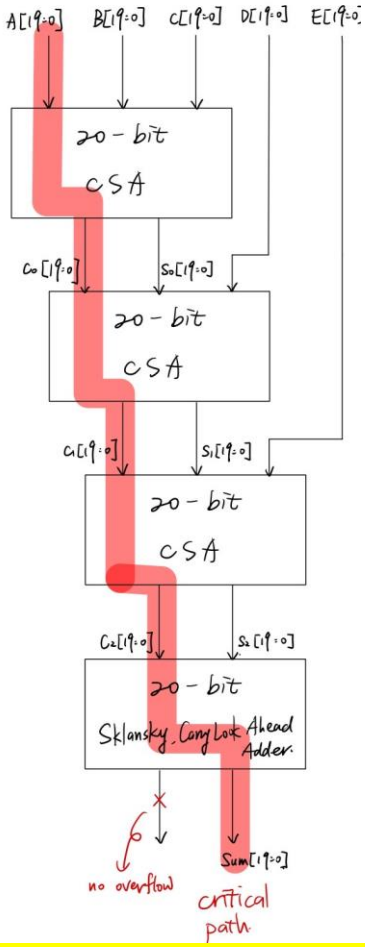
1. Design a 20-bit 2's complement multioperand adder (A+B+C+D+E) using Carry-Save Adder + Sklansky carry look ahead adder. To further optimize the delay, each module can have different size. **The goal is to have minimum critical path delay time with the least gate count. Assum there is no overflow during the additions. The internal word length of each module is 20 bits (60)**
(a) Show your block diagram like that shown in Fig.1. The CPA in Fig.1 shall be the Sklansky carry lookahead adder with variable group size. Indicate the critical path. Expalin your design concept and the decision of group size (20)

因為題目表示在中間的所有加法運算中不需要考慮 overflow 的問題，因此不管 Carry Save Adder(CSA)抑或是 Sklansky carry look ahead adder，我都將使用 20bit 的長度來設計模組。

下圖為我的 20bit multi-operand adder 的設計:

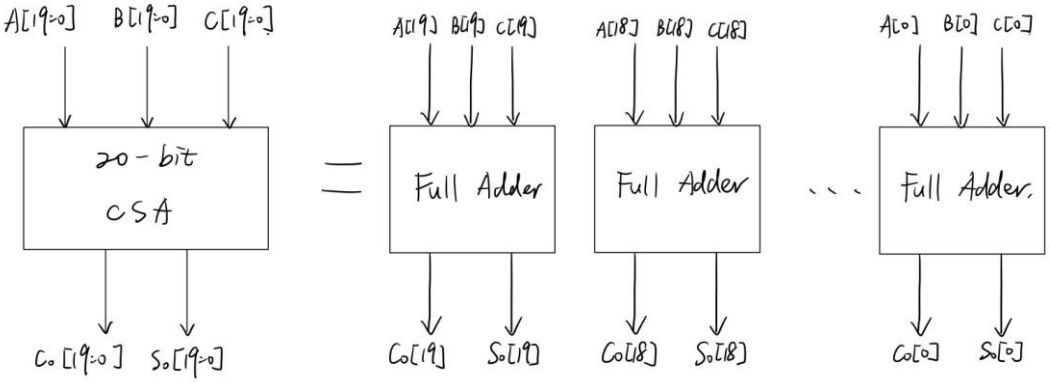


在上面的 block diagram 中，C0, C1, C2 都是已經向左移一位，以表示兩倍的權重(LSB 留空或是填 0，因此仍是從 0 開始取值)。而最後一級的 Sklansky Carry Look Ahead Adder 則是有多保留 Cout(若保證不會 overflow 則可以移除該 pin)。這次是有五個數要相加，因此我使用了三級的 CSA，而最後再使用 Sklansky Carry Look Ahead Adder 來將第三級 CSA 輸出的兩個結果加起來。而關於 Sklansky Carry Look Ahead Adder 內部的結構設計則在下一題說明。(內部包含了 Bitwise PG Logic, Sklansky PG network 及 Sum Logic)

(b) For each block, you shall show its logic design diagram. Indicate the critical path. Explain your design concepts (20).

