ماژول 5

در این ماژول باید به صورت hierarchical سیستمی طراحی کنیم که در نهایت تعداد صفر ها و یک های موجود در new capacity را بشمارد و به ما خروجی بدهد.
به صورت کلی می دانیم اگر تعداد 1 ها در new capacity بشماریم و مطمئن شویم که empty به درستی به دست آمده است می توانیم برای parked عدد 8 را از empty کم کنیم و به جواب برسیم.
پس ابتدا به دنبال پیدا کردن راهی hierarchical می رویم که تعداد 1 های موجود در یک عدد 8 بیتی را بشماریم:
اگر یک عدد 8 بیتی داشته باشیم می توانیم به سه دسته ی زیر بیت های ان را تقسیم کنیم:

1. بیت های شماره ی 0,1,2
2. بیت های شماره ی 3,4,5
3. بیت های شماره ی 6,7

حال اگر برای گروه های شماره ی 1,2 یک full adder داشته باشیم و برای گروه 3 از یک half adder استفاده کنیم هر گروه به ترتیب خروجی های زیر را تولید می کند:

1. result_1 , carry_1
2. result_2 , carry_2
3. result_3, carry_3

حال منطقی که نرم افزار ما دارد به این صورت است که تمام result های درست شده را دوباره به یک full adder داده و از ان خروجی های زیر را می گیرد:

- results_sum_result : حاصل جمع بیت های مربوط به result ها.
- results_sum_carry: خروجی مربوط به carry در جمع کردن همه result ها.

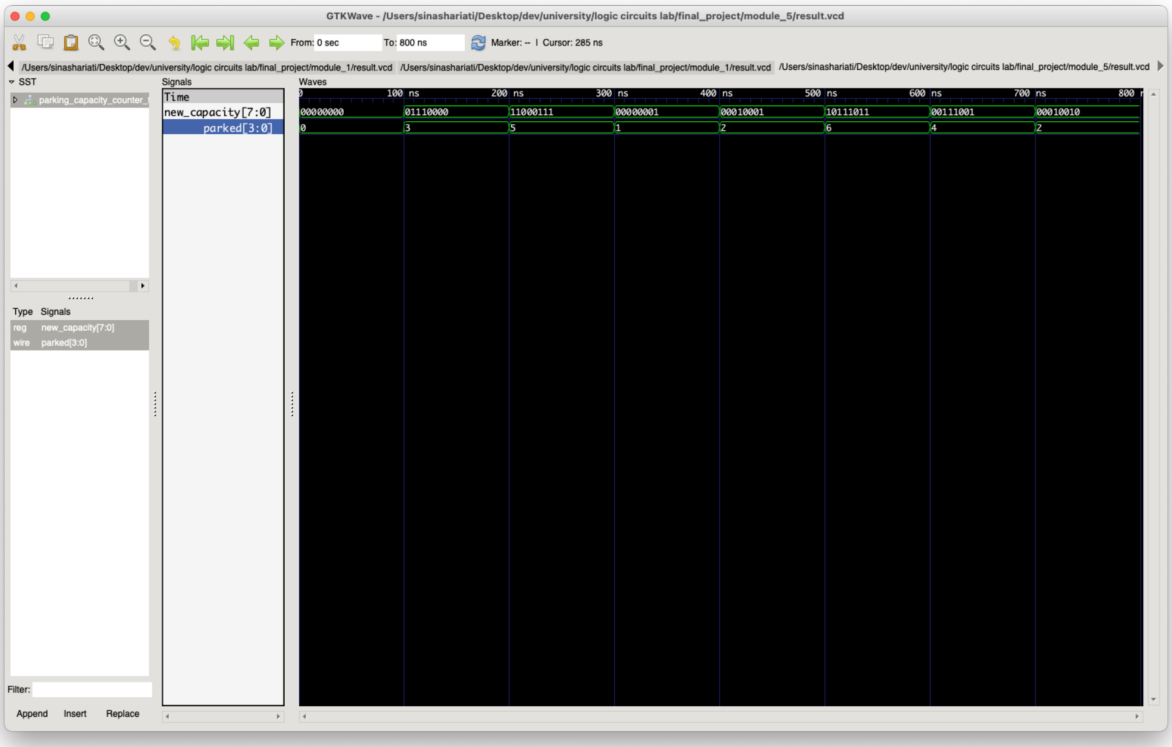
حال همین کار را برای carry های تولید شده در مرحله ی قبل تولید میکند:

- carries_sum_result: حاصل جمع بیت های مربوط به carry ها.
- carries_sum_carry: خروجی carry مربوط به جمع کردن carry ها.

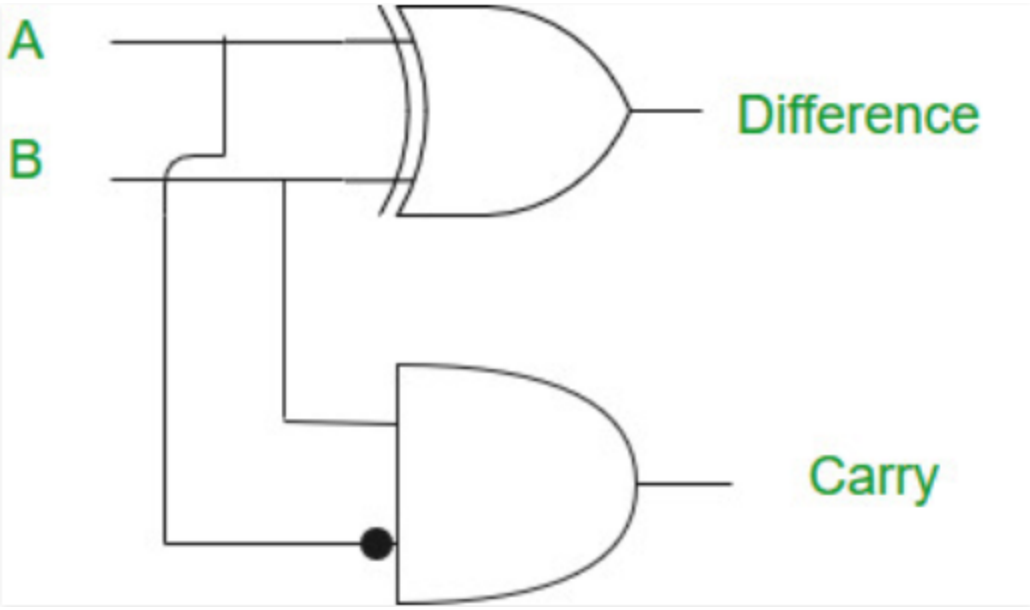
حال اگر بیت های تولید شده را به صورت زیر باهم جمع کنیم دقیقا تعداد 1 ها را به ما میدهد:

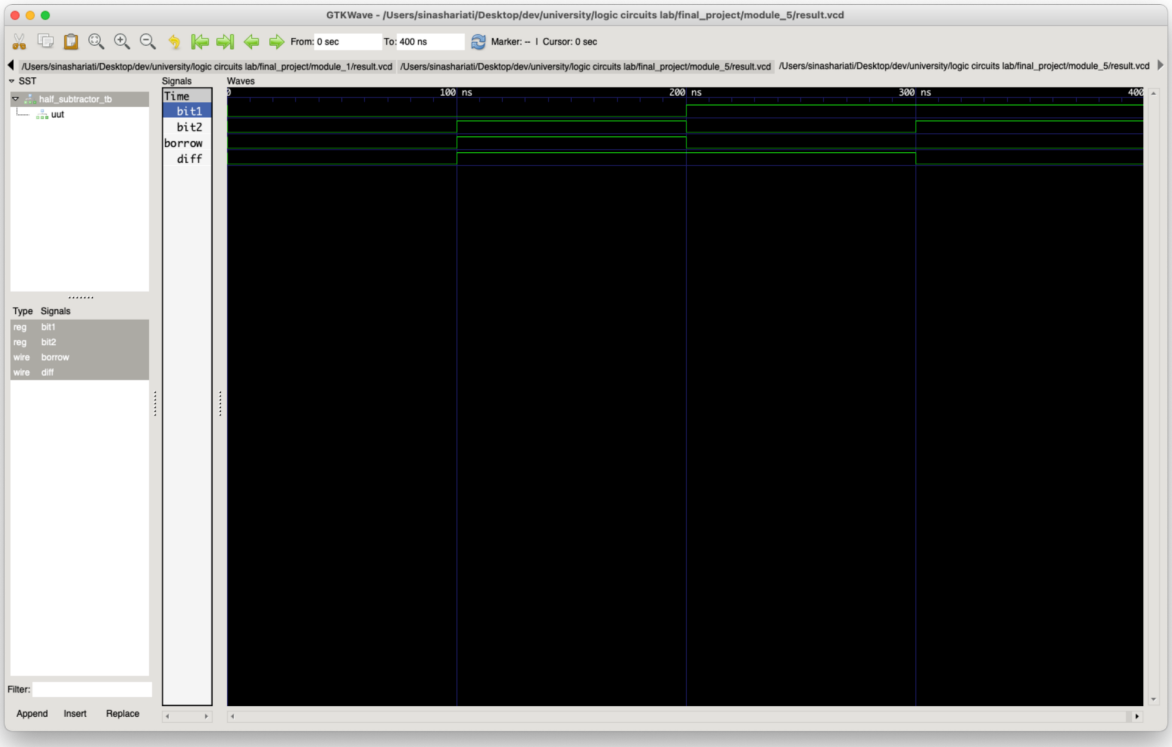
```
Parked[0] = 0 + results_sum_result
Parked[1] = carries_sum_result + results_sum_carry
Parked[2] = bit_1_carry + carries_sum_carry
Parked[3] = 1 + bit_2_carry
```

نکته: نمی توان مطمئن بود که این روش برای هر پارکینگی با هر ظرفیتی کار میکند؛ اما همانطور که در testbench پیوست شده امده است برای تمام حالت های یک پارکینگ 8 جای خالی درست کار میکند:

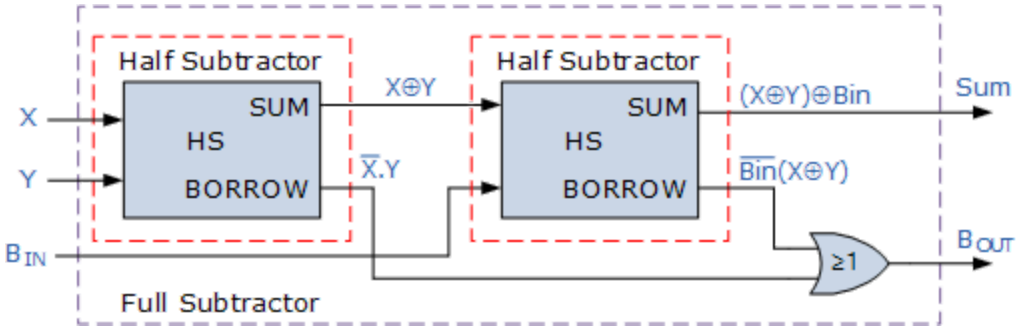


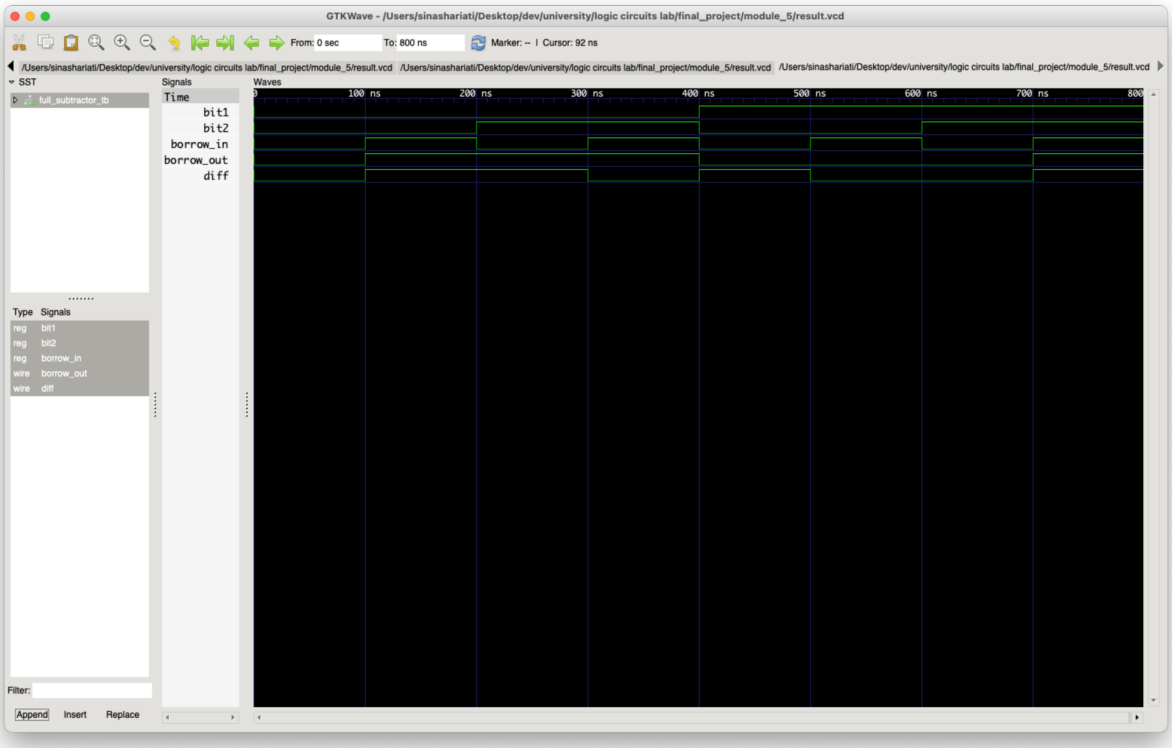
حال که تعداد 1 ها را داریم صرفا تنها مرحله ای که باقی مانده است این است که عدد 8 را از عدد بدست آمده برای empty کم کرد و خروجی parked را تولید کرد.
نکته ای که در اینجا مهم است این است که این کار را باید به صورت hierarchical انجام دهیم و از طرفی به این دلیل که ما از این ماژول هیچ وقت به عنوان جمع کننده استفاده نمیکنیم ؛ منطقی نبود که یک adder subtractor بسازیم پس یک 4bit subtractor ساختیم.
در کلاس درس subtractor تدریس نشد اما می توان با استفاده از ساختاری که برای adder توضیح داده شد به راحتی subtractor هم تولید کرد. به این صورت که ابتدا یک half subtractor میسازیم که وظیفه ی آن از هم کم کردن ۲ ورودی است که ساختار و خروجی test bench ان به صورت زیر است:



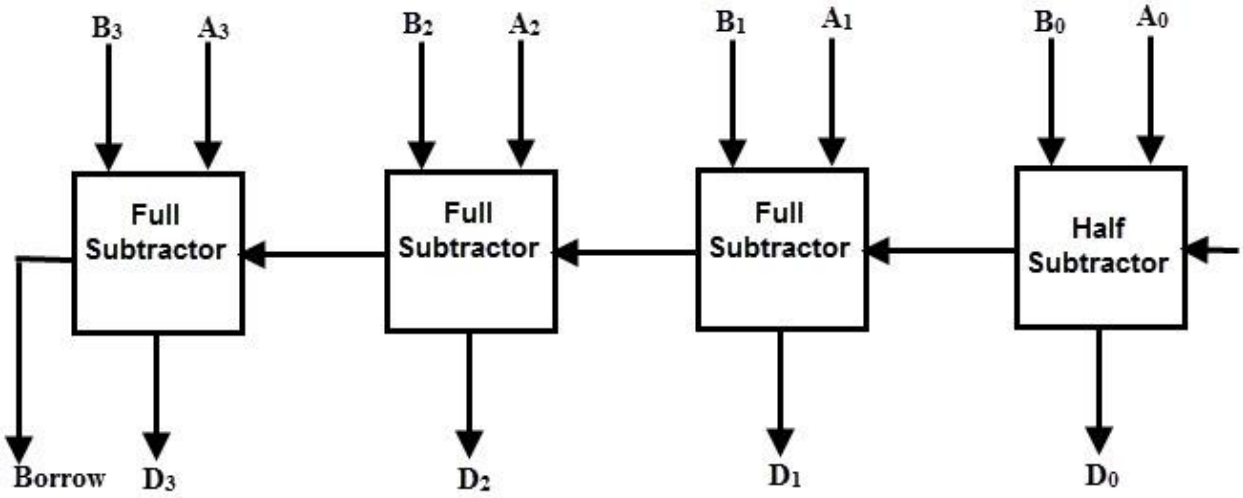


حال با استفاده از این half subtractor یک full subtractor می سازیم که ساختار و خروجی ان به صورت زیر است:

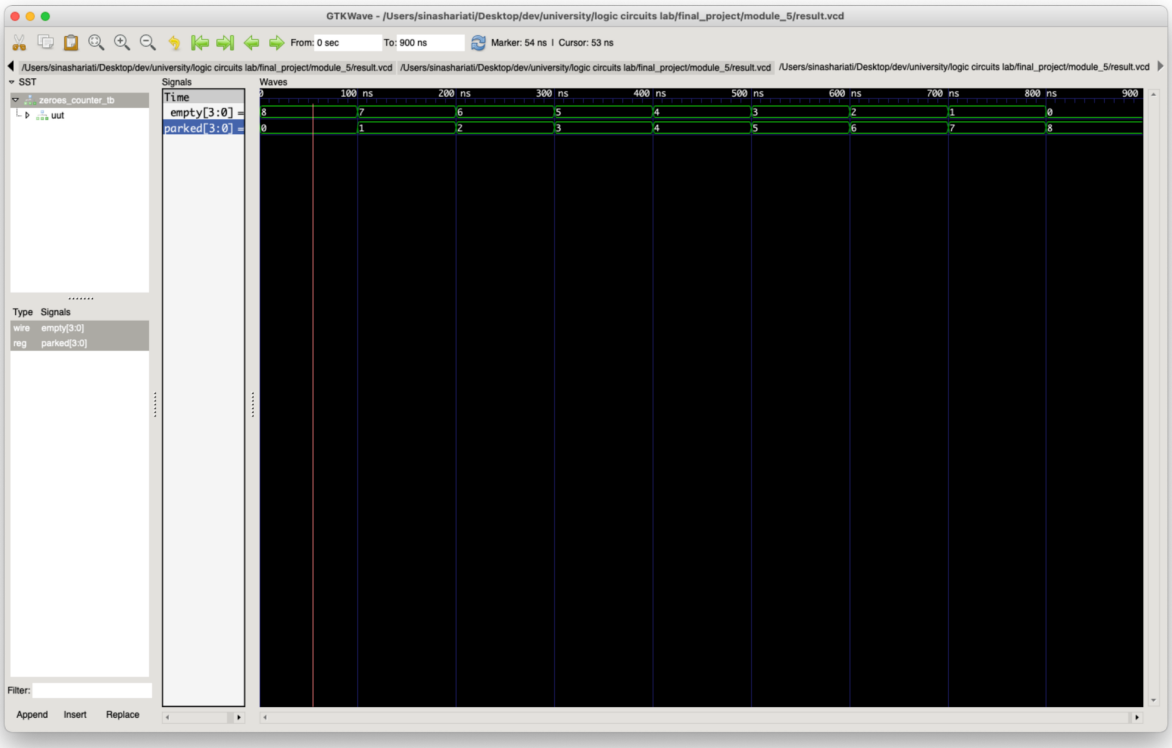




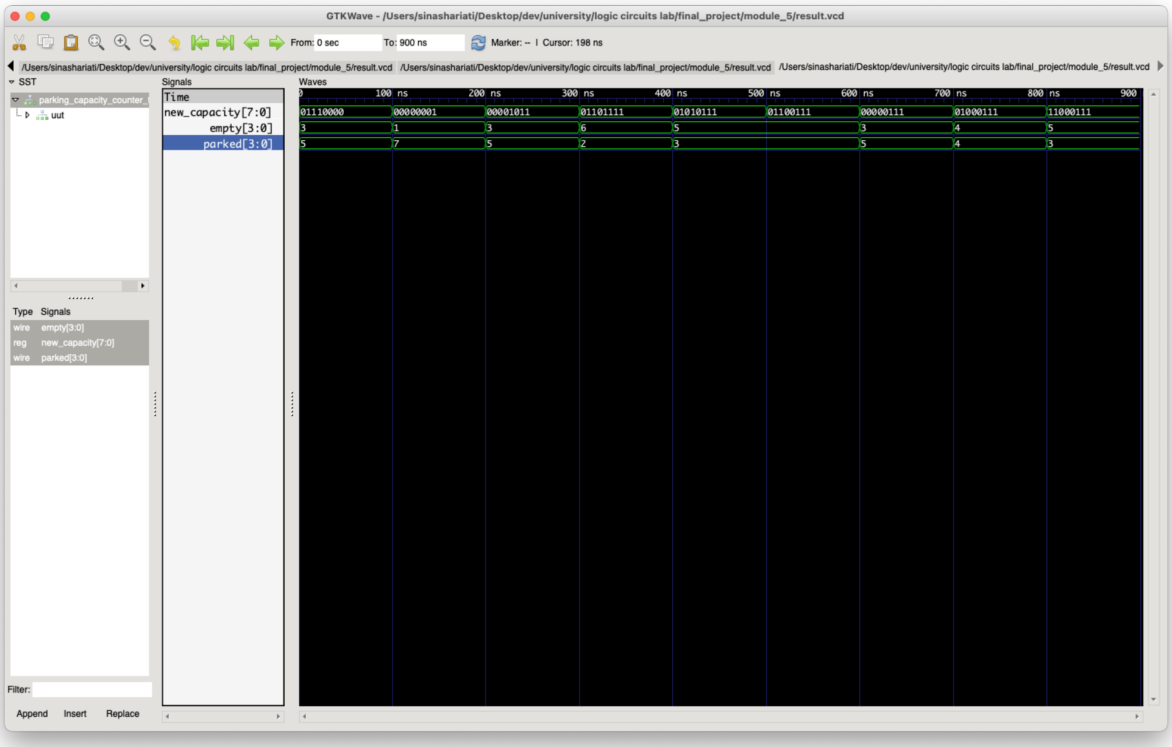
حال با استفاده از full subtractor زیر یک subtractor می سازیم که توانایی subtract کردن ۴ بیت را از هم دارد.



حال می توانیم که $(1000)_2 = (8)_{10}$ را از empty کم کنیم و یک عدد ۴ بیتی تولید کنیم که تعداد parked ها را نشان میدهد و خروجی ان به صورت زیر خواهد بود :



حال که هم تعداد صفر ها و تعداد یک ها درست می شماریم می توانیم top level module را نیز که همان parking capacity counter است نیز تست کنیم که مطمین شویم تمام اجزا به درستی کنار هم قرار میگیرند :



که همانطور که در عکس بالا آمده است کاملا درست کار میکنند.