## LabB -FFT

Github link: https://github.com/aahwu/AAHLS\_labB

Overall system:

本次lab實作了FFT演算法，相較DFT演算法利用了對稱性來加速運算。一開始提供了整個演算法跑在software(CPU)上的版本，但若需要充分的使用硬體加速，就必須改寫其架構。

Project設定:

Device: xc7z020-clg400-1

Period: 10ns

1. Bit reverse

由於FFT使用了butterfly的結構，可以對一開始輸入的資料重新排列，使得之後的FFT中，資料能夠in-place做計算。

下圖是8bit FFT的bit reverse範例:

一張含有 桌 的圖片

自動產生的描述

於是此function可以分為兩部分:1. Address的計算，2. 資料的移動

* 1. Address的計算:

Source code如下圖所示:

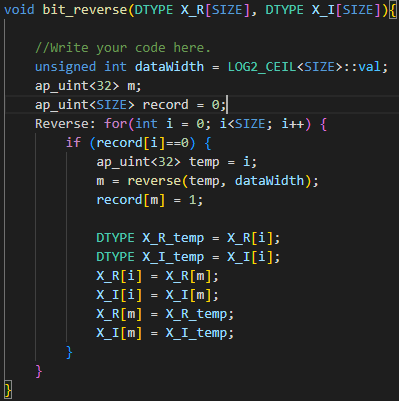
一張含有 文字 的圖片

自動產生的描述

當中的計算皆使用logical operator做運算來優化硬體資源。

* 1. 資料的移動

Source code如下圖所示:



其中有兩個data dependency的部分。一是同個array在相同的cycle中需要寫兩次(WAW)，二是同個array在不同的cycle可能有RAW。但根據實際上演算法的規律，這些dependency都可以忽略，於是我們可以下一些pragma:

一張含有 文字 的圖片

自動產生的描述

但最終的II仍然無法降到1，原因是load operation需要2個cycle，而又因為需同時讀取要互換的資料，用完了2個port，故無法有效地做pipeline。而因為reverse後的address是random access，故也無法使用array partition來增加頻寬。

1. Software implementation

FFT的核心演算法的source code如下圖所示:

一張含有 文字 的圖片

自動產生的描述

在這個架構下，因為最裡層的兩個loop皆有variable bond，故無法有效地做優化。cos和sin的計算可以使用LUT來優化。

1. Hardware acceleration

接下來，stage再分為三個部分:first stage, last stage, other stages，另外stage和stage之間都使用了buffer。其中first stage和last stage因為參數都固定了，可以很有效地做優化，且可以幫助我們了解在general的case下stage怎麼運作的。

下圖是first stage和last stage的source code:

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

因為有了buffer，所以沒有了RAW的問題，且因為這兩個部分memory access是規律了，故可以使用unroll和array partition來進一步增加throughput。

再來是其餘的stage。為了讓tool做優化，可以將內層的兩個loop合併。因為實際上的operation數量是固定的，便能夠做pipeline。

下圖為source code:

一張含有 文字 的圖片

自動產生的描述

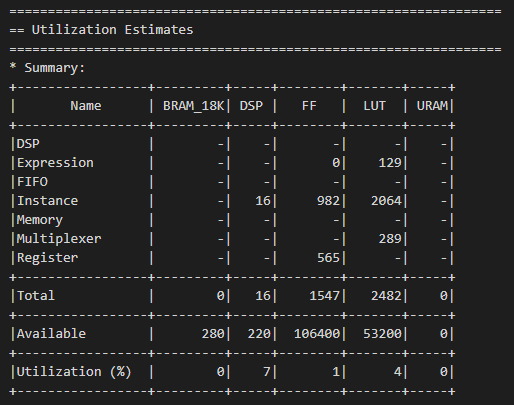
若一開始bit reverse的部分可以不需要in-place而可以也另外用buffer，則可以進一步優化code如下:

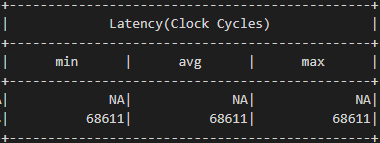
一張含有 文字 的圖片

自動產生的描述

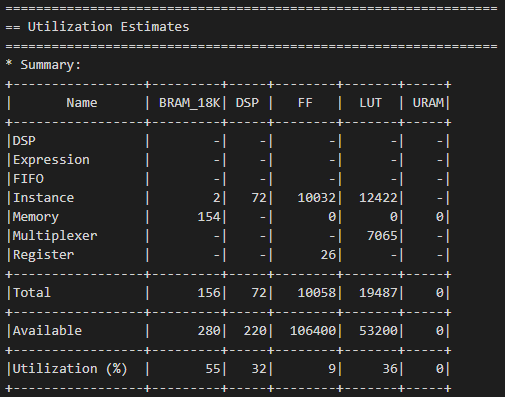
這樣便將II降為1且能夠另外做unroll。

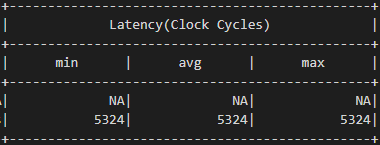
1. Comparison
   1. Software resource & timing





* 1. Hardware resource & timing





可以看到除了因為使用了buffer和array partition所以使用了許多BRAM，其餘的資源約增加了4~9倍，而Latency卻縮短了近13倍。由此可知此為合理的加速。