Chapter 2

Logic Simulation with C/C++ Programming Language

Zainalabedin Navabi

Logic Simulation with C/C++

- Procedural Languages for Hardware Modeling
- Types and Operators for Logic Modeling
- Basic Logic Simulation
  - Logic functions
  - Function overloading
  - Passing logic functions
  - Using default values
  - Building higher level structures
  - Handling 4-value logic
  - Logic vector
  - Sequential circuit modeling
  - Using pointers for logic vectors
- Enhanced logic simulation with timing
  - Using struct for timing and logic
  - Gates that handle timing
  - Utility functions
  - Timing in logic structures
  - Overloading logical operators
  - Using Boolean expressions
- More Functions for Wires and Gates
  - Gate classes
  - Carrier generic modeling
  - Compatible scalar and vector

Logic Simulation with C/C++

- Containing Event Based Timing
  - To include in wires
  - To include in gates
- Gate-based structures
- Gate pointers and objects
- Wire and gate vectors

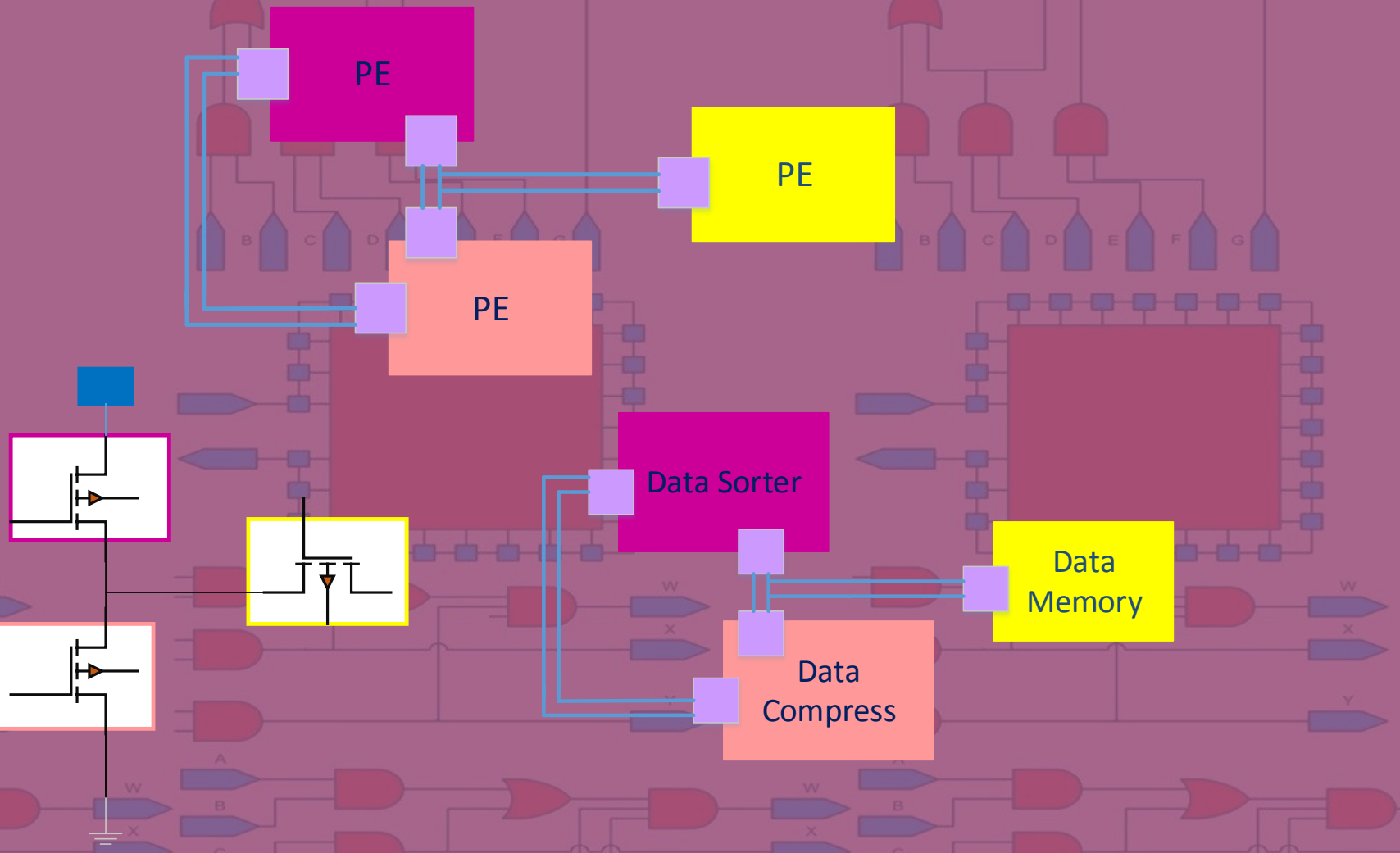
- Inheritance in Logic Structures
  - A generic gate definition
  - Gates to include timing
  - Building structures from objects

- Hierarchal Modeling of Digital Components
  - Wire functionalities
  - Gate functionalities
  - Polymorphic gate base
  - Virtual functions
  - Functions overwriting
  - Flip flop description hierarchal

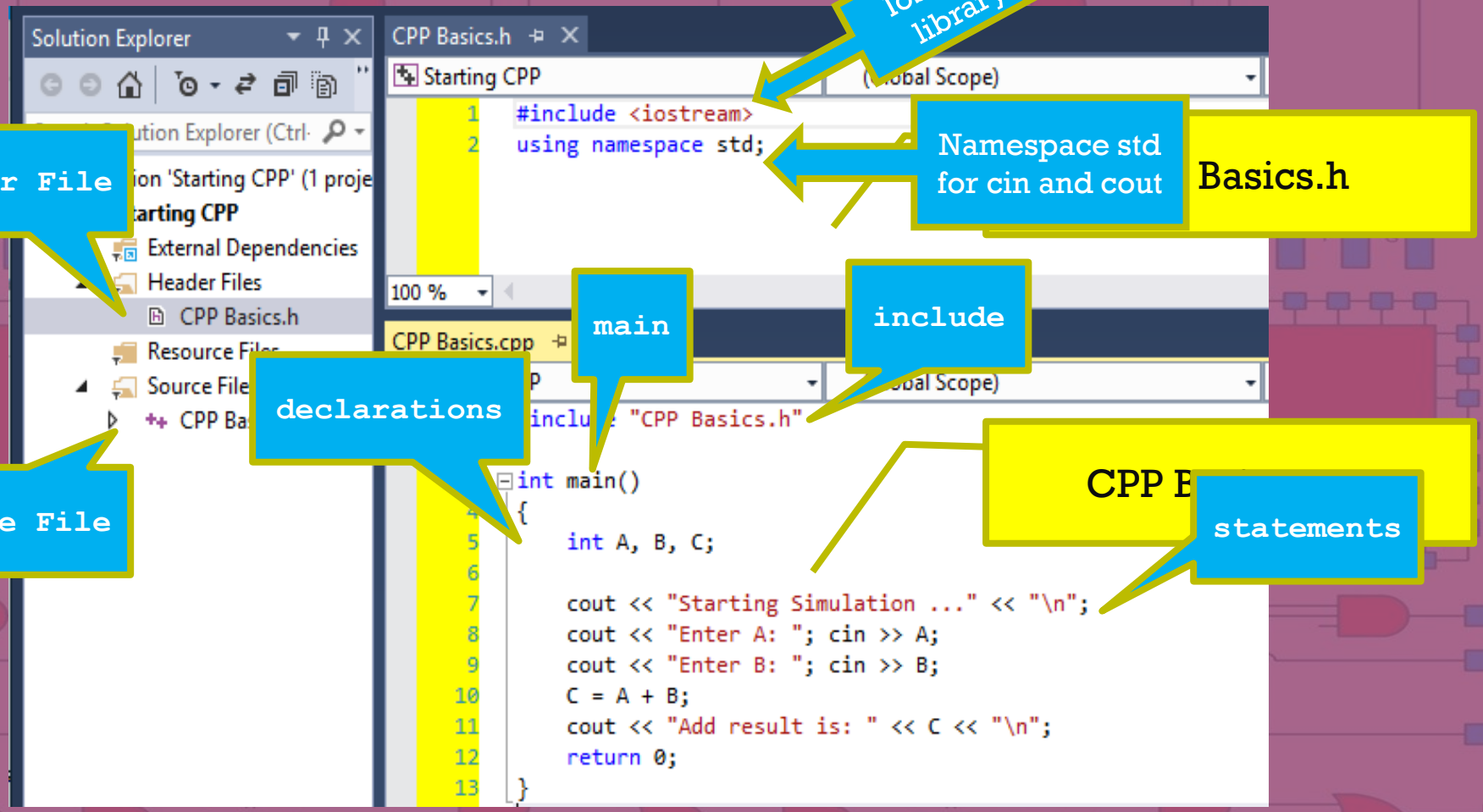
Logic Simulation with C/C++

- Procedural Languages for Hardware Modeling
- Types and Operators for Logic Modeling
- Basic Logic Simulation
  - Logic functions
  - Function overloading
  - Passing logic functions
  - Using default values
  - Building higher level structures
  - Handling 4-value logic
  - Logic vector
  - Sequential circuit modeling
  - Using pointers for logic vectors
- Enhanced logic simulation with timing
  - Using struct for timing and logic
  - Gates that handle timing
  - Utility functions
  - Timing in logic structures
  - Overloading logical operators
  - Using Boolean expressions
- More Functions for Wires and Gates
  - Gate classes
  - Carrier centric modeling
  - Compatible scalar and vector

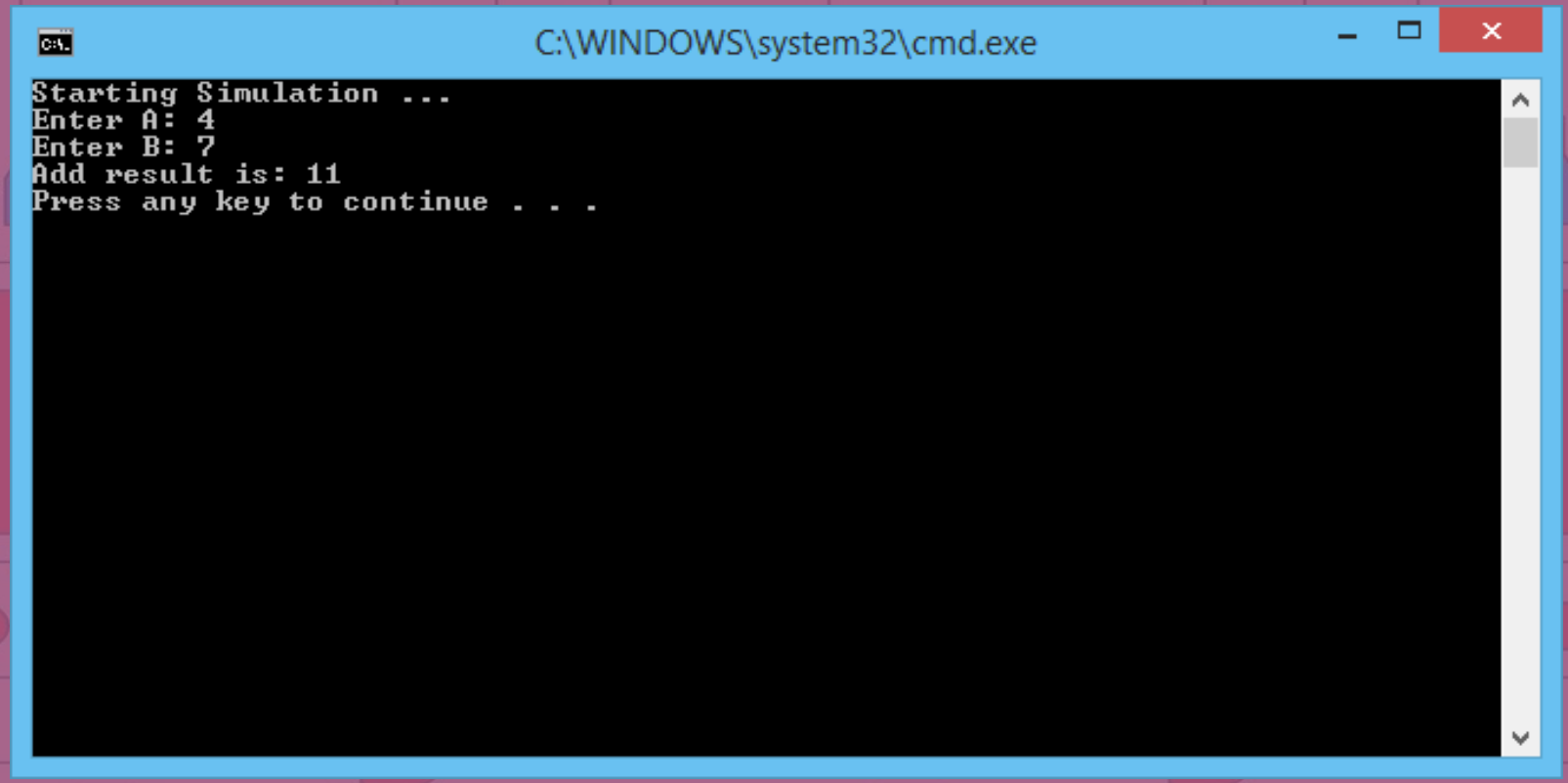
Hardware Modeling



C++ Environment



C++ Environment



Logic Simulation with C/C++

- Procedural Languages for Hardware Modeling
- Types and Operators for Logic Modeling
- Basic Logic Simulation
  - Logic functions
  - Function overloading
  - Passing logic functions
  - Using default values
  - Building higher level structures
  - Handling 4-value logic
  - Logic vector
  - Sequential circuit modeling
  - Using pointers for logic vectors
- Enhanced logic simulation with timing
  - Using struct for timing and logic
  - Gates that handle timing
  - Utility functions
  - Timing in logic structures
  - Overloading logical operators
  - Using Boolean expressions
- More Functions for Wires and Gates
  - Gate classes
  - Carrier centric modeling
  - Compatible scalar and vector

Types and Operators for logic Modeling

Group	Type names	Note on size/Precision
Character Types	Char	Exactly one byte in size. At least 8 bits
Integer Types (signed)	Signed Char	Same size as char. At least 8 bits
	Signed Int	At least 16 bits
Integer Types (unsigned)	Unsigned Char	Same size as char. At least 8 bits
	Unsigned Int	At least 16 bits
Floating-point Type	Float	
	Double	Precision not less than float
	Long Double	Precision not less than float
Boolean Type	Bool	
Void Type	Void	No storage

Using Boolean Type

Boolean Type.h

```
#include <iostream>
using namespace std;
```

Boolean Type.cpp

```
//bool a = true
//bool b = fal
bool a(0);
bool b(1);
bool anding;
int go;
cout << "Performing Lo  Simulation . . .\n";
anding = a && b;
cout << "a:" << a << "; b:" << b << "; anding:" << anding << "\n";
cout << "Enter 0 to exit:";
cin >> go;

return 0;
}
```

declarations

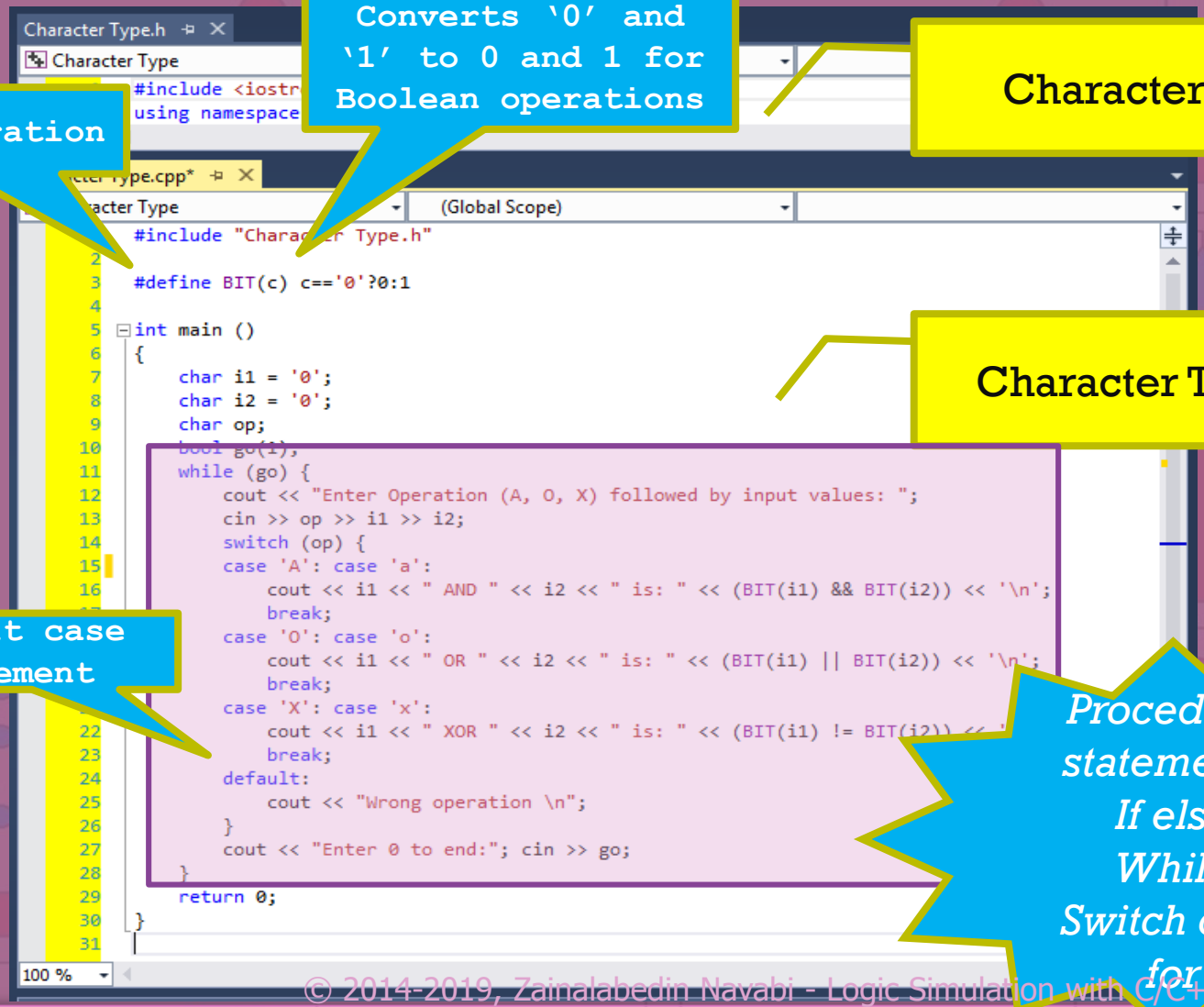
initialization

operations

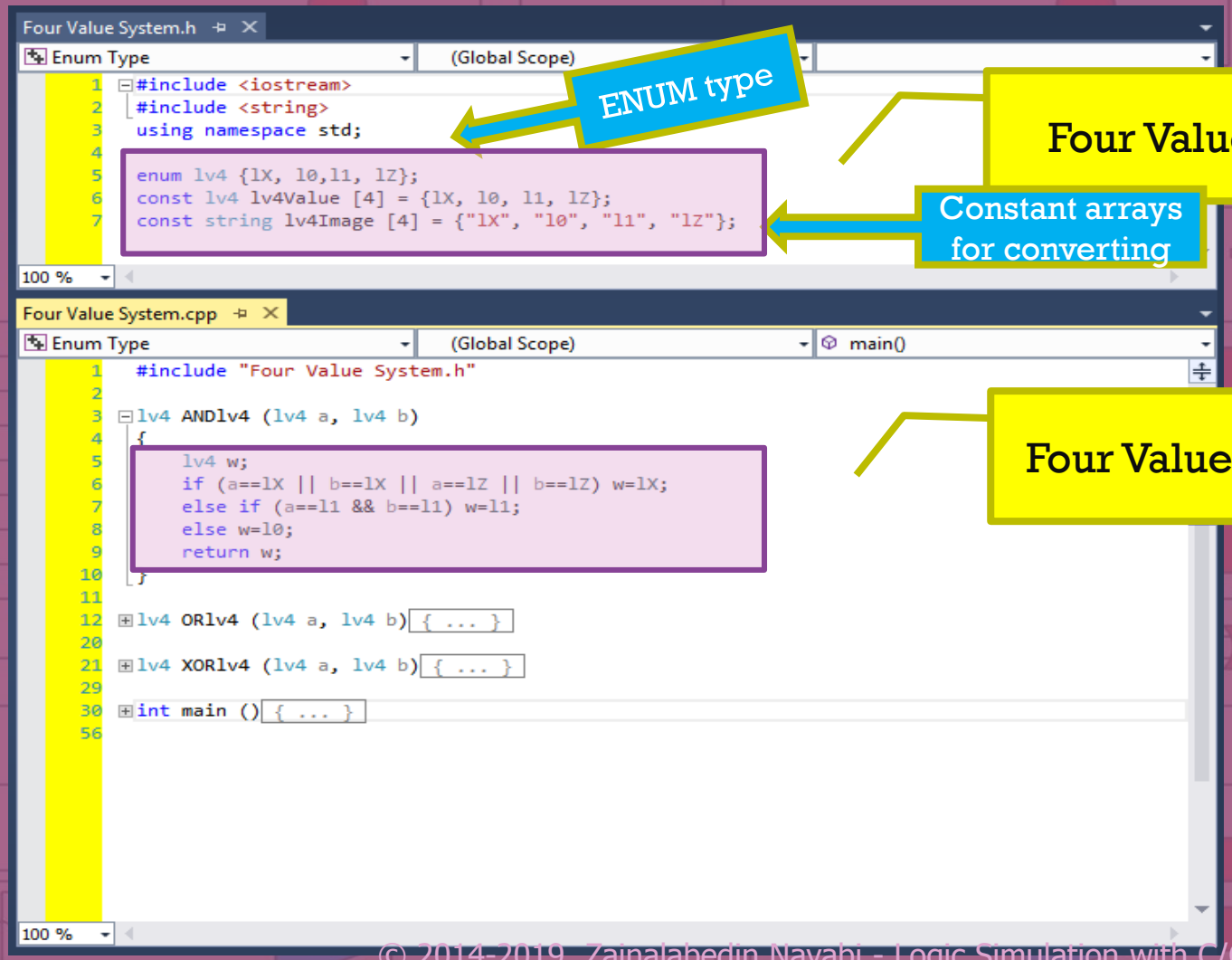
Boolean Type.h

Boolean Type.cpp

Types and Operators for Logic Modeling



Using Enumerators



Using Enumerators

Four Value System.cpp | Four Value System.h

Enum Type | (Global Scope)

```
1   #include "Four Value System.h"
2
3   lv4 ANDlv4 (lv4 a, lv4 b){ ... }
11
12  lv4 ORlv4 (lv4 a, lv4 b){ ... }
20
21  lv4 XORlv4 (lv4 a, lv4 b){ ... }
29
30  int main ()
31  {
32      lv4 i1 = lX;
33      lv4 i2 = lX;
34      lv4 out = lX;
35      int Ii1, Ii2, Iout;
36      char op;
37      bool go;
38      do {
39          cout << "Enter operation (A,O,X), then inputs (0 to 3): ";
40          cin >> op >> Ii1 >> Ii2;
41          i1=lv4Value[Ii1]; i2=lv4Value[Ii2];
42          switch (op) {
43              case 'A': out = ANDlv4 (i1, i2); break;
44              case 'O': out = ORlv4 (i1, i2); break;
45              case 'X': out = XORlv4 (i1, i2); break;
46              default: out = lX;
47          }
48          cout << i1 << " " << op << " " << i2;
49          cout << ", is: " << out << '\n';
50          cout << lv4Image[i1] << " " << op << " " << lv4Image[i2];
51          cout << ", is: " << lv4Image[out] << '\n';
52          cout << "Enter 0 to end:"; cin >> go;
53      } while (go);
54      return 0;
55  }
56
```

