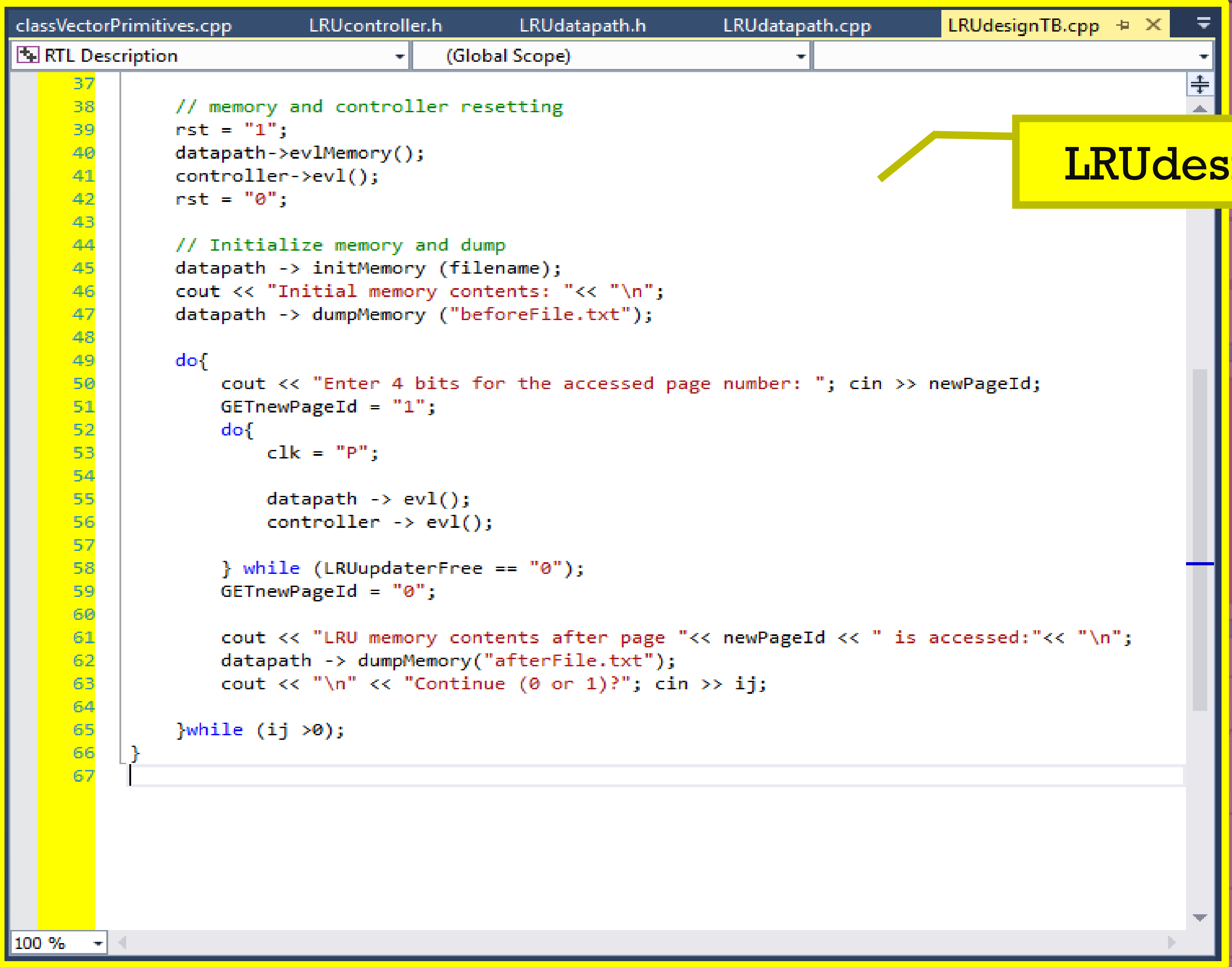
LRU Testbench



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


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

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

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

```

LRUdesignTB.cpp

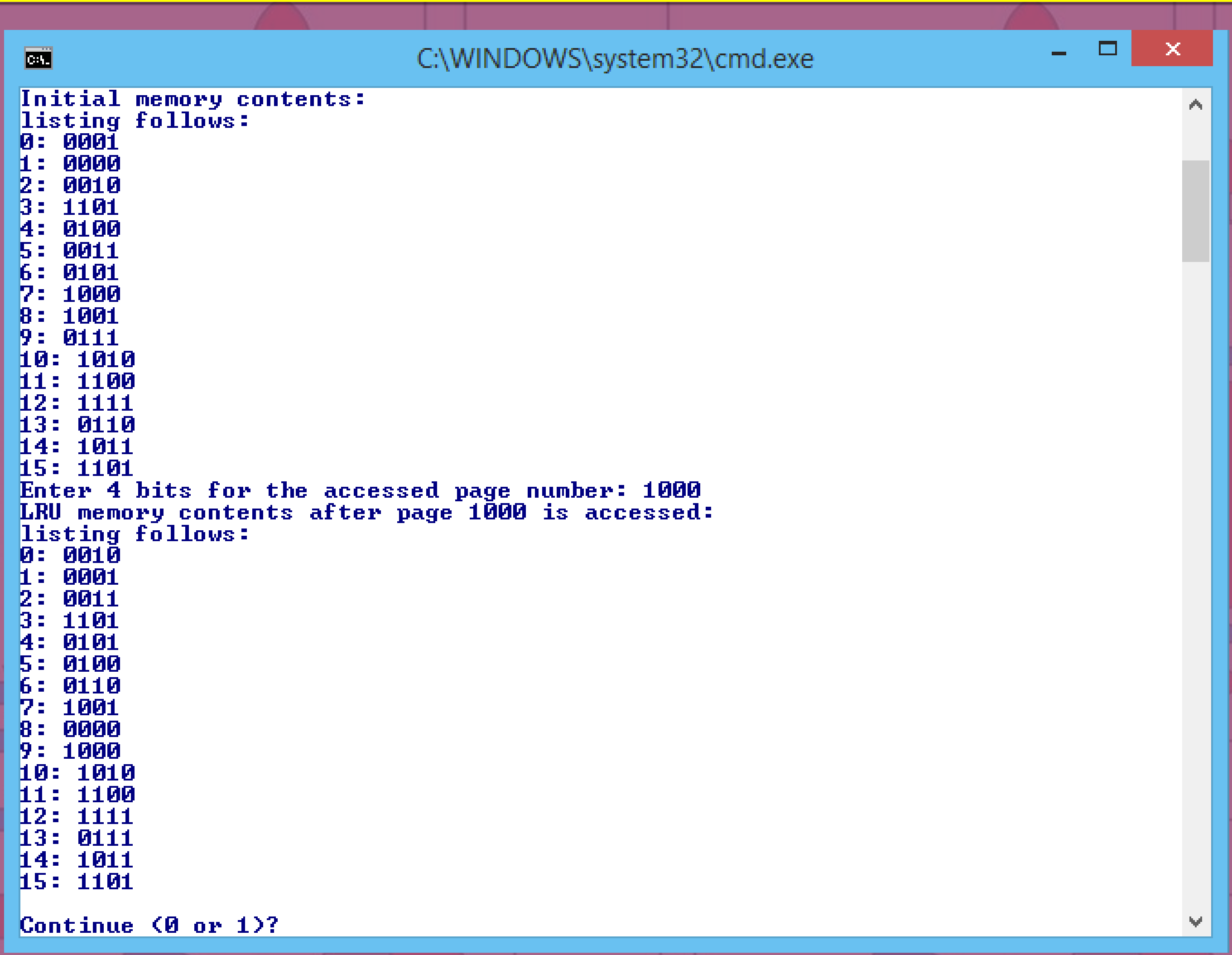
LRU Testbench



```
    // memory and controller resetting
    rst = "1";
    datapath->evlMemory();
    controller->evl();
    rst = "0";

    // Initialize memory and dump
    datapath -> initMemory (filename);
    cout << "Initial memory contents: "<< "\n";
    datapath -> dumpMemory ("beforeFile.txt");

    do{
        cout << "Enter 4 bits for the accessed page number: "; cin >> newPageId;
        GETnewPageId = "1";
        do{
            clk = "P";

            datapath -> evl();
            controller -> evl();

        } while (LRUupdaterFree == "0");
        GETnewPageId = "0";

        cout << "LRU memory contents after page "<< newPageId << " is accessed:"<< "\n";
        datapath -> dumpMemory("afterFile.txt");
        cout << "\n" << "Continue (0 or 1)?"; cin >> ij;

