# 積體電路電腦輔助設計概論

## CAD　LAB2

# Scheduling

**實驗日期： 2019/04/24**

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 scheduling 的方法。其中，應用 List Scheduling 為這次實驗的主要內容。

以 List Scheduling 的方式來做到 resource constrained 的目的，做法為先算出整個運算的 longest path，並將 input 已經 ready 的 operation 分別放入 ready list ( mul, add )，再以 longest path 為 priority function 去選擇 path 最長的優先運算，成為 critical-path list scheduling。

# (二) 實驗過程說明

**主要過程：**

一開始想不太到要用什麼資料結構來完成這次的實驗，最初選用 vector 一筆一筆存資料再去作分析，但到了要算 Longest Path 的時候就出了很大的問題，後來改用 SIZE 固定 200 大小的陣列，來每個 operation number 來當陣列 index，才解決看不到 inpu1、input2 是否 ready 的情況。

第二部分就是做 LongestPath 的運算，我是從整個運算的最後一個 result( 不會有其他運算拿來當 input )，往上做遞迴，每次都挑需要時間比較長的部分一個一個往上增加 path 長度，作為 priority 的依據。

最後一部分就是實作 List Scheduling，主要是以 for 迴圈計算 step count，直到所有的 operation 全部做完，而 for 迴圈中最主要的工作就是，將 ADD/MUL 的 input1/2 皆 ready 可用的 operation 分別推入 ADD/MUL 的 ready list( vector )，經由 resource constrained 的個數，放入閒置中的 operator，最後再做運算倒數，如果倒數至 0 就把這個 result 設為 ready 可提供別人使用( ADD/MUL 皆分開做 )。For 迴圈結束後則 return stepCount，就是最後所需要花費的時間。

程式碼簡要說明：

主要使用 3 個 struct：

```
struct dfg {

    int op ;                // add 1 or mutiply 2 none -1
    int input1 ;            // first input
    int input2 ;            // second input
    int result ;            // output
    int pathPriority ;      // longestpath
    bool last ;             // last operator
    bool ready ;            // ready to operate
    bool pushtoreadylist ;  // already push to ready list

};
```

▲ dfg 最主要的架構(記錄所有內容)

```
struct alu {

    int mul ;       // mul resource constaint / mul operating time
    int add ;       // add                    / add

};
```

▲ Alu 紀錄有幾個 operator / 紀錄要用的時間(兩種用途)

```
struct process {

    int resultNum ;         // process operator num
    int count ;             // time counter

};
```

▲ Process 紀錄目前運算中的 operation 編號、及其還需要多少時間能完成這個運算

計算 LongestPath Priority：

```
void LongestPathCal( dfg *buffer, alu time ) {
    // calculate LongestPath driver
    for ( int i = 0 ; i < MAX ; i++ ) {
        if ( buffer[i].op != -1 && buffer[i].last ) {
            buffer[i].pathPriority = ( buffer[i].op-1 ) ? time.mul : time.add ;
            CalLength( buffer, time, buffer[i].result ) ;
        } // end if
    } // end for

} // LongestPathCal()
```

```
void CalLength( dfg *buffer, alu time, int result ) {
    // a Recursion function to calculate the longest path with every operator
    int addmul[] = { 0, time.add, time.mul } ;
    if ( buffer[buffer[result].input1].ready && buffer[buffer[result].input2].ready )
        return ;
    if ( !buffer[buffer[result].input1].ready &&\
         buffer[result].pathPriority+addmul[buffer[buffer[result].input1].op]\
         > buffer[buffer[result].input1].pathPriority ) {
        buffer[buffer[result].input1].pathPriority = buffer[result].pathPriority+addmul[buffer[buffer[result].input1].op];
        CalLength( buffer, time, buffer[result].input1 ) ;
    } // end if
    if ( !buffer[buffer[result].input2].ready &&\
         buffer[result].pathPriority+addmul[buffer[buffer[result].input2].op]\
         > buffer[buffer[result].input2].pathPriority ) {
        buffer[buffer[result].input2].pathPriority = buffer[result].pathPriority+addmul[buffer[buffer[result].input2].op];
        CalLength( buffer, time, buffer[result].input2 ) ;
    } // end if

} // CalLength()
```