100 %

Four Value System.cpp

Convert Integer value to lv4 enumeration type

Conversion to string for printing

Waveform Generation

String Character.h

```
#include <iostream>
#include <string>
using namespace std;
```

String Character.cpp

```
#define MIN(a,b) (a<b)?a:b

int wave (string seq)
{   int i, l;
    l = seq.length();
    for (i=0; i < l; i++)
    {
        if (seq[i]=='0') cout << "__";
        else cout << "--";
    }
    cout << '\n';
    return l;
}

bool char2bool (char c)
{
    if (c=='0') return 0;
    else return 1;
}

bool operation (string fn, bool in1, bool in2)
{
    bool out;
    if (fn == "AND" || fn == "and") out = in1 && in2;
    else if (fn == "OR" || fn == "or") out = in1 || in2;
    else if (fn == "XOR" || fn == "xor") out = in1 != in2;
    else out = 0;
    return out;
    }
```

MIN Macro

Got a sequence of 1s and 0s. Turn into waveform

Equivalent to BIT macro

Bool is good for logical operations

Types and Operators for Logic Modeling

String Character.cpp

```
17  bool char2bool (char c) { ... }
22
23  bool operation (string fn, bool in1, bool in2) { ... }
32
33  int main ()
34  {
35      string i1Seq, i2Seq;
36      string logic;
37      int i, i1Len, i2Len, outLen;
38      bool i1=0, i2=0, out=0;
39      bool go(1);
40      while (go) {
41          cout << "Enter logic type and input sequences
42          cin >> logic >> i1Seq >> i2Seq;
43          i1Len=wave (i1Seq);
44          i2Len=wave (i2Seq);
45          outLen = MIN (i1Len, i2Len);
46          string outSeq (outLen, '0');
47          for (i=0; i<outLen; i++) {
48              i1 = char2bool (i1Seq[i]);
49              i2 = char2bool (i2Seq[i]);
50              out=operation(logic, i1,i2);
51              outSeq[i] = out ? '1' : '0';
52          }
53          outLen=wave (outSeq); cout << '\n';
54          cout << "Enter 0 to end:"; cin >> go;
55      }
56      return 0;
57  }
58
```

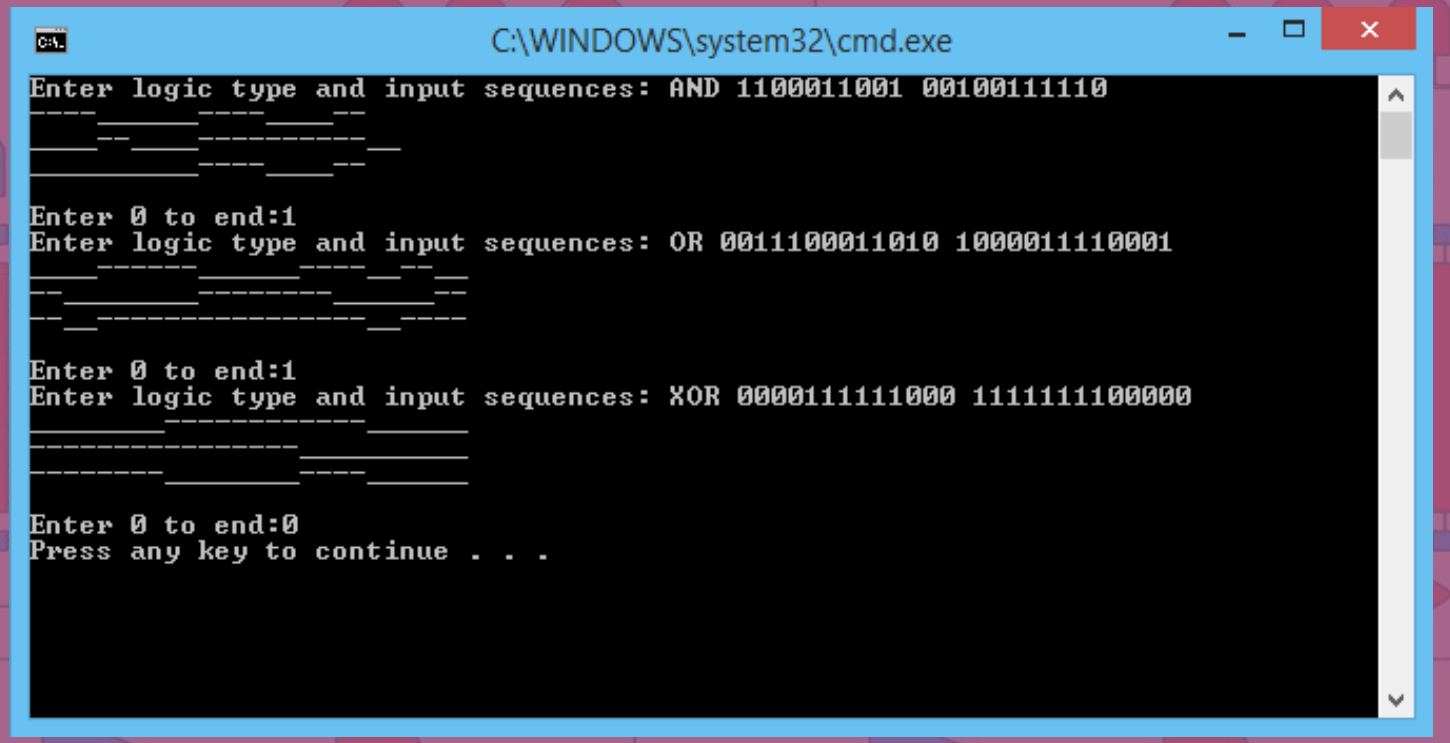
Output the waveform for input sequence

MIN macro calculating output waveform length

Apply a certain logic operation

Output the waveform for output sequence

Types and Operators for Logic Modeling



Logic Simulation with C/C++

- Procedural Languages for Hardware Modeling
- Types and Operators for Logic Modeling
- Basic Logic Simulation
  - Logic functions
  - Function overloading
  - Passing logic functions
  - Using default values
  - Building higher level structures
  - Handling 4-value logic
  - Logic vector
  - Sequential circuit modeling
  - Using pointers for logic vectors
- Enhanced logic simulation with timing
  - Using struct for timing and logic
  - Gates that handle timing
  - Utility functions
  - Timing in logic structures
  - Overloading logical operators
  - Using Boolean expressions
- More Functions for Wires and Gates
  - Gate classes
  - Carrier centric modeling
  - Compatible scalar and vector

Basic Logic Simulation

logicGates.h

Logic Simulation | (Global Scope)

```
1  #include <iostream>
2  using namespace std;
```

100 %

LogicGates.h

primitives.h

Logic Simulation | (Global Scope)

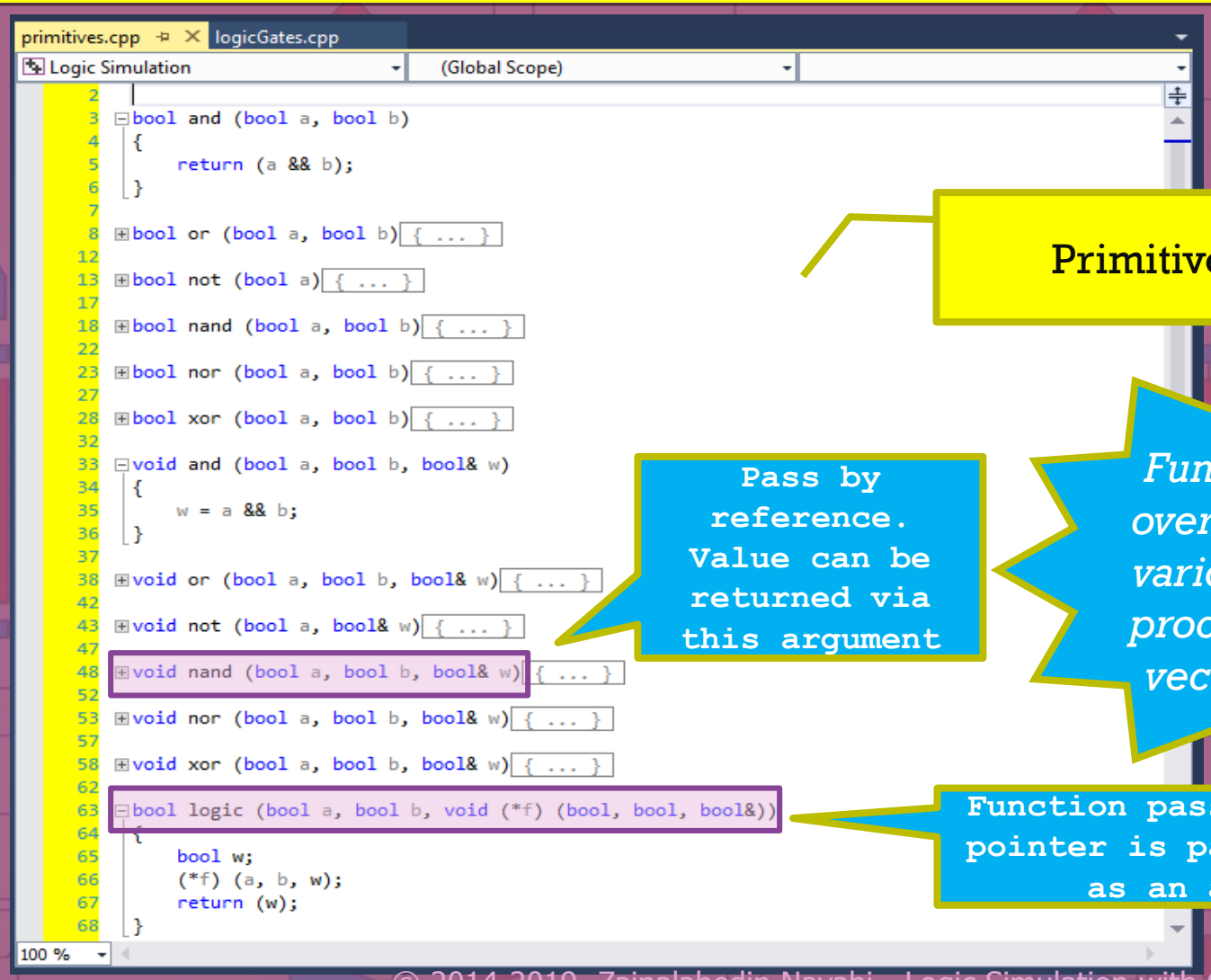
```
1  bool and (bool a, bool b);
2  bool or (bool a, bool b);
3  bool not (bool a);
4  bool nand (bool a, bool b);
5  bool nor (bool a, bool b);
6  bool xor (bool a, bool b);
7
8  void and (bool a, bool b, bool& w);
9  void or (bool a, bool b, bool& w);
10 void not (bool a, bool& w);
11 void nand (bool a, bool b, bool& w);
12 void nor (bool a, bool b, bool& w);
13 void xor (bool a, bool b, bool& w);
14
15 bool logic (bool a, bool b, void (*f) (bool, bool, bool&));
16
17 bool and5 (bool a=true, bool b=true, // up to 5 inputs
18            bool c=true, bool d=true, bool e=true);
19 bool or5  (bool a=false, bool b=false,
20            bool c=false, bool d=false, bool e=false);
21 bool xor5 (bool a=false, bool b=false,
22            bool c=false, bool d=false, bool e=false);
23
24 void and (bool[], bool[], bool[], const int);
25 void or (bool[], bool[], bool[], const int);
26
27
```

100 %

Primitives.h

Gate function prototypes

Logic Functions



Primitives.cpp

Pass by reference. Value can be returned via this argument

Functions are overloaded for various type of procedure and vector format

Function passing. Function pointer is passed to logic as an argument

Using Default Values

logicGates.h

Logic Simulation | (Global Scope)

```
1   #include <iostream>
2   using namespace std;
```

LogicGates.h

primitives.h

Logic Simulation | (Global Scope)

```
1   bool and (bool a, bool b);
2   bool or (bool a, bool b);
3   bool not (bool a);
4   bool nand (bool a, bool b);
5   bool nor (bool a, bool b);
6   bool xor (bool a, bool b);
7
8   void and (bool a, bool b, bool& w);
9   void or (bool a, bool b, bool& w);
10  void not (bool a, bool& w);
11  void nand (bool a, bool b, bool& w);
12  void nor (bool a, bool b, bool& w);
13  void xor (bool a, bool b, bool& w);
14
15  bool logic (bool a, bool b, void (*f) (bool, bool, bool&));
16
17  bool and5 (bool a=true, bool b=true, // up to 5 inputs
18             bool c=true, bool d=true, bool e=true);
19  bool or5  (bool a=false, bool b=false,
20             bool c=false, bool d=false, bool e=false);
21  bool xor5 (bool a=false, bool b=false,
22             bool c=false, bool d=false, bool e=false);
23
24  void and (bool[], bool[], bool[], const int);
25  void or (bool[], bool[], bool[], const int);
26
27
```

Primitives.h

To use this function with fewer arguments, all arguments must have default values

Building Higher Level Structures

primitives.cpp | logicGates.cpp*

Logic Simulation | (Global Scope)

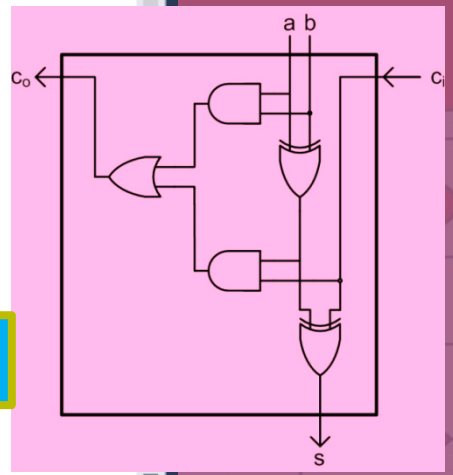
```
1   #include "logicGates.h"
2   #include "primitives.h"
3
4   /* ... */
16
17  /* ... */
29
30  void fullAdder (bool a, bool b, bool ci, bool& co, bool& sum)
31  {
32      bool axb, ab, abc;
33
34      axb = logic (a, b, xor); // uses: void xor (bool, bool, bool&);
35      ab  = logic (a, b, and);
36      abc = logic (axb, ci, and);
37      co  = logic (ab, abc, or);
38      sum = logic (axb, ci, xor);
39  }
40
41  void fullAdder (bool a, bool b, bool ci, bool& co, bool& sum)
42  {
43      bool ab, bc, ac;
44
45      ab = and5 (a, b);
46      bc = and5 (b, ci);
47      ac = and5 (a, ci);
48      co = or5 (ab, bc, ac);
49      sum = xor5 (a, b, ci);
50  }
51
52  int main () { ... }
66
67
```

100 %

logicgates.cpp

Use logic function and pass specific function

Full-adder implementation



Building Higher Level Structures

primitives.cpp | logicGates.cpp*

Logic Simulation | (Global Scope)

```
1   #include "logicGates.h"
2   #include "primitives.h"
3
4   void fullAdder (bool a, bool b, bool ci, bool& co, bool& sum)
5   {
6       bool axb, ab, abc;
7
8       axb = xor (a, b);
9       ab  = and (a, b);
10      abc = and (axb, ci);
11      co = or (ab, abc);
12      sum = xor (axb, ci);
13  }
14
15  /* ... */
27
28  /* ... */
40
41  /* ... */
53
54  int main ()
55  {
56      bool a, b, c, co, sum;
57
58      do {
59          cout << "Enter a, b, c: "; cin >> a >> b >> c;
60
61          fullAdder (a, b, c, co, sum);
62
63          cout << "Carry:" << co << " Sum:" << sum << "\n";
64
65          cout << "\n" << "Continue?"; cin >> a;
66      } while (a != false);
67  }
68
69
```

100 %

logicgates.cpp

Calling full-adder

4-value Logic

Value	Description
0	Forcing 0 or Pulled 0
1	Forcing 1 or Pulled 1
Z	Float or High Impedance
X	Uninitialized or Unknown

- **Four-Value Logic System**

Handling 4-value Logic

characterFunctions.h

Character Logic | (Global Scope)

```
1  #include <iostream>
2  using namespace std;
3
```

100 %

characterPrimitives.h

Character Logic | (Global Scope) | and(char a, char b)

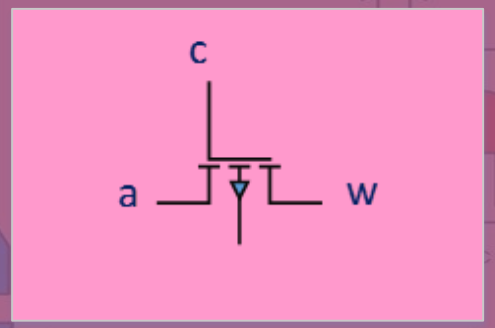
```
1  char and (char a, char b);
2  char or (char a, char b);
3  char not (char a);
4  char tri (char a, char c);
5  char resolve (char a, char c);
6  char xor (char a, char b);
7
8  void fullAdder (char a, char b, char ci, char & co, char & sum);
9
```

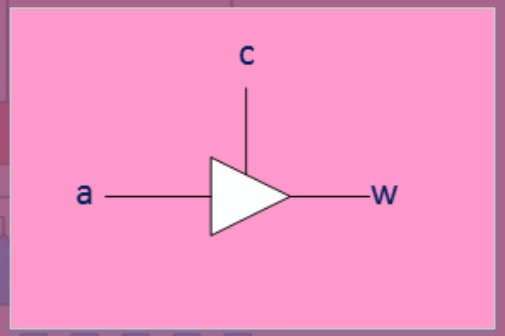
CharacterFunctions.h

CharacterPrimitives.h

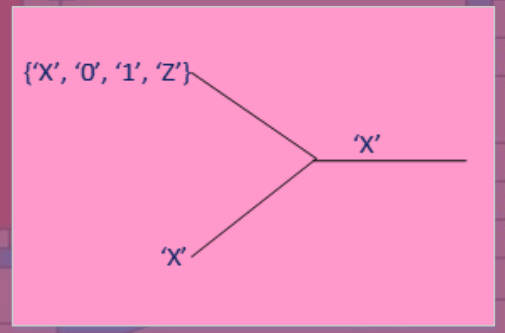
Using char for easier and more expressive in and out instead of directly input logic value

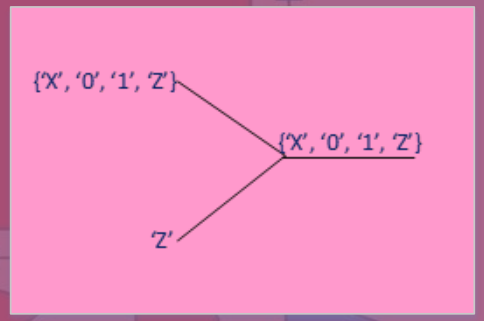
Handling 4-value Logic





Tri-state





And resolution function

	x	0	1	z
x	x	x	x	x
0	x	0	x	0
1	x	x	1	1
z	x	0	1	z

Handling 4-value Logic

characterPrimitives.cpp | characterFunctions.cpp

Character Logic | (Global Scope)

```
1   #include "characterPrimitives.h"
2
3   char and (char a, char b)
4   {
5       if ((a=='0')||(b=='0')) return '0';
6       else if ((a=='1')&&(b=='1')) return '1';
7       else return 'X';
8   }
9
10  char or (char a, char b) { ... }
16
17  char not (char a) { ... }
23
24  char tri (char a, char c) { ... }
29
30  char resolve (char a, char b)
31  {
32      if (a=='Z' || a==b) return b;
33      else if (b=='Z') return a;
34      else return 'X';
35  }
36
37  char xor (char a, char b) { ... }
43
44  void fullAdder (char a, char b, char ci, char & co, char & sum)
45  {
46      char axb, ab, abc;
47
48      axb = xor (a, b);
49      ab  = and (a, b);
50      abc = and (axb, ci);
51      co = or (ab, abc);
52      sum = xor (axb, ci);
53  }
54
55
```

100 %

CharacterPrimitives.cpp

But the drawback is that we have to generate our own logical functions. This happens one and can easily be reused.

Handling 4-value Logic

characterPrimitives.cpp | characterFunctions.cpp

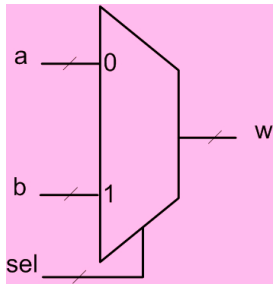
Character Logic | (Global Scope)

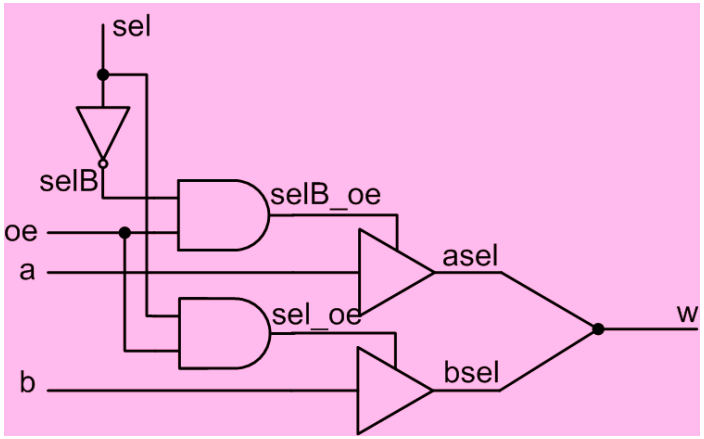
```
1  #include "characterPrimitives.h"
2  #include "characterFunctions.h"
3
4  void muxStd2TO1 (char a, char b, char& w, char sel)
5  {
6      w = (sel=='1') ? b : a;
7  }
8
9  void muxTri2TO1 (char a, char b, char& w, char sel, char oe)
10 {
11     char selB, selB_oe, sel_oe;
12     char asel;
13     char bsel;
14
15     selB = not(sel);
16     selB_oe = and(selB, oe);
17     sel_oe = and(sel, oe);
18     asel = tri(a, selB_oe);
19     bsel = tri(b, sel_oe);
20     w = resolve(asel, bsel);
21 }
22
23 int main (){ ... }
43
44
```

multiplexer

Multiplexer with OE

CharacterFunctions.cpp





Logic Vector

vectorFunctions.h

Logic Vector Simulation | (Global Scope)

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
```

100 %

vectorPrimitives.h

Logic Vector Simulation | (Global Scope) | and(bool a, bool b)

```
1  bool and (bool a, bool b);
2  bool or (bool a, bool b);
3  bool not (bool a);
4
5  void and (bool a[], bool b[], bool w[], const int SIZE);
6  void or (bool a[], bool b[], bool w[], const int SIZE);
7
```

vectorFunctions.cpp

VectorFunctions.h

VectorPrimitives.h

Arrays are passed by reference to first location

Logic Vector

vectorPrimitives.cpp | vectorFunctions.cpp

Logic Vector Simulation | (Global Scope)

```
1   #include "vectorPrimitives.h"
2
3   bool and (bool a, bool b)
4   {
5       return (a && b);
6   }
7
8   bool or (bool a, bool b) { ... }
12
13  bool not (bool a) { ... }
17
18  void and (bool a[], bool b[], bool w[], const int SIZE)
19  {
20      int i;
21      for (i=0; i<SIZE; i++) {
22          w[i] = a[i] && b[i];
23      }
24  }
25
26  void or (bool a[], bool b[], bool w[], const int SIZE) { ... }
33
34
```

100 %

VectorPrimitives.cpp

Loop and index need size

Logic vectors overloaded basic functions

Logic Vector

vectorPrimitives.cpp | vectorFunctions.cpp

Logic Vector Simulation | (Global Scope)

