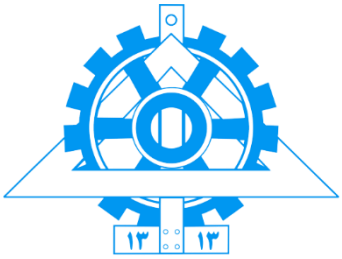
باسمه‌ی تعالی



Object-Oriented Modeling of Electronic Circuits

Computer Assignment 3

RTL Design and SystemC Simulation

Student Name: Moein Maleki
Student Number: 810197591

Electrical and Computer Engineering Department
Spring 1401

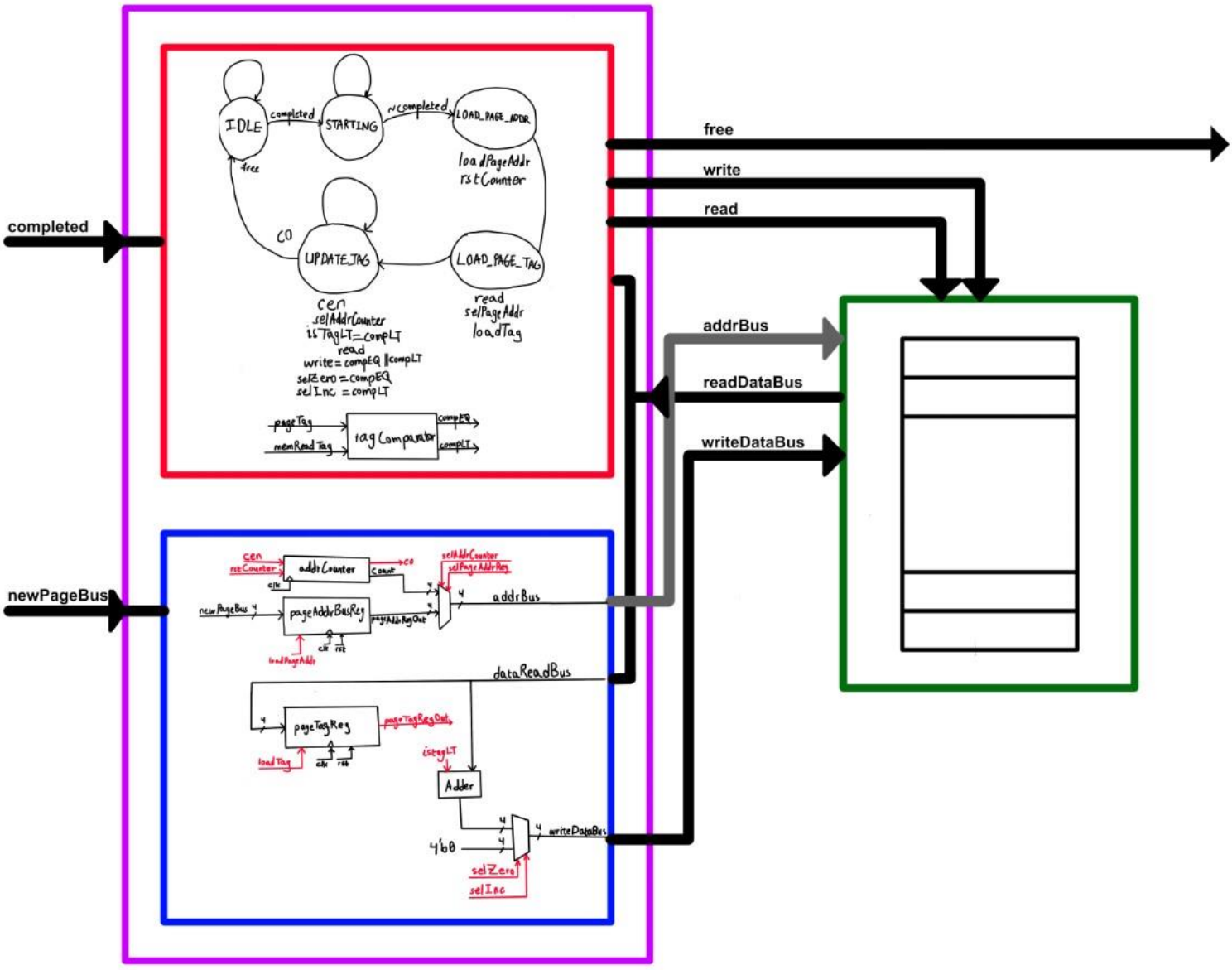
Table of Contents:

Deliverables:

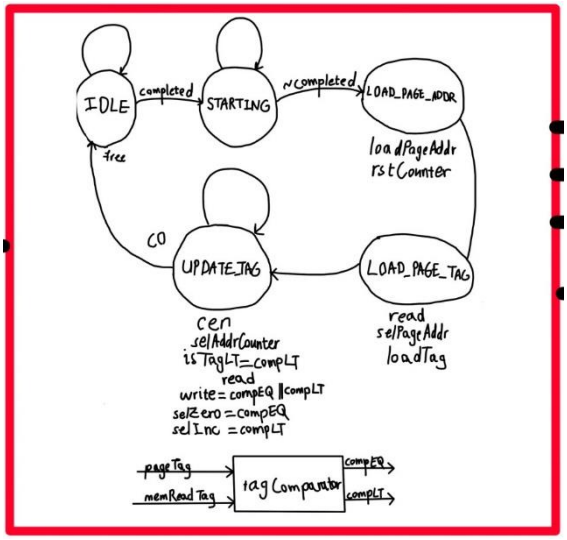
A)-1 Circuit Diagram, Datapath, and State diagram:

Solution 1- LRU Updater

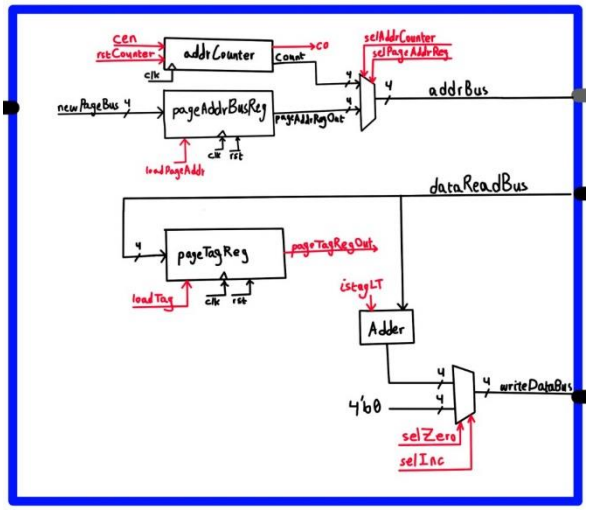
Full Circuit Diagram



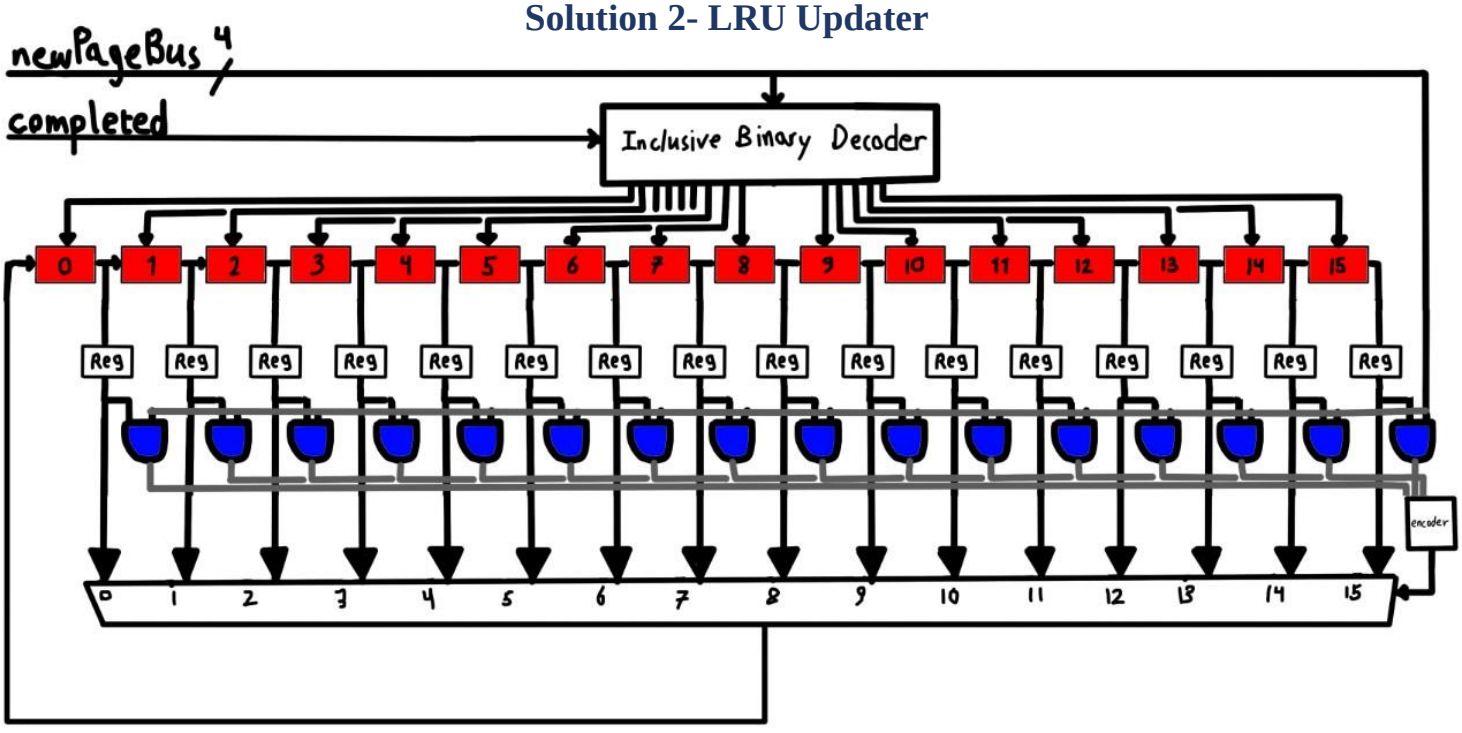
Controller and State diagram:



Datapath diagram:



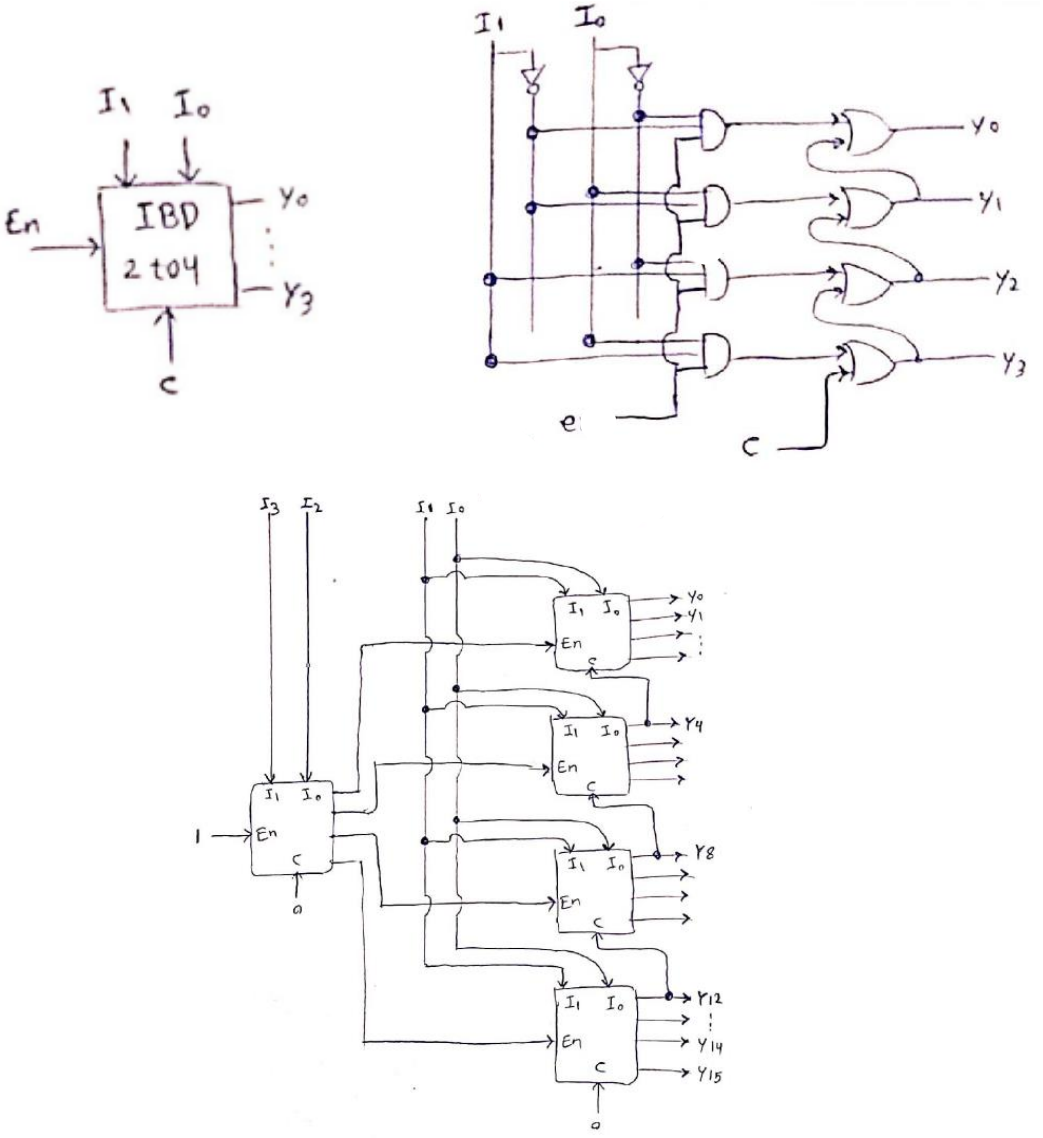
Solution 2- LRU Updater



Inclusive Binary Decoder:

این مدار، در واقع یک decoder عادی است، تنها تفاوتی که با یک decoder عادی دارد اینست که به جای آنکه فقط خطِ خروجیِ شماره‌ی n ام را Drive کند، تمامی خطوط برابر یا کوچک تر از شماره‌ی n را Drive میکند. یعنی اگر این مدار، مقدار باینری معادل ۹ را دریافت کند، خروجی این مدار، یک شدنِ ۱۰ عدد از خطوط خروجی، از خط شماره‌ی صفر تا خط شماره‌ی ۹ است. این مدار یک ورودی Enable نیز دارد که در طراحی بالا به Completed وصل شده است.