在第(a)題的 block diagram 中，共用了兩種不同的模塊。一個是 20-bit Carry Save Adder(CSA)，而另一個是 20-bit Sklansky Carry Look Ahead Adder。將在這個部分詳細解釋這兩個模塊內部是如何設計與實作的。

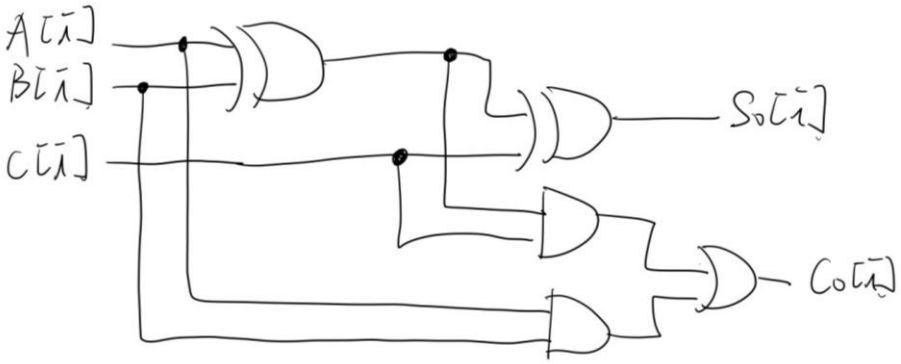
1. 20-bit CSA:



在(a)題中我使用了三個 20-bit CSA，因為使用方法相同，因此以第一級來解釋。這個 20-bit 的 CSA 我是使用 20 個個別獨立的 full adder 來實作。與傳統 full adder 不同的是原先 cin 的訊號可以換成給第三個加數拿進來憶起相加。此外，這些 full adder 因為是個別獨立的，並沒有誰串接誰的關係。因此這些 full adder 是可以被平行計算的。換句話說，一級 CSA 的 delay 與一級 full adder 的 delay 會是相同的。缺點是輸出的 sum 與進位仍需要一個 adder 來把他們加起來變成最後的答案。在這個 CSA 中，critical path 為 full adder 的 critical path。而 full adder 的 critical path 則為輸入訊號到 sum/cout(取決於要使用何種 full adder 的架構)。

Full adder 的實作:
這邊會使用最常見的 full adder 實作方法:
由 full adder 的真值表或是行為，我們可以得到以下式子:

$$\begin{cases} S_0[i] = A[i] \oplus B[i] \oplus C[i] \\ C_0[i] = \left(A[i] \cdot B[i]\right) + \left(A[i] \oplus B[i]\right) \cdot C[i] \end{cases}$$



上圖為 full adder 的 gate level 實作。

2. 20-bit Sklansky Carry Look Ahead Adder
對於 Sklansky Carry Look Ahead Adder，內部的架構又可以細分成三個部分:

- 20-bit Bitwise PG Logic
- 20-bit Sklansky PG network
- 20-bit Sum Logic

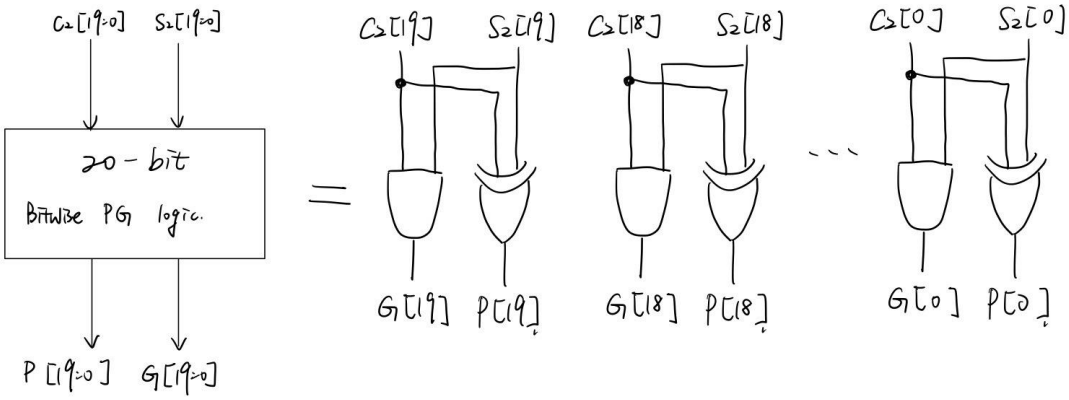
而以下將會針對 Bitwise PG Logic, Sklansky PG network,與 Sum Logic 進行個別的解釋。Bitwise PG Logic 的部分主要是要產生單一 bit 的加法是否會產生 carry generation 或是 propagation 的信號。而 PG network 則是透過在前一步算出來的 bitwise 的 generation/propagation 進行 grouping，來看這一群 bit 是否會產生 carry generation 或是 propagation，以此來將 critical path 縮短。而最後的 sum logic 則是使用中間算出來的 propagation 訊號與不同 bit 間的進位訊號 Ci 來做運算，進一步求出最終 sum 出來的結果。這邊一樣假設不會有 overflow 的問題，因此 sum logic 最終的輸出便可以簡化為只有 20-bit 的 sum 結果。存的值便是五個數字相加起來的結果。