```
1   #include "vectorPrimitives.h"
2   #include "vectorFunctions.h"
3
4   void getBits (string vectorName, int numBits, bool values[])
5   {
6       string valuesS;
7       int i;
8       cout << "Enter " << numBits << " bits of " << vectorName << ": ";
9       cin >> valuesS;
10      for (i=0; i<numBits; i++){
11          if (valuesS[i] == '1') values[i] = true;
12          else values[i] = false;
13      }
14  }
15
16  void putBits (string vectorName, int numBits, bool values[]) { ... }
27  void two2OneMux (bool a[], bool b[], bool w[], bool sel, int SIZE=8)
28  {
29      bool as [8];
30      bool bs [8];
31
32      int i;
33      for (i=0; i<SIZE; i++) {
34          as[i] = and (a[i], not(sel));
35      }
36      for (i=0; i<SIZE; i++) {
37          bs[i] = and (b[i], sel);
38      }
39
40      or (as, bs, w, SIZE);
41  }
42
43  void two2OneMuxB (bool a[], bool b[], bool w[], bool sel, int SIZE=8) { ... }
50
51  int main () { ... }
82
```

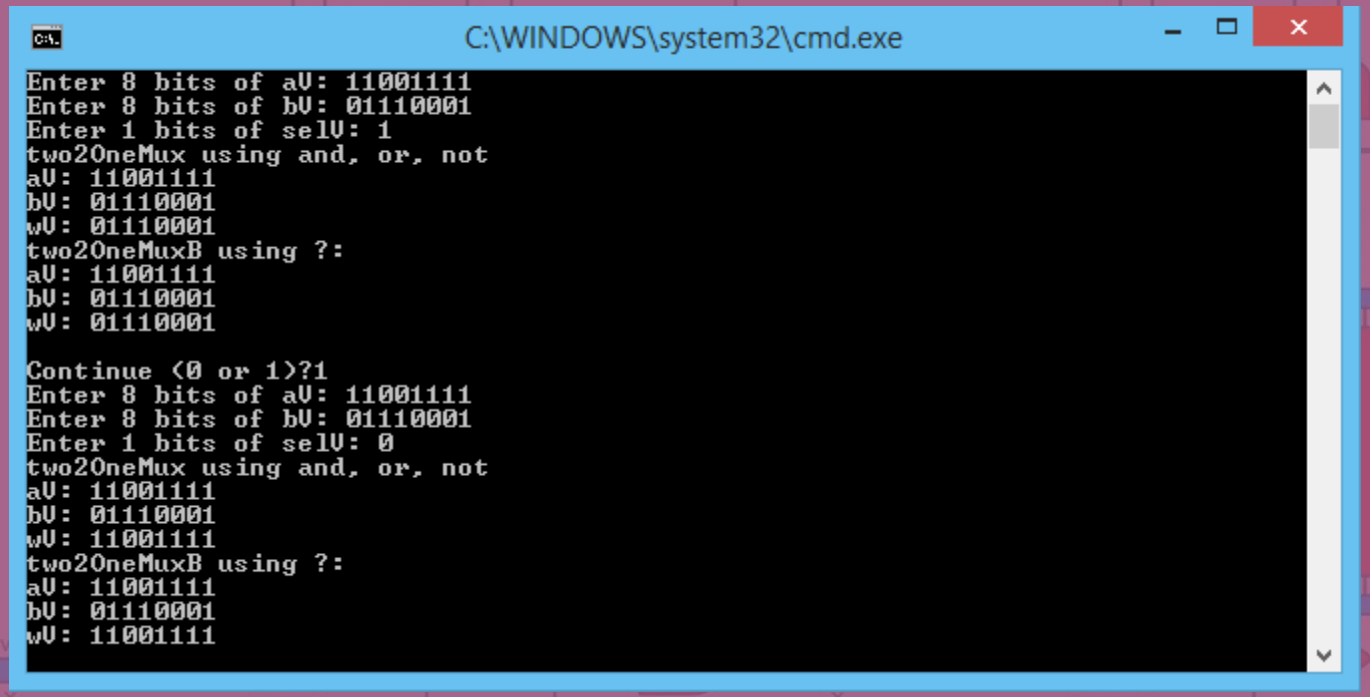
100 %

VectorFunctions.cpp

Read string and turns it into an array of bool

Array indexing

Logic Vector



Logic Vector

characterVectorFunctions.cpp | characterVectorFunctions.h | characterVectorPrimitives.h

Character Vector Logic | (Global Scope)

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  char and (char a, char b);
6  char or (char a, char b);
7  char not (char a);
8  char tri (char a, char c);
9  char resolve (char a, char c);
10
11 void and (char a[], char b[], char w[]);
12 void or (char a[], char b[], char w[]);
13 void tri (char a[], char c, char w[]);
14 void resolve (char a[], char b[], char w[]);
15
16 char xor (char a, char b);
17 void fullAdder (char a, char b, char ci, char & co, char & sum);
18
```

100 %

characterVectorPrimitives.h

Shows arrays of characters

Char-based primitives and their vector overloading

Logic Vector

characterVectorFunctions.cpp | characterVectorPrimitives.h | characterVectorPrimitives.cpp

Character Vector Logic | (Global Scope)

```
38  void and (char a[], char b[], char w[])
39  {
40      int i=0;
41      while (a[i] != '\0') {
42          w[i] = and (a[i], b[i]);
43          i++;
44      };
45      w[i] = '\0';
46  }
47
48  void or (char a[], char b[], char w[]) { ... }
57
58  void tri (char a[], char c, char w[]) { ... }
67
68  void resolve (char a[], char b[], char w[])
69  {
70      int i=0;
71      while (a[i] != '\0') {
72          w[i] = resolve (a[i], b[i]);
73          i++;
74      };
75      w[i] = '\0';
76  }
77
78  char xor (char a, char b) { ... }
84
85  void fullAdder (char a, char b, char ci, char & co, char & sum)
86  {
87      char axb, ab, abc;
88
89      axb = xor (a, b);
90      ab  = and (a, b);
91      abc = and (axb, ci);
92      co = or (ab, abc);
93      sum = xor (axb, ci);
94  }
95
```

100 %

Null character marks the end of the vector

characterVectorPrimitives.cpp

Logic Vector

characterVectorFunctions.cpp | characterVectorPrimitives.h | characterVectorPrimitives.cpp

Character Vector Logic | (Global Scope)

```
1  #include "characterVectorPrimitives.h"
2  #include "characterVectorFunctions.h"
3
4  void mux8Std2TO1 (char a[], char b[], char w[], char sel)
5  {
6      int i=0;
7      do {
8          w[i] = (sel=='1') ? b[i] : a[i];
9      } while (a[i++] != '\0');
10 }
11
12 void mux8Tri2TO1 (char a[], char b[], char w[], char sel, char oe)
13 {
14     char selB, selB_oe, sel_oe;
15     char asel [9];
16     char bsel [9];
17
18     selB = not(sel);
19     selB_oe = and(selB, oe);
20     sel_oe = and(sel, oe);
21     tri(a, selB_oe, asel);
22     tri(b, sel_oe, bsel);
23     resolve(asel, bsel, w);
24 }
25
26 int main ()
27 {
28     char aCV [9], bCV [9];
29     char sel, oe;
30     char wCV [9];
31     int ai;
32     do {
33         cout << "Enter eight bits of aCV <space> bCV: "; cin >> aCV >> bCV;
34         cout << "Enter sel <space> oe: "; cin >> sel >> oe;
35
36         mux8Std2TO1 (aCV, bCV, wCV, sel);
37         cout << "The " << strlen(wCV) << " bits of wC become as follows: \n";
```

100 %

characterVectorFunctions.cpp

8-Bit character vector based mux

If fewer than 8-bits are entered, using cin automatically puts '\0' at the end of string

Sequential Circuit Modeling

sequentialFunctions.h | characterPrimitives.cpp | sequentialFunctions.cpp

Sequential Model | (Global Scope)

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std;
5
```

100 %

characterPrimitives.h

Sequential Model | (Global Scope)

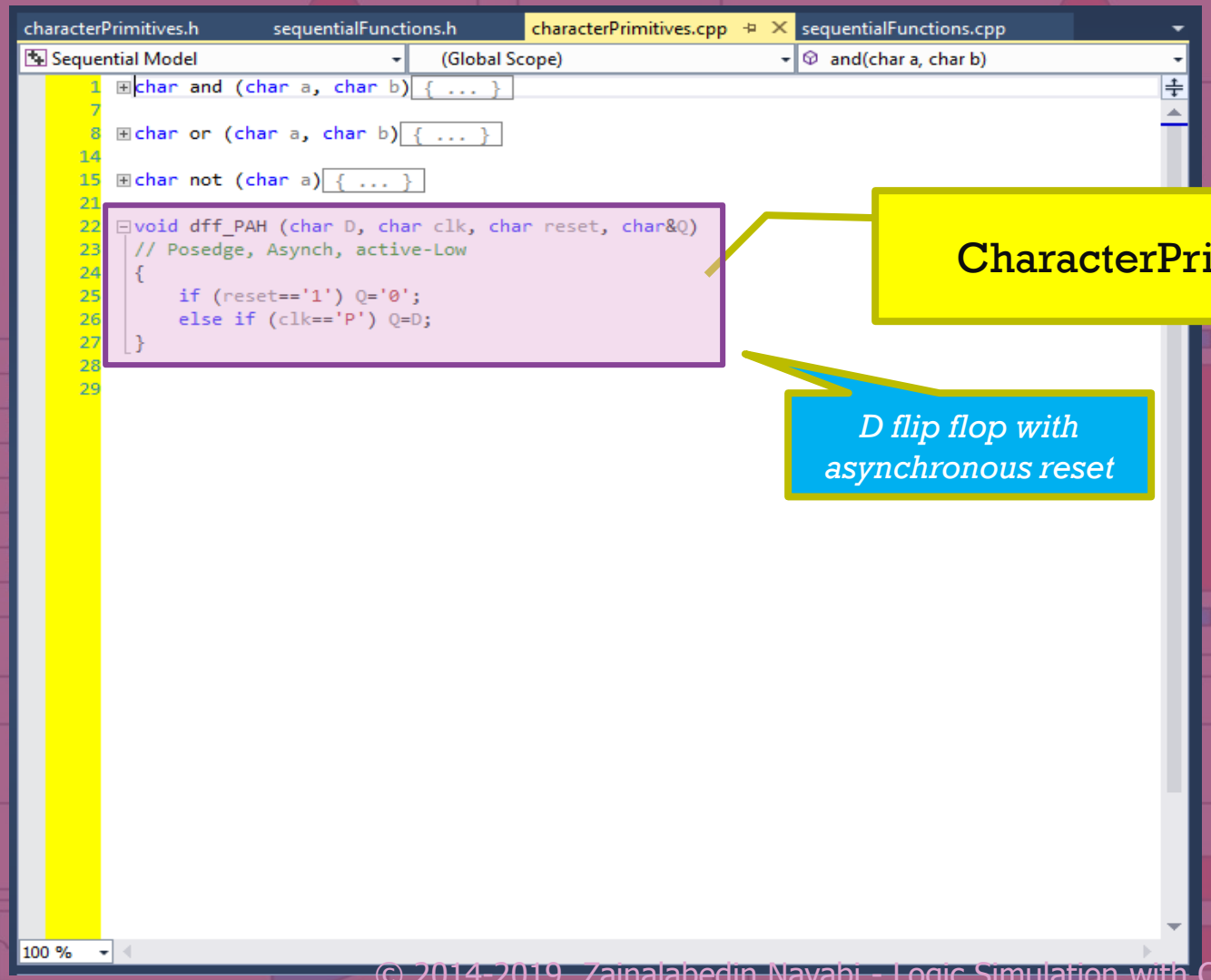
```
1  char and (char a, char b);
2  char or (char a, char b);
3  char not (char a);
4  void dff_PAH (char D, char clk, char reset, char&Q);
5
```

SequentialFunctions.h

CharacterPrimitives.h

DFFaLRhE =
D filp flop
active low
reset active
high enable

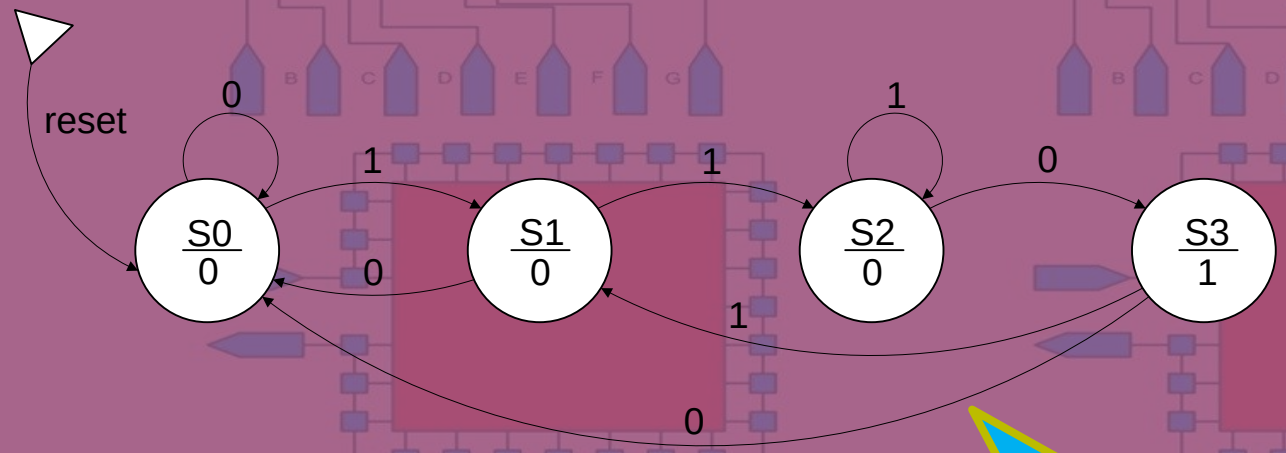
Sequential Circuit Modeling



CharacterPrimitives.cpp

D flip flop with asynchronous reset

Sequential Circuit Modeling



Moore 110 sequence detector

Sequential Circuit Modeling

characterPrimitives.h | sequentialFunctions.h | characterPrimitives.cpp | sequentialFunctions.cpp

Sequential Model | (Global Scope)

```
1  #include "characterPrimitives.h"
2  #include "sequentialFunctions.h"
3
4  int main ()
5  {
6      string inVec;
7      string outVec = ",,,,";
8      char ain('0'), reset, clock;
9
10     char Y1('X'), Y0('X'), D1, D0, w;
11
12     ifstream finp ( "indata.tst");
13     ofstream fout ("outdata.tst");
14
15     fout << "Inp -> Output, Next state\n";
16
17     do {
18         finp >> inVec;
19             ain = inVec[0];
20             reset = inVec[1];
21             clock = inVec[2];
22         // combinational parts in proceduaral fashion
23         // followed by the sequential parts
24
25         D1 = or(and(Y1, Y0), and (ain, Y0));
26         D0 = ain;
27         w = and(Y1, not(Y0));
28
29         outVec[0] = w;  // These values are after
30         outVec[2] = Y1; //   application of the
31         outVec[3] = Y0; //   previous inputs
32         fout << outVec+"\n" << inVec << " -> ";
33
34         dff_PAH (D1, clock, reset, Y1);
35         dff_PAH (D0, clock, reset, Y0);
36     } while (!finp.eof());
37 }
```

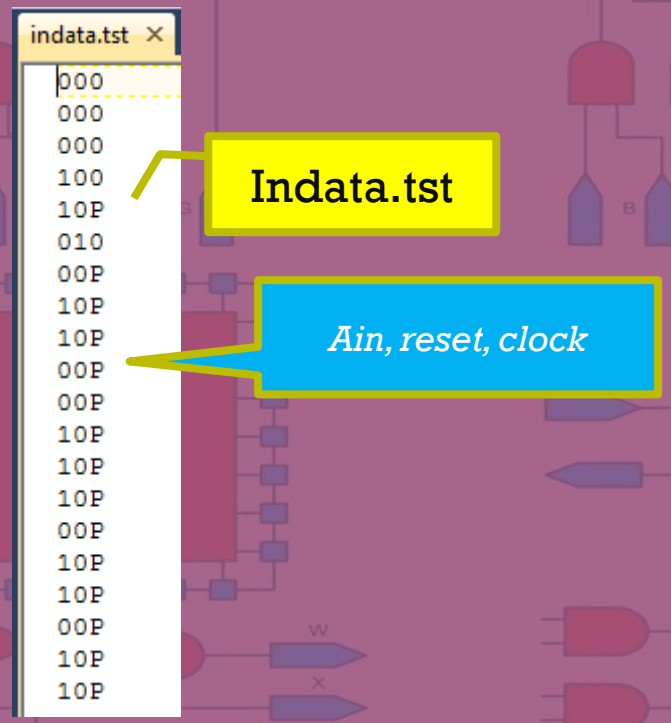
100 %

SequentialFunctions.cpp

File handling

Convert string to char. Operations in char

Sequential Circuit Modeling



Sequential Circuit Modeling

indata.tst

```
000
000
000
100
10P
010
00P
10P
10P
00P
00P
10P
10P
10P
00P
10P
10P
00P
10P
10P
```

outdata.tst

```
Inp -> Output, Next state
X,XX
000 -> X,XX
000 -> X,XX
000 -> X,XX
100 -> X,XX
10P -> 0,X1
010 -> 0,00
00P -> 0,00
10P -> 0,01
10P -> 0,11
00P -> 1,10
00P -> 0,00
10P -> 0,01
10P -> 0,11
10P -> 0,11
00P -> 1,10
10P -> 0,01
10P -> 0,11
00P -> 1,10
10P -> 0,01
10P -> 0,11
10P ->
```

outdata.tst

W and 2 bits of states

Clock by clock output

Using Pointers for Logic Vectors

pointerFunctionsFileData.h | pointerPrimitives.h | pointerPrimitives.cpp | pointerFunctionsFileData.cpp

Pointer Logic File Data | (Global Scope)

```
1   void and (char a, char b, char & w);
2   void or (char a, char b, char & w);
3   void not (char a, char & w);
4   void tri (char a, char c, char & w);
5   void resolve (char a, char c, char & w);
6
7   void and (char* a, char* b, char* w);
8   void or (char *a, char *b, char *w);
9   void not (char *a, char *w);
10  void tri (char *a, char *c, char *w);
11  void resolve (char *a, char *b, char *w);
12
13  void mux8Std2TO1 (char*, char*, char*, char);
14  void mux8Tri2TO1 (char*, char*, char*, char, char);
15
```

pointerPrimitives.h

Pointers instead of arrays

Using Pointers for Logic Vectors

pointerFunctionsFileData.h | pointerPrimitives.h | pointerPrimitives.cpp | pointerFunctionsFileData.cpp

Pointer Logic File Data | (Global Scope)

```
1   #include <iostream>
2   using namespace std;
3
4   void and (char a, char b, char & w)
5   {
6       w = ((a=='0')||(b=='0')) ? '0':
7           ((a=='1')&&(b=='1')) ? '1':
8                                  'X';
9   }
10
11  void or (char a, char b, char  & w) { ... }
17
18  void not (char a, char & w) { ... }
24
25  void tri (char a, char c, char & w) { ... }
29
30  void resolve (char a, char b, char & w) { ... }
36
37  void and (char* a, char* b, char* w)
38  {
39      int i=0;
40      do {
41          and (*(a+i), *(b+i), *(w+i));
42          i++;
43      } while (*(a+i) != '\0');
44      *(w+i) = '\0';
45  }
46
47  void or (char *a, char *b, char *w) { ... }
56
57  void not (char *a, char *w) { ... }
66
67  void tri (char *a, char *c, char *w) { ... }
76
77  void resolve (char *a, char *b, char *w) { ... }
86
87  void mux8Std2T01 (char *a, char *b, char *w, char sel)
```

100 %

Overloading AND

pointerPrimitives.cpp

And with Pointer Arguments

Using Pointers for Logic Vectors

pointerFunctionsFileData.h | pointerPrimitives.h | pointerPrimitives.cpp* | pointerFunctionsFileData.cpp

Pointer Logic File Data | (Global Scope) | resolve(char * a, char * b, char * w)

```
30  void resolve (char a, char b, char & w) { ... }
36
37  void and (char* a, char* b, char* w)
38  {
39      int i=0;
40      do {
41          and (*(a+i), *(b+i), *(w+i));
42          i++;
43      } while (*(a+i) != '\0');
44      *(w+i) = '\0';
45  }
46
47  void or (char *a, char *b, char *w) { ... }
56  void not (char *a, char *w) { ... }
65  void tri (char *a, char *c, char *w) { ... }
74  void resolve (char *a, char *b, char *w) { ... }
83
84  void mux8Std2TO1 (char *a, char *b, char *w, char sel)
85  {
86      int i=0;
87      do {
88          *(w+i) = (sel=='1') ? *(b+i) : *(a+i);
89          i++;
90      } while (*(a+i) != '\0');
91      *(w+i) = '\0';
92  }
93
94  void mux8Tri2TO1 (char *a, char *b, char *w, char sel, char oe)
95  {
96      int i=0;
97      do {
98          if (oe == '1') *(w+i) = (sel=='1') ? *(b+i) : *(a+i);
99          else *(w+i) = 'Z';
100         i++;
101     } while (*(a+i) != '\0');
102     *(w+i) = '\0';
103 }
```

100 %

pointerPrimitives.cpp

Pointer referencing in a multi bit multiplexer

Using Pointers for Logic Vectors

pointerFunctionsFileData.h | pointerPrimitives.h | pointerPrimitives.cpp* | pointerFunctionsFileData.cpp

Pointer Logic File Data | (Global Scope)

```
1  #include "pointerPrimitives.h"
2  #include "pointerFunctionsFileData.h"
3
4  int main ()
5  {
6      ifstream inp ("inpdata.tst"); //declare and initialize inp
7      ofstream out ("outdata.tst"); //declare and initialize out
8
9      int ii;
10     inp >> ii;
11     out << "All vector lengths are " << ii << " bits.\n";
12
13     char sel, oe;
14     char* aC = new char [ii+1];
15     char* bC = new char [ii+1];
16     char* wC = new char [ii+1];
17
18     while (inp >> aC >> bC >> sel >> oe)
19     {
20         out << "Inputs are a, b vectors and sel, oe bits: ";
21         out << aC << " " << bC << " " << sel << " " << oe << "\n";
22
23         mux8Std2TO1 (aC, bC, wC, sel);
24         out << "Std Mux: " << wC << '\n';
25
26         mux8Tri2TO1 (aC, bC, wC, sel, oe);
27         out << "Tri Mux: " << wC << '\n';
28     }
29 }
30
```

100 %

pointerFunctionsFileData.cpp

Testing Multiplexers using ifstream and ofstream

Using Pointers for Logic Vectors

outdata.tst

```
All vector lengths are 8 bits.
Inputs are a, b vectors and sel, oe bits: 11001111 11110001 0 0
Std Mux: 11001111
Tri Mux: ZZZZZZZZ
Inputs are a, b vectors and sel, oe bits: 11110001 00010101 0 1
Std Mux: 11110001
Tri Mux: 11110001
Inputs are a, b vectors and sel, oe bits: 10101011 11110000 1 0
Std Mux: 11110000
Tri Mux: ZZZZZZZZ
Inputs are a, b vectors and sel, oe bits: 11001111 11001100 1 1
Std Mux: 11001100
Tri Mux: 11001100
Inputs are a, b vectors and sel, oe bits: 11110000 11101010 1 1
Std Mux: 11101010
Tri Mux: 11101010
Inputs are a, b vectors and sel, oe bits: 00111110 00110011 0 0
Std Mux: 00111110
Tri Mux: ZZZZZZZZ
Inputs are a, b vectors and sel, oe bits: 01110001 00101001 1 0
Std Mux: 00101001
Tri Mux: ZZZZZZZZ
Inputs are a, b vectors and sel, oe bits: 00001110 01010101 0 1
Std Mux: 00001110
Tri Mux: 00001110
```

Outdata.tst

Logic Simulation with C/C++

- Procedural Languages for Hardware Modeling
- Types and Operators for Logic Modeling
- Basic Logic Simulation
  - Logic functions
  - Function overloading
  - Passing logic functions
  - Using default values
  - Building higher level structures
  - Handling 4-value logic
  - Logic vector
  - Sequential circuit modeling
  - Using pointers for logic vectors
- Enhanced logic simulation with timing
  - Using struct for timing and logic
  - Gates that handle timing
  - Utility functions
  - Timing in logic structures
  - Overloading logical operators
  - Using Boolean expressions
- More Functions for Wires and Gates
  - Gate classes
  - Carrier centric modeling
  - Compatible scalar and vector

Using Struct for Logic and Timing

timedFunctions.cpp | timedPrimitives.h | timedPrimitives.cpp | timedFunctions.h

Timed Logic Structs | (Global Scope)

