オブジェクト指向プログラミング入門（その４）

加賀　赳寛

論理回路シミュレータを作ってみよう

オブジェクト指向プログラミングの雰囲気がわかってきたと思います。オブジェクト指向プログラミングのメリットを感じていただくために、今までの例題よりも、もう少し実用的な例題をやってみましょう。私が選んだ例題は論理回路シミュレータの製作です。
名前は LogicSim としました。

LogicSim は AND、OR などのロジック回路の動きをシミュレーションするものです。

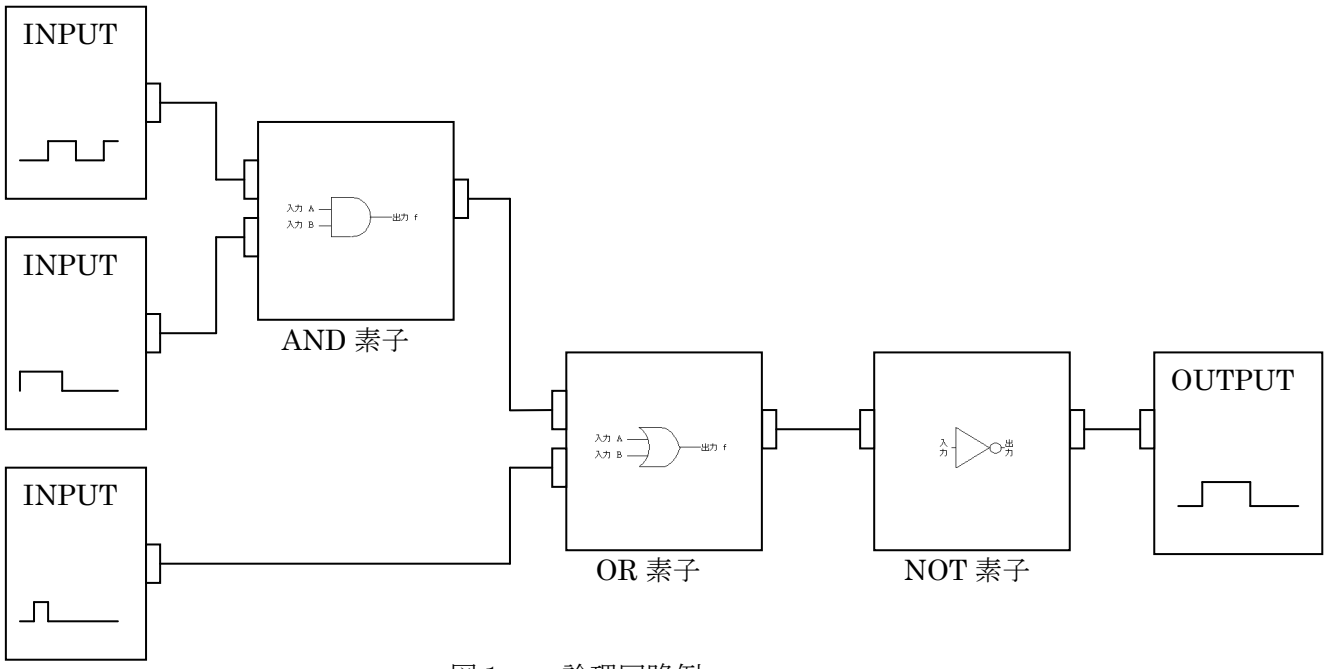

図１　　論理回路例

さて、どんなクラスが必要になるでしょうか？
すぐには思い浮かばないのは当たり前ですが、まずは下記のようなクラスが必要となりそうです。

AND, OR などのロジック素子とうまく設計できればロジック素子の共通部分

ロジック素子を結合する線分

入力信号（あるいは入力端子）に相当するもの

出力信号（あるいは出力端子）に相当するもの

ロジック素子の信号変化を蓄積、伝播するための機構部分

時刻を進め、シミュレーションを実施する本体

などがまず思い浮かびました。

私自身、論理シミュレータを作った経験はありませんでしたので、一部試行錯誤をして、概略の設計とプログラミングの過程を進めながら、最終的には下記のクラスとなりました。最初にぼんやりと考えていたクラスからかなり増えているのが分かります。どうやら最初から完璧なクラスを詳細に設計するのは一般的には無理がありそうです。このあたりがウォーターフォールモデルの設計方式ではうまく行きにくく、スパイラル方式の設計が推奨されるところではないかと感じます。