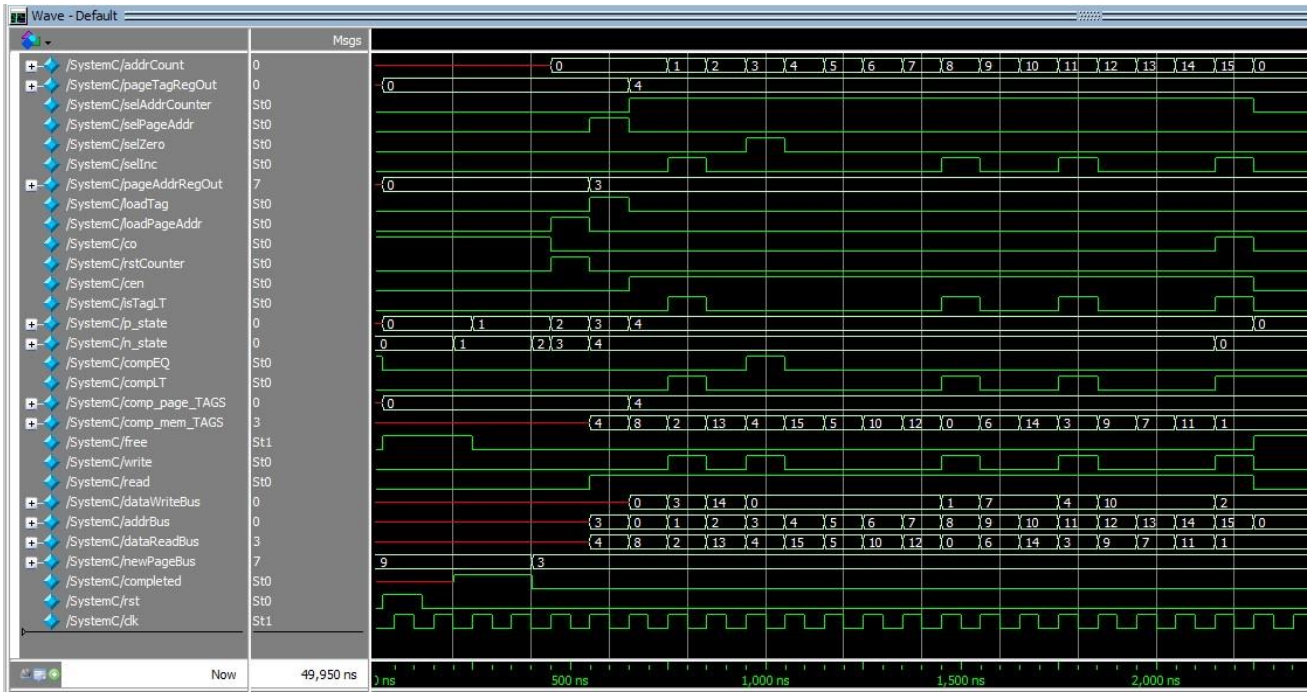
در ادامه مدار Cascadable یک 2-bit IBD را مشاهده میکنیم. و در ادامه نحوه‌ی Cascade شدن این مدار، برای ساخته شدن یک 4-bit IBD را مشاهده میکنیم.



A)-2 Images Of Testbenches Ran For The Two Designs

Solution 1- RTL - LRU Updater

Test One – Accessing Page Number 9