```
1  struct tlogic {
2      char logic;
3      int time;
4  };
5
6  tlogic and (tlogic a, tlogic b, int delay);
7  tlogic or (tlogic a, tlogic b, int delay);
8  tlogic not (tlogic a, int delay);
9  tlogic xor (tlogic a, tlogic b, int delay);
10
```

timedPrimitives.h

Structure to accommodate time as well as logic

Gates that Handle Timing

timedFunctions.cpp | timedPrimitives.h | timedPrimitives.cpp | timedFunctions.h

Timed Logic Structs | (Global Scope) | not(tlogic a, int delay)

```
1   #include "timedPrimitives.h"
2
3   tlogic and (tlogic a, tlogic b, int delay)
4   {
5       tlogic tl;
6       if ((a.logic=='0')||(b.logic=='0')) {
7           tl.logic = '0';
8           if (a.logic=='0') tl.time = a.time + delay;
9           else tl.time = b.time + delay;
10      }
11      else if ((a.logic=='1')&&(b.logic=='1')) {
12          tl.logic = '1';
13          if (a.time > b.time) tl.time = a.time + delay;
14          else tl.time = b.time + delay;
15      }
16      else {
17          tl.logic = 'X';
18          if (a.logic != '1') tl.time = a.time + delay;
19          else tl.time = b.time + delay;
20      };
21      return tl;
22  }
23
24  tlogic or (tlogic a, tlogic b, int delay) { ... }
44
45  tlogic not (tlogic a, int  delay) { ... }
54
55  tlogic xor (tlogic a, tlogic b, int delay)
56  {
57      tlogic tl;
58      if (a.logic==b.logic) tl.logic = '0';
59      else tl.logic = '1';
60      if (a.time > b.time) tl.time = a.time + delay;
61      else tl.time = b.time + delay;
62      return tl;
63  }
64
```

100 %

timedPrimitives.cpp

And logic function with timing

A more accurate delay propagation requires the gate function to be aware of its previous output value

Utility Functions

timedFunctions.cpp | timedPrimitives.h | timedPrimitives.cpp | timedFunctions.h

Timed Logic Structs | (Global Scope) | main()

```
1  #include "timedPrimitives.h"
2  #include "timedFunctions.h"
3
4  #define MAX(a,b)a>b?a:b;
5  #define MIN(a,b)a<b?a:b;
6
7  void getVect (string vectorName, int numBits, tlogic values[])
8  { //order according to bit significance
9      string valuesS;
10     int i, bits, delay;
11     cout << "Enter " << numBits << " bits of " << vectorName << ": ";
12     cin >> valuesS;
13     bits = MIN (valuesS.length(), numBits); // if fewer are entered
14     cout << "Enter vector delay: "; cin >> delay;
15     for (i=bits-1; i>=0; i--) {
16         values[i].logic = char(valuesS[bits-1-i]); // reverse bits
17 //      values[i].time = delay; // This or two below are good
18 //      (*(values+i)).time = delay;
19         (values+i)->time = delay;
20     }
21 }
22
23 void putVect (string vectorName, int numBits, tlogic values[])
24 {
25     int i, delay;
26     delay = 0;
27     cout << vectorName << ": ";
28     for (i=numBits-1; i>=0; i--) {
29         cout << values[i].logic;
30         if (values[i].time > delay) delay=values[i].time;
31     }
32     cout << " AT " << delay << "\n";
33 }
34
35 void fullAdder (tlogic a, tlogic b, tlogic ci, tlogic& co, tlogic& sum) { ... }
45
46 void nBitAdder (tlogic a[], tlogic b[], tlogic ci[], tlogic co[], tlogic sum[], int
```

100 %

timedFunctions.cpp

Entered: 1011
ValuesS: 1011
Values: 1101

Pointer referencing

This method starts from bit 0 and treat bit 0 as logical LSB value

Timing in Logic Structures

timedFunctions.cpp | timedPrimitives.h | timedPrimitives.cpp | timedFunctions.h

Timed Logic Structs | (Global Scope)

```
35  void fullAdder (tlogic a, tlogic b, tlogic ci, tlogic& co, tlogic& sum)
36  {
37      tlogic axb, ab, abc;
38
39      axb = xor (a, b, 5);
40      ab  = and (a, b, 3);
41      abc = and (axb, ci, 3);
42      co = or (ab, abc, 4);
43      sum = xor (axb, ci, 5);
44  }
45
46  void nBitAdder (tlogic a[], tlogic b[], tlogic ci[], tlogic co[], tlogic sum[], int bits)
47  {
48      // assumes 0 is LSB
49      int i;
50      tlogic* c = new tlogic[bits+1];
51      c[0] = ci[0];
52      for (i = 0; i<bits; i++)
53      {
54          fullAdder(a[i], b[i], c[i], c[i+1], sum[i]);
55      }
56      co[0] = c[bits];
57  }
58
59  int main () { ... }
94
```

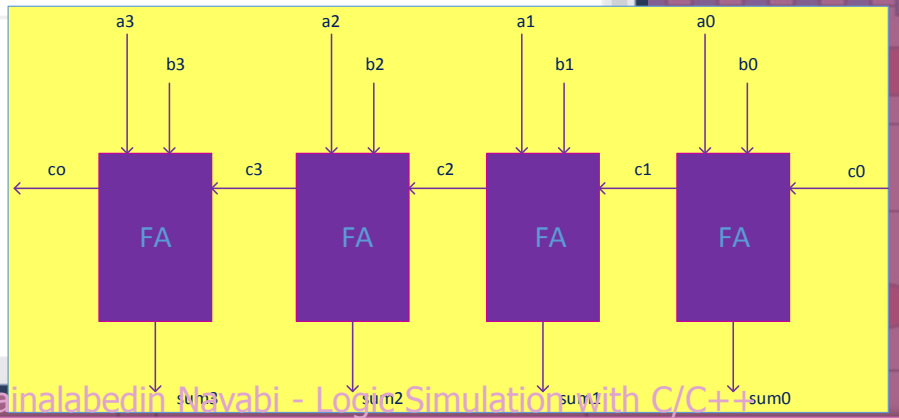
100 %

timedFunctions.cpp

Full adder using timed logic

nBitAdder function using timed logic

4 bit adder made of four FAs



Timing in Logic Structures

timedFunctions.cpp | timedPrimitives.h | timedPrimitives.cpp | timedFunctions.h

Timed Logic Structs | (Global Scope)

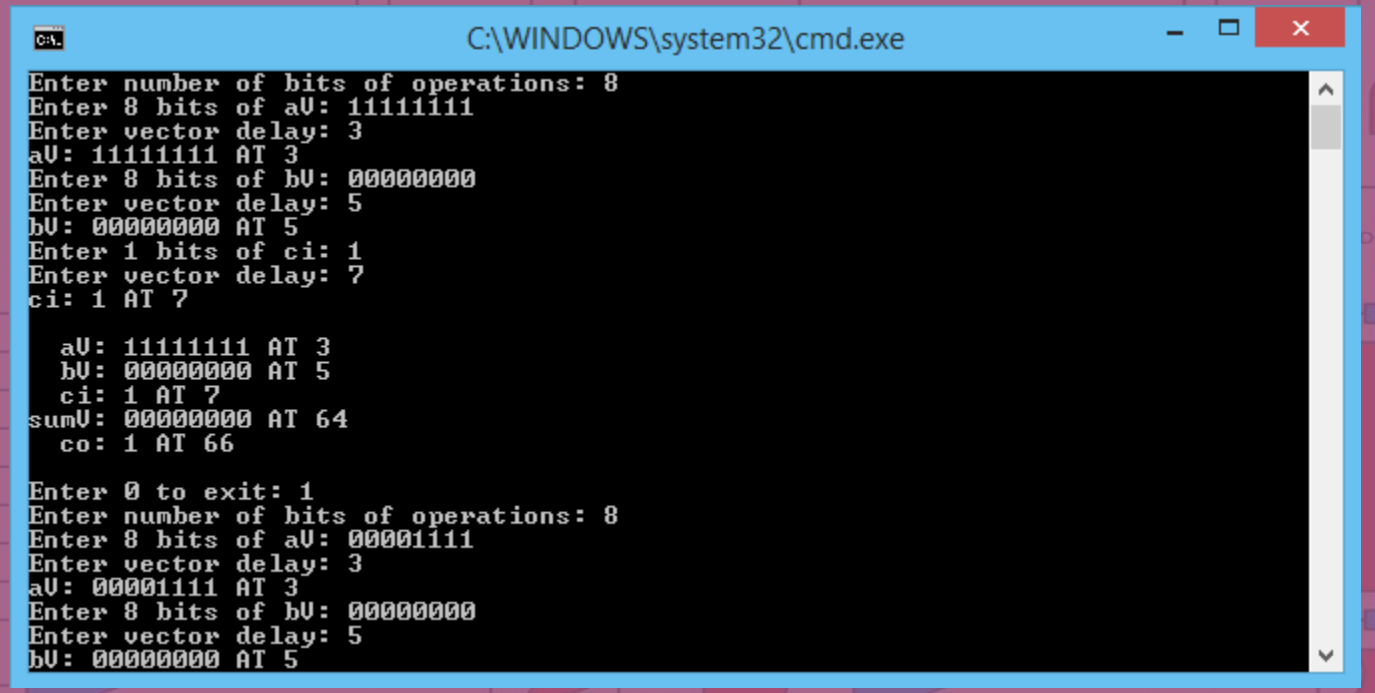
```
58
59  int main ()
60  {
61      tlogic *aV, *bV, *ci, *co, *sumV;
62
63      int bits, go(1);
64
65      while (go)
66      {
67          cout << "Enter number of bits of operations: "; cin >> bits;
68          aV = new tlogic[bits];
69          bV = new tlogic[bits];
70          ci = new tlogic[1];
71          co = new tlogic[1];
72          sumV = new tlogic[bits];
73
74          getVect ("aV", bits, aV); putVect ("aV", bits, aV);
75          getVect ("bV", bits, bV); putVect ("bV", bits, bV);
76          getVect ("ci", 1, ci); putVect ("ci", 1, ci);
77          cout << "\n";
78
79          nBitAdder (aV, bV, ci, co, sumV, bits); // calculates all propagations
80
81          putVect ("  aV", bits, aV); putVect ("  bV", bits, bV);
82          putVect ("  ci", 1, ci);
83          putVect ("sumV", bits, sumV); putVect ("  co", 1, co);
84
85          delete [] aV;
86          delete [] bV;
87          delete [] ci;
88          delete [] co;
89          delete [] sumV;
90
91          cout << "\nEnter 0 to exit: "; cin >> go;
92      }
93  }
94
```

100 %

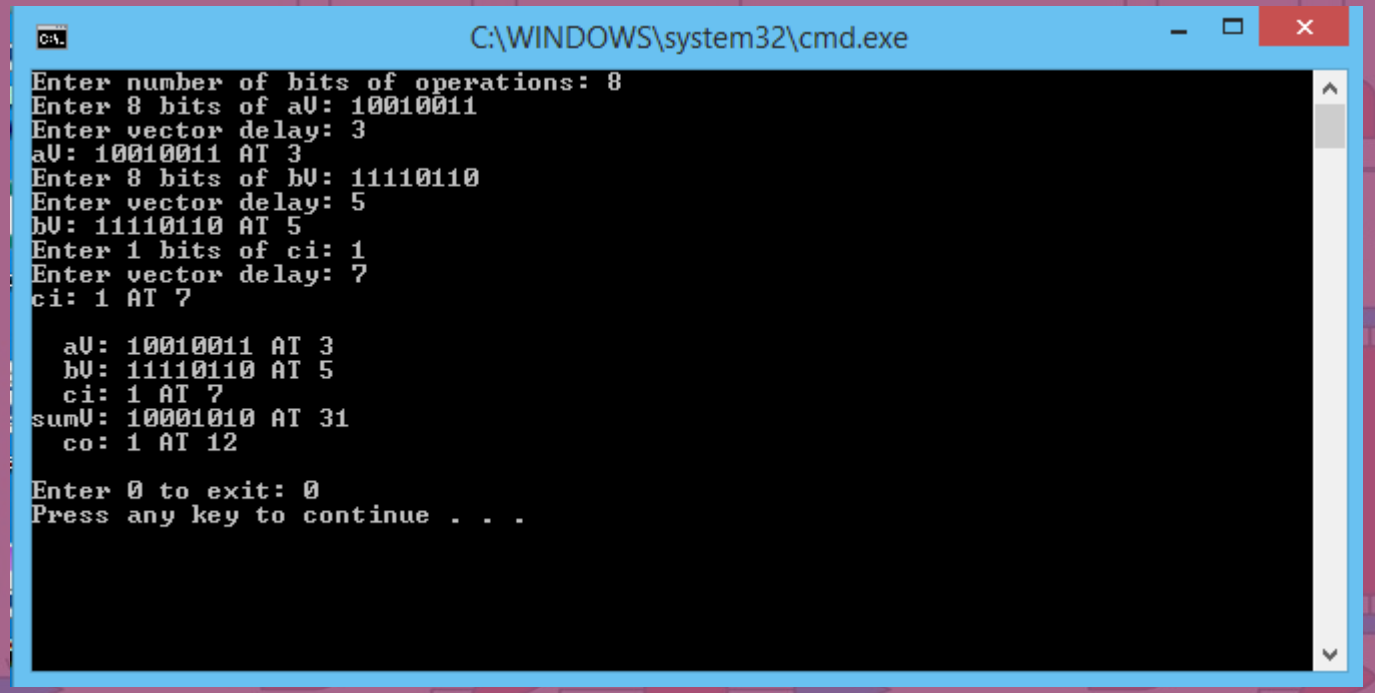
timedFunctions.cpp

Char based adder with tlogic

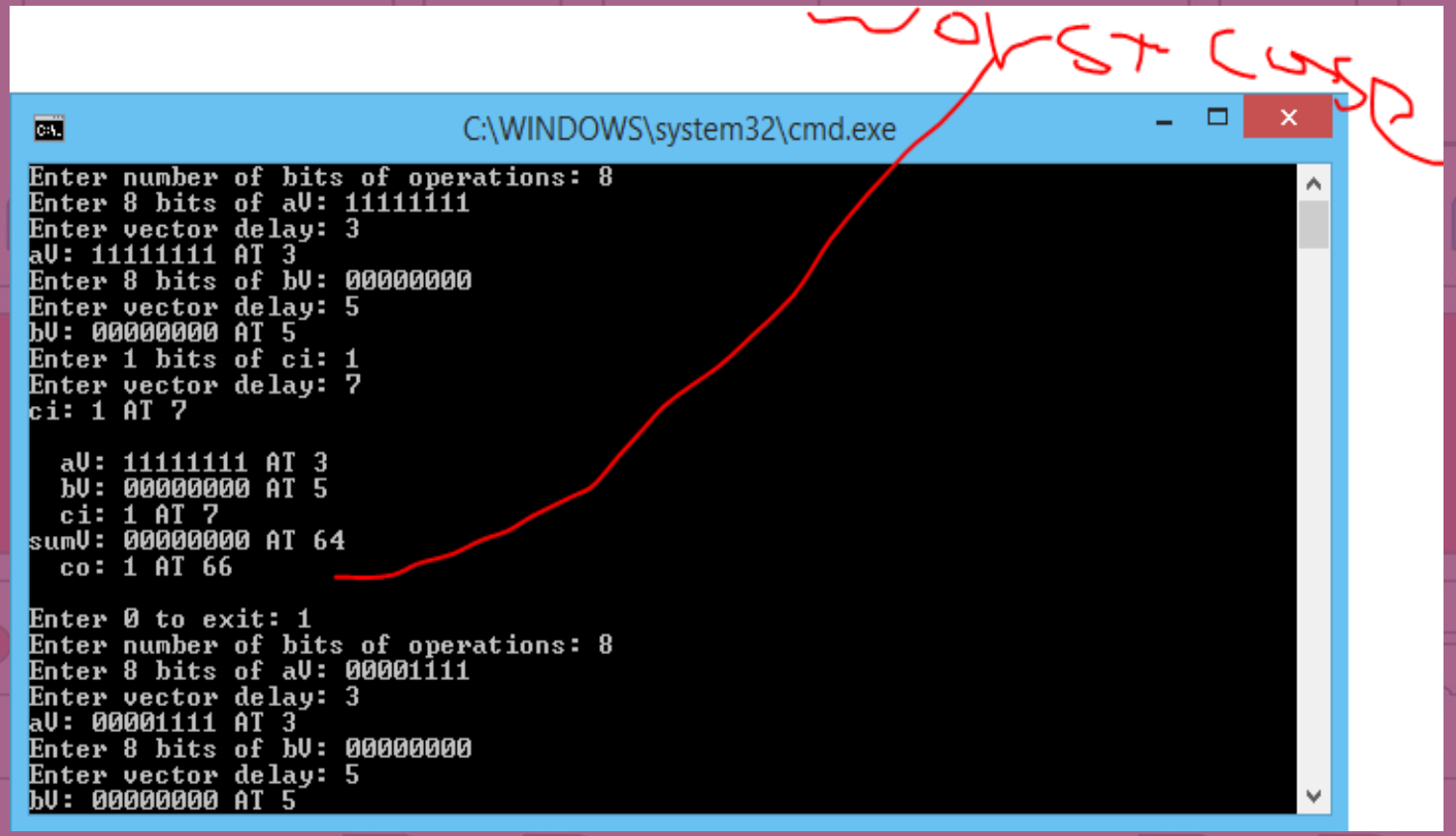
Timing in Logic Structures



Timing in Logic Structures



Timing in Logic Structures



Overloading Logical Operators

timedFunctions.cpp | timedFunctions.h | timedOperators.h* | timedOperators.cpp

Timed Logic Overloading | (Global Scope)

```
1  struct tlogic {
2      char logic;
3      int time;
4  };
5
6  tlogic operator& (tlogic a, tlogic b);
7  tlogic operator| (tlogic a, tlogic b);
8  tlogic operator~ (tlogic a);
9  tlogic operator^ (tlogic a, tlogic b);
10
11
```

100 %

Operator overloading for tlogic

timedOperators.h

Overloading Logical Operators

timedFunctions.cpp | timedFunctions.h | timedOperators.h* | timedOperators.cpp*

Timed Logic Overloading | (Global Scope)

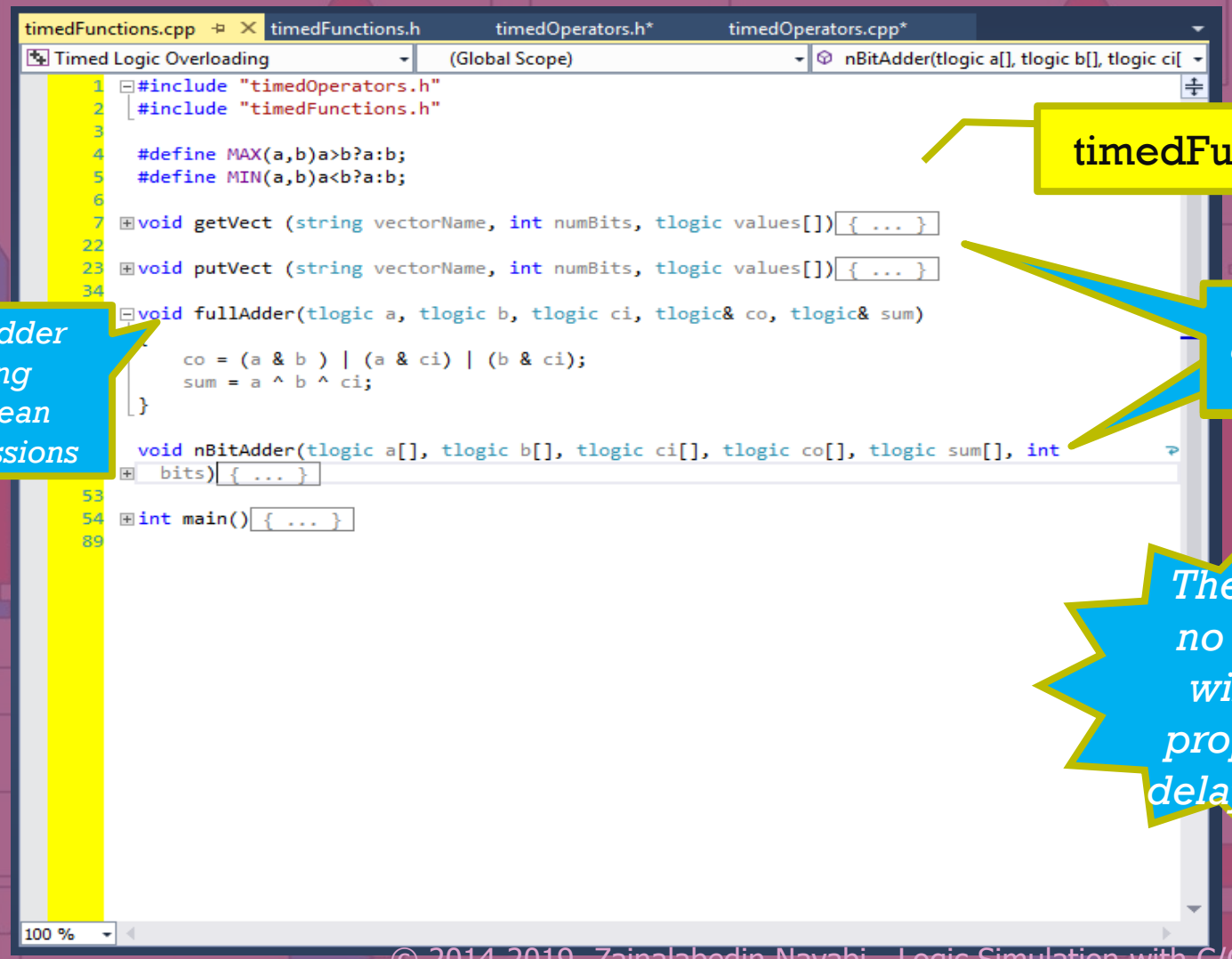
```
1   #include "timedOperators.h"
2
3   tlogic operator& (tlogic a, tlogic b) { ... }
23
24  tlogic operator| (tlogic a, tlogic b) { ... }
44
45  tlogic operator~ (tlogic a)
46  {
47      tlogic tl;
48      if (a.logic=='1') tl.logic = '0';
49      else if (a.logic=='0') tl.logic = '1';
50      else tl.logic='X';
51      tl.time = a.time;
52      return tl;
53  }
54
55  tlogic operator^ (tlogic a, tlogic b)
56  {
57      tlogic tl;
58      if (a.logic==b.logic) tl.logic = '0';
59      else tl.logic = '1';
60      if (a.time > b.time) tl.time = a.time;
61      else tl.time = b.time;
62      return tl;
63  }
64
65
```

100 %

timedOperators.cpp

Overloaded Operators for struct type

Using Boolean Expressions



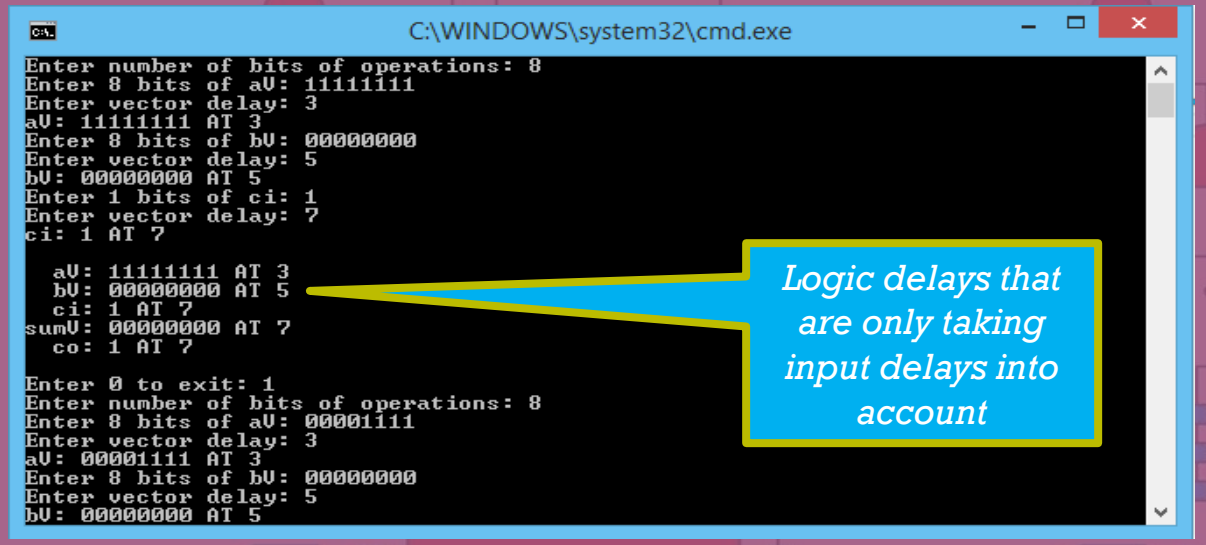
timedFunction.cpp

Full adder using Boolean expressions

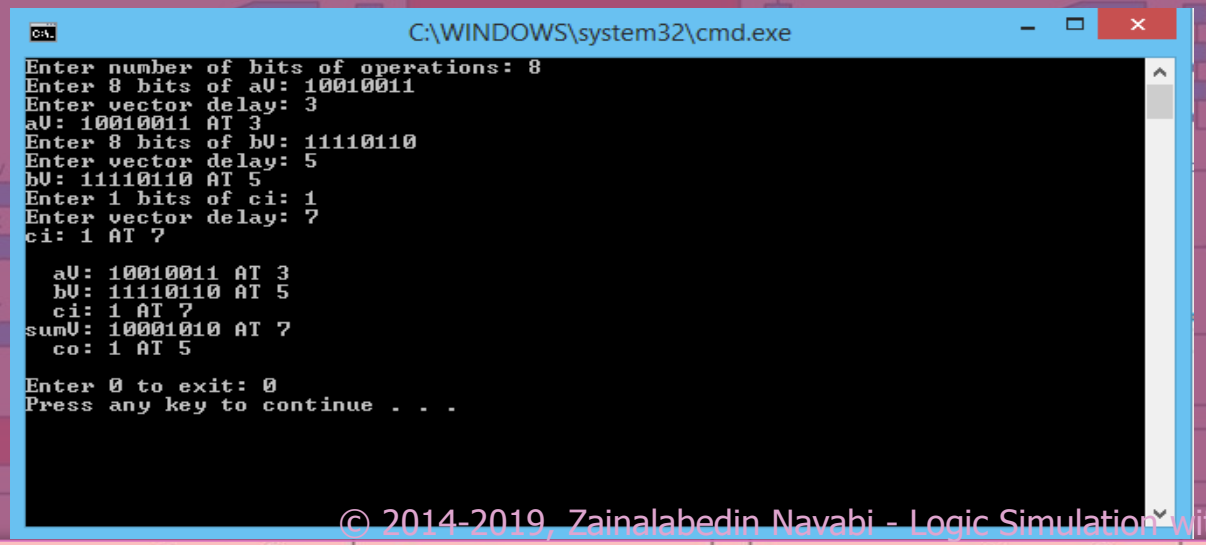
Full adder considers logic and timing

There are no inside wires to propagate delay values

Using Boolean Expressions



Logic delays that are only taking input delays into account



Logic Simulation with C/C++

- Procedural Languages for Hardware Modeling
- Types and Operators for Logic Modeling
- Basic Logic Simulation
  - Logic functions
  - Function overloading
  - Passing logic functions
  - Using default values
  - Building higher level structures
  - Handling 4-value logic
  - Logic vector
  - Sequential circuit modeling
  - Using pointers for logic vectors
- Enhanced logic simulation with timing
  - Using struct for timing and logic
  - Gates that handle timing
  - Utility functions
  - Timing in logic structures
  - Overloading logical operators
  - Using Boolean expressions
- More Functions for Wires and Gates
  - Gate classes
  - Carrier centric modeling
  - Compatible scalar and vector

Gate Classes

class3Functions.cpp | class3Primitives.cpp | class3Primitives.h

Class3 Logic | (Global Scope)

