112-1-LSV PA1

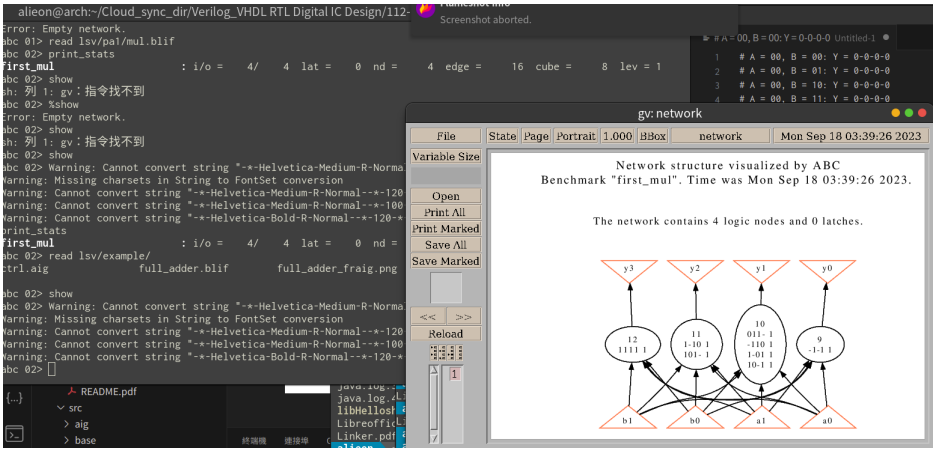
- Github Page: **https://github.com/NTU-ALComLab/LSV-PA** (https://github.com/NTU-ALComLab/LSV-PA)
  - 檔案要放在 `lsv/pa1` 當中，並且壓縮成 `.tgz` 的形式。
  - Exercise 4 則是要將檔案放在 `src/ext-lsv` 當中。
  - 要先自己 fork 後，在本地端改完東西、推上去後送 Pull request.

Getting Familiar with Github

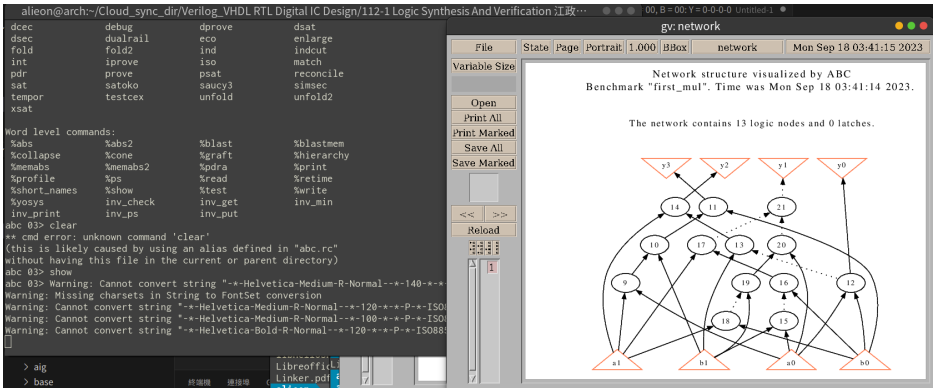
- Forked, modified and have pulled request.

Using ABC

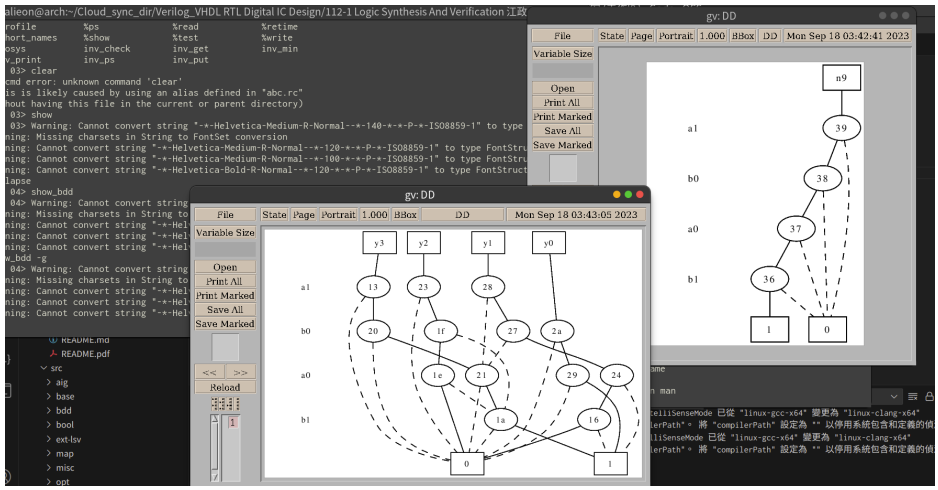
- Statistics & Graph directly after *visualize the network structure (command “show”)*:



- After *convert to AIG (command “strash”)* and *visualize the AIG (command “show”)* :



- After *convert to BDD (command “collapse”)* and *isualize the BDD (command “show bdd -g)* :



Issue Report

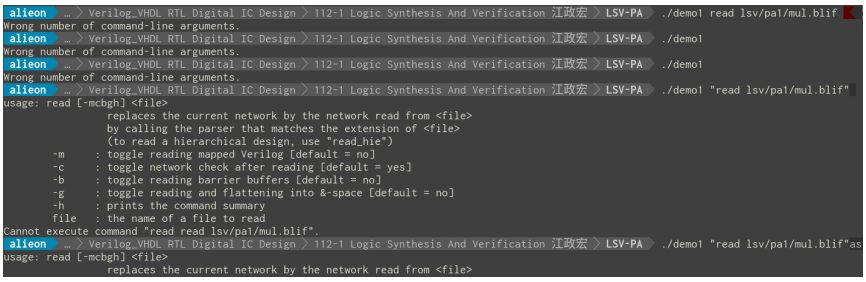
- 在 `lecture01-abc.pdf` 講義中提到 `graphviz` 的 `dot` 功能需要安裝在同一目錄下，也提到 `GSview` 和 `Ghostscript` 需要被安裝，但是關於後兩者的安裝過程其實有改善空間 ：
  - 我的個人環境： `Linux arch 6.5.3-arch1-1 #1 SMP PREEMPT_DYNAMIC Wed, 13 Sep 2023 08:37:40 +0000 x86_64 GNU/Linux`
  - 在 ABC 中，預設會使用 `gv` package:

```
abc 02> print_stats
first_mul                 : i/o =    4/    4  lat =    0  nd =     4  edge =     16  cube =     8  lev = 1
abc 02> show
sh: 列 1: gv：指令找不到
```

    - 原因是在 `graphviz` 中有進行 `import gv` 以及其他等義的 library 引入動作
  - 但這對於非 GNE `gv` 用戶，不一定會知道 `gv` Library 能來自於 `GSview` (in Windows, 雖然有 Linux 版但在各個發行板的 issue 有點多)或 `Ghostscript` 。
    - 而且大部分人有類似的 Postscript Viewer 的需求時，也會是直接安裝 **gv** 套件 ，甚至在部份發行板中還不需要對 pip 也安裝 `gv` 。(e.g: 我使用的 Arch Linux 版本)
  - 因此我個人建議， 在讓大家裝 ABC 的環境時，可以直接讓大家安裝 `sudo apt install gv` 或 `sudo pacman -S gv` 會讓大家少更多對 abc 內部的 script

進行人肉 debug 的時間，也不會讓使用者必須要對 Linux 的各種 utilities 還得花時間去做深度了解。
    - 同樣地， `sudo apt-get install graphviz` or `sudo pacman -S graphviz` 也可以讓大家的 path 預設就安裝 `dot` library.

- 關於一開始如何 `make` ABC 的部份，其實可能需要在 `NTU-ALComLab/LSV-PA/README.md` 多一點 Guide 。
    - 在 `NTU-ALComLab/LSV-PA` 中沒有註記外， `berkeley-abc/abc` 一開始引導使用者進行 `make` 時，給出的 Hello world 示例是請使用者直接將整個 repo `make libabc.a`
    - 由於在 `src/demo.c` 中提供的引數是有限制數量跟讀取方式的，使得我一開始在餵作業中的各種 `Read` 或是其他指令時會有引數無法正常輸入的情況發生：