▲ 上圖為 CalculateLongestPath 的 driver / 下圖為 recursion 計算長度

其中每次都去看 input1 跟 input2 兩個都 ready 的話，代表這是最一開始的 operation，也就是 base case。但只要其中一個不是最上層的 operation 就要繼續做 recursion。接下來，每次都去檢查目前的 pathPriority 加上 input1(or2)所花費的運算時間，是否大於 input1(or2)目前所計算到的 pathPriority，藉由這個檢查，就可以選擇最長的那個運算時間，得到最後的 longest path。

**List Scheduling：**

```
int ListScheduling( dfg *buffer, int cnt, alu resCons, alu time ) {
    // list scheduling with longest path piority
    int step ;              // step count
    vector<dfg> addReady ;  // add operator ready piority queue
    vector<dfg> mulReady ;  // mul operator ready piority queue
    process *addcntr = new process[resCons.add] ;   // add resource time counter
    process *mulcntr = new process[resCons.mul] ;   // mul resource time counter
    for ( int i = 0 ; i < resCons.add ; i++ )
        addcntr[i].count = 0 ;                      // initiate/unuse, set count to 0
    for ( int i = 0 ; i < resCons.mul ; i++ )
        mulcntr[i].count = 0 ;                      // initiate/unuse, set count to 0
```

▲ List Scheduling 事前宣告

```
for( step = 0 ; cnt != 0 || addReady.size() != 0 || mulReady.size() != 0 ; step++ ) {
    // step count with resource constraint and piority
    for ( int i = 0 ; i < MAX ; i++ ) {
        // for loop to push the ready operator into ready list
        if ( !buffer[i].ready && !buffer[i].pushtoreadylist &&\
             buffer[buffer[i].input1].ready && buffer[buffer[i].input2].ready ) {
            buffer[i].pushtoreadylist = true ;
            if ( buffer[i].op == 1 )
                addReady.push_back( buffer[i] ) ;
            else
                mulReady.push_back( buffer[i] ) ;
            cnt-- ;
        }
    } // end for

    // sort the ready list with piority
    if ( addReady.size() > 1 )
        sort( &addReady[0], &addReady[0]+addReady.size(), compare ) ;
    if ( mulReady.size() > 1 )
        sort( &mulReady[0], &mulReady[0]+mulReady.size(), compare ) ;
    // sort the ready list with piority
```

▲ List Scheduling // Step counting，推入 ready list，並以 pathPriority 排序。

```
for ( int i = 0 ; i < resCons.add ; i++ ) {
    // check if there is a resource is in idle( counter = 0 ) ADD
    if ( addcntr[i].count == 0 && addReady.size() != 0 ) {
        // start to operate
        addcntr[i].resultNum = addReady[0].result ;
        addcntr[i].count = time.add ;
        addReady.erase( addReady.begin() ) ;
        // remove from ready list
    }
} // end for
for ( int i = 0 ; i < resCons.mul ; i++ ) {
    // check if there is a resource is in idle( counter = 0 ) MUL
    if ( mulcntr[i].count == 0 && mulReady.size() != 0 ) {
        // start to operate
        mulcntr[i].resultNum = mulReady[0].result ;
        mulcntr[i].count = time.mul ;
        mulReady.erase( mulReady.begin() ) ;
        // remove from ready list
    }
} // end for
```