```
1   class and { ... };
9
10  class or {
11      char i1, i2, o1;
12      public:
13          or (); // constructor
14          void inp (char a, char b) {i1=a; i2=b;}
15          void evl ();
16          void out (char& w) {w=o1;}
17  };
18
19  class not { ... };
27
28  class xor { ... };
36
```

100 %

Member variable

class3Primitives.h

Inline implementation

Member function

external implementation

Gate Classes

class3Functions.cpp | class3Primitives.cpp | class3Primitives.h

Class3 Logic | (Global Scope)

```
1   #include "class3Primitives.h"
2
3   and::and() {o1='X';}
4  void and::evl () { ... }
9
10  or::or() {o1='X';}
11 void or::evl () {
12      if ((i1=='1')||(i2=='1')) o1='1';
13      else if ((i1=='0')&&(i2=='0')) o1='0';
14      else o1='X';
15  }
16
17  not::not() {o1='X';}
18 void not::evl () { ... }
23
24  xor::xor() {o1='X';}
25 void xor::evl () { ... }
30
```

100 %

class3Primitives.cpp

OR class member function

Gate Classes

class3Functions.cpp | class3Primitives.cpp | class3Primitives.h | class3Functions.h

Class3 Logic | (Global Scope)

```
1  #include "class3Primitives.h"
2  #include "class3Functions.h"
3
4  void fullAdder (char a, char b, char ci, char & co, char & sum)
5  {
6      char axb, ab, abc;
7      xor xor1, xor2;
8      and and1, and2;
9      or or1;
10
11     xor1.inp(a, b);
12         xor1.evl();
13         xor1.out(axb);
14     and1.inp(a, b);
15         and1.evl();
16         and1.out(ab);
17     and2.inp(axb, ci);
18         and2.evl();
19         and2.out(abc);
20     or1.inp(ab, abc);
21         or1.evl();
22         or1.out(co);
23     xor2.inp(axb, ci);
24         xor2.evl();
25         xor2.out(sum);
26 }
27
28
29 int main () { ... }
65
```

100 %

class3Functions.cpp

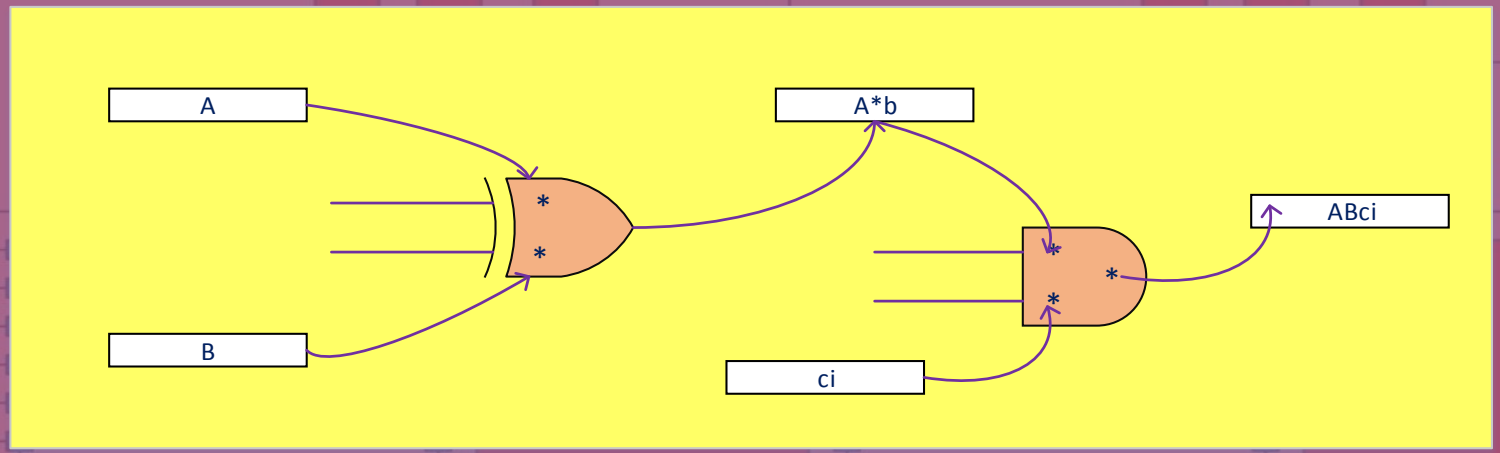
Full adder function using gate classes

Gate Classes

class3Functions.cpp

```
28
29  int main ()
30  {
31      char aC;
32      char bC;
33      char ciC;
34      char coC;
35      char sumC;
36
37      int ai;
38
39      do {
40
41          cout << "Enter a: ";
42          cin >> aC; cout << aC << '\n';
43
44          cout << "Enter b: ";
45          cin >> bC; cout << bC << '\n';
46
47          cout << "Enter ci: ";
48          cin >> ciC; cout << ciC << '\n';
49
50  //      and (aC, bC, wC);
51  //      cout << "and:" << wC << '\n';
52
53  //      or (aC, bC, wC);
54  //      cout << "or: " << wC << '\n';
55
56          fullAdder (aC, bC, ciC, coC, sumC);
57
58          cout << "Carry: " << coC << '\n';
59          cout << "  Sum: " << sumC << '\n';
60
61          cout << "\n" << "Continue?"; cin >> ai;
62
63      } while (ai>0);
64  }
65
```

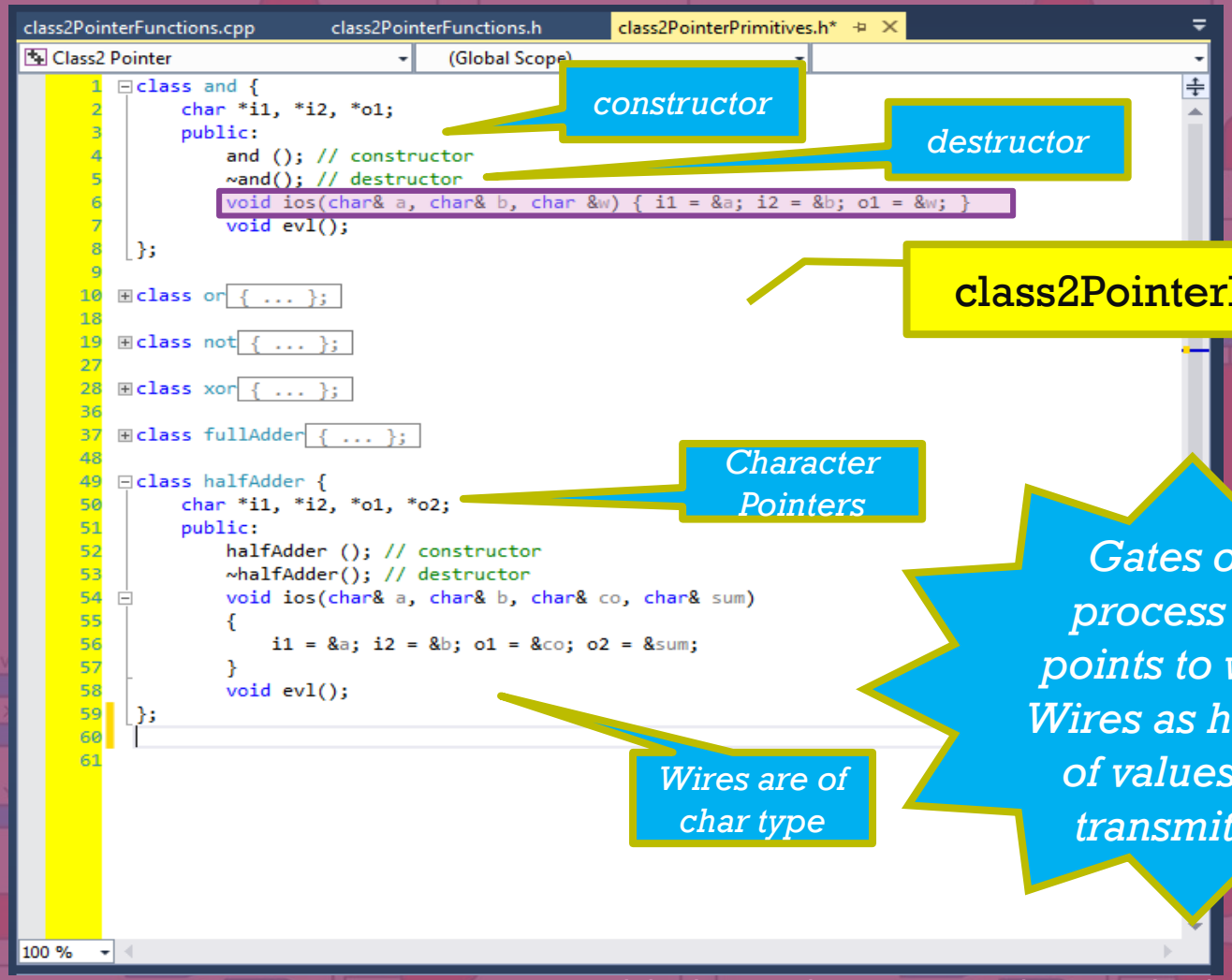
Carrier Centric Modeling



Pointer Based Logic Classes

- Classes do not hold values. Since the lines are just pointers, someone else has to declare them and allocate them.
- evl and out are combined and evl does both. Actually, since the outputs are pointers they will just be updated by evl. Every invocation of evl puts the internal output values on the evl return value.
- Destructor is introduced.

Pointer Based Logic Classes



constructor

destructor

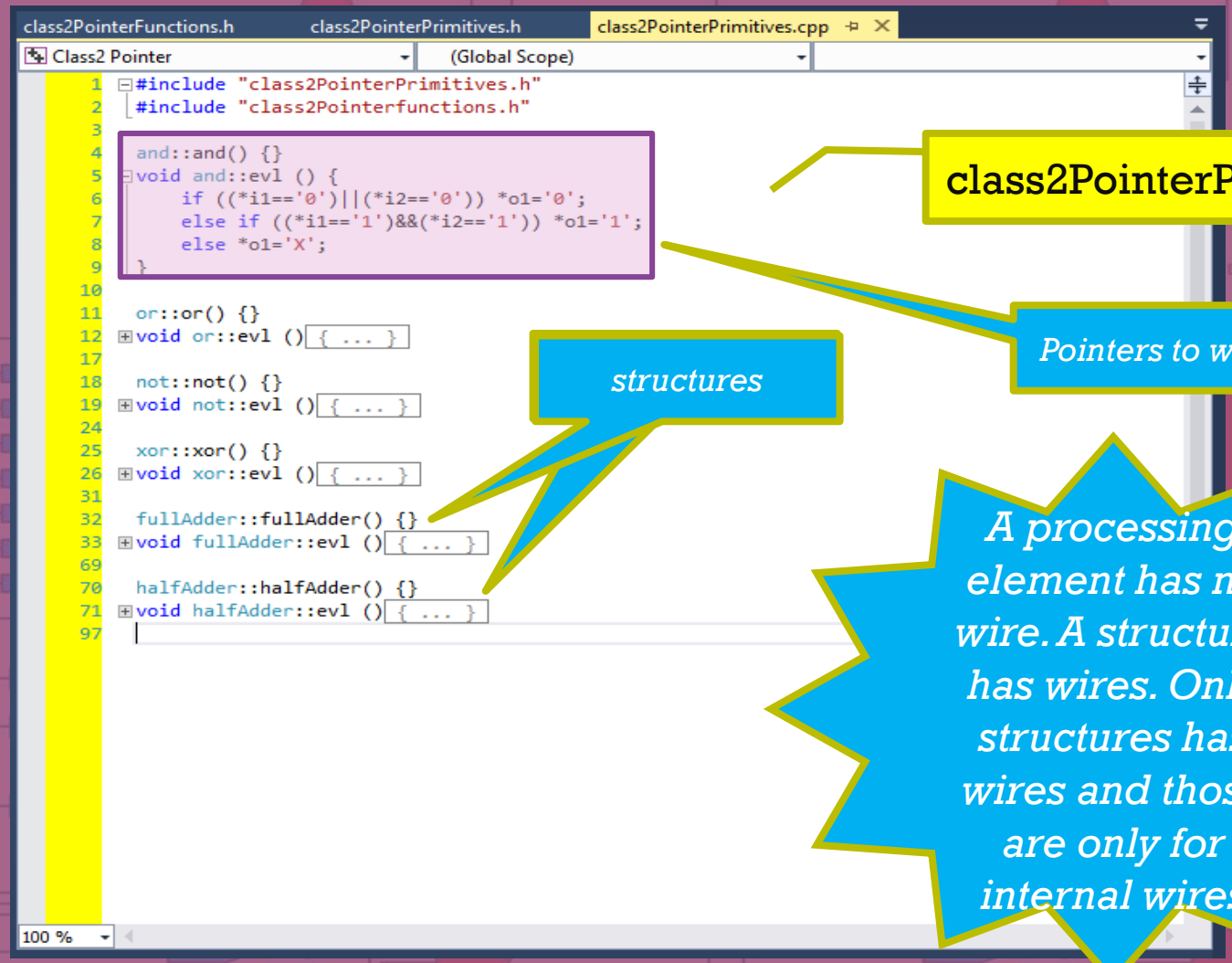
class2PointerPrimitives.h

Character Pointers

Wires are of char type

Gates only process and points to wires. Wires as holders of values and transmitters

Pointer Based Logic Classes



class2PointerPrimitives.cpp

Pointers to wires

structures

A processing element has no wire. A structure has wires. Only structures has wires and those are only for internal wires

Pointer Based Logic Classes

class2PointerFunctions.h | class2PointerPrimitives.h | class2PointerPrimitives.cpp

Class2 Pointer | (Global Scope)

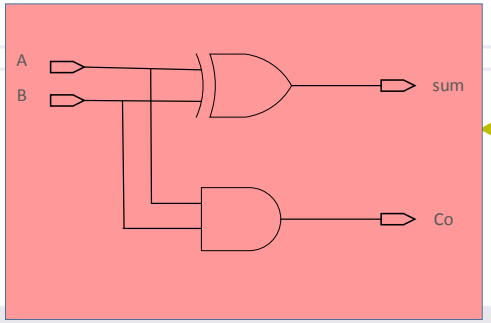
```
70  halfAdder::halfAdder() {}
71  void halfAdder::evl () {
72      // halfadder Local wires
73      char aL('X'), bL('X');
74      char coL('X'), sumL('X');
75
76      // Declare necessary gate instances
77      xor *xor1 = new xor;
        and *and1 = new and;

        // Associate ports of the gates with the Local HA wires
        xor1->ios(aL, bL, sumL);
        and1->ios(aL, bL, coL);

84      // Via the HA pointers, read wire values that connect to
85      // the HA from outside, and assign them to HA Local wires
86      aL = *i1; bL = *i2;
87
88      // Evaluate gates in the proper order
89      xor1->evl();
90      and1->evl();
91
92      // Take calculated local wire values and assign the values
93      // to the outside wires via pointers of FA
94      *o1 = coL; *o2 = sumL;
95
96  }
97
```

100 %

class2PointerPrimitives.cpp

Or
*xor1.ios()

Order evl() functions according to logic



Structures also have gates that have no internal wires

Pointer Based Logic Classes

class2PointerFunctions.h | class2PointerPrimitives.h | class2PointerPrimitives.cpp

Class2 Pointer | (Global Scope)

```
32  fullAdder::fullAdder() {}
33  void fullAdder::evl () {
34
35      // fulladder Local wires
36      char aL('X'), bL('X'), ciL('X');
37      char coL('X'), sumL('X');
38      char axbL('X'), abL('X'), abcL('X');
39
40      // Declare necessary gate instances
41      xor *xor1=new xor;
42      xor *xor2=new xor;
43      and *and1=new and;
44      and *and2=new and;
45      or  *or1= new or;
46
47      // Associate ports of the gates with the Local FA wires
48      xor1->ios(aL, bL, axbL);
49      and1->ios(aL, bL, abL);
50      and2->ios(axbL, ciL, abcL);
51      or1->ios(abL, abcL, coL);
52      xor2->ios(axbL, ciL, sumL);
53
54      // Via the FA pointers, read wire values that connect to
55      // the FA from outside, and assign them to FA Local wires
56      aL = *i1; bL = *i2; ciL = *i3;
57
58      // Evaluate gates in the proper order
59      xor1->evl();
60      and1->evl();
61      and2->evl();
62      or1->evl();
63      xor2->evl();
64
65      // Take calculated local wire values and assign the values
66      // to the outside wires via pointers of FA
67      *o1 = coL; *o2 = sumL;
68  }
```

100 %

class2PointerPrimitives.cpp

Port association

Only evl() functions must be ordered

Pointer Based Logic Classes

class2PointerPrimitives.h | class2PointerPrimitives.cpp | class2PointerFunctions.cpp

Class2 Pointer | (Global Scope)

```
1  #include "class2PointerPrimitives.h"
2  #include "class2PointerFunctions.h"
3
4  int main ()
5  {
6      char aC('X'), bC('X'), ciC('X'), coCF('X'), sumCF('X'),
7           coCH('X'), sumCH('X');
8
9      fullAdder *FA;
10     FA=new fullAdder();
11     halfAdder *HA;
12     HA=new halfAdder();
13
14     FA->ios(aC, bC, ciC, coCF, sumCF);
15     HA->ios(aC, bC, coCH, sumCH);
16
17     int ai;
18
19     do {
20         cout << "Enter a: "; cin >> aC;
21         cout << "Enter b: "; cin >> bC;
22         cout << "Enter ci: "; cin >> ciC;
23
24         FA->evl();
25         HA->evl();
26
27         cout << "FA - Carry: " << coCF << ";  Sum: " << sumCF << '\n';
28         cout << "HA - Carry: " << coCH << ";  Sum: " << sumCH << '\n';
29
30         cout << "\n" << "Continue?"; cin >> ai;
31
32     } while (ai>0);
33 }
34
```

100 %

class2PointerFunctions.cpp

Shows main for full adder and half adder functions

Gate Classes with Power and Timing Calculation

timedLogicFunctions.cpp | timedLogicUtilities.cpp | timedLogicFunctions.h | timedLogicPrimitives.h

Timed Logic Classes | (Global Scope)

```
1   int calculateEventTime(char lastValue, char newValue,
2       int in1LastEvent, int in2LastEvent, int gateDelay, int lastEvent);
3
4   class wire {
5   public:
6       char value;
7       int eventTime;
8       int activityCount=0;
9       public:
10          wire(char c, int d) : value(c), eventTime(d) {}
11          wire(){};
12          void put(char a, int d) { value = a; eventTime = d; }
13          void get(char& a, int& d) { a = value; d = eventTime; }
14          int activity() { return activityCount; }
15  };
16
17  class and {
18      wire *i1, *i2, *o1;
19      int gateDelay, lastEvent;
20      char lastValue;
21      public:
22          and(wire& a, wire& b, wire& w, int d) :
23              i1(&a), i2(&b), o1(&w), gateDelay(d) {};
24          ~and();
25          void evl();
26  };
27
28  class or { ... };
37
38  class not { ... };
47
48  class xor { ... };
57
58  class dff_ar {
59      wire *D, *clk, *R, *Q;
60      int clkQDelay, rstQDelay;
61      int lastEvent; // last time output changed
```

100 %

EventTime to propagate delay

ActivityCount to carry power consumption

timedLogicPrimitives.h

Wires have constructor for value and event time.

They have put and get for accessing their value and event time

Wires have access function to activityCount

Declare wire to contain more information than just logic value

Gate Classes with Power and Timing Calculation

Xor constructor just ties port pointers to wires

timedLogicPrimitives.h

O1 is a pointer. This pointer is tied to pointer of w

timedLogicFunctions.cpp | timedLogicUtilities.cpp | timedLogicFunctions.h | timedLogicPrimitives.h

Timed Logic Classes | (Global Scope)