    }while (ij >0);
}
```

LRUdesignTB.cpp

LRU Output



```
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)?
```

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

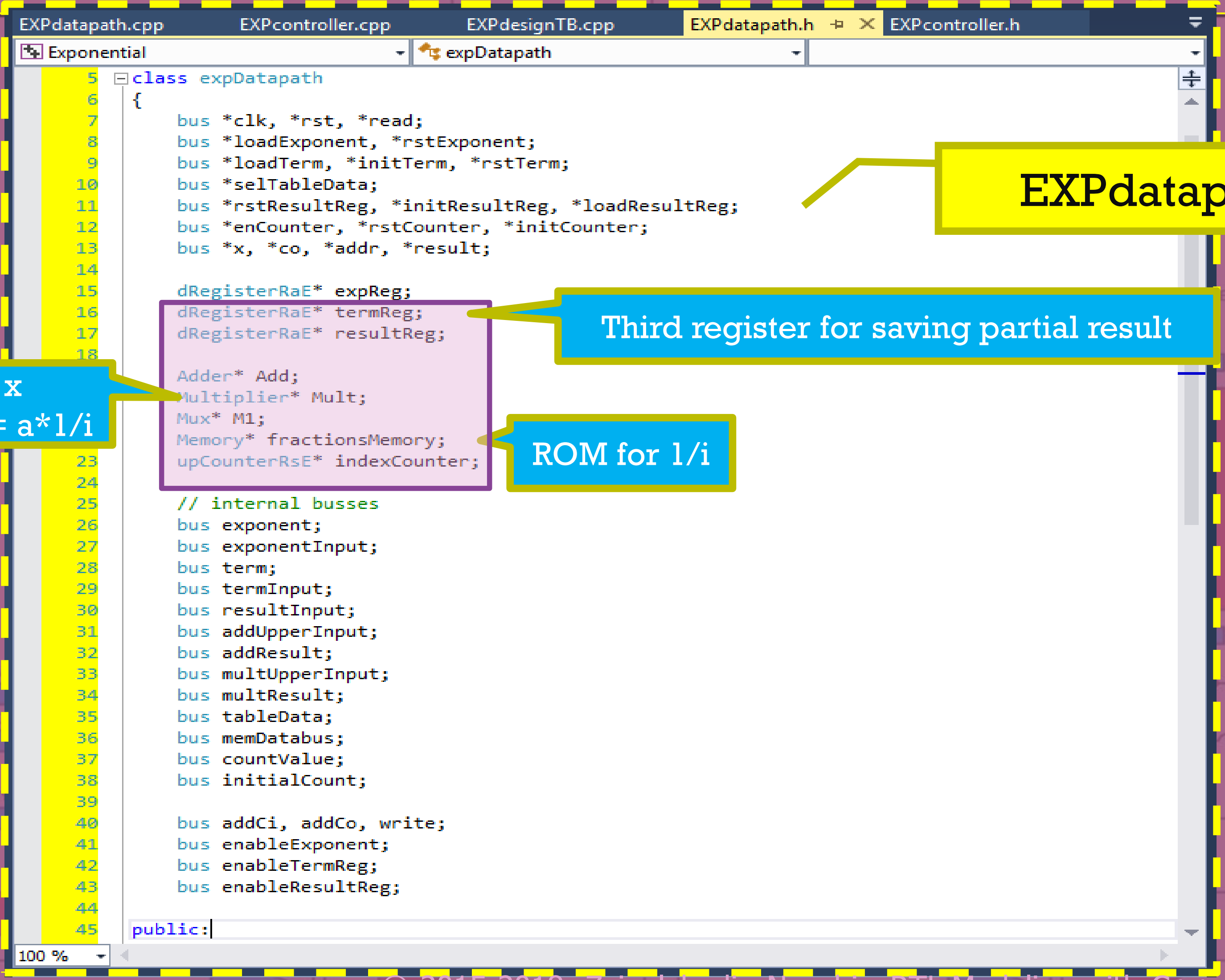
- With 8 bit input size, x is in the range between 0.00000000 for the smallest and 0.11111111 for the largest value.
- Smallest output = 1 when e^0
- Largest output = 10.1010111 (2.68357) when e^1

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 × ( 1 / i );
   e = e + a;
}
```

Exponential Circuit Datapath



```
class expDatapath
{
    bus *clk, *rst, *read;
    bus *loadExponent, *rstExponent;
    bus *loadTerm, *initTerm, *rstTerm;
    bus *selTableData;
    bus *rstResultReg, *initResultReg, *loadResultReg;
    bus *enCounter, *rstCounter, *initCounter;
    bus *x, *co, *addr, *result;

    dRegisterRaE* expReg;
    dRegisterRaE* termReg;
    dRegisterRaE* resultReg;

    Adder* Add;
    Multiplier* Mult;
    Mux* M1;
    Memory* fractionsMemory;
    upCounterRsE* indexCounter;

    // internal busses
    bus exponent;
    bus exponentInput;
    bus term;
    bus termInput;
    bus resultInput;
    bus addUpperInput;
    bus addResult;
    bus multUpperInput;
    bus multResult;
    bus tableData;
    bus memDatabus;
    bus countValue;
    bus initialCount;

    bus addCi, addCo, write;
    bus enableExponent;
    bus enableTermReg;
    bus enableResultReg;

public:
```