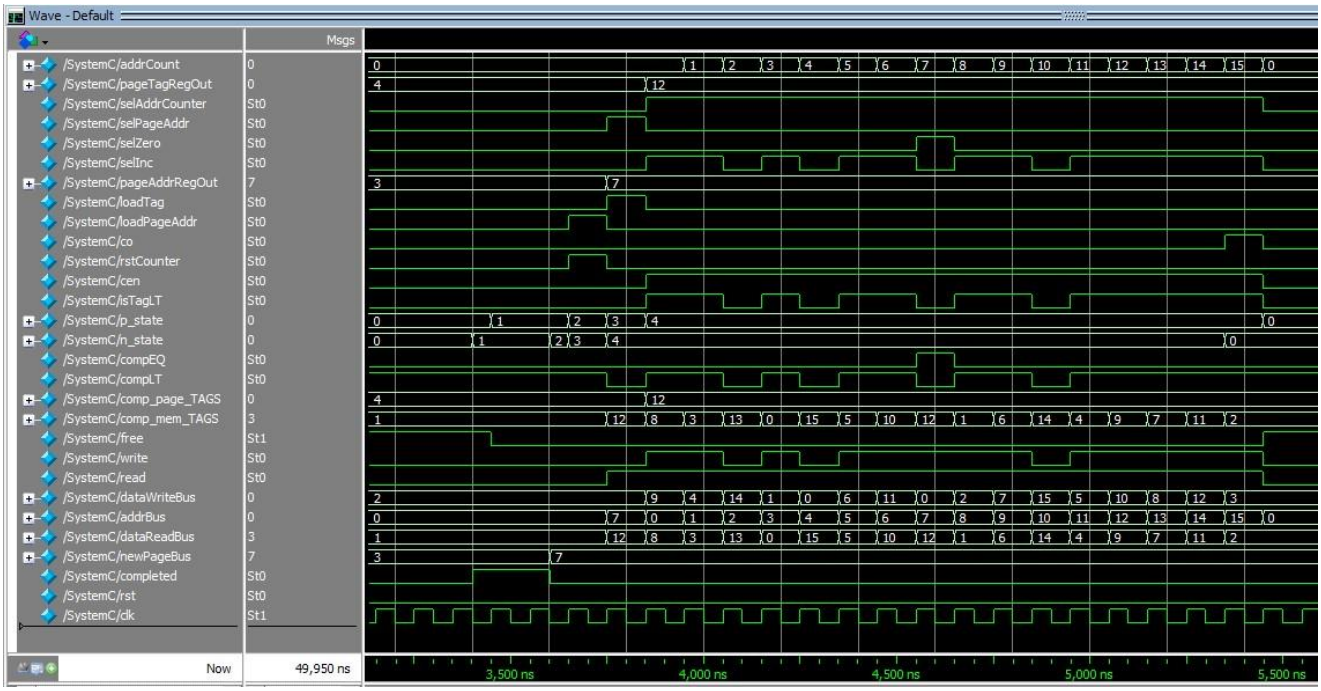
Memory State Before Running the Test:

```
0:      1000
1:      0010
2:      1101
3:      0100
4:      1111
5:      0101
6:      1010
7:      1100
8:      0000
9:      0110
10:     1110
11:     0011
12:     1001
13:     0111
14:     1011
15:     0001
```