```
48  class xor {
49      wire *i1, *i2, *o1;
50      int gateDelay, lastEvent;
51      char lastValue = 'X';
52      public:
            xor(wire& a, wire& b, wire& w, int d) : i1(&a), i2(&b), o1(&w)
            ~xor();
55          void evl();
56  };
57
58  class dff_ar {
59      wire *D, *clk, *R, *Q;
60      int clkQDelay, rstQDelay;
61      int lastEvent; // last time output changed
62      char lastValue;
63
64      public:
65          dff_ar(wire& d, wire& c, wire& r, wire& q, int dC, int dR) :
66              D(&d), clk(&c), R(&r), Q(&q), clkQDelay(dC), rstQDelay(dR) {};
67          ~dff_ar();
68          void evl();
69  };
70
71  // Structures based on above primitives begin here
72
73  class fullAdder {
74      wire *i1, *i2, *i3, *o1, *o2;
75
76      // Declare necessary gate instances
77      xor *xor1;
78      xor *xor2;
79      and *and1;
80      and *and2;
81      or  *or1;
82
83      // fulladder Local wires
84      wire aL, bL, ciL;
```

100 %

Gate Classes with Power and Timing Calculation

timedLogicFunctions.cpp | timedLogicUtilities.cpp | timedLogicFunctions.h | timedLogicPrimitives.h

Timed Logic Classes | (Global Scope)

```
71  // Structures based on above primitives begin here
72
73  class fullAdder {
74      wire *i1, *i2, *i3, *o1, *o2;
75
76      // Declare necessary gate instances
77      xor *xor1;
78      xor *xor2;
79      and *and1;
80      and *and2;
81      or  *or1;
82
83      // fulladder Local wires
84      wire aL, bL, ciL;
85      wire coL, sumL;
86      wire axbL, abL, abcL;
87
88      public:
89          fullAdder(wire& a, wire& b, wire& ci, wire& co, wire& sum) :
90              i1(&a), i2(&b), i3(&ci), o1(&co), o2(&sum),
91              aL('X', 0), bL('X', 0), ciL('X', 0),
92              coL('X', 0), sumL('X', 0),
93              axbL('X', 0), abL('X', 0), abcL('X', 0) {
94
95          // Associate ports of the gates with the Local FA wires
96          xor1 = new xor(aL, bL, axbL, 5); // 5 is gate delay
97          xor2 = new xor(axbL, ciL, sumL, 5);
98          and1 = new and(aL, bL, abL, 3);
99          and2 = new and(axbL, ciL, abcL, 3);
100         or1 = new or(abL, abcL, coL, 3);
101         };
102         ~fullAdder();
103         void evl();
104 };
105
106 class halfAdder {
107     wire *i1, *i2, *o1, *o2;
```

100 %

timedLogicPrimitives.h

Full adder class definition declares gates and internal wires

Full adder constructor ties ports of the full adder to external wires and initialize internal wires

Then It has evl() function that call gate classes in proper order

Gate Classes with Power and Timing Calculation

timedLogicFunctions.cpp | timedLogicUtilities.cpp | timedLogicFunctions.h | timedLogicPrimitives.h

Timed Logic Classes | (Global Scope)

```
105
106 class halfAdder {
107     wire *i1, *i2, *o1, *o2;
108
109     // Declare necessary gate instances
110     xor *xor1;
111     and *and1;
112
113     // halfadder Local wires
114     wire aL, bL;
115     wire coL, sumL;
116
117     public:
118         halfAdder(wire& a, wire& b, wire& co, wire& sum) :
119             i1(&a), i2(&b), o1(&co), o2(&sum),
120             aL('X', 0), bL('X', 0), coL('X', 0), sumL('X', 0){
121
122         // Associate ports of the gates with the Local HA wires
123         xor1 = new xor(aL, bL, sumL, 5);
124         and1 = new and(aL, bL, coL, 3);
125         };
126         ~halfAdder();
127         void evl();
128 };
129
130
```

100 %

timedLogicPrimitives.h

half adder constructor ties ports of the full adder to external wires and initialize internal wires

Then It has evl() function that call gate classes in proper order

Gate Classes with Power and Timing Calculation

timedLogicUtilities.h | timedLogicFunctions.cpp | timedLogicPrimitives.cpp

Timed Logic Classes | (Global Scope) | inpBit(string, wire &)

```
1  #include "timedLogicPrimitives.h"
2  #include "timedLogicFunctions.h"
3
4  void inpBit(string, wire&);
5  void outBit(string, wire);
```

100 %

timedLogicUtilities.h

timedLogicUtilities.cpp | Timed Logic Classes | (Global Scope)

```
1  #include "timedLogicUtilities.h"
2
3  void inpBit(string wireName, wire& valtim) {
4      char value;
5      int time;
6      cout << "Enter value followed by @ time for " << wireName << ": ";
7      cin >> value; cin >> time;
8      valtim.put(value, time);
9  }
10
11 void outBit(string wireName, wire valtim) {
12     char value;
13     int time;
14     valtim.get(value, time);
15     cout << wireName << ": " << value << " @ " << time << "\n";
16 }
```

100 %

timedLogicUtilities.cpp

For implementing this we need several utility functions. For inbit and outbit to get time and value for wires

Gate Classes with Power and Timing Calculation

timedLogicUtilities.cpp | timedLogicUtilities.h | timedLogicFunctions.cpp | timedLogicPrimitives.cpp

Timed Logic Classes → fullAdder → evl()

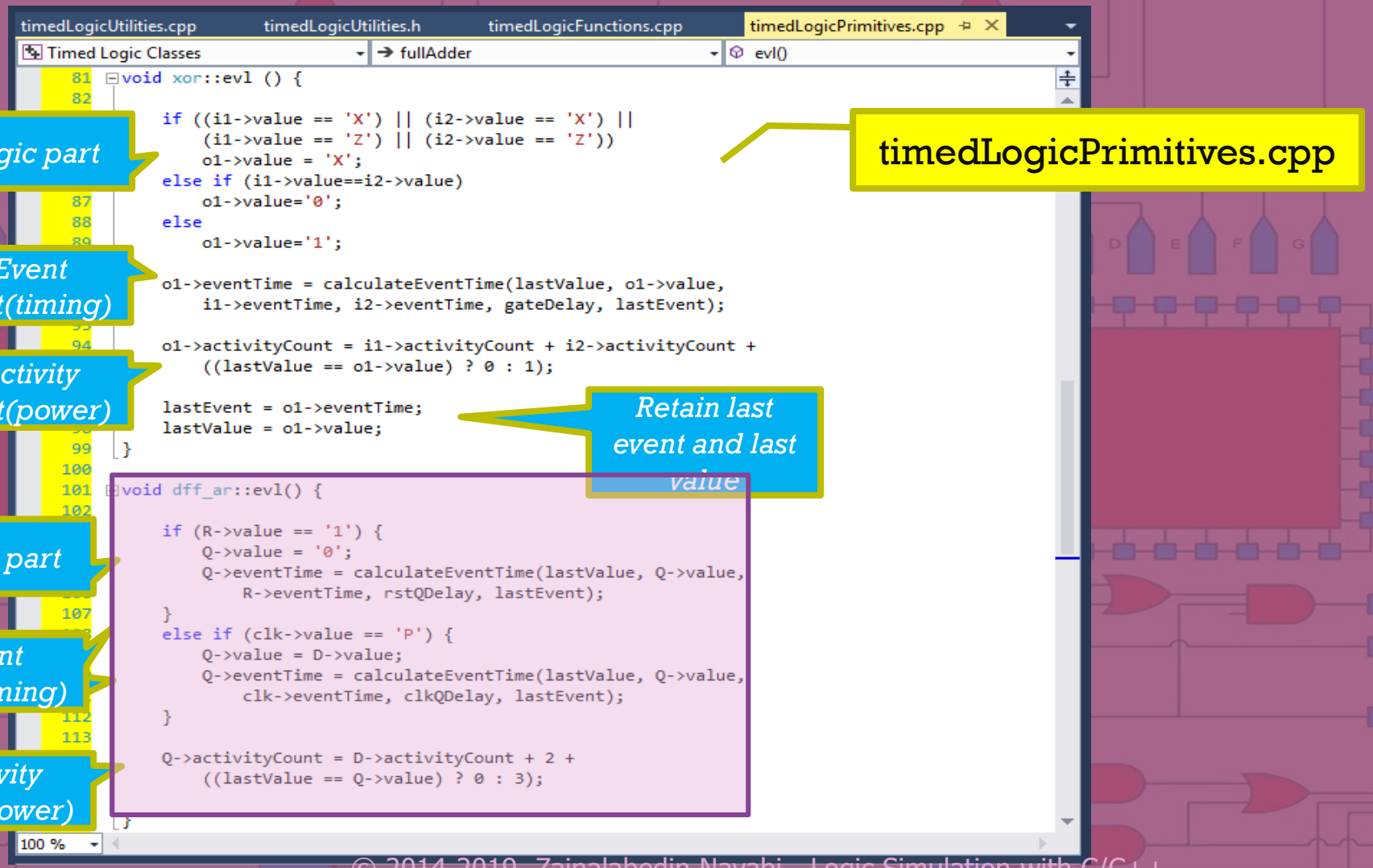
```
1  #include "timedLogicPrimitives.h"
2  #include "timedLogicFunctions.h"
3
4  #define MAX(a,b) ((a>b)?a:b)
5
6  int calculateEventTime(char lastValue, char newValue,
7      int in1LastEvent, int in2LastEvent, int gateDelay, int lastEvent){
8
9      if (lastValue == newValue)
10         return lastEvent;
11     else
12         return gateDelay + MAX (in1LastEvent, in2LastEvent);
13 }
14
15 int calculateEventTime(char lastValue, char newValue,
16     int in1LastEvent, int gateDelay, int lastEvent){
17
18     if (lastValue == newValue)
19         return lastEvent;
20     else
21         return gateDelay + in1LastEvent;
22 }
23
24 void and::evl () {
25
26     if ((i1->value == '0') || (i2->value == '0'))
27         o1->value = '0';
28     else if ((i1->value == '1') && (i2->value == '1'))
29         o1->value = '1';
30     else
31         o1->value='X';
32
33     o1->eventTime = calculateEventTime(lastValue, o1->value,
34         i1->eventTime, i2->eventTime, gateDelay, lastEvent);
35
36     o1->activityCount = i1->activityCount + i2->activityCount +
37         ((lastValue == o1->value) ? 0 : 1);
```

100 %

timedLogicPrimitives.cpp

If output has changed, the last event time on output is the larger of the inputs plus gate delay

Gate Classes with Power and Timing Calculation




```
81  void xor::evl () {
82
        if ((i1->value == 'X') || (i2->value == 'X') ||
            (i1->value == 'Z') || (i2->value == 'Z'))
            o1->value = 'X';
        else if (i1->value==i2->value)
            o1->value='0';
87      else
88          o1->value='1';

        o1->eventTime = calculateEventTime(lastValue, o1->value,
            i1->eventTime, i2->eventTime, gateDelay, lastEvent);

94      o1->activityCount = i1->activityCount + i2->activityCount +
            ((lastValue == o1->value) ? 0 : 1);

        lastEvent = o1->eventTime;
        lastValue = o1->value;
99  }
100
101 void dff_ar::evl() {
102
        if (R->value == '1') {
            Q->value = '0';
            Q->eventTime = calculateEventTime(lastValue, Q->value,
                R->eventTime, rstQDelay, lastEvent);
        }
107     else if (clk->value == 'P') {
            Q->value = D->value;
            Q->eventTime = calculateEventTime(lastValue, Q->value,
                clk->eventTime, clkQDelay, lastEvent);
        }
112
113     Q->activityCount = D->activityCount + 2 +
            ((lastValue == Q->value) ? 0 : 3);
    }
```

Gate Classes with Power and Timing Calculation

timedLogicPrimitives.cpp

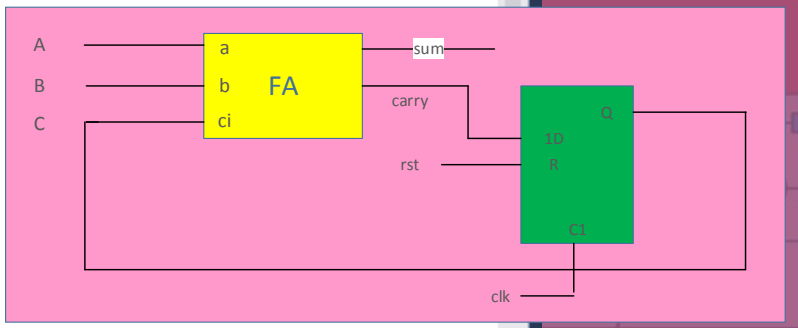
```
timedLogicFunctions.h | timedLogicPrimitives.cpp | timedLogicFunctions.cpp | timedLogicPrimitives.h
Timed Logic Classes | (Global Scope) | calculateEventTime(char lastValue, char

120
121 void fullAdder::evl () {
122
123     // Via the FA pointers, read wire values that connect to
124     // the FA from outside, and assign them to FA Local wires
125     aL = *i1; bL = *i2; ciL = *i3;
126
127     // Evaluate gates in the proper order
128     xor1->evl();
129     and1->evl();
130     and2->evl();
131     or1->evl();
132     xor2->evl();
133
134     // Take calculated local wire values and assign the values
135     // to the outside wires via pointers of FA
136     *o1 = coL; *o2 = sumL;
137 }
138
139 void halfAdder::evl () {
140
141     // Via the HA pointers, read wire values that connect to
142     // the HA from outside, and assign them to HA Local wires
143     aL = *i1; bL = *i2;
144
145     // Evaluate gates in the proper order
146     xor1->evl();
147     and1->evl();
148
149     // Take calculated local wire values and assign the values
150     // to the outside wires via pointers of FA
151     *o1 = coL; *o2 = sumL;
152 }
153

100 %
```

Gate Classes with Power and Timing Calculation

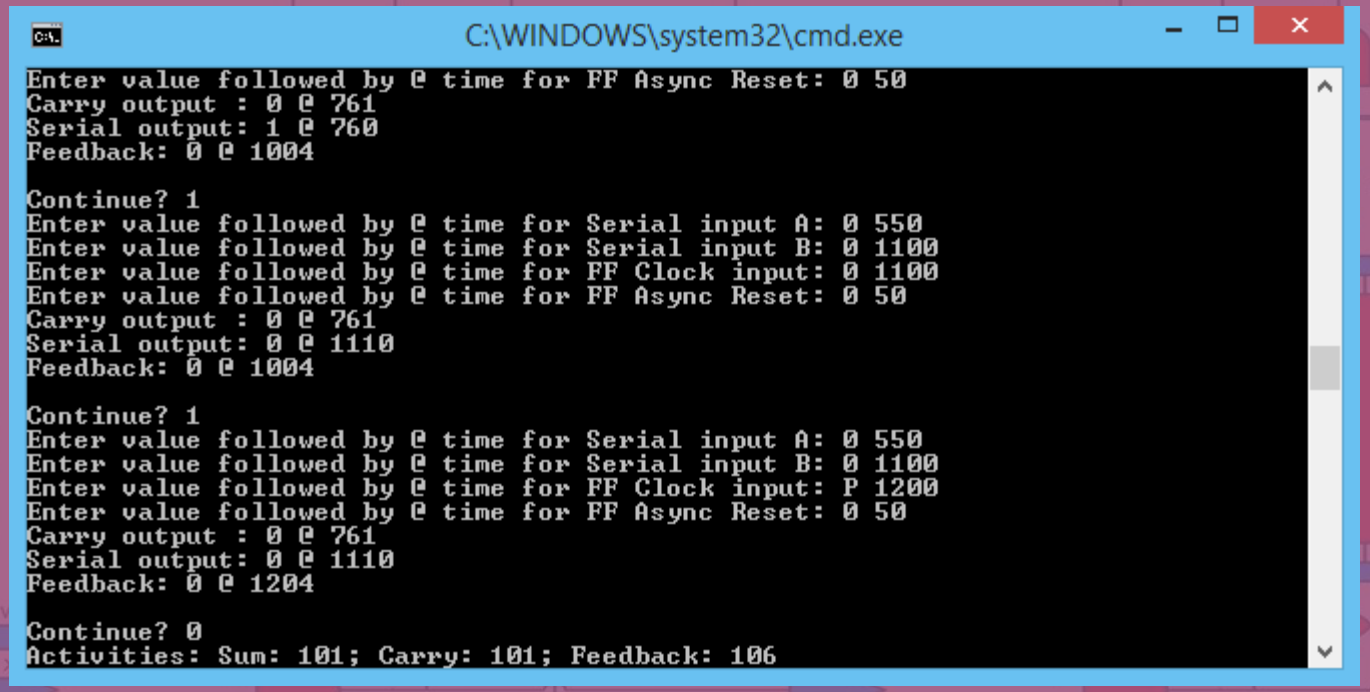
timedLogicFunctions.cpp

Serial Adder

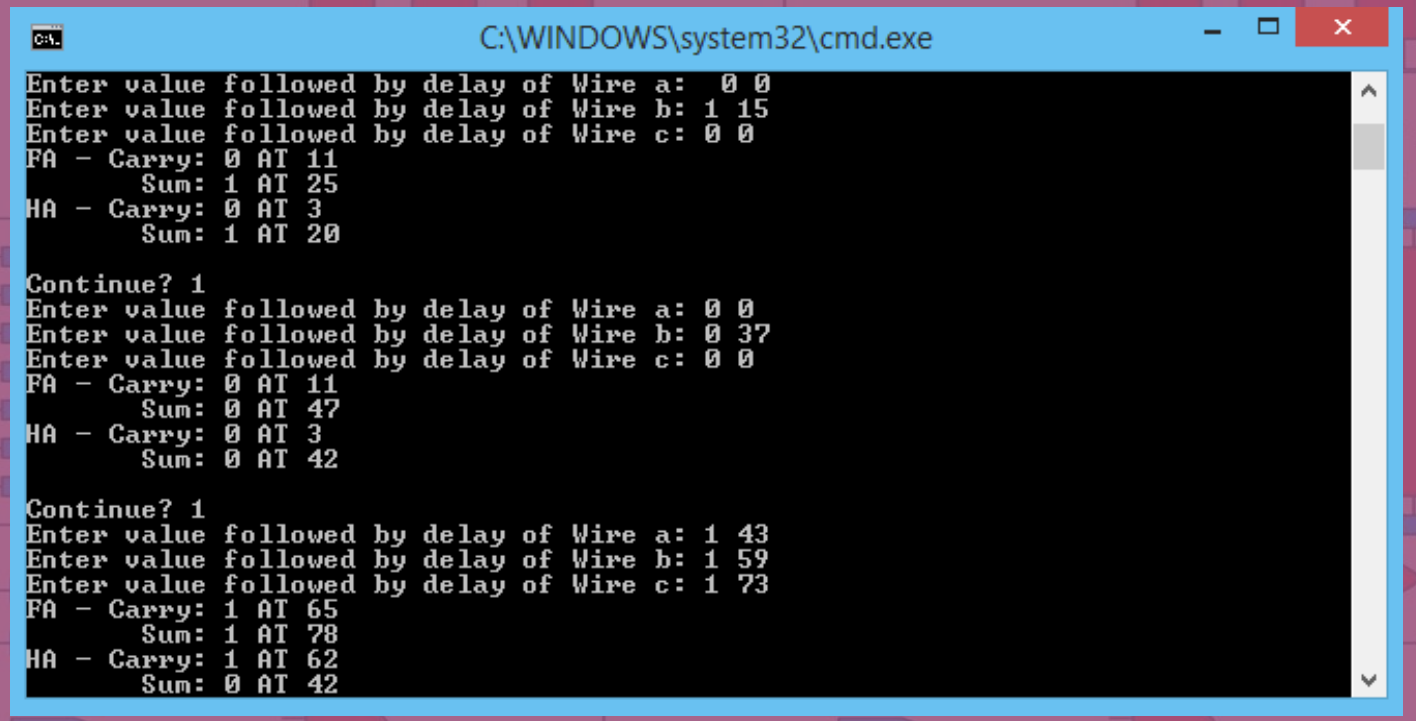
```
30
31  int main ()
32  {
33      wire A, B;
34      wire clk, rst, fb('X',0);
35      wire sum, carry;
36
37      fullAdder *FA = new fullAdder(A, B, fb, carry, sum);
38      dff_ar *FF = new dff_ar(carry, clk, rst, fb, 4, 6);
39
40      int ai=1;
41
42      do {
43          inpBit("Serial input A", A);
44          inpBit("Serial input B", B);
45          inpBit("FF Clock input", clk);
46          inpBit("FF Async Reset", rst);
47
48          FA->evl();
49
50          outBit("Carry output ", carry);
51          outBit("Serial output", sum);
52
53          FF->evl();
54
55          outBit("Feedback", fb);
56
57          cout << "\n" << "Continue? "; cin >> ai;
58          } while (ai>0);
59
60      cout << "Activities: Sum: " << sum.activity()
61          << "; Carry: " << carry.activity()
62          << "; Feedback: " << fb.activity() << '\n'
63  }
64
65
```



Gate Classes with Power and Timing Calculation



Gate Classes with Power and Timing Calculation



Wire and Gate Vectors

timedVectorLogicFunctions.cpp | timedVectorLogicUtilities.cpp | timedVectorLogicPrimitives.h

Timed Vector Logic Classes | halfAdder | evl()

```
133
134 class wireV {
135 public:
136     char* value;
137     int n; //Bits
138     int eventTime;
139     int activityCount = 0;
140 public:
141     wireV(string v, int d, int size);
142     wireV(){};
143     ~wireV(){};
144     void put(string a, int d);
145     void get(string& a, int& d);
146     int activity() { return activityCount; }
147 };
148
149 class andV {
150     wireV *i1, *i2, *o1;
151     int gateDelay, lastEvent;
152     char* lastValue;
153 public:
154     andV(wireV& a, wireV& b, wireV& w, int d) :
155         i1(&a), i2(&b), o1(&w), gateDelay(d) {
156         lastValue = new char[w.n+1];
157     };
158     ~andV(){};
159     void evl();
160 };
161
162 class orV {
163     wireV *i1, *i2, *o1;
164     int gateDelay, lastEvent;
165     char* lastValue;
166 public:
167     orV(wireV& a, wireV& b, wireV& w, int d) :
168         i1(&a), i2(&b), o1(&w), gateDelay(d) {
169         lastValue = new char[w.n + 1];
```

100 %

timedVectorLogicPrimitives.h

Main difference with wire

Last value for timing calculation

Wire vector has an event time for a group of wires and an activityCount for a group of wires. This model is not accurate since all individual wires are treated the same

Wire and Gate Vectors

timedVectorLogicUtilities.cpp | timedVectorLogicPrimitives.h | timedVectorLogicUtilities.h

Timed Vector Logic Classes | (Global Scope) | inpBit(string wireName, wireV & valtim)