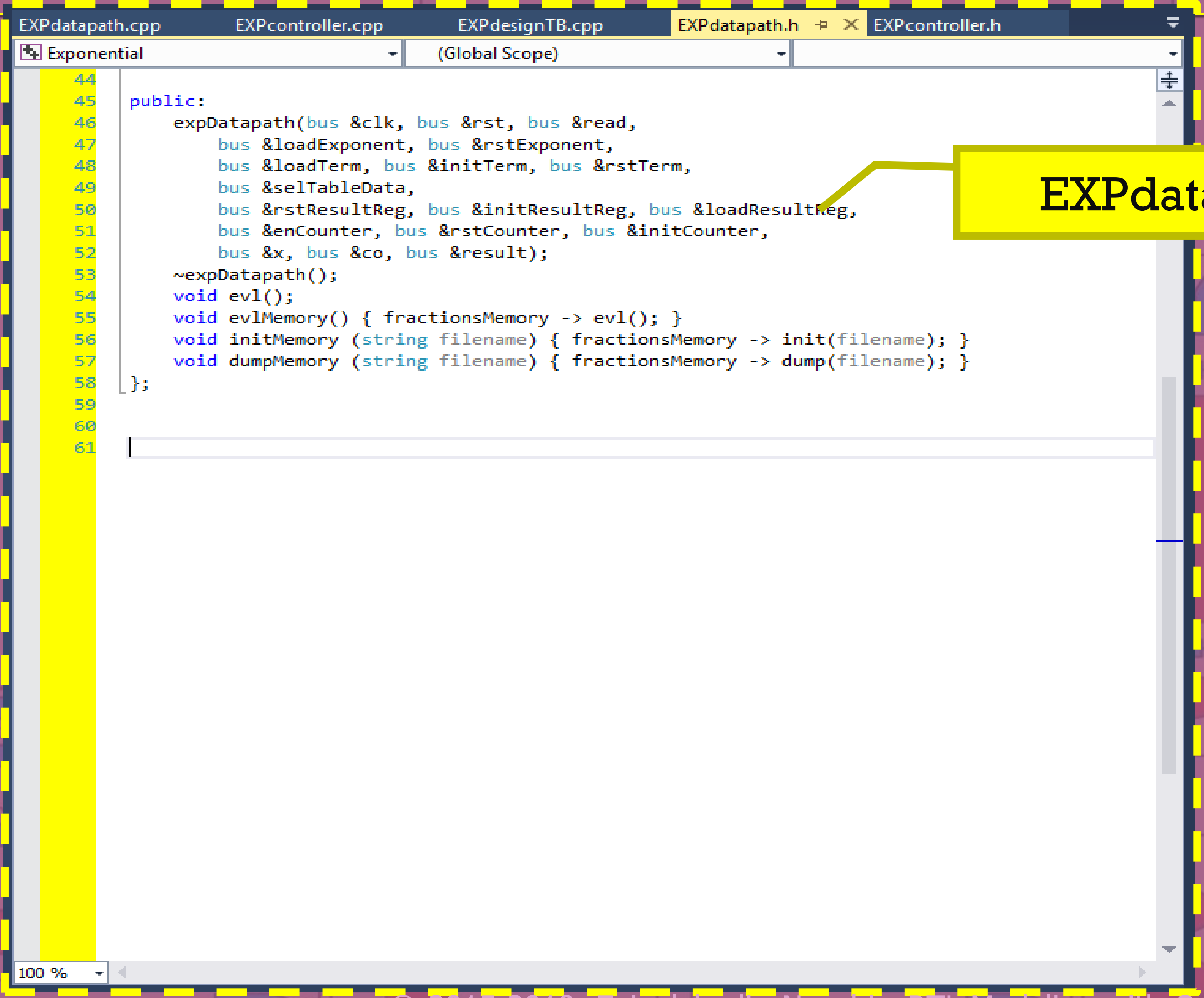
EXPdatapath.h

Third register for saving partial result

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

ROM for 1/i

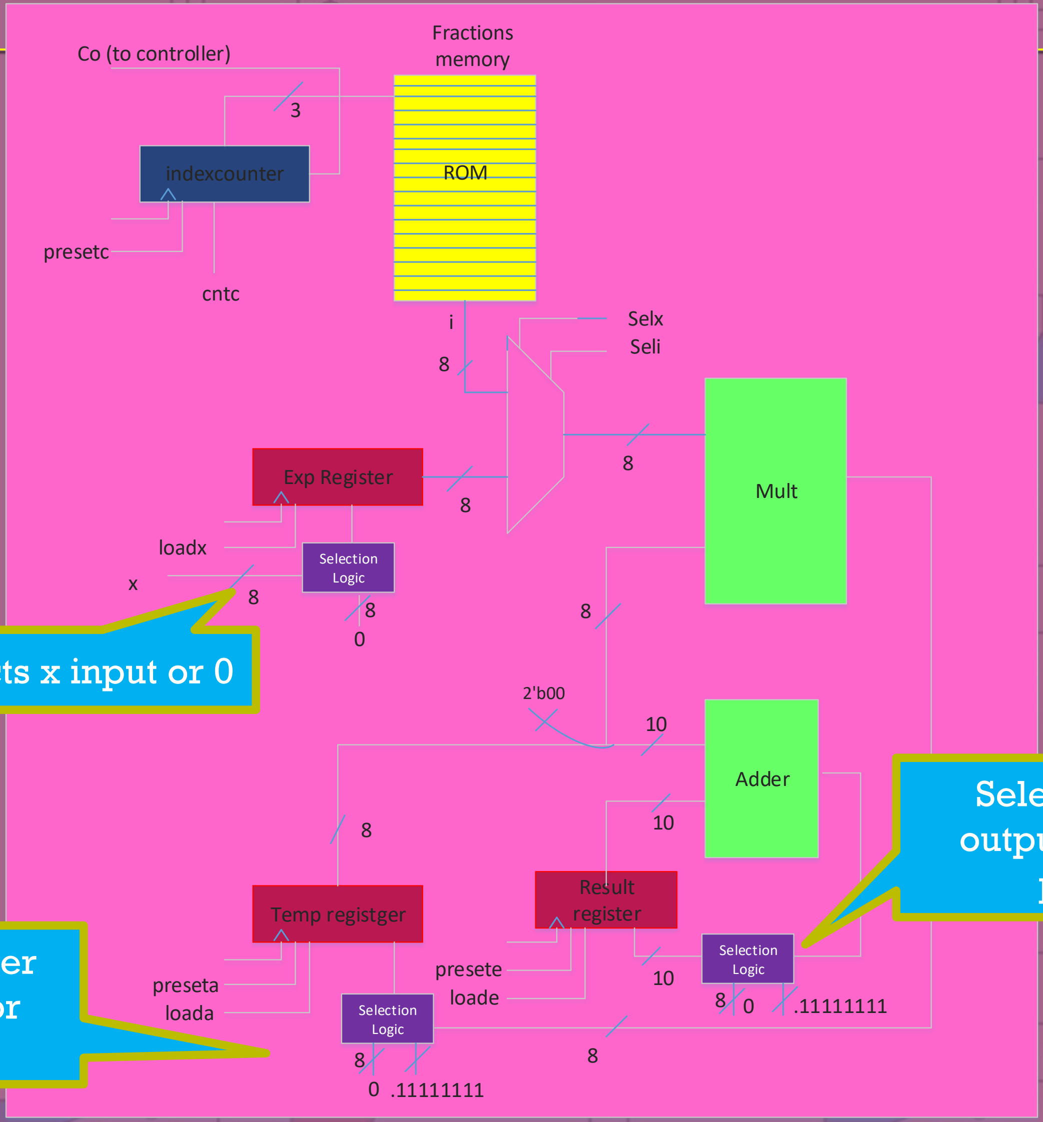
Exponential Circuit Datapath



```

public:
    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);
    ~expDatapath();
    void evl();
    void evlMemory() { fractionsMemory -> evl(); }
    void initMemory (string filename) { fractionsMemory -> init(filename); }
    void dumpMemory (string filename) { fractionsMemory -> dump(filename); }
};
```

EXPdatapath.h

Exponential Circuit Datapath

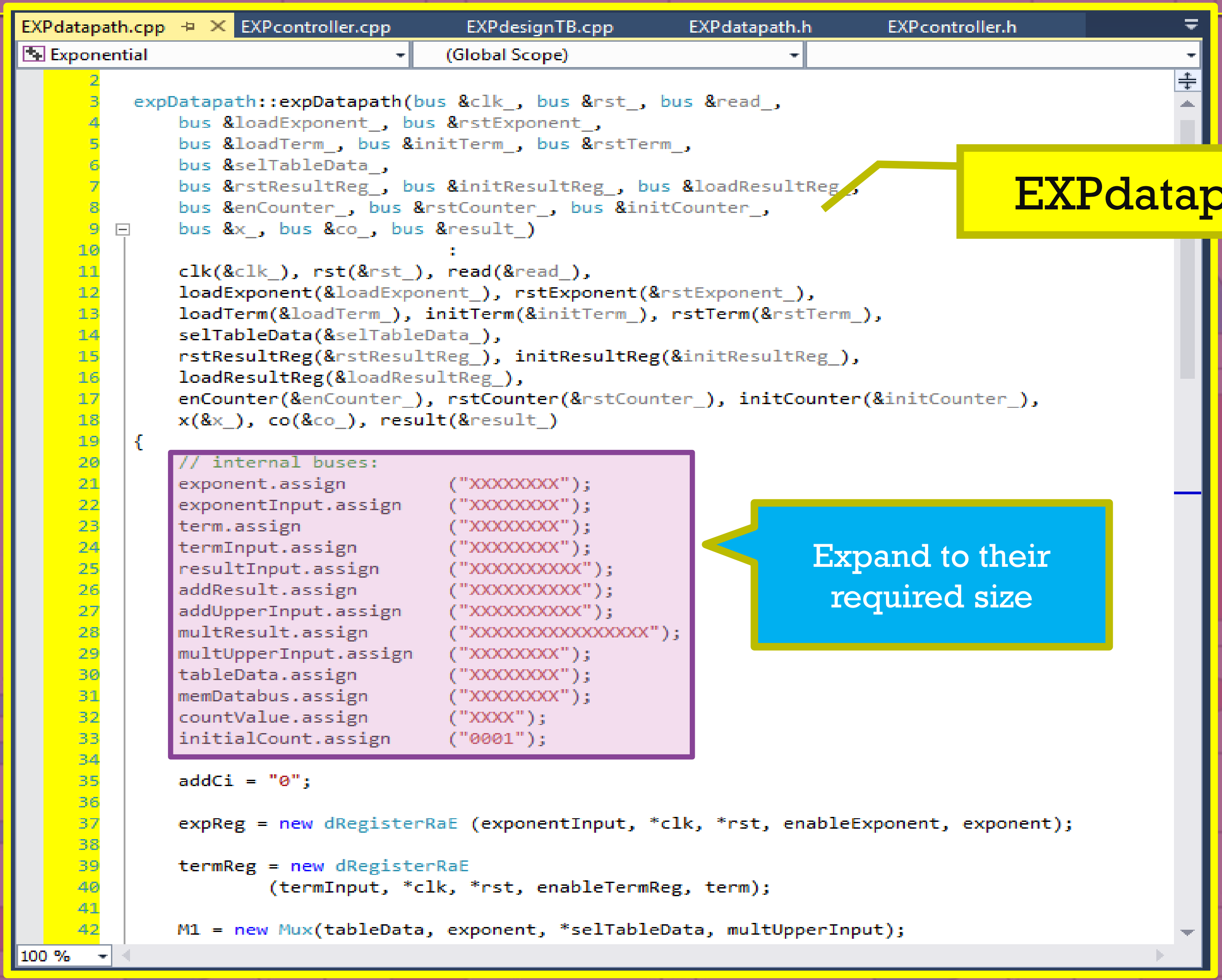


Selects x input or 0

Selects adder output, Reset or preset

Selects multiplier output, Reset or preset

Exponential Circuit Datapath

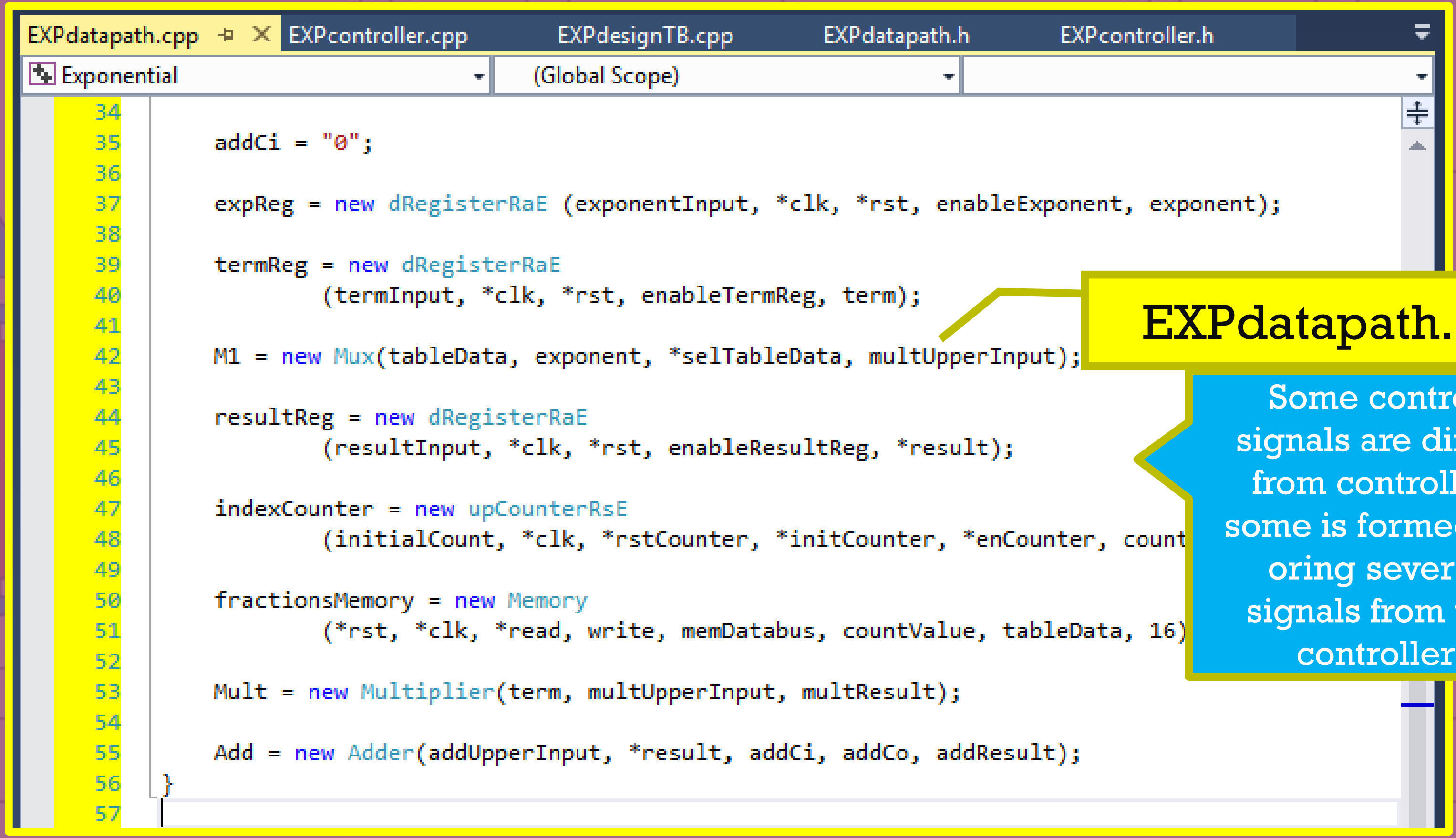


```

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);
```