Memory State After Running the Test:

```
0:      1000
1:      0011
2:      1101
3:      0000
4:      1111
5:      0101
6:      1010
7:      1100
8:      0001
9:      0110
10:     1110
11:     0100
12:     1001
13:     0111
14:     1011
15:     0010
```

Test Two – Accessing Page Number 3



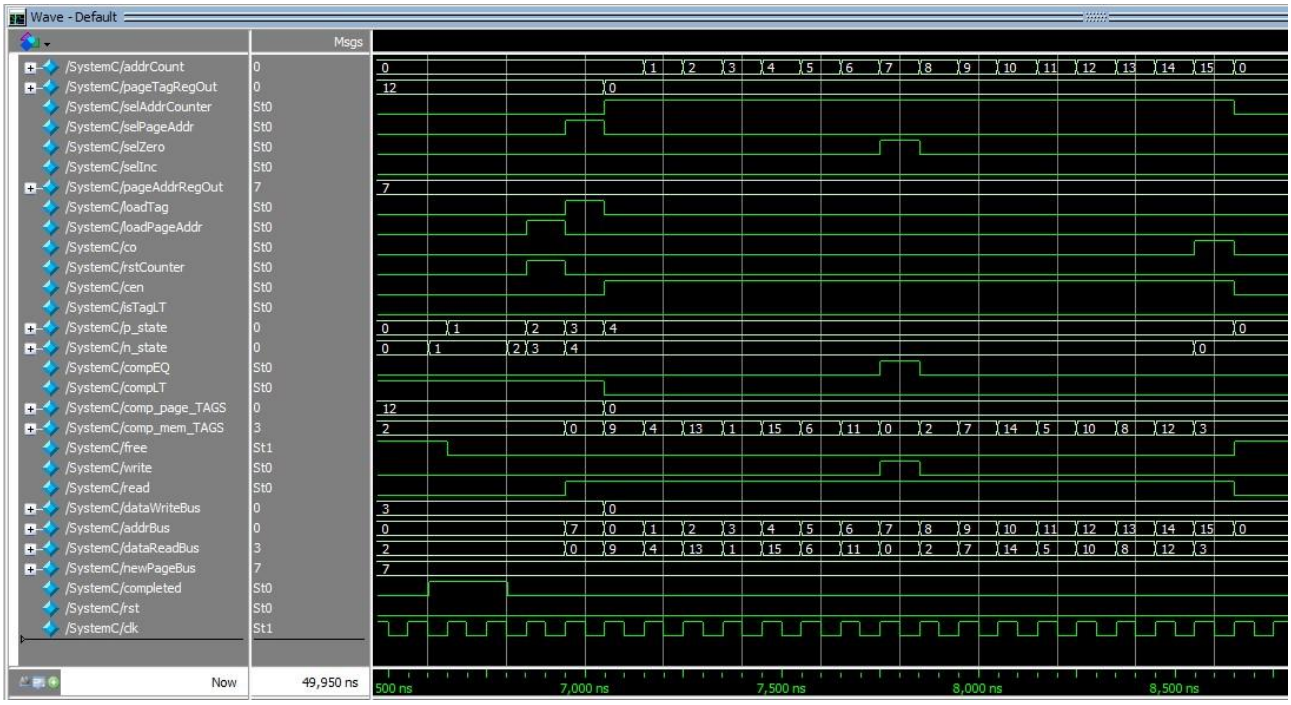
Memory State Before Running the Test:

```
0:      1000
1:      0011
2:      1101
3:      0000
4:      1111
5:      0101
6:      1010
7:      1100
8:      0001
9:      0110
10:     1110
11:     0100
12:     1001
13:     0111
14:     1011
15:     0010
```