```
1  #include "timedVectorLogicUtilities.h"
2
3  void inpBit(string wireName, wire& valtim) {
4      char value;
5      int time;
6      cout << "Enter value followed by @ time for " << wireName << ": ";
7      cin >> value; cin >> time;
8      valtim.put(value, time);
9  }
10
11 void outBit(string wireName, wire valtim) {
12     char value;
13     int time;
14     valtim.get(value, time);
15     cout << wireName << ": " << value << " @ " << time << "\n";
16 }
17
18 void inpBit(string wireName, wireV& valtim) {
19     string value;
20     int time;
21     cout << "Enter value followed by @ time for " << wireName << ": ";
22     cin >> value; cin >> time;
23     valtim.put(value, time);
24 }
25
26 void outBit(string wireName, wireV valtim) {
27     string value;
28     int time;
29     valtim.get(value, time);
30     cout << wireName << ": " << value << " @ " << time << "\n";
31 }
```

100 %

timedVectorLogicPrimitives.h

Utility for individual wires

Utility for arrays

Wire and Gate Vectors

timedVectorLogicUtilities.cpp | timedVectorLogicUtilities.h | timedVectorLogicPrimitives.cpp*

```
156 wireV::wireV(string v, int d, int size) : eventTime(d), n(size) {
157     int i;
158     value = new char[n + 1];
159     v.resize(n, 'X');
160     for (i = 0; i < n; i++){ *(i + value) = v.at(i); };
161     *(n + value) = '\0';
162 }
163 void wireV::put(string a, int d){
164     int i;
165     eventTime = d;
166     a.resize(n, '0');
167     for (i = 0; i < n; i++){ *(i + value) = a.at(i);};
168 }
169 void wireV::get(string& a, int& d){
170     int i;
171     d = eventTime;
172     a.resize(n, '0');
173     for (i = 0; i < n; i++){ a.at(i) = *(i + value); };
174 }
175
176 void andV::evl() {
177     int i = 0;
178
179     while (i1->value[i] != '\0'){
180         if (((i1->value[i]) == '0') || ((i2->value[i]) == '0'))
181             o1->value[i] = '0';
182         else if ((i1->value[i] == '1') && (i2->value[i] == '1'))
183             o1->value[i] = '1';
184         else
185             o1->value[i] = 'X';
186         i++;
187     };
188 }
189
190 void orV::evl() {
191     int i = 0;
192     while (i1->value[i] != '\0'){
```

timedVectorLogicPrimitives.cpp

Adding \0 to make it compatible with the c++ predefined string class

Evl() function for wireV. Since they are clusters, individual delay and power do not apply

Wire and Gate Vectors

timedVectorLogicUtilities.cpp | timedVectorLogicPrimitives.cpp* | timedVectorLogicFunctions.cpp

Timed Vector Logic Classes | (Global Scope)

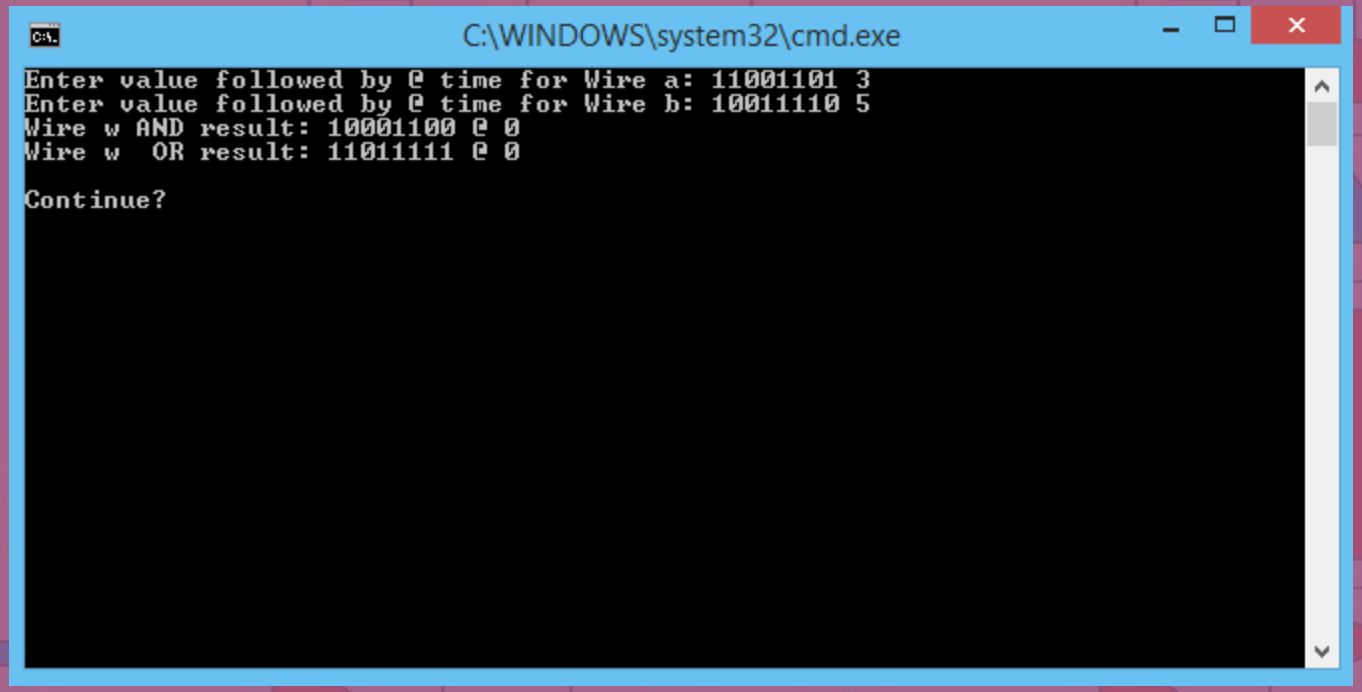
```
64
65  int main ()
66  {
67      wireV aWV("10101111", 0, 8), bWV("00110000", 0, 8), cWV("00001111", 0, 8),
68          wWV("00001111", 0, 8), yWV("XXXX0000", 0, 8);
69
70      andV *AND = new andV(aWV, bWV, wWV, 0);
71      orV *OR = new orV(aWV, bWV, yWV, 0);
72
73      int ai;
74
75      do {
76          inpBit("Wire a", aWV);
77          inpBit("Wire b", bWV);
78
79          AND->evl();
80          OR->evl();
81
82          outBit("Wire w AND result", wWV);
83          outBit("Wire w  OR result", yWV);
84
85      cout << "\n" << "Continue? "; cin >> ai;
86
87      } while (ai>0);
88  }
89
90
```

100 %

timedVectorLogicFunctions.cpp

Vectors have character pointer instead of char

Wire and Gate Vectors



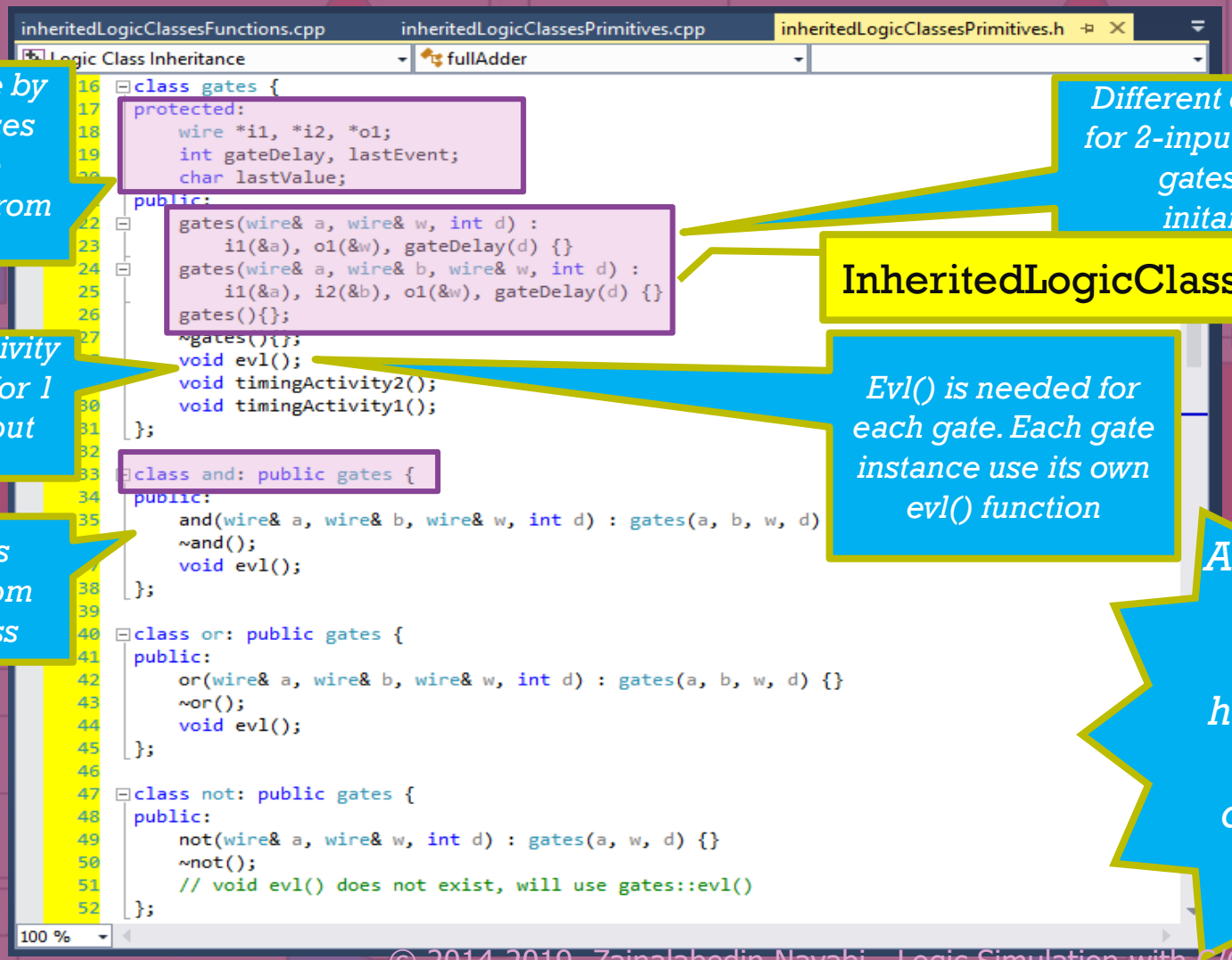
Logic Simulation with C/C++

- Containing Event Based Timing
  - To include in wires
  - To include in gates
- Gate-based structures
- Gate pointers and objects
- Wire and gate vectors

- Inheritance in Logic Structures
  - A generic gate definition
  - Gates to include timing
  - Building structures from objects

- Hierarchal Modeling of Digital Components
  - Wire functionalities
  - Gate functionalities
  - Polymorphic gate base
  - Virtual functions
  - Functions overwriting
  - Flip flop description hierarchal

Inheritance in Logic Functions



Accessible by gate classes that are inherited from gates

Different constructors for 2-input and 1-input gates and no initailization

InheritedLogicClassPrimitives.h

Timing activity functions for 1 and 2 input gates

Evl() is needed for each gate. Each gate instance use its own evl() function

All gates inherit from gate class

An inherited class that does not have its own evl() can depend on the base class

Inheritance in Logic Functions

inheritedLogicClassesFunctions.cpp | inheritedLogicClassesPrimitives.cpp | inheritedLogicClassesPrimitives.h

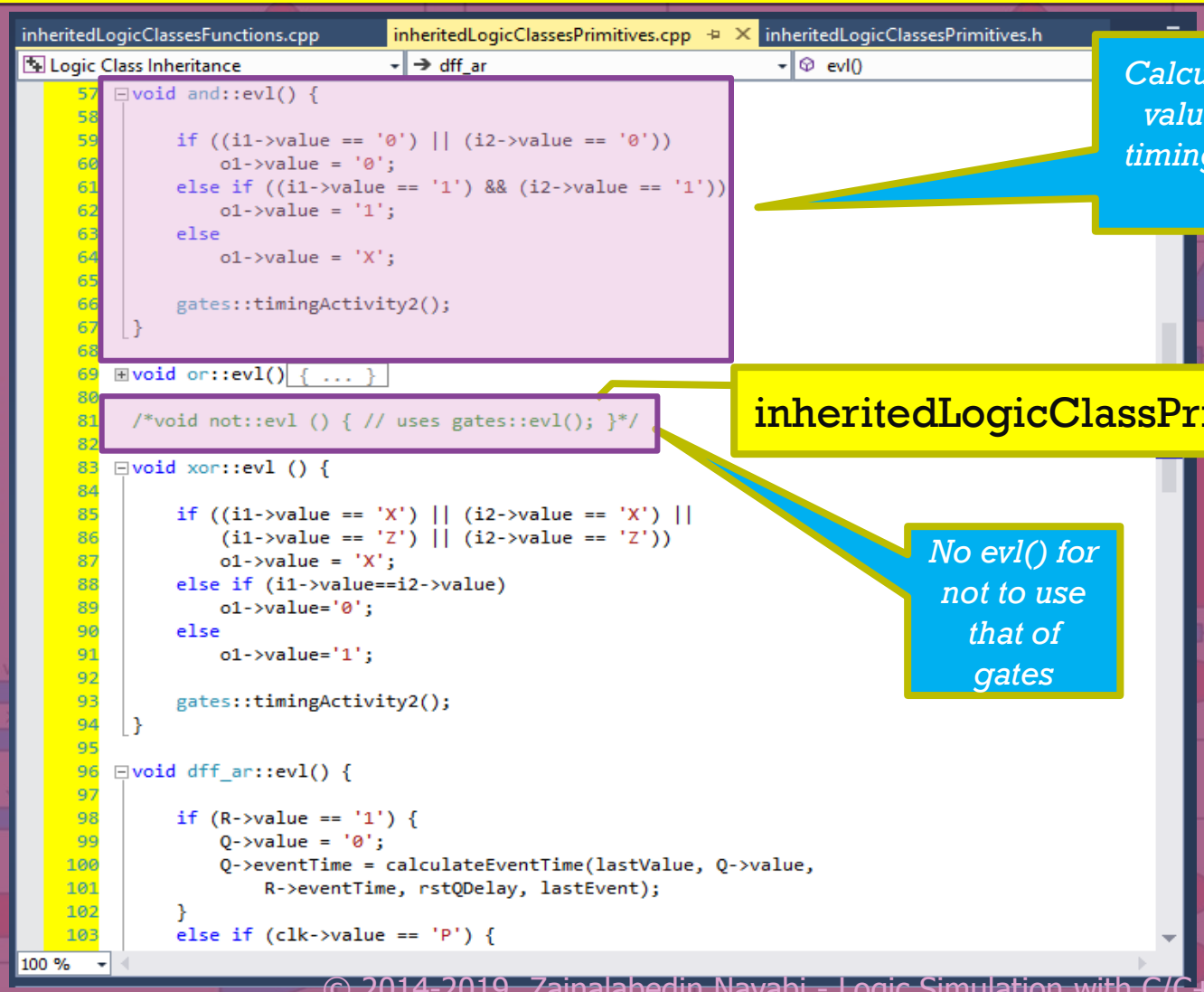
Logic Class Inheritance → xor → evl()

```
4   #define MAX(a,b) ((a>b)?a:b)
5
6   int calculateEventTime(char lastValue, char newValue,
7       int in1LastEvent, int in2LastEvent, int gateDelay, int lastEvent){
8
9       if (lastValue == newValue)
10          return lastEvent;
11      else
12          return gateDelay + MAX (in1LastEvent, in2LastEvent);
13  }
14
15  int calculateEventTime(char lastValue, char newValue,
16      int in1LastEvent, int gateDelay, int lastEvent) { ... }
23
24  void gates::evl() { // inverts its input 1
25
26      if (i1->value == '0')
27          o1->value = '1';
28      else if (i1->value == '1')
29          o1->value = '0';
30      else
31          o1->value = 'X';
32
33      gates::timingActivity1();
34  }
35  void gates::timingActivity2() {
36
37      o1->eventTime = calculateEventTime(lastValue, o1->value,
38          i1->eventTime, i2->eventTime, gateDelay, lastEvent);
39
40      o1->activityCount = i1->activityCount + i2->activityCount +
41          ((lastValue == o1->value) ? 0 : 1);
42
43      lastEvent = o1->eventTime;
44      lastValue = o1->value;
45  }
46  void gates::timingActivity1() { ... }
```

100 %

InheritedLogicClassPrimitives.cpp

Inheritance in Logic Functions



Calculate output value and call timing activity at gates

inheritedLogicClassPrimitives.cpp

No evl() for not to use that of gates

Structures from Inherited Gates

inheritedLogicClassesFunctions.cpp | inheritedLogicClassesPrimitives.cpp | inheritedLogicClassesPrimitives.h

Logic Class Inheritance | (Global Scope)

```
76  class fullAdder {
77      wire *i1, *i2, *i3, *o1, *o2;
78
79      // Declare necessary gate instances
80      xor *xor1;
81      xor *xor2;
82      and *and1;
83      and *and2;
84      or  *or1;
85
86      // fulladder Local wires
87      wire aL, bL, ciL;
88      wire coL, sumL;
89      wire axbL, abL, abcL;
90
91      public:
92          fullAdder(wire& a, wire& b, wire& ci, wire& co, wire& sum) :
93              i1(&a), i2(&b), i3(&ci), o1(&co), o2(&sum),
94              aL('X', 0), bL('X', 0), ciL('X', 0),
95              coL('X', 0), sumL('X', 0),
96              axbL('X', 0), abL('X', 0), abcL('X', 0) {
97
98          // Associate ports of the gates with the Local FA wires
99          xor1 = new xor(aL, bL, axbL, 5); // 5 is gate delay
100         xor2 = new xor(axbL, ciL, sumL, 5);
101         and1 = new and(aL, bL, abL, 3);
102         and2 = new and(axbL, ciL, abcL, 3);
103         or1 = new or(abL, abcL, coL, 3);
104         };
105         ~fullAdder();
106         void evl();
107 };
108
109 class halfAdder {
110     wire *i1, *i2, *o1, *o2;
111
112     // Declare necessary gate instances
```

100 %

inheritedLogicClassPrimitives.cpp

Full adder uses inherited gates. Wiring is done here.

Inheritance in Logic Structures

inheritedLogicClassesFunctions.cpp | inheritedLogicClassesPrimitives.cpp | inheritedLogicClassesPrimitives.h

Logic Class Inheritance → xor → evl()

```
116
117 void fullAdder::evl () {
118
119     // Via the FA pointers, read wire values that connect to
120     // the FA from outside, and assign them to FA Local wires
121     aL = *i1; bL = *i2; ciL = *i3;
122     and1->timingActivity1();
123     // Evaluate gates in the proper order
124     xor1->evl();
125     and1->evl();
126     and2->evl();
127     or1->evl();
128     xor2->evl();
129
130     // Take calculated local wire values and assign the values
131     // to the outside wires via pointers of FA
132     *o1 = coL; *o2 = sumL;
133 }
134
135 void halfAdder::evl () {
136
137     // Via the HA pointers, read wire values that connect to
138     // the HA from outside, and assign them to HA Local wires
139     aL = *i1; bL = *i2;
140
141     // Evaluate gates in the proper order
142     and1->evl();
143     xor1->evl();
144
145     // Take calculated local wire values and assign the values
146     // to the outside wires via pointers of FA
147     *o1 = coL; *o2 = sumL;
148 }
149
150 // --------------------- Vector Logics --------------------- //
151
152 wireV::wireV(string v, int d, int size) : eventTime(d), n(size) {
```

100 %

inheritedLogicClassPrimitives.cpp

Evl() of full adder order the gates

Inheritance in Logic Structures

inheritedLogicClassesFunctions.cpp | inheritedLogicClassesPrimitives.cpp | inheritedLogicClassesPrimitives.h

Logic Class Inheritance | (Global Scope)

```
3  int main ()
4  {
5      wire aW('0', 3), bW('1', 5), ciW('X', 0), coWF('X', 0), sumWF('X', 0),
6          coWH('X', 0), sumWH('X', 0);
7      wire dW('X', 4), eW('X', 4);
8
9      fullAdder *FA = new fullAdder(aW, bW, ciW, coWF, sumWF);
10     halfAdder *HA = new halfAdder(aW, bW, coWH, sumWH);
11     //not *NOT = new not(aW, dW, 5); // or use gates as below
12     gates *NOT = new not(aW, dW, 5); // "not" can do everything that "gates" can.
13
14     int ai;
15
16     do {
17         inpBit("Wire a", aW);
18         inpBit("Wire b", bW);
19         inpBit("Wire c", ciW);
20
21         FA->evl();
22         HA->evl();
23         NOT->evl();
24
25         outBit("FA - Carry", coWF);
26         outBit("       Sum", sumWF);
27
28         outBit("HA - Carry", coWH);
29         outBit("       Sum", sumWH);
30
31         outBit("NOT - Gate", dW);
32
33         cout << "\n" << "Continue? "; cin >> ai;
34
35     } while (ai>0);
36 }
37
38 /*int main ()
39 {
```

100 %

inheritedLogicClassFunctions.cpp

Logic Simulation with C/C++

- Containing Event Based Timing
  - To include in wires
  - To include in gates
- Gate-based structures
- Gate pointers and objects
- Wire and gate vectors

- Inheritance in Logic Structures
  - A generic gate definition
  - Gates to include timing
  - Building structures from objects

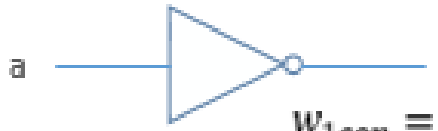
- Hierarchal Modeling of Digital Components
  - Wire functionalities
  - Gate functionalities
  - Polymorphic gate base
  - Virtual functions
  - Functions overwriting
  - Flip flop description hierarchal

Logic Testability Analysis

$$W_{1con} = a_{1con} * b_{1con}$$

$$W_{0con} = a_{0con} * b_{0con}$$

$$W_{1con} = a_{1con} + b_{1con} - a_{1con} * b_{1con}$$



$$w_{1con} = 1 - a_{1con}$$

Wire Functionality

polymorphismLogicClassesFunctions.h | polymorphismLogicClassesPrimitives.h*

Logic Class Polymorphism | (Global Scope)

```
2
3   class wire {
4   protected:
5       static int numberOfWires;
6   public:
7       char value;
8       int eventTime;
9       int activityCount = 0;
10      float controlability = 0.5;
11  public:
12      int wireIdentifier;
13      wire(char c, int d) : value(c), eventTime(d) {
14          wireIdentifier = numberOfWires;
15          numberOfWires++;
16      }
17      wire(){};
18      void put(char a, int d) { value = a; eventTime = d; }
19      void get(char& a, int& d) { a = value; d = eventTime; }
20      int activity() { return activityCount; }
21  };
22
23  class gates {
24  protected:
25      wire *i1, *i2, *o1;
26      int gateDelay, lastEvent;
27      char lastValue;
28
29      void timingActivity2();
30      void timingActivity1();
31      static int numberOfGates;
32  public:
33      int gateIdentifier;
34      float outputControlability = 1.0;
35      gates(wire& a, wire& w, int d) :
36          i1(&a), o1(&w), gateDelay(d) {
37          gateIdentifier = numberOfGates;
38          numberOfGates++;
```

100 %

Only a copy of it is generated for every instance of wire

PolymorphismLogicClassesPrimitives.h

Any new wire increments number of wires

Wire class has wire identifier and static number of wires

Polymorphic Gate Base

polymorphismLogicClassesFunctions.h | polymorphismLogicClassesPrimitives.h*

Logic Class Polymorphism | (Global Scope)