EXPdatapath.cpp

Expand to their required size

Exponential Circuit Datapath

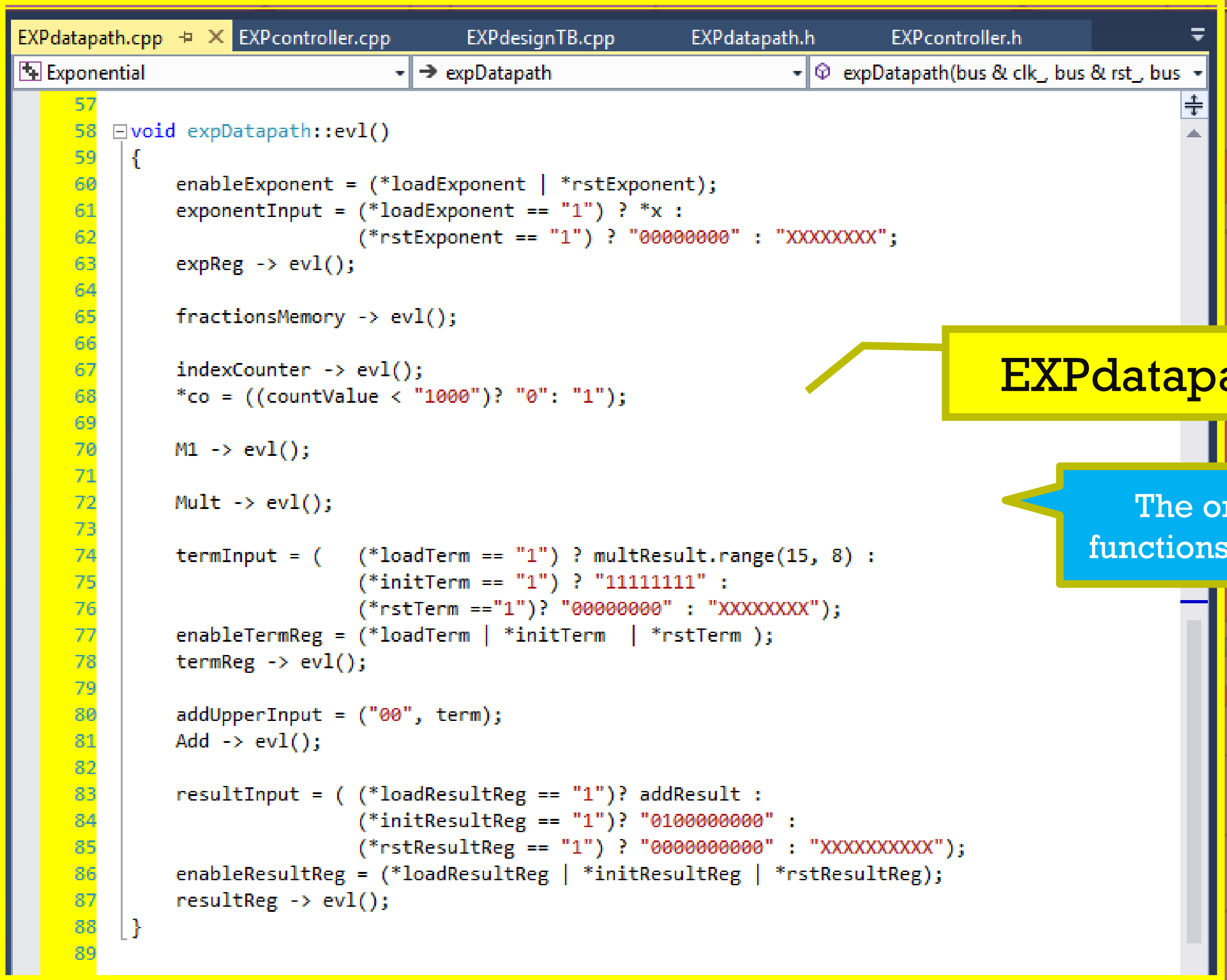


```

    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);

    resultReg = new dRegisterRaE
            (resultInput, *clk, *rst, enableResultReg, *result);

    indexCounter = new upCounterRsE
            (initialCount, *clk, *rstCounter, *initCounter, *enCounter, count

    fractionsMemory = new Memory
            (*rst, *clk, *read, write, memDatabus, countValue, tableData, 16)

    Mult = new Multiplier(term, multUpperInput, multResult);

    Add = new Adder(addUpperInput, *result, addCi, addCo, addResult);
}

```

EXPdatapath.cpp

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

Exponential Circuit Datapath

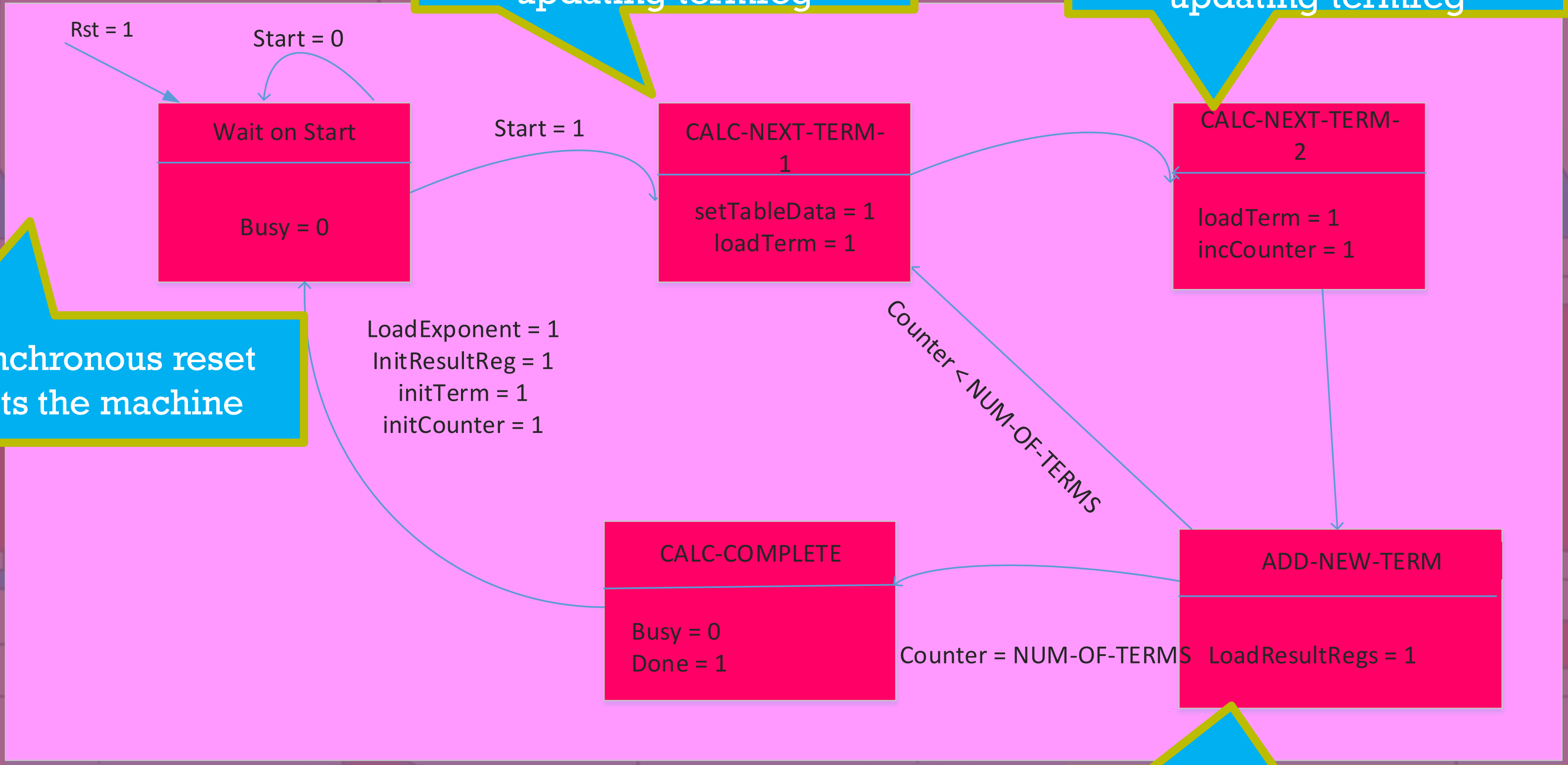


```cpp

void expDatapath::evl()
{
    enableExponent = (*loadExponent | *rstExponent);
    exponentInput = (*loadExponent == "1") ? *x :
                    (*rstExponent == "1") ? "00000000" : "XXXXXXXX";
    expReg -> evl();

    fractionsMemory -> evl();

    indexCounter -> evl();
    *co = ((countValue < "1000")? "0": "1");

    M1 -> evl();

    Mult -> evl();

    termInput = (   (*loadTerm == "1") ? multResult.range(15, 8) :
                    (*initTerm == "1") ? "11111111" :
                    (*rstTerm =="1")? "00000000" : "XXXXXXXX");
    enableTermReg = (*loadTerm | *initTerm  | *rstTerm );
    termReg -> evl();

    addUpperInput = ("00", term);
    Add -> evl();

    resultInput = ( (*loadResultReg == "1")? addResult :
                    (*initResultReg == "1")? "0100000000" :
                    (*rstResultReg == "1") ? "0000000000" : "XXXXXXXXXX");
    enableResultReg = (*loadResultReg | *initResultReg | *rstResultReg);
    resultReg -> evl();
}