結果としては添付「論理シミュレータクラス図」に示すクラス構成になりました。
以下に論理シミュレータで用いたクラスの概要を説明します。

（1）クラス　Node

Logic 素子（AND,　OR, NOT など）の共通部分の処理を行う
インスタンスのデータとして、入力ピン配列@inPin, 出力ピン配列@outPin,
出力ピンに対応する出力値@out を持つ

メソッド　outConnect
　出力ピンに線分の from 側端子を接続する

メソッド　inConnect
　入力ピンに線分の to 側端子を接続する

メソッド　outConnectNode
　出力ピンに接続しているノードを求め、結果を@outNodeArray に格納する。
　なお、分岐ノードの場合は分岐先のノードを求める。
　求めたノードをイベント発生としてプライオリティ・キューに入れる。

メソッド　bunkiShori
　出力ピンから出ている線分の T 分岐処理を行う

メソッド　inConnectNode
　入力ピンに接続している前段ノードを求める

メソッド　bunkiShori2
　T 分岐の前段ノードを求める

メソッド　event
　時刻と処理 node をプライオリティ・キューに格納する

（2）クラス　AND

二入力 AND ゲートの処理を行う。

メソッド　connect(input1, input2, output1)
　出力の初期値は不定状態を示す２とする。

入力ピン１に Wire のインスタンス input1 を接続
input1 の to 側に AND ノードのピン１が接続
入力ピン２に Wire のインスタンス input２を接続
input２の to 側に AND ノードのピン２が接続
出力ピン１に Wire のインスタンス output1 を接続
output1 の from 側に AND ノードの出力ピン１が接続

メソッド　exec
入力ピンに接続しているノードを見つけ出す
そのノードの値に一つでも２を含めば出力は２
そのノードの値に一つでも０を含めば出力は０
それ以外なら出力は１
以前の出力と比較して、出力変化があれば outConnectNode(1)をコールする

（３）クラス　OR

二入力 OR ゲートの処理を行う。

メソッド　connect(input1, input2, output1)
出力の初期値は不定状態を示す２とする。
入力ピン１に Wire のインスタンス input1 を接続
input1 の to 側に OR ノードのピン１が接続
入力ピン２に Wire のインスタンス input２を接続
input２の to 側に OR ノードのピン２が接続
出力ピン１に Wire のインスタンス output1 を接続
output1 の from 側に OR ノードの出力ピン１が接続

メソッド　exec
入力ピンに接続しているノードを見つけ出す
そのノードの値に一つでも２を含めば出力は２
そのノードの値に一つでも 1 を含めば出力は 1
それ以外なら出力は 0
以前の出力と比較して、出力変化があれば outConnectNode(1)をコールする

（4）クラス　NOT

一入力 NOT ゲートの処理を行う。

メソッド　connect(input1, output1)
出力の初期値は不定状態を示す２とする。
入力ピン１に Wire のインスタンス input1 を接続
input1 の to 側に NOT ノードのピン１が接続
出力ピン１に Wire のインスタンス output1 を接続
output1 の from 側に NOT ノードの出力ピン１が接続

メソッド　exec
入力を反転する
ただし入力が２（不定）の場合は出力は２となる
以前の出力と比較して、出力変化があれば outConnectNode(1)をコールする

（5）クラス　INPUT

INPUT 素子　入力信号を発生する処理を行う。

メソッド　connect(output1)
出力の初期値は不定状態を示す２とする。
出力ピン１に Wire のインスタンス output1 を接続
output1 の from 側に INPUT ノードの出力ピン１が接続