- 20-bit Bitwise PG Logic

承接(a)小題的圖，傳進這個 module 的訊號為 C2 與 S2，由下式:

$$\begin{cases} G_i = G_{i:i} = C_2[i] \cdot S_2[i] \\ P_i = P_{i:i} = C_2[i] \oplus S_2[i] \end{cases}$$

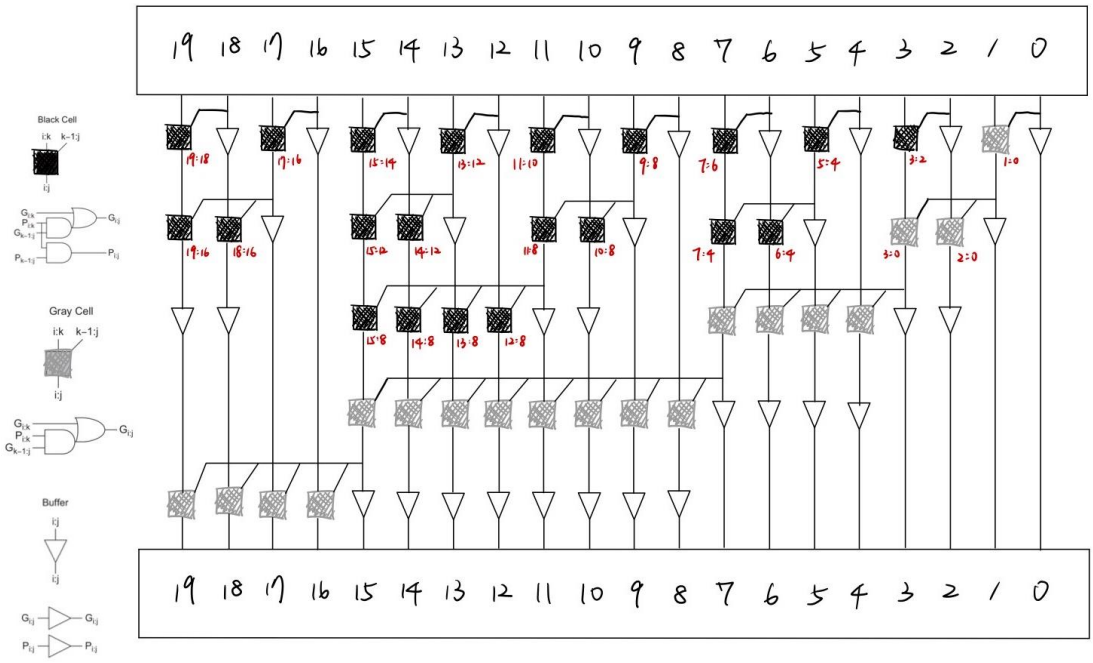
可以求出兩個 bit 的加法是否會產生(Generation)進位，或是兩個 bit 的加法是否會傳遞(Propagation)進位的可能，如果有，則必須要得到進位的 bit 來進一步判斷是否會有進位產生。因此對於一個 bit 的 bitwise PG logic 可以使用一個 and gate 及一個 exclusive or gate 來達成。而對於 20-bit 的 bitwise PG 則可以使用二十個 and gate 及二十個 exclusive or gate 來達成。這 20 個 bitwise 的 PG logic 是可以同時進行運算的，因此跑完這二十個 bitwise PG logic 的時間會與只跑一個 bitwise 的 PG logic 的時間相同。



上圖為 20-bit bitwise PG logic 的 gate level 實作

- 20-bit Sklansky PG network

這邊就到了這個 Sklansky Carry Look Ahead Adder 最厲害的地方了。由上一頁的 bitwise PG logic 先算出 single bit 之間是否會有 carry generation 或是 carry 的 propagation。而在這邊將會介紹如何產生 Grouping 的 P 及 G 來透過 prefix 的特性讓這個 adder 的多個 bit 可以類似平行計算。透過產生 grouping 的 P&G，我們便可以使用 Grouping G 來算出 Cout[19:0]，而使用 Grouping P 及 Cout[19:0]來求得 Sum[19:0]，這個部分將會在下一個部分詳述。