▲ List Scheduling // 上為 ADD 下為 MUL，檢查有沒有閒置的 operator，有就丟進去並設置 time

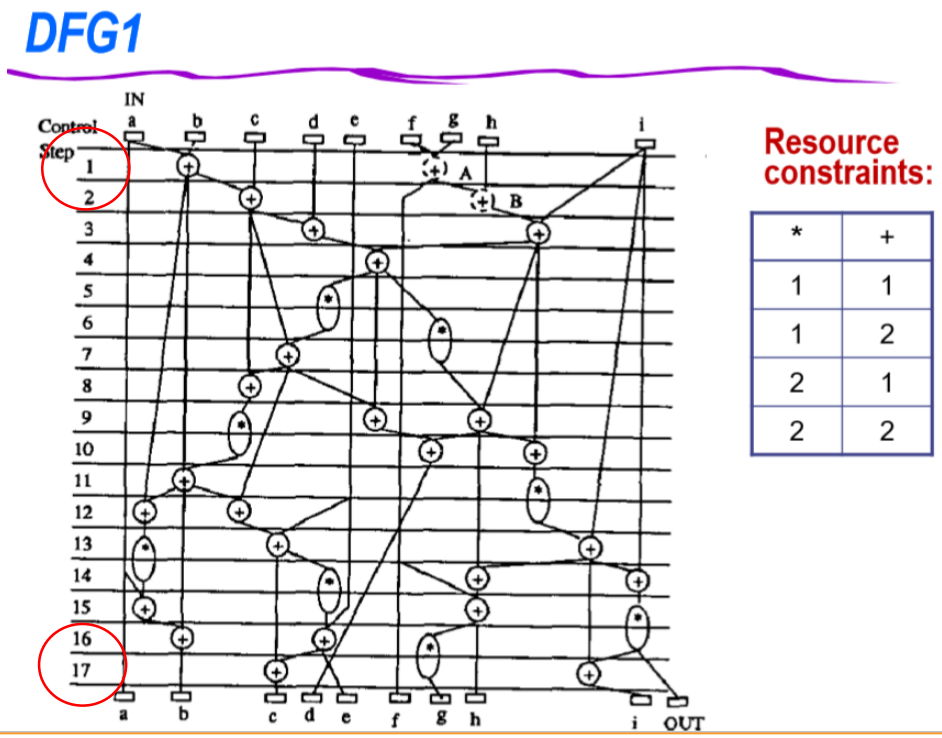
```
for ( int i = 0 ; i < resCons.add ; i++ ) {
    // time count with operator time ADD
    if ( addcntr[i].count != 0 ) {
        addcntr[i].count-- ;
        if ( addcntr[i].count == 0 )
            // operator done, set this operator output to ready
            buffer[addcntr[i].resultNum].ready = true ;
    } // end if
} // end for
for ( int i = 0 ; i < resCons.mul ; i++ ) {
    // time count with operator time MUL
    if ( mulcntr[i].count != 0 ) {
        mulcntr[i].count-- ;
        if ( mulcntr[i].count == 0 )
            // operator done, set this operator output to ready
            buffer[mulcntr[i].resultNum].ready = true ;
    } // end if
} // end for
} // end for
return step ;
```

▲ List Scheduling // 上為 ADD 下為 MUL，把設置的 time-1，若 time-1 為 0，表示完成 operate

▲ List Scheduling // for 迴圈結束則 return step count，最終結果

## (三) 實驗結果分析說明

測資一：

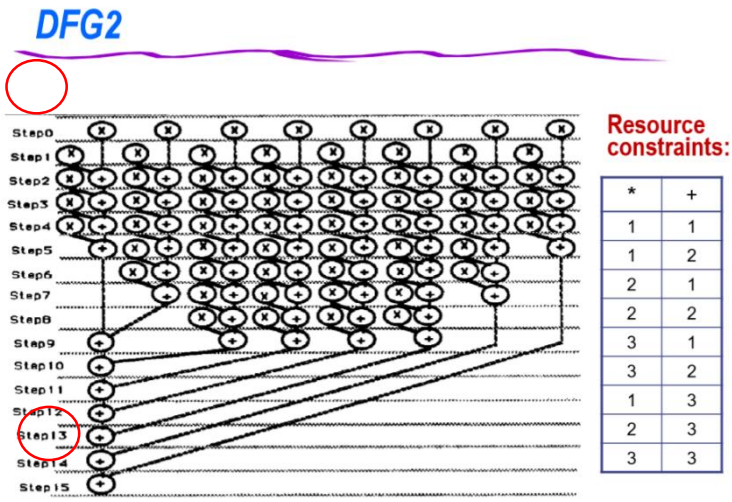



```
C:\Users\yang8\Desktop\CAD_LAB2_B063040061_B063040056.exe
Please enter input file name ( except".txt" ) : DFG1
Please input the ADD operator time : 1
Please input the MUL operator time : 2
Please input all resource constraint you want to test ( -1 to end ) :
MUL resource / ADD resource ( with white space ): 1 1
MUL resource / ADD resource ( with white space ): 1 2
MUL resource / ADD resource ( with white space ): 2 1
MUL resource / ADD resource ( with white space ): 2 2
MUL resource / ADD resource ( with white space ): 5 5
MUL resource / ADD resource ( with white space ): 8 8
MUL resource / ADD resource ( with white space ): -1
MUL, ADD ( 1, 1 ) : 28 steps ( counter start from step 1 )
MUL, ADD ( 1, 2 ) : 19 steps ( counter start from step 1 )
MUL, ADD ( 2, 1 ) : 28 steps ( counter start from step 1 )
MUL, ADD ( 2, 2 ) : 17 steps ( counter start from step 1 )
MUL, ADD ( 5, 5 ) : 17 steps ( counter start from step 1 )
MUL, ADD ( 8, 8 ) : 17 steps ( counter start from step 1 )

--------------------------------
Process exited after 21.1 seconds with return value 0
請按任意鍵繼續 . . .
```