Memory State After Running the Test:

```
0:      1001
1:      0100
2:      1101
3:      0001
4:      1111
5:      0110
6:      1011
7:      0000
8:      0010
9:      0111
10:     1110
11:     0101
12:     1010
13:     1000
14:     1100
15:     0011
```

Test Three – Accessing Page Number 7



Memory State Before Running the Test:

```
0:      1001
1:      0100
2:      1101
3:      0001
4:      1111
5:      0110
6:      1011
7:      0000
8:      0010
9:      0111
10:     1110
11:     0101
12:     1010
13:     1000
14:     1100
15:     0011
```

Memory State After Running the Test:

```
0:      1001
1:      0100
2:      1101
3:      0001
4:      1111
5:      0110
6:      1011
7:      0000
8:      0010
9:      0111
10:     1110
11:     0101
12:     1010
13:     1000
14:     1100
15:     0011
```

Solution 1- Functional - LRU Updater

Test One – Accessing Page Number 9

Memory State Before Running the Test:

```
0:      1000
1:      0010
2:      1101
3:      0100
4:      1111
5:      0101
6:      1010
7:      1100
8:      0000
9:      0110
10:     1110
11:     0011
12:     1001
13:     0111
14:     1011
15:     0001
```

Memory State After Running the Test:

```
0:      1000
1:      0011
2:      1101
3:      0101
4:      1111
5:      0110
6:      1010
7:      1100
8:      0001
9:      0000
10:     1110
11:     0100
12:     1001
13:     0111
14:     1011
15:     0010
```

Test Two – Accessing Page Number 3

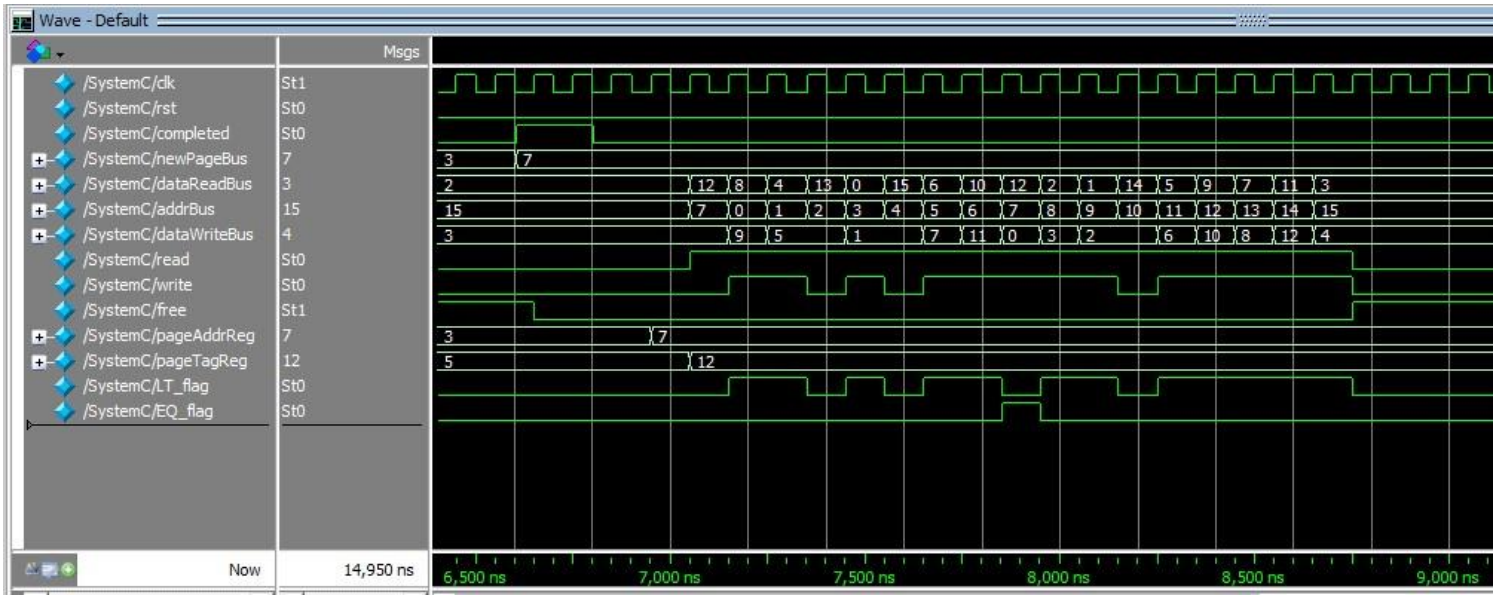
Memory State Before Running the Test:

```
0:      1000
1:      0011
2:      1101
3:      0101
4:      1111
5:      0110
6:      1010
7:      1100
8:      0001
9:      0000
10:     1110
11:     0100
12:     1001
13:     0111
14:     1011
15:     0010
```

