# 積體電路電腦輔助設計概論

## CAD　LAB3

# Allocation

**實驗日期： 2019/05/15**

資工 110

B063040061

陳少洋

# (一) 實驗內容說明

在課堂中，我們有學到各種不同的 allocation 的方法，例如 Greddy Constructive Approach、Clique Partioning、Iterative Refinement Approach 等等。其中，我們應用 Left-Edge Algorithm 為這次實驗的主要內容方式。

以 Left-Edge Algorithm 的方式來做到 Allocation 的目的，來確保我們可以運用最少的 register file 個數。這個演算法的最大的限制就是，在同一個 register file 內，不能有重複 life time 的運算資料，也就是說我們只要記錄運算 data 的 life time 起始與結束就能在眾多 data 抓出可以共用 register file 的 data life time。

# (二) 實驗過程說明

**主要過程：**

**Step1** 讀入所有運算的名字、起始時間與結束時間
**Step2** 以起始使間去做運算的分類並給予排序
**Step3** 從第 1 個 Register 開始抓不衝突的資料進入 Register vector
**Step4** 如果起始時間比暫存器目前使用時間大就可以共用此 Register
**Step5** Register last time 設為此運算結束時間、並記錄其名字
**Step6** 換下一個 Register 抓 data 進來共用
**Step7** loop step4~6 直到所有 data 都有分配到 Register

**程式碼簡要說明：**

主要使用 2 個 struct：

```
struct LifeTime {

    string name ;
    int startTime ;
    int endTime ;
    bool done ;

};

struct Register {

    vector<string> regStream ;
    int lastTime ;

};
```

◤Life Time 代表此運算的生成時間直到其被用掉的時間

◤Register 代表目前暫存器所存內容

Left-Edge Allocation Algorithm：

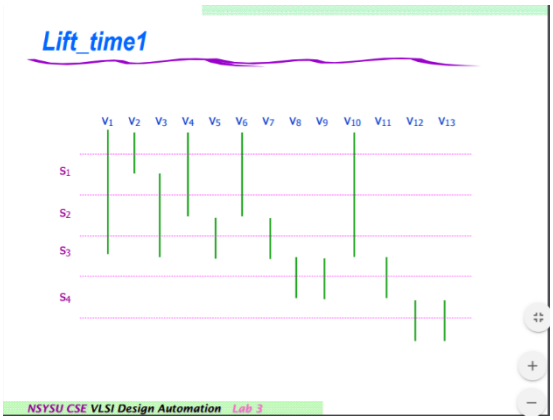
```
int LeftEdgeAllocation( vector<LifeTime> &lifeTimeVec, vector<Register> &regVec ) {
    // move the time to the left most register whether there has no conflict between their life time
    Register curReg ;
    int cnt = lifeTimeVec.size(), regCnt ;
    for ( regCnt = 0 ; cnt > 0 ; regCnt++ ) {
        curReg.lastTime = 0 ;
        curReg.regStream.clear() ;
        for ( int i = 0 ; i < lifeTimeVec.size() ; i++ ) {
            if ( !lifeTimeVec[i].done && lifeTimeVec[i].startTime >= curReg.lastTime ) {
                curReg.regStream.push_back( lifeTimeVec[i].name ) ;
                lifeTimeVec[i].done = true ;
                curReg.lastTime = lifeTimeVec[i].endTime ;
                cnt-- ;
            }
        }
        regVec.push_back( curReg ) ;
    } // end while
    return regCnt ;

}
```

▲ LeftEdgeAllocation 一個 for 迴圈就代表需要新增一個新的 Register 來使用。將可以共用的資料存入當前 Register 中，計算出最少需要用到的 Register 以及共用資訊。

# (三) 實驗結果分析說明

**測資一：**



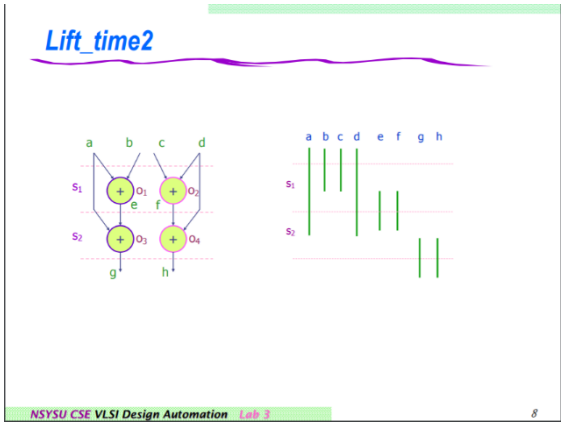

```
13
v1 0 3
v2 0 1
v3 1 3
v4 0 2
v5 2 3
v6 0 2
v7 2 3
v8 3 4
v9 3 4
v10 0 3
v11 3 4
v12 4 5
v13 4 5
```

```
Please enter the file name : test.txt
The minimun register used by left edge allocation : 5
Reg#0 : v1      v8      v12
Reg#1 : v2      v3      v9      v13
Reg#2 : v4      v5      v11
Reg#3 : v6      v7
Reg#4 : v10

--------------------------------
Process exited after 8.481 seconds with return value 0
請按任意鍵繼續 . . .
```