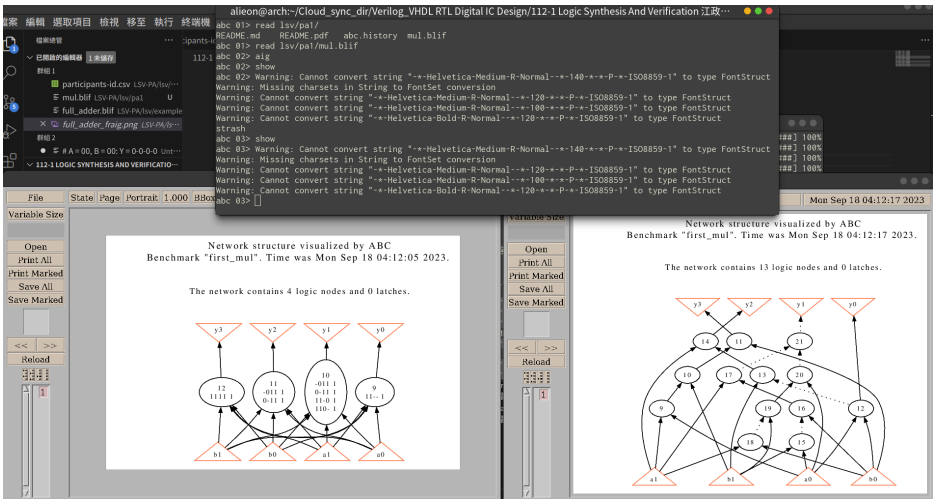


ABC Boolean Function Representations

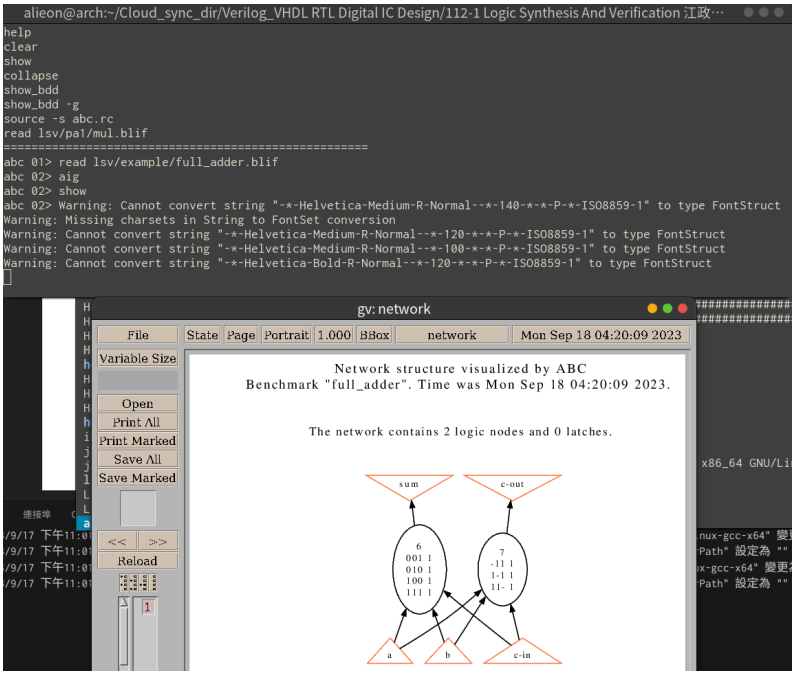
Compare

- 以下來自 **https://people.eecs.berkeley.edu/~alanmi/abc/** (https://people.eecs.berkeley.edu/~alanmi/abc/) 都會下稱 *~alanmi/abc/*

- Compare AIG & Structurally AIG:



  - LHS: AIG, RHS: Structurally AIG
  - 根據 *~alanmi/abc/*， `aig` 指令: *local functions of the nodes*，因此在本範例的前提下，只能將 Truth table 內部所映射的 directed graph 包成四個 node——對應我們一開始輸入的 `.name` 節點數量，實際邏輯實現的方式及算法在圖中並沒有辦法看出相關資訊。
    - 在我的環境下，我實作 `lsv/example/full_adder.blif` 的 `aig` 指令時亦會得到以下圖片：



    - 上述圖片能看出，`aig` 和在 `lsv/example` 底下的 `full_adder_fraig.png` (使用 `fraig`，`show` 能製作出來同一張圖) 以及

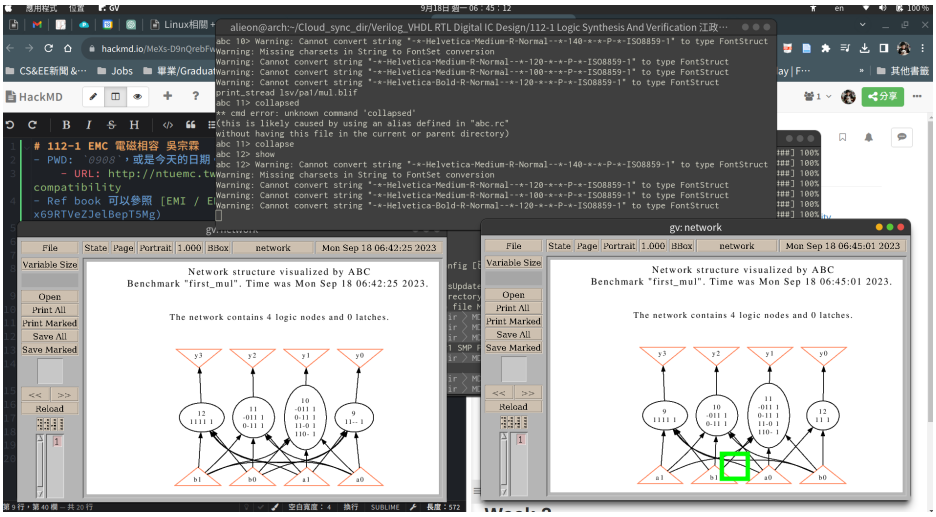
`full_adder_strash.png` 並不同，因此可以排除指令本身有問題，上圖和本節的第一張圖片的 LHS 處，確實就是 `aig` 正常生成的圖片。