上圖為改良自課本的 Sklansky PG network 為 20-bit 的結果圖。沿用課本對於 black cell 與 gray cell 的定義: black cell 會產 grouping G&P，而 gray cell 則只產 grouping G。此外，Sklansky Carry Look Ahead Adder 最大的缺點就是在於他的 fanout 太多了，也因此速度會比課本另外一個 Adder 的架構:Kogge-Stone 還要慢。

這邊可以稍微分析一下上面的 PG network，使用到的公式如下:

$$\begin{cases} G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j} \\ P_{i:j} = P_{i:k} \cdot P_{k-1:j} \end{cases} \quad i \geq k > j$$

也就是說當要計算 $G_{i:j}$ 的時候，必須確定 $G_{i:k}, P_{i:k}, G_{k-1:j}$ 都是已經被算好的，透過這個限制，並且使用 Sklansky 的架構，便可以設計出上圖的 PG network。

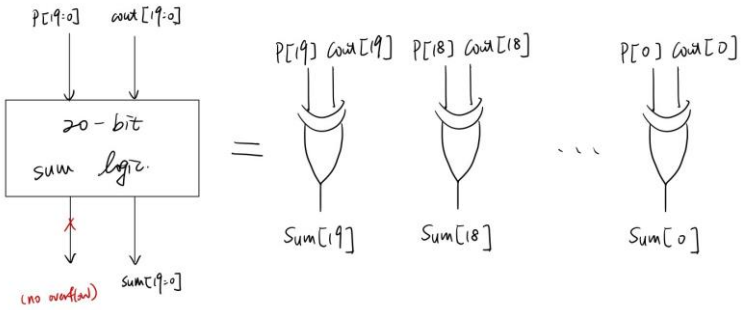
Ex: Black Cell 7:4

$$\begin{cases} G_{7:4} = G_{7:6} + P_{7:6} \cdot G_{5:4} \\ P_{7:4} = P_{7:6} \cdot P_{5:4} \end{cases}$$

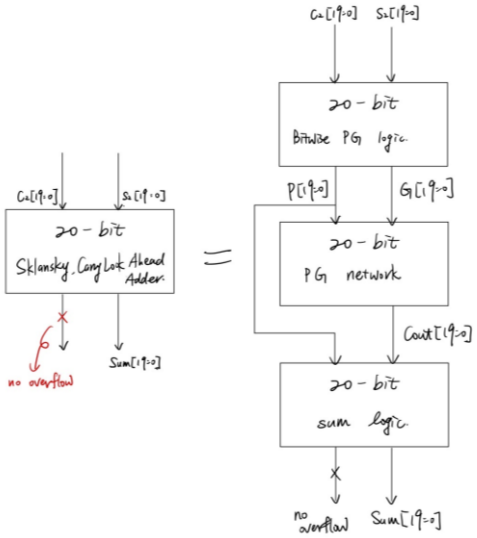
可以發現當輪到 Black Cell 7:4 計算的時候，該準備好的 $G_{7:6}, P_{7:6}$ 已經在 Black Cell 7:6(上一層)計算好了。而 $G_{5:4}, P_{5:4}$ 也同樣在 Black Cell 5:4(上一層)計算好了。如此使用 prefix 的方式來計算，使得整體的速度會相較最簡單的 carry ripple adder 快了許多。在這層會輸出的只有一個部分: Cout[19:0]，而 Cout[i]便是我們的 Grouping G($G_{i:0}$)。

- 20-bit Sum Logic

在 Sum Logic 這 part 便會將前一個 part 的 Cout[19:0]與前前一個 part 的 P[19:0]作為 input，進一步去算出最終的 Sum[19:0]為多少。基本上就是將 Cout[i] xor P[i]便是 Sum[i]了。細節實作如下:



因此整個 20-bit Sklansky Carry Look Ahead Adder 的架構圖如下:



剛好在網路上也有找到 Sklansky Adder 的程式化輸出 cell 架構圖:

```
Sklansky 20 bitwidth
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
-------------------------------------------------
0   2   4   6   8    10    12    14    16    18
  1 1     5 5     9  9       13 13       17 17
      3 3 3 3           11 11 11 11
              7 7  7  7  7  7  7  7
                                   15 15 15 15
-------------------------------------------------
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
[coherent17@NVL4 ~/AdderCircuitGenerator]$
```