```
22
23  class gates {
24  protected:
25      wire *i1, *i2, *o1;
26      int gateDelay, lastEvent;
27      char lastValue;
28
29      void timingActivity2();
30      void timingActivity1();
31      static int numberOfGates;
32  public:
33      int gateIdentifier;
34      float outputControlability = 1.0;
35      gates(wire& a, wire& w, int d) :
36          i1(&a), o1(&w), gateDelay(d) {
37          gateIdentifier = numberOfGates;
38          numberOfGates++;
39      }
40      gates(wire& a, wire& b, wire& w, int d) :
41          i1(&a), i2(&b), o1(&w), gateDelay(d) {
42          gateIdentifier = numberOfGates;
43          numberOfGates++;
44      }
45      gates(){};
46      ~gates(){};
47      virtual void evl();
48      virtual void prob(){};
49  };
50
51  float getProb(gates*);
52
53  class and: public gates {
54  public:
55      and(wire& a, wire& b, wire& w, int d) : gates(a, b, w, d) {}
56      ~and();
57      void evl();
58      void prob();
```

100 %

PolymorphismLogicClassesPrimitives.h

Gates constructor assigns an id and increments the gate count

Virtual can be overwritten by classes that inherit from it. If not overwritten, the same evl() of gates will be used for an inherited class

Polymorphic Gate Base

polymorphismLogicClassesPrimitives.h | polymorphismLogic...ssesPrimitives.cpp

Logic Class Polymorphism | (Global Scope)

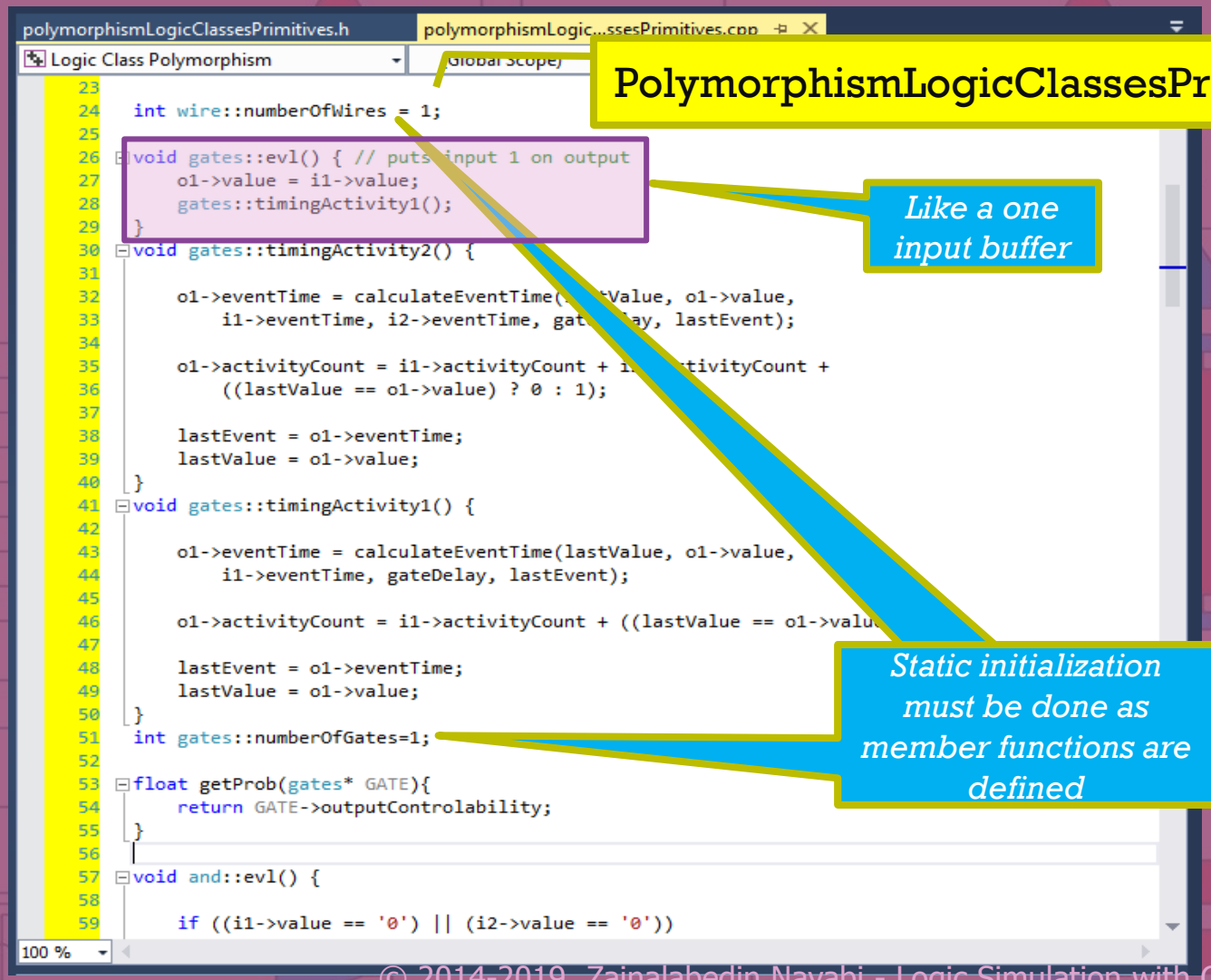
```
53  class and: public gates {
54  public:
55      and(wire& a, wire& b, wire& w, int d) : gates(a, b, w, d) {}
56      ~and();
57      void evl();
58      void prob();
59  };
60
61  class or: public gates {
62  public:
63      or(wire& a, wire& b, wire& w, int d) : gates(a, b, w, d) {}
64      ~or();
65      void evl();
66      void prob();
67  };
68
69  class not: public gates {
70  public:
71      not(wire& a, wire& w, int d) : gates(a, w, d) {}
72      ~not();
73      void evl();
74      void prob();
75  };
76
77  class xor: public gates {
78  public:
79      xor(wire& a, wire& b, wire& w, int d) : gates(a, b, w, d) {}
80      ~xor();
81      void evl();
82      void prob();
83  };
84
85  class flipflop {
86  protected:
87      wire *D, *clk, *rst, *cen, *Q;
88      int clkQDelay;
89      int rstQDelay;
```

100 %

PolymorphismLogicClassesPrimitives.h

Each gate just uses the constructor of gates and declares member functions to overwrite evl() and prob() of gates

Polymorphic Gate Base




```
23
24  int wire::numberOfWires = 1;
25
26  void gates::evl() { // puts input 1 on output
27      o1->value = i1->value;
28      gates::timingActivity1();
29  }
30  void gates::timingActivity2() {
31
32      o1->eventTime = calculateEventTime(lastValue, o1->value,
33          i1->eventTime, i2->eventTime, gateDelay, lastEvent);
34
35      o1->activityCount = i1->activityCount + i2->activityCount +
36          ((lastValue == o1->value) ? 0 : 1);
37
38      lastEvent = o1->eventTime;
39      lastValue = o1->value;
40  }
41  void gates::timingActivity1() {
42
43      o1->eventTime = calculateEventTime(lastValue, o1->value,
44          i1->eventTime, gateDelay, lastEvent);
45
46      o1->activityCount = i1->activityCount + ((lastValue == o1->value
47
48      lastEvent = o1->eventTime;
49      lastValue = o1->value;
50  }
51  int gates::numberOfGates=1;
52
53  float getProb(gates* GATE){
54      return GATE->outputControlability;
55  }
56
57  void and::evl() {
58
59      if ((i1->value == '0') || (i2->value == '0'))
```

Polymorphic Gate Base

polymorphismLogicClassesPrimitives.h | polymorphismLogic...ssesPrimitives.cpp

Logic Class Polymorphism | (Global Scope)

```
56
57  void and::evl() {
58
59      if ((i1->value == '0') || (i2->value == '0'))
60          o1->value = '0';
61      else if ((i1->value == '1') && (i2->value == '1'))
62          o1->value = '1';
63      else
64          o1->value = 'X';
65
66      gates::timingActivity2();
67  }
68  void and::prob() {
69      outputControlability = i1->controlability * i2->controlability;
70      o1->controlability = outputControlability;
71  }
72
73  void or::evl() { ... }
84  void or::prob() { ... }
89
90  void not::evl() { ... }
101 void not::prob() { ... }
106
107 void xor::evl () { ... }
119 void xor::prob() { ... }
124
125 int flipflop::numberOfFlipflops = 1;
126
127 void DFF::evl() {
128     char valueToLoad = '0';
129
130     if (!containsReset) valueToLoad = D->value;
131     else valueToLoad = (rst->value == '1') ? '0' : D->value;
132
133     if (clk->value == 'P') {
134         Q->value = valueToLoad;
135         Q->eventTime = calculateEventTime(lastValue, Q->value,
```

100 %

Redefine virtual functions of gate

PolymorphismLogicClassesPrimitives.cpp

DFF is inherited from flip flop

Polymorphic Gate Base

```
57  float evl(gates* GATE){
58      GATE->evl();
59      return GATE->outputControlability;
60  }
```

PolymorphismLogicClassesPrimitives.cpp

Polymorphic Gate Base

polymorphismLogic...ssesPrimitives.cpp | polymorphismLogic...ssesFunctions.cpp

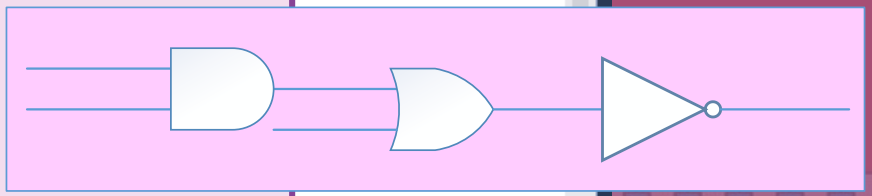
Logic Class Polymorphism | (Global Scope)

```
73
74  int main()
75  {
76      wire a('0', 3), b('1', 5), c('X', 0);
77      wire v('0', 3), w('0', 3), y('1', 5);
78
79      gates *NOT = new not(y, w, 5);
80      gates *AND = new and(a, b, v, 7);
81      gates *OR1 = new or(v, c, y, 6);
82
83      AND->prob();
84      OR1->prob();
85      NOT->prob();
86
87      int ai; int time = 0;
88
89      do {
90          inpBit("Wire a", a, time);
91          inpBit("Wire b", b, time);
92          inpBit("Wire c", c, time);
93
94          cout << evl(AND) << " :AND\n";
95          cout << evl(OR1) << " :OR1\n";
96          cout << evl(NOT) << " :NOT\n";
97
98          outBit("AOI  output: ", w);
99          cout << "AOI  output activity count: " << w.activity() << '\n';
100
101         time += 17;
102         cout << "\n" << "Continue? "; cin >> ai; cout << "\n";
103     } while (ai>0);
104 }
105
106
107 /*
108 int main()
109 {
```

100 %

Base-pointer type compatibility

PolymorphismLogicClassesFunctions.cpp



Polymorphic Gate Base

polymorphismLogicClassesPrimitives.h* | polymorphismLogic...ssesPrimitives.cpp

Logic Class Polymorphism | (Global Scope)

```
84
85  class flipflop {
86  protected:
87      wire *D, *clk, *rst, *cen, *Q;
88      int clkQDelay;
89      int rstQDelay;
90      int lastEvent; // last time output changed
91      char lastValue;
92      bool containsReset = false;
93      float clockControlability = 0.5;
94      static int numberOfFlipflops;
95  public:
96      int flipflopIdentifier;
97      float outputControlability = 1.0;
98      flipflop(wire& d, wire& c, wire& q, int dC) :
99          D(&d), clk(&c), Q(&q), clkQDelay(dC) {
100         flipflopIdentifier = numberOfFlipflops;
101         numberOfFlipflops++;
102     };
103     ~flipflop(){};
104     virtual void evl() = 0;
105     virtual void prob() = 0;
106     virtual void init(float, char) = 0;
107 };
108
109 class DFF : public flipflop {
110 public:
111     DFF(wire& d, wire& c, wire& q, int dC) : flipflop(d, c, q, dC)
112     { containsReset = false; };
113     ~DFF(){};
114     virtual void evl();
115     virtual void prob();
116     virtual void init(float, char);
117 };
118
119 class DFFsR : public DFF {
120 public:
```

100 %

PolymorphismLogicClassesPrimitives.h

Pure virtual functions

First_Level Derived flip-flop classes

Flip Flop Description Hierarchies

polymorphismLogicClassesPrimitives.h* | polymorphismLogic...ssesPrimitives.cpp

Logic Class Polymorphism | (Global Scope)

```
125  int flipflop::numberOfFlipflops = 1;
126
127  void DFF::evl() {
128      char valueToLoad = '0';
129
130      if (!containsReset) valueToLoad = D->value;
131      else valueToLoad = (rst->value == '1') ? '0' : D->value;
132
133      if (clk->value == 'P') {
134          Q->value = valueToLoad;
135          Q->eventTime = calculateEventTime(lastValue, Q->value,
136              clk->eventTime, clkQDelay, lastEvent);
137      }
138
139      Q->eventTime = calculateEventTime(lastValue, Q->value,
140          clk->eventTime, clkQDelay, lastEvent);
141
142      Q->activityCount = (D->activityCount + clk->activityCount) * 2 +
143          ((lastValue == Q->value) ? 0 : 3);
144
145      lastEvent = Q->eventTime;
146      lastValue = Q->value;
147  }
148  void DFF::prob(){
149      outputControlability = D->controlability * clockControlability;
150      Q->controlability = outputControlability;
151  }
152  void DFF::init(float clkCon, char iniOut) {
153      clockControlability = clkCon; Q->value = iniOut;
154  }
155
156  void DFFsR::prob(){
157      outputControlability = (D->controlability + rst->controlability -
158                              D->controlability * rst->controlability ) *
159                              clockControlability;
160      Q->controlability = outputControlability;
161  }
```

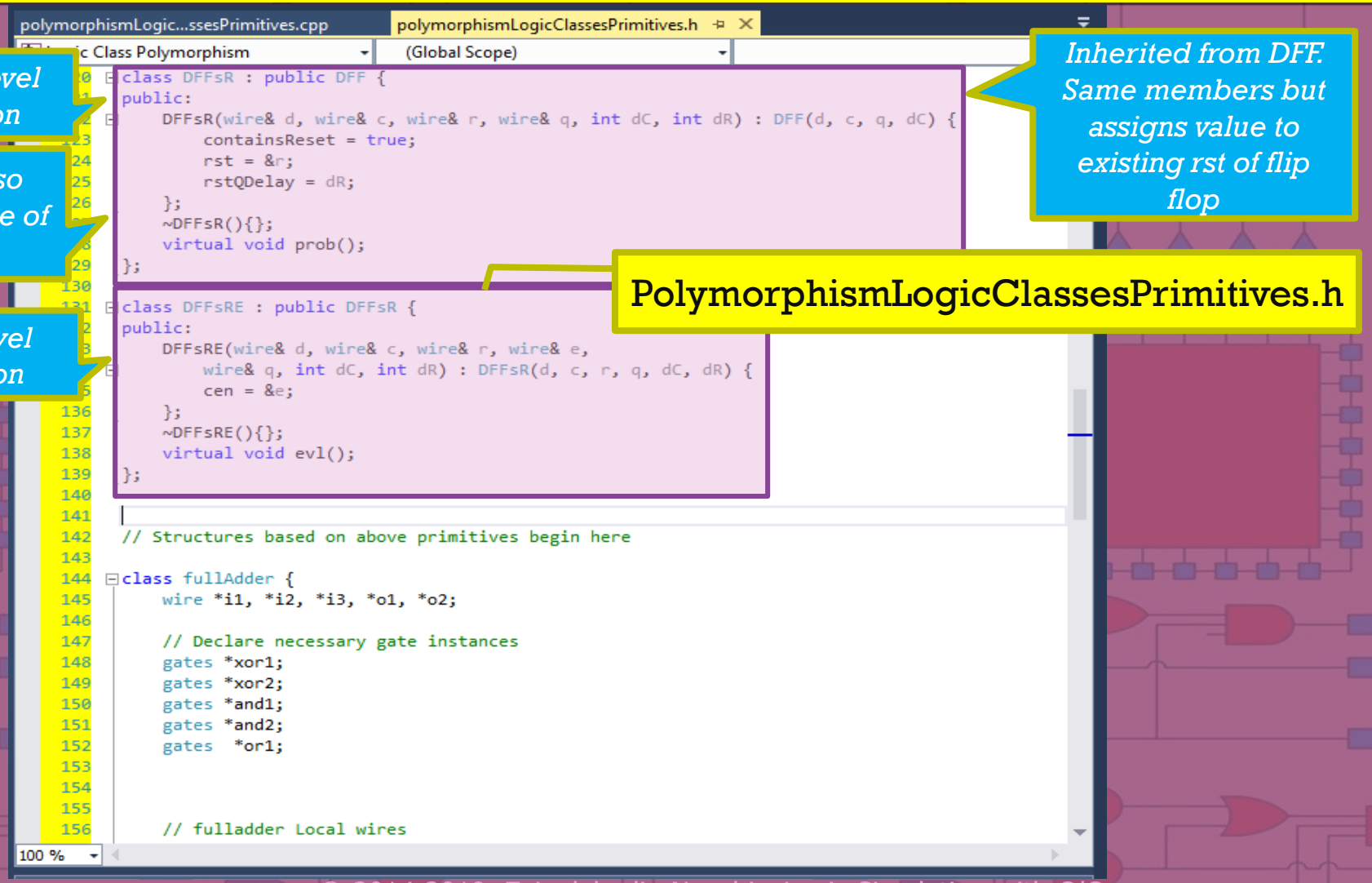
100 %

PolymorphismLogicClassesPrimitives.cpp

Basic DFF with synchronous reset

DFFsR is inherited from DFF

Flip Flop Description Hierarchies



Second Level Derivation

No evl(), so uses the one of DFF

Third Level derivation

Inherited from DFF. Same members but assigns value to existing rst of flip flop

PolymorphismLogicClassesPrimitives.h

Flip Flop Description Hierarchal

polymorphismLogicClassesPrimitives.h* | polymorphismLogic...ssesPrimitives.cpp

Logic Class Polymorphism | (Global Scope)

```
155
156 void DFFsR::prob(){
157     outputControlability = (D->controlability + rst->controlability -
158                             D->controlability * rst->controlability ) *
159                             clockControlability;
160     Q->controlability = outputControlability;
161 }
162
163 void DFFsRE::evl() {
164     if (en->value == '1') DFFsR::evl();
165 }
166
167  // Structures based on above primitives begin
168
169 void fullAdder::prob(){
170
171     // Calculate probabilities in the proper order
172     xor1->prob();
173     and1->prob();
174     and2->prob();
175     or1->prob();
176     xor2->prob();
177
178     o1Controlability = getProb(or1);
179     o2Controlability = getProb(xor2);
180 }
181 void fullAdder::evl () {
182
183     // Via the FA pointers, read wire values that connect to
184     // the FA from outside, and assign them to FA Local wires
185     aL = *i1; bL = *i2; ciL = *i3;
186
187     // Evaluate gates in the proper order
188     xor1->evl();
189     and1->evl();
190     and2->evl();
191     or1->evl();
```

100 %

PolymorphismLogicClassesPrimitives.cpp

DFFsRE calls DFFsR when value is one

Flip Flop Description Hierarchal

polymorphismLogic...ssesPrimitives.cpp | polymorphismLogicClassesUtilities.cpp

Logic Class Polymorphism | (Global Scope)

```
3  void inpBit(string wireName, wire& valtim) {
4      char value;
5      int time;
6      cout << "Enter value followed by @ time for " << wireName << ": ";
7      cin >> value; cin >> time;
8      valtim.put(value, time);
9  }
10
11 void inpBit(string wireName, wire& valtim, int t
12     char value;
13     cout << "For @ time " << time << ", enter logic value for " << wireName << ": ";
14     cin >> value;
15     valtim.put(value, time);
16 }
17
18 void outBit(string wireName, wire valtim) {
19     char value;
20     int time;
21     valtim.get(value, time);
22     cout << wireName << ": " << value << " @ " << time << "\n";
23 }
24
25 void inpBit(string wireName, wireV& valtim) {
26     string value;
27     int time;
28     cout << "Enter value followed by @ time for " << wireName << ": ";
29     cin >> value; cin >> time;
30     valtim.put(value, time);
31 }
32
33 void outBit(string wireName, wireV valtim) {
34     string value;
35     int time;
36     valtim.get(value, time);
37     cout << wireName << ": " << value << " @ " << time << "\n";
38 }
39
```

100 %

PolymorphismLogicClassesPrimitives.cpp

Flip Flop Description Hierarchal

polymorphismLogicClassesUtilities.cpp | polymorphismLogic...ssesFunctions.cpp

Logic Class Polymorphism | (Global Scope) | main()

```
3   int main()
4   {
5       wire a('0', 3), b('1', 5), c('X', 0), clk('X', 0), rst('X', 0),
6           en('X', 0),
7           Q1('X', 0), Q2('X', 0), Q3('X', 0);
8       wire v('0', 3), w('0', 3), y('1', 5);
9
10      flipflop *FF1 = new DFF(a, clk, Q1, 401);
11      flipflop *FF2 = new DFFsR(a, clk, rst, Q2, 502, 6);
12      flipflop *FF3 = new DFFsRE(a, clk, rst, en, Q3, 603, 7);
13      FF1->init(float(0.37), '1');
14      FF2->init(float(0.37), '1');
15      FF3->init(float(0.37), '1');
16
17      gates *NOT = new not(y, w, 5);
18      gates *AND = new and(a, b, v, 7);
19      gates *OR1 = new or(v, c, y, 6);
20
21      AND->prob();
22      OR1->prob();
23      NOT->prob();
24      FF1->prob();
25      FF2->prob();
26      FF3->prob();
27
28      cout << "AND gate Id: " << AND->gateIdentifier << '\n';
29      cout << "OR1 gate Id: " << OR1->gateIdentifier << '\n';
30      cout << "NOT gate Id: " << NOT->gateIdentifier << "\n\n";
31
32      cout << "DFF2 output 1-probability: " << FF2->outputControlability << '\n';
33      cout << "DFF3 output 1-probability: " << FF3->outputControlability << "\n\n";
34
35      cout << "AOI  output 1-probability: " << getProb(NOT) << '\n';
36      cout << "DFF1 output 1-probability: " << FF1->outputControlability << '\n';
37      cout << "DFF2 output 1-probability: " << FF2->outputControlability << '\n';
38      cout << "DFF3 output 1-probability: " << FF3->outputControlability << "\n\n";
39
```

100 %

PolymorphismLogicClassesFunctions.cpp

Pointer compatibility

Flip Flop Description Hierarchal

polymorphismLogicClassesUtilities.cpp | polymorphismLogic...ssesFunctions.cpp

Logic Class Polymorphism | (Global Scope)

```
39
40      int ai; int time = 0;
41
42      do {
43          inpBit("Wire a", a, time);
44          inpBit("Wire b", b, time);
45          inpBit("Wire c", c, time);
46          inpBit("Clock input", clk, time);
47          inpBit("Reset input", rst, time);
48          inpBit("Enable input", en, time);
49
50          AND->evl();
51          OR1->evl();
52          NOT->evl();
53          FF1->evl();
54          FF2->evl();
55          FF3->evl();
56
57          outBit("AOI  output: ", w);
58          outBit("DFF1 output: ", Q1);
59          outBit("DFF2 output: ", Q2);
60          outBit("DFF3 output: ", Q3);
61
62          cout << "AOI  output activity count: " << w.activity() << '\n';
63          cout << "DFF1 output activity count: " << Q1.activity() << '\n';
64          cout << "DFF2 output activity count: " << Q2.activity() << '\n';
65          cout << "DFF3 output activity count: " << Q3.activity() << "\n\n";
66
67          time += 17;
68          cout << "\n" << "Continue? "; cin >> ai; cout << "\n";
69
70      } while (ai>0);
71  }
72  /*
73  int main()
74  {
75      wire aW('0', 3), bW('1', 5), ciW('X', 0), coWF('X', 0), sumWF('X', 0),
```

100 %

PolymorphismLogicClassesFunctions.cpp

Conclusion

This chapter presented:

- Procedural Languages for Hardware Modeling
- Types and Operators for Logic Modeling
- Basic Logic Simulation
- Enhanced logic simulation with timing
- More Functions for Wires and Gates
- Inheritance in Logic Structures
- Hierarchal Modeling of Digital Components

Copyright and Acknowledgment

- Slides prepared by Hanieh Hashemi, ECE graduate student