- 而 `strash` = structurally hash 方法，則能夠看出有向圖被結構化的樣子，並且被分成 13 個邏輯節點。
- Compare both in `print_stats`：

```
abc 13> read lsv/pa1/mul.blif
abc 14> print_stats
first_mul                 : i/o =    4/    4  lat =    0  nd =     4  edge =     16  cube =     8  lev = 1
abc 14> aig
abc 14> print_stats
first_mul                 : i/o =    4/    4  lat =    0  nd =     4  edge =     16  aig  =    14  lev = 1
abc 14> read lsv/pa1/mul.blif
abc 15> strash
abc 16> print_stats
first_mul                 : i/o =    4/    4  lat =    0  and =    13  lev =  4
```

  - 無論從上面的有向圖、或是從 `print_stats` 得到得資訊，`strash` 都會將層 (lev) 分的更細。

- Compare BDD (LHS) & Collapsed BDD (RHS):



  - 綠框框處可以看見有一部份的 edge 在 collapsed 被最佳化了，其餘大致不變。
  - Compare both in `print_stats`：

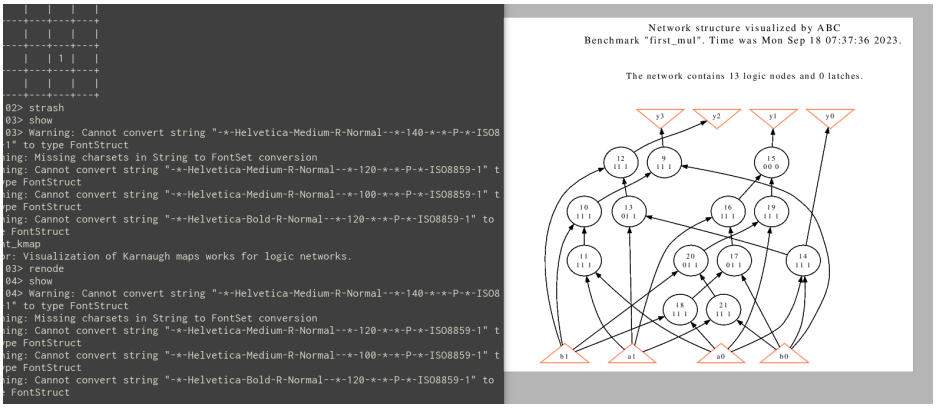
```
abc 16> read lsv/pa1/mul.blif
abc 17> bdd
abc 17> print_stats
first_mul                 : i/o =    4/    4  lat =    0  nd =     4  edge =     16  bdd  =    17  lev = 1
abc 17> read lsv/pa1/mul.blif
abc 18> collapsed
** cmd error: unknown command 'collapsed'
(this is likely caused by using an alias defined in "abc.rc"
without having this file in the current or parent directory)
abc 18> collapse
abc 19> print_stats
first_mul                 : i/o =    4/    4  lat =    0  nd =     4  edge =     14  bdd  =    14  lev = 1
```

    - 從這邊也可以看出 edge 的數量在 collapsed 的部份變少了。

- 根據 *~alanmi/abc/*，關於 `strash` = structurally hash 以及 `collapse`
  - `strash` 會使用 one-level structural hashing 將 node boundry 破壞掉，此時產生的 AIG 集合變換並不會改變 Latch 的數量（如果有呼叫的話）

        - 而這點也解釋了為何 `aig` 指令無法對本範例做出太大的圖像更動，因為 invert 及邏輯實現的部份都包在 node 當中、並未被拆出來。
    - `collapse` 則是使用 BDD 表達方式，用 recursive 的方式將 Fanin 和 Fanout nodes 重劃成 CI / CO (Collapse In / Out)，能夠在不破壞 node boundry 的狀況下整理新的邏輯，但缺點就是複雜度問題，會導致它無法在過大的電路實現邏輯合成。

Convert to Logic Network SOP from `strash` AIG

- 

    - 其中每個 node 裡面的語法， `1` 代表 non-inverted, `0` 代表 inverted, 且數字間的空白代表的是 input / output, 如同 `.blif` 檔案中寫的一樣。