簡單的 python program 來輸出 Sklansky 的 cell code:

```python
def __init__(self, bitwidth):
    super(Sklansky, self).__init__(bitwidth)
    for row_index, row in enumerate(self.matrix):
        shift = pow2(row_index+1)  # Sequence in loop: 2 4 8 16 32..
        repeats = pow2(row_index) # How many times is the value repeated? Sequence in loop: 1 2 4 8 16
        starting_index = repeats - 1 # Sequence 0 1 3 7 15
        last_index = bitwidth - 1 # len(row)
        for i in range(starting_index, last_index, shift):
            for j in range(repeats):
                try:
                    row[j+i] = i
                except IndexError:
                    pass
```

https://github.com/IamFlea/AdderCircuitGenerator/blob/master/adders.py

Critical Path 分析:

- 20-bit Carry Save Adder (CSA)
  與 full adder 的 critical path 相同 from input 到 cout
- 20-bit Bitwise PG Logic
  因為是平行運算，與 single bit 產生 PG 相同，critical path 為 input 到 P
- 20-bit Sklansky PG network
  Critical path 會經過 $\lceil \log_2 20 \rceil = 5$ 層的 and or gate 的運算。
- 20-bit Sum Logic
  因為是平行運算，與 single bit 產生 sum 相同，critical path 為過一個 xor。

(c) Calculate the critical path delay in terms of the sum of 1-bit FA, 1-bit PG, Mux2, Valency-2 cells and XOR2 delay.(10)

由前一題的 critical path 分析可以得出:

$$T_{delay} = 3 \cdot t_{FA} + t_{pg} + 5 \cdot t_{valency2} + t_{xor2}$$

會經過 3 個 CSA，每個 CSA 會經過一個 Full adder 的 delay，而後進入到 Sklansky Carry Look Ahead Adder，會先經過一級 bitwise PG，這邊會有一個 pg 的 delay，而後進入到 Sklansky PG network，這邊會有 $\lceil \log_2 20 \rceil = 5$ 的 valency2 cell 的 delay。最後進入到 sum logic，僅需要一級 xor2 的 delay。

(d) Indicate the overall module used in terms of the number of 1-bit FA, 1-bit PG, Mux2, Valency-2 cells and XOR2. (10)

Module name	Number of Modules
1-bit FA	$3(CSA) \times 20(bit) = 60$
1-bit PG	20
Mux2	0
Valency-2 cells	$21(Black) + 19(Gray) = 40$
Buffer	33
XOR2	20
Total	173

2. Pipeling design of Q1 (40)

(a) Show the block diagram of the design with three pipelining stages (with DFFs) within the mutioperand adder. (20)

首先，先將前一小題有使用到的 module(1-bit FA, 1-bit PG, Valency-2 cells. Buffer, XOR2)先定義好相對經過所需要的時間。才可以了解到 pipeline 切在哪邊可以切得平均，使得 clk period 可以越短。為了簡單估計，我將 and/or gate 及 buffer 的 propagation delay 視為單位延遲=unit gate delay。在 FA 中與 bitwise PG logic & Sklansky Carry Look Ahead Adder 中都有使用到 XOR2 gate，這邊將其訂為 2 倍的 unit size delay。

Module name	Relative Gate Delay
1-bit FA	4 gate delay
1-bit PG	2 gate delay
Valency-2 cells	2 gate delay
Buffer	1 gate delay
XOR2	2 gate delay

在這之中，FA 的 sum 會經過兩次 XOR2，因此會有 4 個 gate delay。Bitwise 的 PG logic 有並聯的 and 及 XOR2，因此會有 2 個 gate delay。Valency-2 cells 則為 AO gate，因此會有 2 個 gate delay。