メソッド　exec(vale)
入力信号 value を出力端子に出す
以前の出力と比較して、出力変化があれば outConnectNode(1)をコールする

（6）クラス　OUTPUT

OUTPUT 素子　この素子に接続している信号を記録するために用いる。

メソッド　connect(input1, name)
入力ピン１に Wire のインスタンス input1 を接続

input1 の to 側に OUTPUT ノードの入力ピン１が接続
記録時に用いる名称 name をセットする

メソッド　exec
入力ピン１に接続された信号の値を時刻と名称とともに記録する。
（OutputEvent を生成し、Report.ary へ push する）

（７）クラス　OutputEvent

時刻、名称、値を持つ記録用の項目

（８）クラス　Report

記録用の行列 Report.ary を持つ

（９）クラス　Wire

一本の線である。from 側端子と to 側端子がある。

メソッド　connectFrom(node, pinNo)
from 側端子と node 素子（例えば AND 素子）の出力ピン pinNo を接続する

メソッド　connectTo(node, pinNo)
to 側端子と node 素子（例えば AND 素子）の入力ピン pinNo を接続する

（１０）クラス　Bunki

T 分岐の処理を行う。　T 分岐は一入力を二出力へ分岐する。

メソッド　outConnect(pinNo, edge)
T 分岐の出力ピン pinNo に線 edge の from 側端子を接続する

メソッド　inConnect(pinNo, edge)
T 分岐の入力ピン pinNo に線 edge の to 側端子を接続する

（１１）クラス　EventItem

Pque にエンキューするイベントであり、index と content からなる

（１２）クラス　InputEvent

INPUT 素子用のイベントであり、index, value, inputnode を持つ
Index は時刻、value は入力信号の値、inputnode は対応する INPUT 素子であり、このイベントを受けて INPUT 素子が信号を発生することになる。

（１３）クラス　Pque

プライオリティ・キューをヒープ構造を使って実現している

insert メソッド：　挿入
　ヒープにデータを追加するときは、まず木構造の最後に追加する
　このままではヒープ構造の条件を満たさないので、親と比較して
　この方が小さければ入れ替える。この操作を根にたどり着くか、
　または親の方が小さくなるまで繰り返す。

メソッド　lookmin
　先頭の要素（最小値）を見る。見るだけでデキューはしない。

deletemin メソッド：
先頭の要素（最小値）を取り除き、返す。
ヒープから最小のデータを取り出す場合は、その根のデータを
取り出せばよい。しかし、これだけでは根がなくなってヒープ
構造がなくなって壊れてしまう。そこで、ヒープの最後のデータ
を根に異動させる。さらに、ヒープ構造の条件を満たすために、
子の小さい方のデータと比較して親の方が大きければ入れ替える。
これを最後まで繰り返す。

（１４）クラス　LogicSim

プライオリティ・キューよりイベントを取り出す

INPUT 素子用のイベントならば、クラス INPUT の exec(value)を実行する
その他のイベントならば、対応する各クラス(AND, OR, NOTnado)の exec を実行する

なお、クラス Pque はプライオリティ・キューでありヒープ構造を使用しました。
このキューに入れると必ず小さいもの（今回の場合、インデックスすなわち時刻が古いもの）からデキューされます。ちょっとわかりにくいかも知れませんので特別に解説用の説明を添付しました。

