1　はじめに

トランジスタを用いた双安定マルチバイブレータを設計・製作し，外部からトリガーパルスを周期的に入力したときの各部の波形を測定して，回路の動作を明確に理解することを目的とする．

2　課題 1(組合せ回路の設計)

セルフバイアス型双安定マルチバイブレータの回路を図 1 に示す.

2.1　半加算器

2.2　全加算器

2.3　4 ビット並列加算器

2.3.1　課題内容

2.3.2　設計結果

2.3.3　実機検証

2.3.4　シミュレーション

本実験においては，テストベンチでカバーしなければならない各 PIN への入力・出力の組み合わせが 2,560 ケースに上った．そこで，すべてのケースをカバーするとともに，理論値との比較がコンピュータで行えるよう，テストベンチ並びに理論値データの自動生成を行う Python スクリプトを作成した．また，シミュレーション結果の波形を目視で検証するコストを削減するため，テストベンチコードが「入力が入ってから 3ns 経過地点での各出力値」を理論値データと同じフォーマットで ModelSim の Transcript に出力するように設計した.

付録に，テストベンチを生成したスクリプト，生成されたテストベンチコード，生成された理論値データ，シミュレーション結果の波形を記載する.

理論値を算出するプログラム部分のみを抜粋して以下に掲載する.

Listing 1: hoge

```
1
2 ##############################
3 # statics
4 ...
5 ARRAYSTR = ["Theoretrical Value", "A3", "A2", "A1", "A0", "B3", "B2",
      "B1", "B0", "C3", "S3", "S2", "S1", "S0"]
6 ...
7 ##############################
8 # Calucurating Theoretrical Value
9 for pattern in patternList:
```

```
10      A = int("".join(map(str, pattern[0:4])),2)
11      B = int("".join(map(str, pattern[4:8])),2)
12      if LOG_LEVEL >= 2:
13          print("A = " + str(A) + " / A = " + str(pattern[0:4]) + " / B
                = " + str(B) + " / B = " + str(pattern[4:8]), end="")
14      pattern.insert(0, A + B)
15      if LOG_LEVEL >= 2:
16          print(" / Cal = " + str(A+B))
17  ...
18  ############################
19  # Generating Test Code
20  resultFile = open(RESULT_PATH, mode="w")
21  for i in patternList:
22      if LOG_LEVEL >= 1:
23          for p in range(1,9):
24              print(ARRAYSTR[p] + ":[\'" + str(i[p]) + "\'] ", file=
                    resultFile, end="")
25          caledResVal = str(bin(i[0])).replace("b","").zfill(5)
26          if len(caledResVal) > 5:
27              caledResVal = caledResVal[1:6]
28          print(caledResVal)
29          for q in range (9,14):
30              print(ARRAYSTR[q] + ":[\'" + str(caledResVal[q-9:q-8]) +
                    "\'] ", file=resultFile, end="")
31
32          print("\n", file=resultFile, end="")
33          print("A = " + str(int("".join(map(str, i[1:5])),2)) + " / A =
                " + str(i[1:5]) + " / B = " + str(int("".join(map(str, i
                [5:9])),2)) + " / B = " + str(i[5:9]) + " / cal = " + str
                (i[0]))
```

結果, 理論値データである”fourbitcounter_tb.result”と, シミュレーション結果である”simulatedResult”を入手した. これを Linux にデフォルトで実装されている diff コマンドにて差分確認したところ, 差分は見つからなかった. これにより, 正常に設計されていることが確認できた.

Listing 2: diff コマンドの実行結果

```
1  [root@XXX result]# diff fourbitcounter_tb.result simulatedResult
2  [root@XXX result]#
```