```

EXPdatapath.cpp

The ordering of functions is important

Exponential Circuit Controller



Rst = 1

Start = 0

Wait on Start

Busy = 0

Start = 1

CALC-NEXT-TERM-1

setTableData = 1
loadTerm = 1

CALC-NEXT-TERM-2

loadTerm = 1
incCounter = 1

LoadExponent = 1
InitResultReg = 1
initTerm = 1
initCounter = 1

Counter < NUM-OF-TERMS

CALC-COMPLETE

Busy = 0
Done = 1

Counter = NUM-OF-TERMS

ADD-NEW-TERM

LoadResultRegs = 1

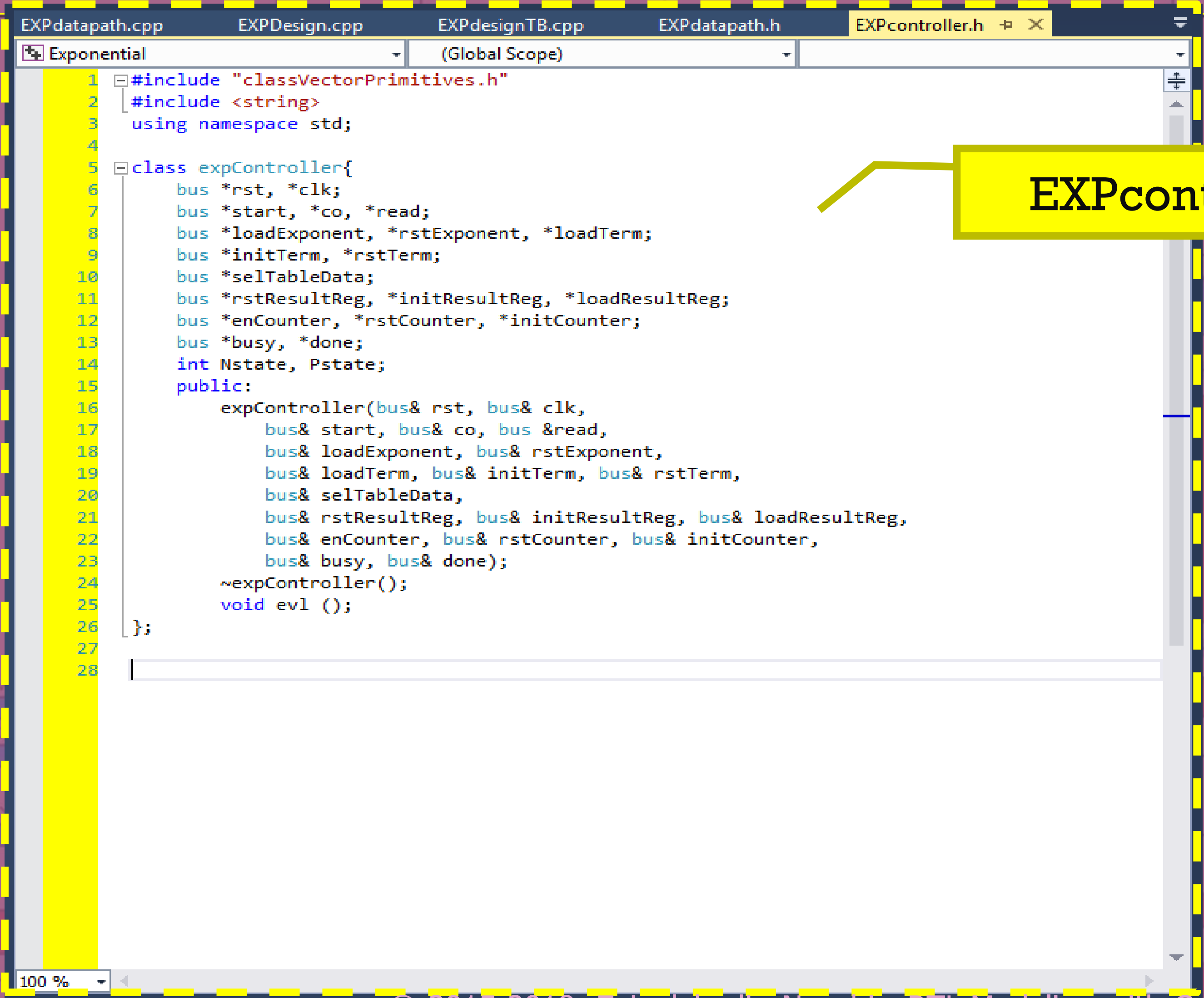
Multiplying the termreg with fractional memory, updating termreg