また、論理 SIM の全体のコードについても添付　logicsim.rb　に示しました。

添付の回路シミュレーション例に対する実行結果を次ページに示していますので、コードとともに見てプログラムの動作を理解する参考にしてください。

実行結果

```
#<LogicSim:0x100c3bb8 @pq=#<PQue:0x100c3ac8 @size=0, @heap=[]>>
  que size up  1
  que size up  2
  que size up  3
  que size up  4
  que size up  5
  que size up  6
  que size up  7

----------run------------------------------

event time:  0  sim time:  0
  deque
  que size down  6
  time: 0  InputEvent
  INPUT value 0
  event enque  0  AND
  que size up  7
event time:  0  sim time:  0
  deque
  que size down  6
  time: 0  InputEvent
  INPUT value 0
  event enque  0  AND
  que size up  7
event time:  0  sim time:  0
  deque
  que size down  6
  time: 0  InputEvent
  INPUT value 0
  event enque  0  OR
  que size up  7
event time:  0  sim time:  0
```

```
  deque
  que size down  6
  time: 0  node:  AND
  AND output 0
  event enque  0  OR
  que size up  7
event time:  0  sim time:  0
  deque
  que size down  6
  time: 0  node:  AND
event time:  0  sim time:  0
  deque
  que size down  5
  time: 0  node:  OR
  OR output 0
  event enque  0  NOT
  que size up  6
event time:  0  sim time:  0
  deque
  que size down  5
  time: 0  node:  OR
event time:  0  sim time:  0
  deque
  que size down  4
  time: 0  node:  NOT
  NOT output 1
  event enque  0  OUTPUT
  que size up  5
event time:  0  sim time:  0
  deque
  que size down  4
  time: 0  node:  OUTPUT
event time:  10  sim time:  0
event time:  10  sim time:  1
event time:  10  sim time:  2
event time:  10  sim time:  3
```

```
event time:  10  sim time:  4
event time:  10  sim time:  5
event time:  10  sim time:  6
event time:  10  sim time:  7
event time:  10  sim time:  8
event time:  10  sim time:  9
event time:  10  sim time:  10
  deque
  que size down  3
  time: 10  InputEvent
  INPUT value 1
  event enque  10  OR
  que size up  4
event time:  10  sim time:  10
  deque
  que size down  3
  time: 10  node:  OR
  OR output 1
  event enque  10  NOT
  que size up  4
event time:  10  sim time:  10
  deque
  que size down  3
  time: 10  node:  NOT
  NOT output 0
  event enque  10  OUTPUT
  que size up  4
event time:  10  sim time:  10
  deque
  que size down  3
  time: 10  node:  OUTPUT
event time:  25  sim time:  10
event time:  25  sim time:  11
event time:  25  sim time:  12
event time:  25  sim time:  13
event time:  25  sim time:  14
```

```
event time:  25  sim time:  15
event time:  25  sim time:  16
event time:  25  sim time:  17
event time:  25  sim time:  18
event time:  25  sim time:  19
event time:  25  sim time:  20
event time:  25  sim time:  21
event time:  25  sim time:  22
event time:  25  sim time:  23
event time:  25  sim time:  24
event time:  25  sim time:  25
  deque
  que size down  2
  time: 25  InputEvent
  INPUT value 0
  event enque  25  OR
  que size up  3
event time:  25  sim time:  25
  deque
  que size down  2
  time: 25  node:  OR
  OR output 0
  event enque  25  NOT
  que size up  3
event time:  25  sim time:  25
  deque
  que size down  2
  time: 25  node:  NOT
  NOT output 1
  event enque  25  OUTPUT
  que size up  3
event time:  25  sim time:  25
  deque
  que size down  2
  time: 25  node:  OUTPUT
event time:  30  sim time:  25
```

```
event time:  30  sim time:  26
event time:  30  sim time:  27
event time:  30  sim time:  28
event time:  30  sim time:  29
event time:  30  sim time:  30
  deque
  que size down  1
  time: 30  InputEvent
  INPUT value 1
  event enque  30  AND
  que size up  2
event time:  30  sim time:  30
  deque
  que size down  1
  time: 30  node:  AND
event time:  35  sim time:  30
event time:  35  sim time:  31
event time:  35  sim time:  32
event time:  35  sim time:  33
event time:  35  sim time:  34
event time:  35  sim time:  35
  deque
  que size down  0
  time: 35  InputEvent
  INPUT value 1
  event enque  35  AND
  que size up  1
event time:  35  sim time:  35
  deque
  que size down  0
  time: 35  node:  AND
  AND output 1
  event enque  35  OR
  que size up  1
event time:  35  sim time:  35
  deque
```

```
  que size down  0
  time: 35  node:  OR
  OR output 1
  event enque  35  NOT
  que size up  1
event time:  35  sim time:  35
  deque
  que size down  0
  time: 35  node:  NOT
  NOT output 0
  event enque  35  OUTPUT
  que size up  1
event time:  35  sim time:  35
  deque
  que size down  0
  time: 35  node:  OUTPUT
---------------REPORT-------------------------
0  OUTPUT1  1
10  OUTPUT1  0
25  OUTPUT1  1
35  OUTPUT1  0
```