總共有 4 個資源限制，我多測了兩個，來證明 critical path 是 17 steps，而老師 PPT 的圖為 resource constraints( mul = 2, add = 2 )為 17 steps。

**測資二：**

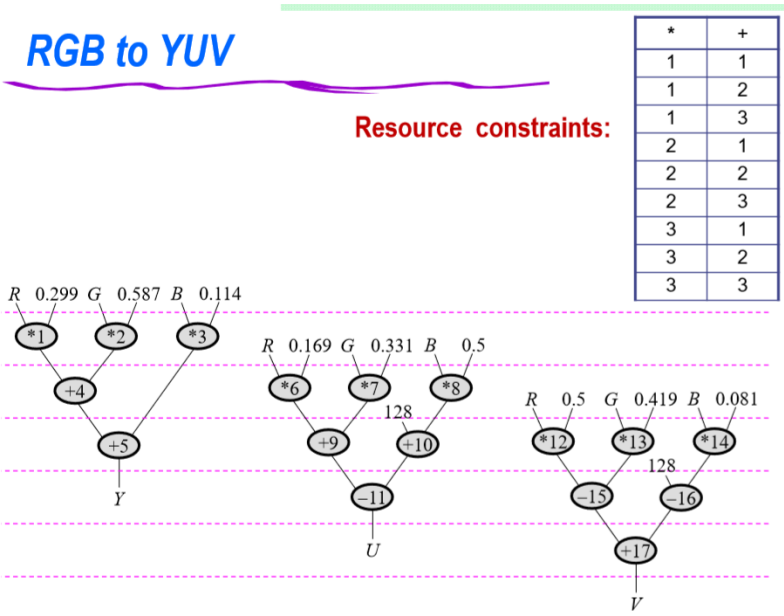



我一樣多試了兩個 resource constarints，得到 critical path 為 16 steps，而左圖為 resource constraints( mul = 8, add = 8 )為 16 steps。( 0 ~ 15 = 16 )

```
C:\Users\yang8\Desktop\CAD_LAB2_B063040061_B063040056.exe
Please enter input file name ( except".txt" ) : DFG2
Please input the ADD operator time : 1
Please input the MUL operator time : 1
Please input all resource constraint you want to test ( -1 to end ) :
MUL resource / ADD resource ( with white space ): 1 1
MUL resource / ADD resource ( with white space ): 1 2
MUL resource / ADD resource ( with white space ): 2 1
MUL resource / ADD resource ( with white space ): 2 2
MUL resource / ADD resource ( with white space ): 3 1
MUL resource / ADD resource ( with white space ): 3 2
MUL resource / ADD resource ( with white space ): 1 3
MUL resource / ADD resource ( with white space ): 2 3
MUL resource / ADD resource ( with white space ): 3 3
MUL resource / ADD resource ( with white space ): 5 5
MUL resource / ADD resource ( with white space ): 8 8
MUL resource / ADD resource ( with white space ): 9 9
MUL resource / ADD resource ( with white space ): -1
MUL, ADD ( 1, 1 ) : 71 steps ( counter start from step 1 )
MUL, ADD ( 1, 2 ) : 62 steps ( counter start from step 1 )
MUL, ADD ( 2, 1 ) : 68 steps ( counter start from step 1 )
MUL, ADD ( 2, 2 ) : 37 steps ( counter start from step 1 )
MUL, ADD ( 3, 1 ) : 68 steps ( counter start from step 1 )
MUL, ADD ( 3, 2 ) : 36 steps ( counter start from step 1 )
MUL, ADD ( 1, 3 ) : 62 steps ( counter start from step 1 )
MUL, ADD ( 2, 3 ) : 33 steps ( counter start from step 1 )
MUL, ADD ( 3, 3 ) : 26 steps ( counter start from step 1 )
MUL, ADD ( 5, 5 ) : 18 steps ( counter start from step 1 )
MUL, ADD ( 8, 8 ) : 16 steps ( counter start from step 1 )
MUL, ADD ( 9, 9 ) : 16 steps ( counter start from step 1 )

--------------------------------
Process exited after 36.25 seconds with return value 0
請按任意鍵繼續 . . .
```