我們可以由上圖得知各個 Register 所需要存取的運算，並將其往左移動歸類在同一個 Registerd 可以發現他們的 life time 並不衝突。

**測資二：**



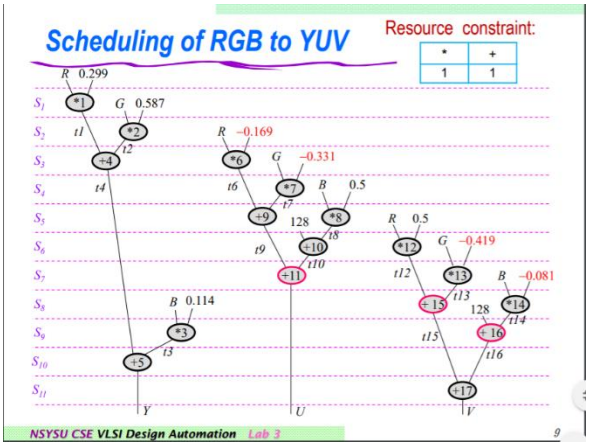

```
8
a 0 2
b 0 1
c 0 1
d 0 2
e 1 2
f 1 2
g 2 3
h 2 3
```

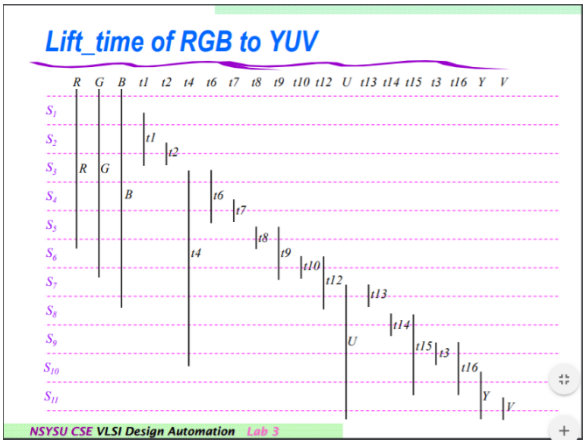
```
Please enter the file name : test2.txt
The minimun register used by left edge allocation : 4
Reg#0 : a       g
Reg#1 : b       e       h
Reg#2 : c       f
Reg#3 : d

--------------------------------
Process exited after 4.411 seconds with return value 0
請按任意鍵繼續 . . .
```

我們可以由上圖得知各個 Register 所需要存取的運算，並將其往左移動歸類在同一個 Registerd 可以發現他們的 life time 並不衝突。

測資三：




```
20
R 0 6
G 0 7
B 0 8
t1 1 2
t2 2 3
t4 3 10
t6 3 5
t7 4 5
t8 5 6
t9 5 7
t10 6 7
t12 6 8
U 7 12
t13 7 8
t14 8 9
t15 8 11
t3 9 10
t16 9 11
Y 10 12
V 11 12
```

```
Please enter the file name : test3.txt
The minimun register used by left edge allocation : 6
Reg#0 : G       U
Reg#1 : B       t14     t3      Y
Reg#2 : R       t10     t13     t15     V
Reg#3 : t1      t2      t4
Reg#4 : t6      t8      t12     t16
Reg#5 : t7      t9

--------------------------------
Process exited after 3.366 seconds with return value 0
請按任意鍵繼續 . . .
```

Reg#0：G( 0~6 ) -> U( 7~12 )

Reg#1：B( 0~8 ) -> t14( 8~9 ) -> t3( 9~10 ) -> Y( 10~12 )

Reg#2：R(0~6)->t10(6~7)->t13(7~8)->t15(8~11)->V(11~12)

Reg#3：t1( 1~2 ) -> t2( 2~3 ) -> t4( 3~10 )

Reg#4：t6( 3~5 ) -> t8( 5~6 ) - >t12( 6~8 ) -> t16( 9~11 )

Reg#5：t7( 4~5 ) -> t9( 5~7 )

## (四) 實驗心得

這次的實驗比較容易理解，Left-Edge Algorithm 算是一個很好理解的 Allocation 的方法，也因此這次能比較快速的寫出這個演算法所要求的條件，只差在需要自己想好想要使用的架構，所以我分了兩種架構來區別不同的運作。

雖然說這次實驗不難，但這只是整個模擬的一個小部分，且這只是其中一種實作方法，老師上課也講了許多不同的 Allocation 方法，我認為自己也都有吸收理解，雖然理解不一定能轉換成程式碼，但這次透過實驗內容，讓我又對 Allocation 的方式有更深入的認識，希望未來有機會在實作這部分的話，能更快的對整個完整的架構上手。