Memory State After Running the Test:

```
0:      1000
1:      0100
2:      1101
3:      0000
4:      1111
5:      0110
6:      1010
7:      1100
8:      0010
9:      0001
10:     1110
11:     0101
12:     1001
13:     0111
14:     1011
15:     0011
```

Test Three – Accessing Page Number 7



Memory State Before Running the Test:

```
0:      1000
1:      0100
2:      1101
3:      0000
4:      1111
5:      0110
6:      1010
7:      1100
8:      0010
9:      0001
10:     1110
11:     0101
12:     1001
13:     0111
14:     1011
15:     0011
```

Memory State After Running the Test:

```
0:      1001
1:      0101
2:      1101
3:      0001
4:      1111
5:      0111
6:      1011
7:      0000
8:      0011
9:      0010
10:     1110
11:     0110
12:     1010
13:     1000
14:     1100
15:     0100
```

Comparing the RTL and Functional Design Waveforms:

تشابه این دو Waveform اینست که در نهایت هر دو یک خروجی را نشان میدهند. پس از اجرای تست ها، میتوان مشاهده کرد که وضعیت حافظه یکسان است و سیگنال های اصلی نیز دقیقا یک نوع رفتار را از خود نشان میدهند و هر دو به از نظر Cycle-Accurate بودن، به طراحی خود وفادار هستند.

تفاوت این دو پیاده سازی، که در Waveform میتوان آنرا به وضوح مشاهده کرد، تعداد اندک سیگنال های داخلی است که میتوان با آنها این طراحی را Debug کرد. در طراحی RTL، پارامتر P_state یا وضعیت کنونی کنترلر طراحی، میتواند وضعیت مدار را در همه‌ی زمان ها نشان دهد و براحتی وضعیت تمامی سیگنال ها را تعیین کرد و اگر بخشی از مدار اشتباه طراحی شده باشد، میتوان آن قسمت از مدار را با تمرکز بیشتری بررسی کرد. ولی در پیاده سازی Functional همچنین پارامتری وجود ندارد و برای Debug کردن طراحی باید به دقت و خط به خط کد طراحی را بررسی کرد. همچنین در این نوع پیاده سازی نیاز است در سطحی قبلا نیاز نبود، به نحوه‌ی پیاده‌سازی سیمولیشن در SystemC آگاه بود. در پیاده سازی بنده، میتوان مشاهده کرد که گاهی نیاز است تاخیر مصنوعی گذاشت تا مدار حافظه بتواند به درستی با مدار LRU_Updater ارتباط برقرار کند.