測資三　：




一樣多試了兩個 resource constarints，得到 critical path 為 3 steps，而左圖為 resource constraints( mul = 3, add = 3 )為 5 steps。

```
C:\Users\yang8\Desktop\CAD_LAB2_B063040061_B063040056.exe
Please enter input file name ( except".txt" ) : RGBtoYUV
Please input the ADD operator time : 1
Please input the MUL operator time : 1
Please input all resource constraint you want to test ( -1 to end ) :
MUL resource / ADD resource ( with white space ): 1 1
MUL resource / ADD resource ( with white space ): 1 2
MUL resource / ADD resource ( with white space ): 1 3
MUL resource / ADD resource ( with white space ): 2 3
MUL resource / ADD resource ( with white space ): 2 2
MUL resource / ADD resource ( with white space ): 2 3
MUL resource / ADD resource ( with white space ): 3 1
MUL resource / ADD resource ( with white space ): 3 2
MUL resource / ADD resource ( with white space ): 3 3
MUL resource / ADD resource ( with white space ): 5 5
MUL resource / ADD resource ( with white space ): 8 8
MUL resource / ADD resource ( with white space ): 9 9
MUL resource / ADD resource ( with white space ): -1
MUL, ADD ( 1, 1 ) : 11 steps ( counter start from step 1 )
MUL, ADD ( 1, 2 ) : 10 steps ( counter start from step 1 )
MUL, ADD ( 1, 3 ) : 10 steps ( counter start from step 1 )
MUL, ADD ( 2, 3 ) : 6 steps ( counter start from step 1 )
MUL, ADD ( 2, 2 ) : 7 steps ( counter start from step 1 )
MUL, ADD ( 2, 3 ) : 6 steps ( counter start from step 1 )
MUL, ADD ( 3, 1 ) : 9 steps ( counter start from step 1 )
MUL, ADD ( 3, 2 ) : 6 steps ( counter start from step 1 )
MUL, ADD ( 3, 3 ) : 5 steps ( counter start from step 1 )
MUL, ADD ( 5, 5 ) : 4 steps ( counter start from step 1 )
MUL, ADD ( 8, 8 ) : 3 steps ( counter start from step 1 )
MUL, ADD ( 9, 9 ) : 3 steps ( counter start from step 1 )

--------------------------------
Process exited after 39.41 seconds with return value 0
請按任意鍵繼續 . . .
```

## (四) 實驗心得

第二次的積體電路電腦輔助設計實驗，在一開始聽老師講解 scheduling 的時候，概念大致上都了解也聽得懂，但真正要實作時，卻又不知道該如何下手。但也因為有這次實驗，我又重新再了解一次 List scheduling 的精隨，最後也創造出自己想法中的資料結構，雖然過程中並沒有太容易，但大致上都能一一克服，也讓我更加對這些 scheduling 的方法有更進一步的認識。

事實上，理解一個完整的概念與實作出一個完整的架構還是有一些落差，時常與我所想像的做法不同，但慢慢增加東西而完成後，就覺得還蠻有趣的，也有很多不同的做法能達成一樣的目的，希望未來需要我做其他更進一步的軟體輔助硬體設計下，這堂課所學的東西能夠派上用場。