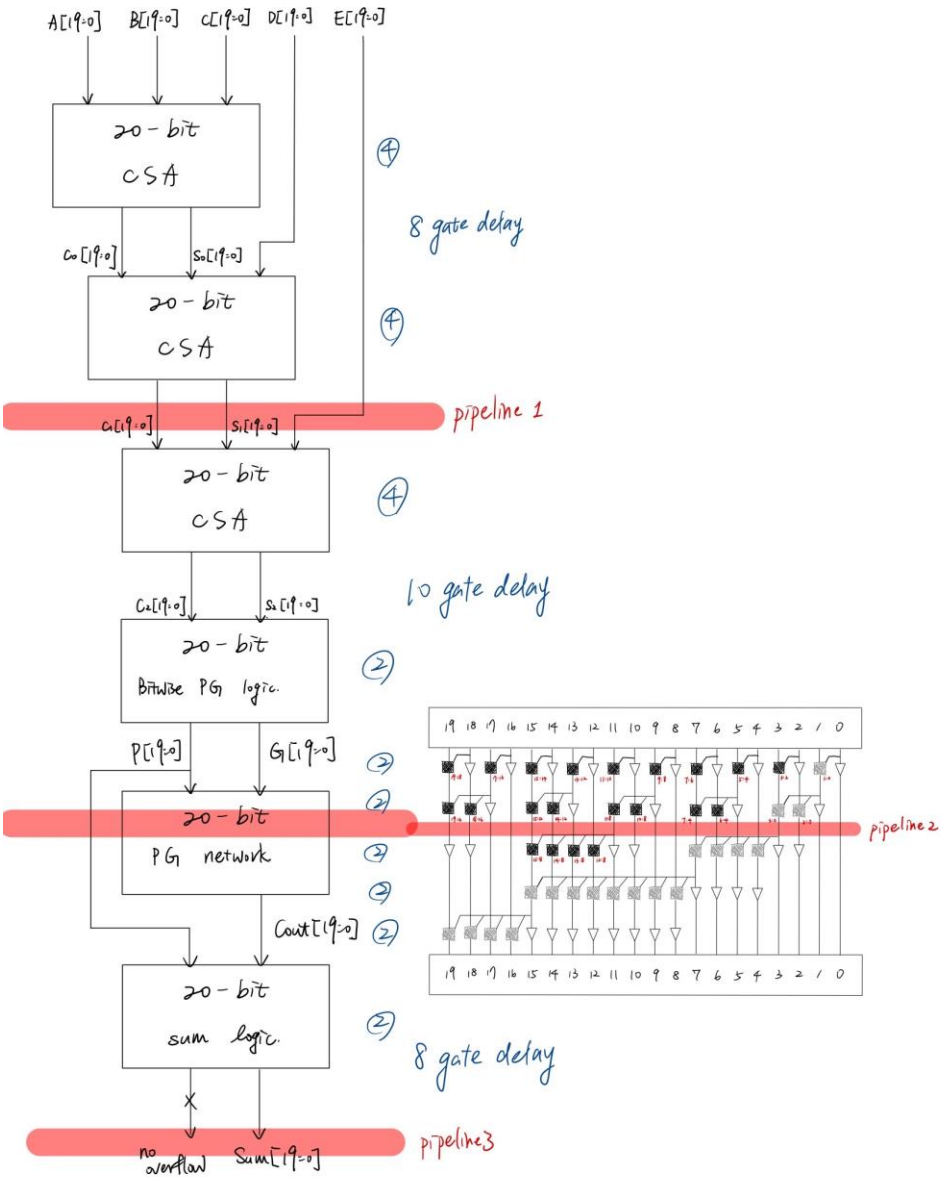
因此由第一題的(c)小題的推估可以得到 critical path 的 delay 為:

$$T_{delay} = 3 \cdot t_{FA} + t_{pg} + 5 \cdot t_{valency2} + t_{xor2}$$

換算為 gate delay:

$$T_{delay} = 3 \cdot 4 + 2 + 5 \cdot 2 + 2 = 26$$

為了盡量切的平均一點，使得 clk frequency 可以高一點，考慮 $26/3 \approx 9$，盡可能將每級的 gate delay 切在 9 附近。因此我將我這次的 20-bit adder 切成三個 stage 分別是:8 gate delay, 10 gate delay, 8 gate delay。第一級 pipeline 會切在第二級 CSA 與第三級 CSA 之間。第二級的 pipeline 會切在 PG network 的第二層(過了兩個 valency-2 cell 後)。下一頁為示意圖。



(b) List the number of DFF used. (10)

Pipeline1: 20-bit C1 + 20-bit S1 = 40 DFF
Pipeline2: 20-bit bitwise P +8 black cells * 2 + 2 gray cells * 1 + 10 buffer out = 48 DFF
Pipeline3: 20-bit sum = 20 DFF

(c) List the clock cycle time of this pipeling design with tpd of the module used in the critical path (10)

$$\begin{cases} t_{pd1} = 8\ gate\ delay \\ t_{pd2} = 10\ gate\ delay \Rightarrow \max\ t_{pd} = 10\ gate\ delay \\ t_{pd3} = 8\ gate\ delay \end{cases}$$

$$T_c \geq t_{pd} + t_{setup} + t_{pcq} = \left(t_{1-bit\ FA} + t_{1-bit\ \mathrm{PG}} + 2 \cdot t_{valency2-cells}\right) + t_{setup} + t_{pcq}$$