Multiplying the updated termreg by exponent and updating termreg

Asynchronous reset starts the machine

Adding the new termreg contenes to result reg

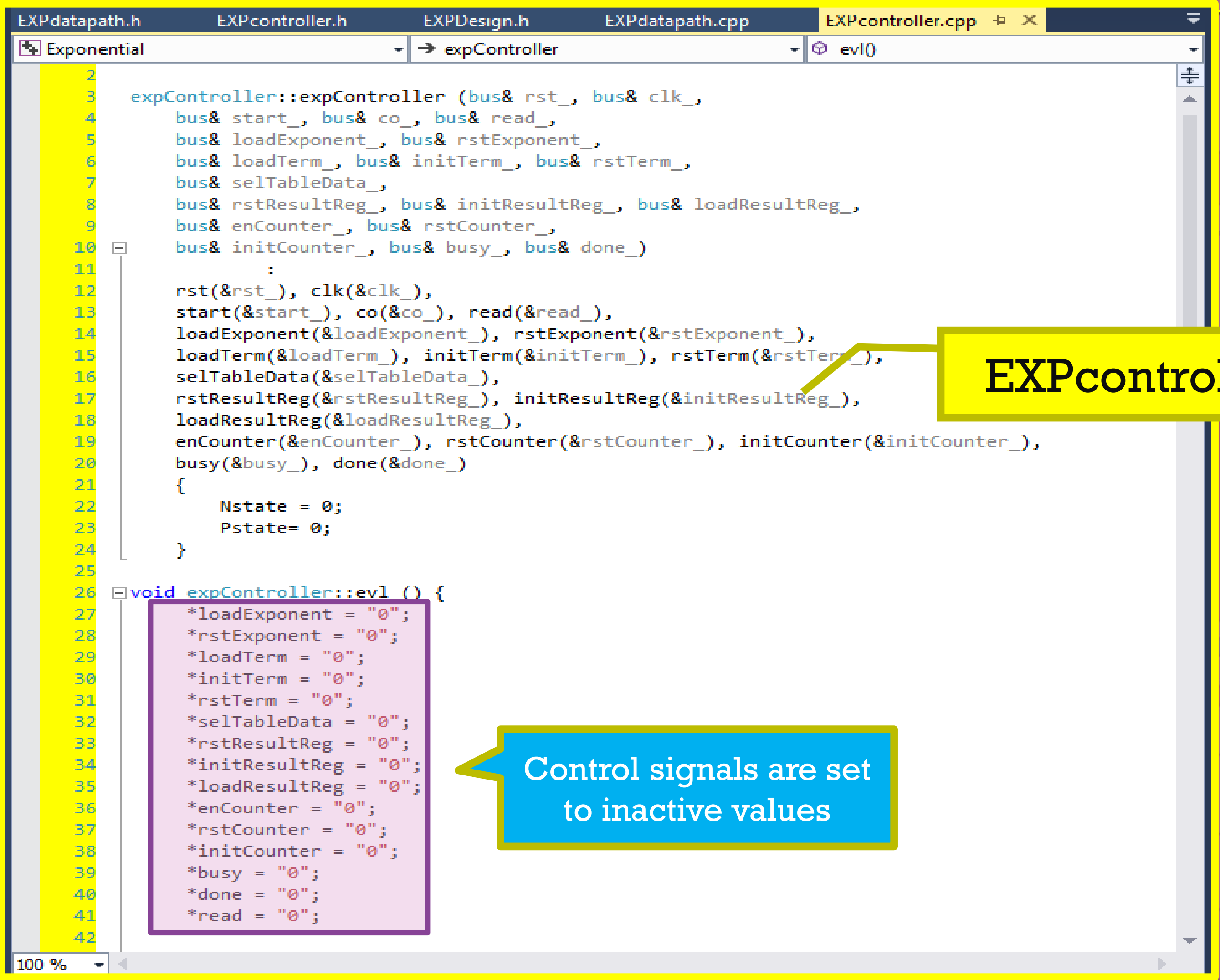
Exponential Circuit Controller



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

class expController{
    bus *rst, *clk;
    bus *start, *co, *read;
    bus *loadExponent, *rstExponent, *loadTerm;
    bus *initTerm, *rstTerm;
    bus *selTableData;
    bus *rstResultReg, *initResultReg, *loadResultReg;
    bus *enCounter, *rstCounter, *initCounter;
    bus *busy, *done;
    int Nstate, Pstate;
    public:
        expController(bus& rst, bus& clk,
            bus& start, bus& co, 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& busy, bus& done);
        ~expController();
        void evl ();
};
```

EXPcontroller.h

Exponential Circuit Controller

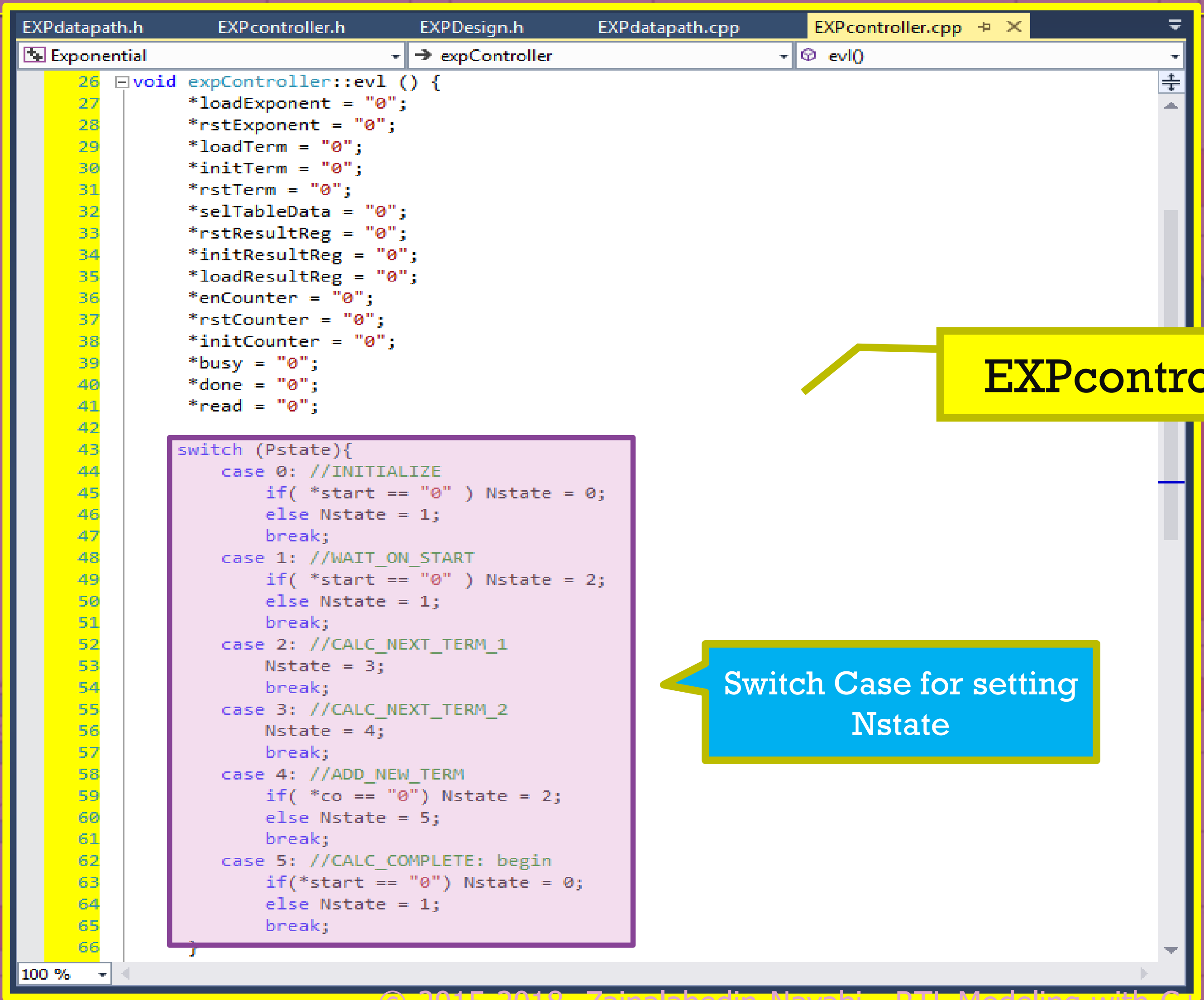


```

expController::expController (bus& rst_, bus& clk_,
    bus& start_, bus& co_, 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& busy_, bus& done_)
            :
    rst(&rst_), clk(&clk_),
    start(&start_), co(&co_), 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_),
    busy(&busy_), done(&done_)
    {
        Nstate = 0;
        Pstate= 0;
    }

void expController::evl () {
     *loadExponent = "0";
     *rstExponent = "0";
     *loadTerm = "0";
     *initTerm = "0";
     *rstTerm = "0";
     *selTableData = "0";
     *rstResultReg = "0";
     *initResultReg = "0";
     *loadResultReg = "0";
     *enCounter = "0";
     *rstCounter = "0";
     *initCounter = "0";
     *busy = "0";
     *done = "0";
     *read = "0";

```

EXPcontroller.cpp

Control signals are set to inactive values

Exponential Circuit Controller

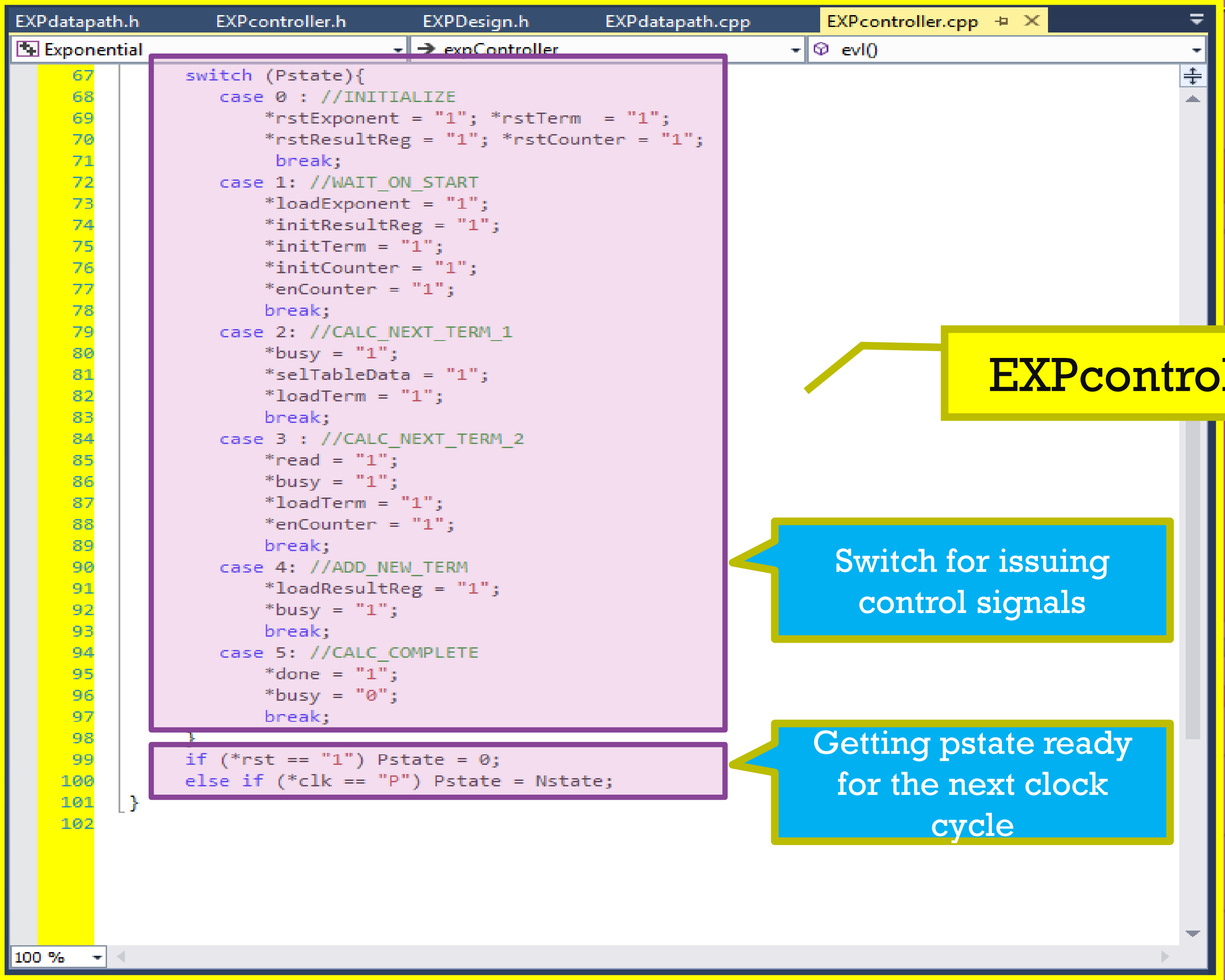


```
void expController::evl () {
    *loadExponent = "0";
    *rstExponent = "0";
    *loadTerm = "0";
    *initTerm = "0";
    *rstTerm = "0";
    *selTableData = "0";
    *rstResultReg = "0";
    *initResultReg = "0";
    *loadResultReg = "0";
    *enCounter = "0";
    *rstCounter = "0";
    *initCounter = "0";
    *busy = "0";
    *done = "0";
    *read = "0";

    switch (Pstate){
        case 0: //INITIALIZE
            if( *start == "0" ) Nstate = 0;
            else Nstate = 1;
            break;
        case 1: //WAIT_ON_START
            if( *start == "0" ) Nstate = 2;
            else Nstate = 1;
            break;
        case 2: //CALC_NEXT_TERM_1
            Nstate = 3;
            break;
        case 3: //CALC_NEXT_TERM_2
            Nstate = 4;
            break;
        case 4: //ADD_NEW_TERM
            if( *co == "0") Nstate = 2;
            else Nstate = 5;
            break;
        case 5: //CALC_COMPLETE: begin
            if(*start == "0") Nstate = 0;
            else Nstate = 1;
            break;
    }
```

EXPcontroller.cpp

Switch Case for setting Nstate

Exponential Circuit Controller



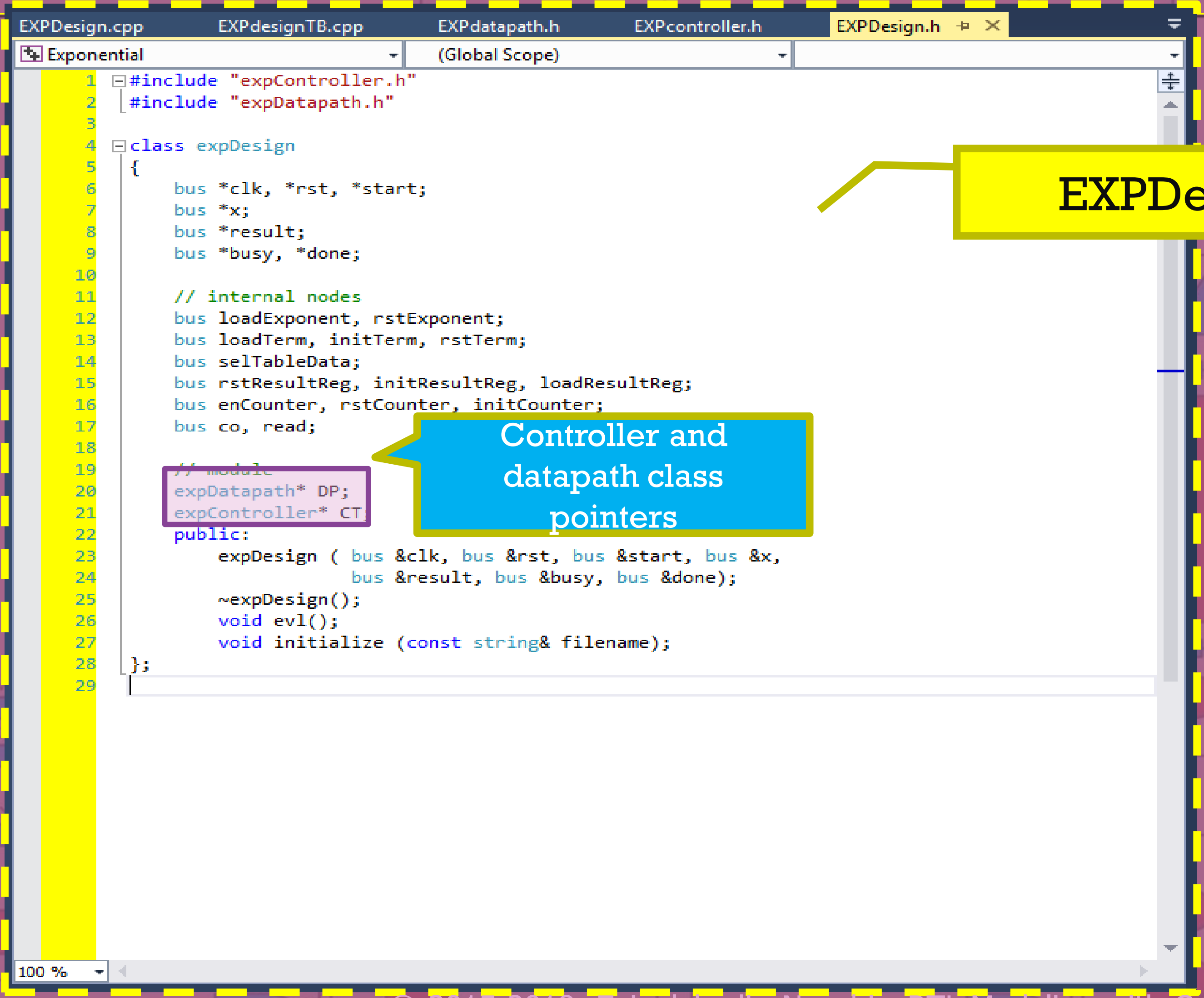
```
     switch (Pstate){
        case 0 : //INITIALIZE
            *rstExponent = "1"; *rstTerm  = "1";
            *rstResultReg = "1"; *rstCounter = "1";
             break;
        case 1: //WAIT_ON_START
            *loadExponent = "1";
            *initResultReg = "1";
            *initTerm = "1";
            *initCounter = "1";
            *enCounter = "1";
            break;
        case 2: //CALC_NEXT_TERM_1
            *busy = "1";
            *selTableData = "1";
            *loadTerm = "1";
            break;
        case 3 : //CALC_NEXT_TERM_2
            *read = "1";
            *busy = "1";
            *loadTerm = "1";
            *enCounter = "1";
            break;
        case 4: //ADD_NEW_TERM
            *loadResultReg = "1";
            *busy = "1";
            break;
        case 5: //CALC_COMPLETE
            *done = "1";
            *busy = "0";
            break;
     }
     if (*rst == "1") Pstate = 0;
     else if (*clk == "P") Pstate = Nstate;
}
```

EXPcontroller.cpp

Switch for issuing control signals

Getting pstate ready for the next clock cycle

Exponential Circuit Design

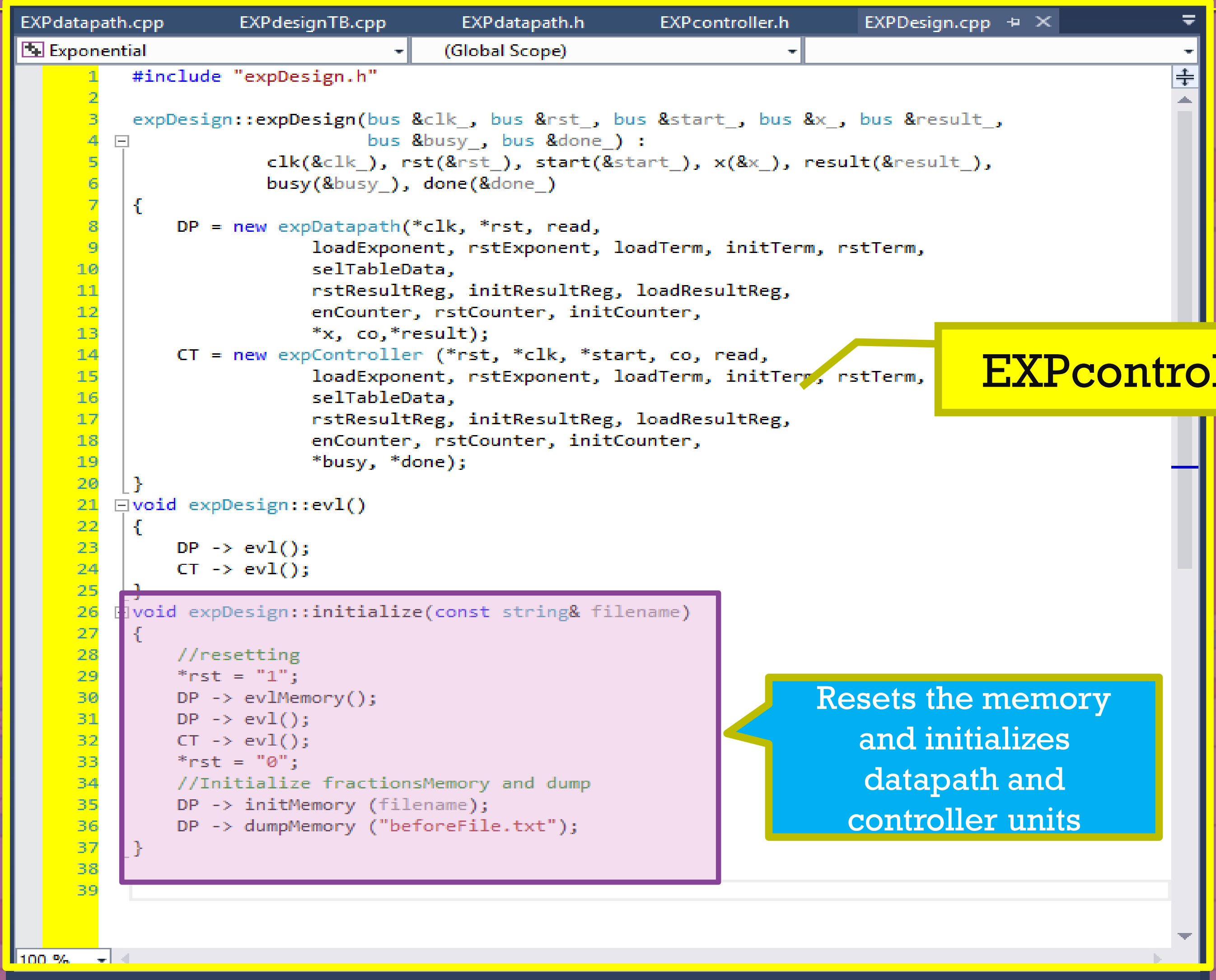


```
#include "expController.h"
#include "expDatapath.h"

class expDesign
{
    bus *clk, *rst, *start;
    bus *x;
    bus *result;
    bus *busy, *done;

    // internal nodes
    bus loadExponent, rstExponent;
    bus loadTerm, initTerm, rstTerm;
    bus selTableData;
    bus rstResultReg, initResultReg, loadResultReg;
    bus enCounter, rstCounter, initCounter;
    bus co, read;

    // module
    expDatapath* DP;
    expController* CT
    public:
        expDesign ( bus &clk, bus &rst, bus &start, bus &x,
                    bus &result, bus &busy, bus &done);
        ~expDesign();
        void evl();
        void initialize (const string& filename);
};

```

EXPDesign.h

Controller and datapath class pointers

Exponential Circuit Design

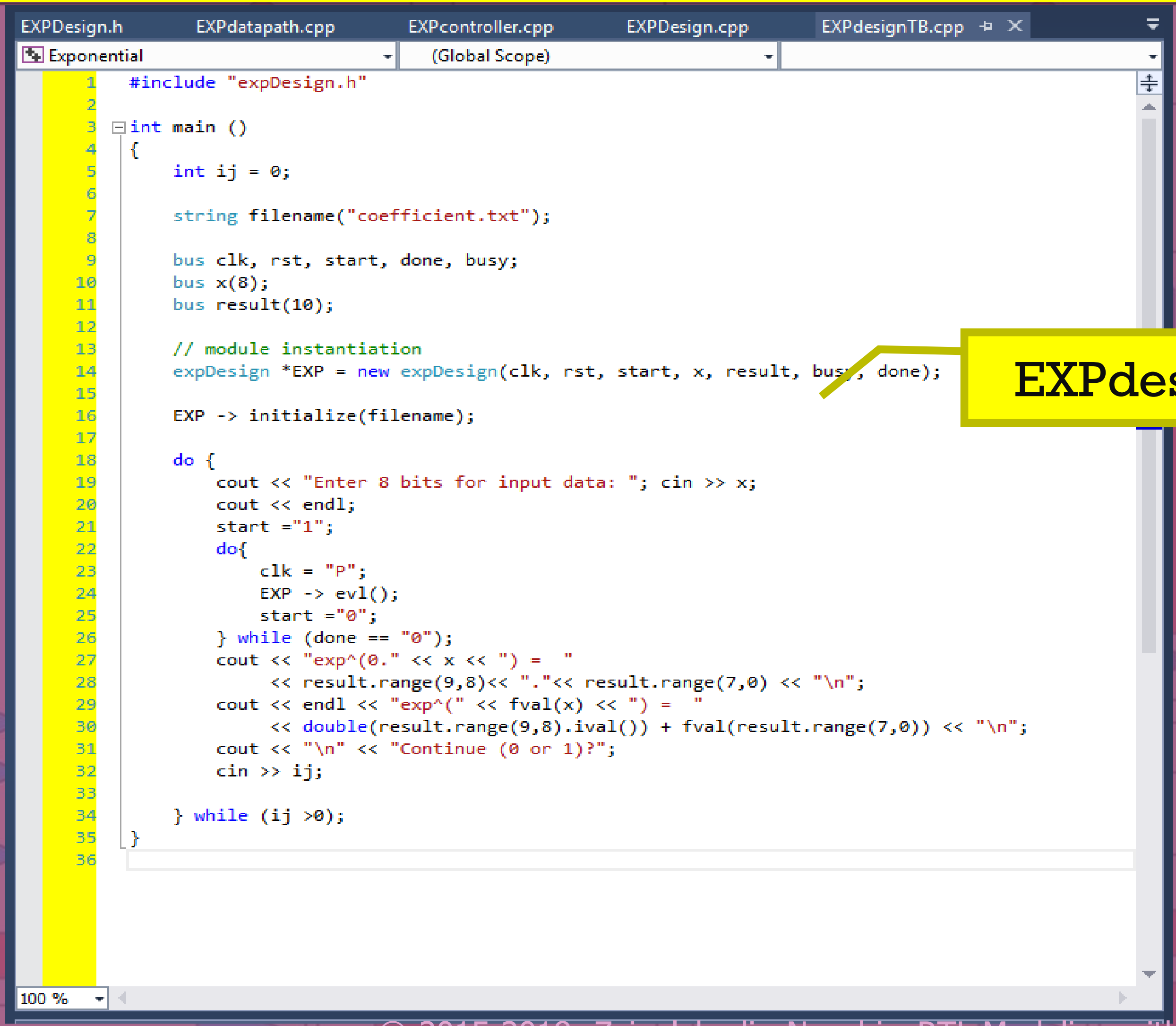


```
#include "expDesign.h"

expDesign::expDesign(bus &clk_, bus &rst_, bus &start_, bus &x_, bus &result_,
                     bus &busy_, bus &done_) :
            clk(&clk_), rst(&rst_), start(&start_), x(&x_), result(&result_),
            busy(&busy_), done(&done_)
{
    DP = new expDatapath(*clk, *rst, read,
                loadExponent, rstExponent, loadTerm, initTerm, rstTerm,
                selTableData,
                rstResultReg, initResultReg, loadResultReg,
                enCounter, rstCounter, initCounter,
                *x, co,*result);
    CT = new expController (*rst, *clk, *start, co, read,
                loadExponent, rstExponent, loadTerm, initTerm, rstTerm,
                selTableData,
                rstResultReg, initResultReg, loadResultReg,
                enCounter, rstCounter, initCounter,
                *busy, *done);
}
void expDesign::evl()
{
    DP -> evl();
    CT -> evl();
}
void expDesign::initialize(const string& filename)
{
    //resetting
    *rst = "1";
    DP -> evlMemory();
    DP -> evl();
    CT -> evl();
    *rst = "0";
    //Initialize fractionsMemory and dump
    DP -> initMemory (filename);
    DP -> dumpMemory ("beforeFile.txt");
}
```

EXPcontroller.cpp

Resets the memory and initializes datapath and controller units

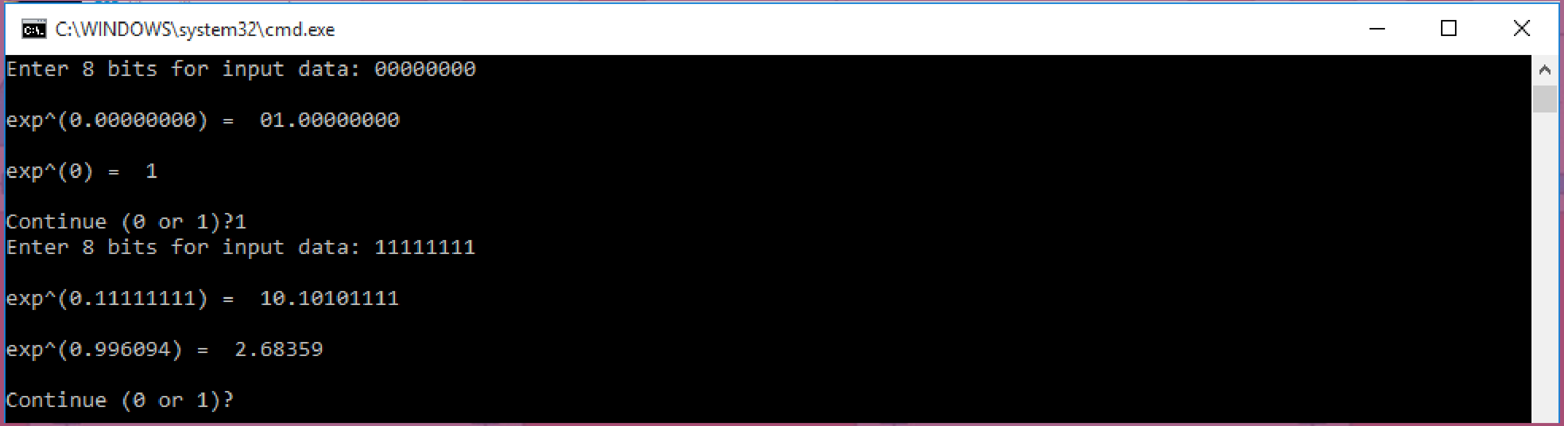
Exponential Circuit Design



```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);
}
```

EXPdesignTB.cpp

Exponential Circuit Design



```
Enter 8 bits for input data: 00000000

exp^(0.00000000) =  01.00000000

exp^(0) =  1

Continue (0 or 1)?1
Enter 8 bits for input data: 11111111

exp^(0.11111111) =  10.10101111

exp^(0.996094) =  2.68359

Continue (0 or 1)?
```

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