5G通訊演算法工程師培訓課程

Course Overview

2021

Outlines

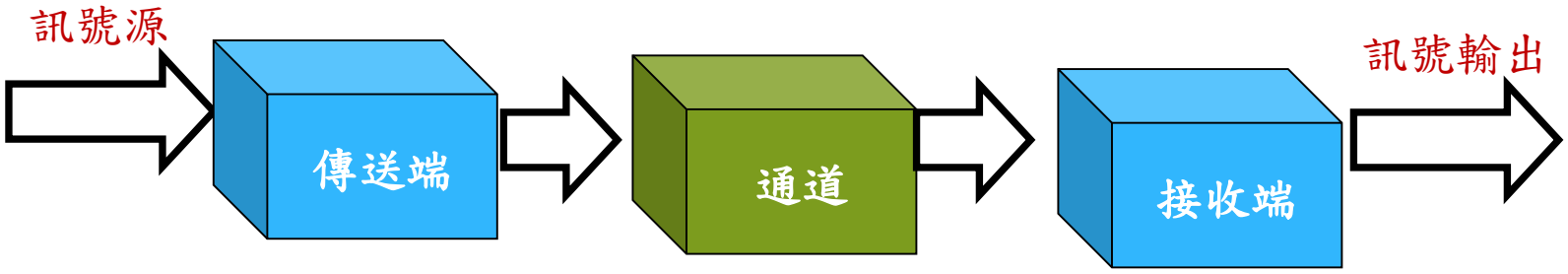
* FPGA/Verilog 數位邏輯電路設計
  * Introduction to FPGA
  * Introduction to Verilog
* **通道編碼技術(Channel Coding)**
* OFDM 與多通道通訊系統

大綱

* 無線電波的傳輸與模型
* 通道編碼的理論
* 錯誤檢測碼
* 線性區塊碼
  - 漢明碼
  - BCH碼
  - 低密度奇偶檢查碼
  - Polar碼

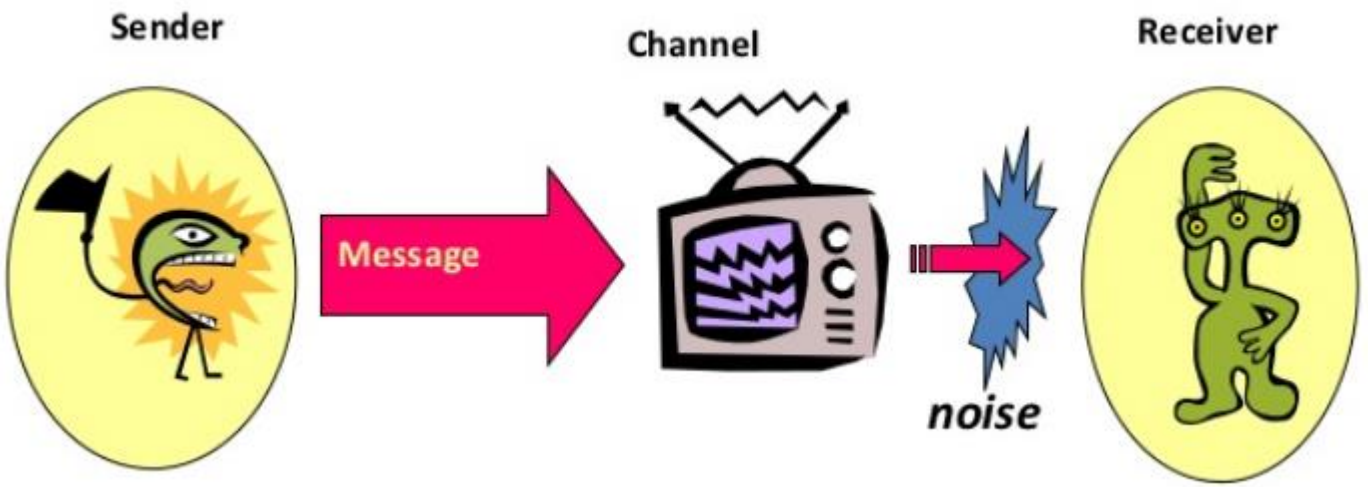
通道 (Channel)

* 通道是用來傳遞訊號的媒介
  - ✓ 在通訊上表示在接收端之前所經過所有路徑以及雜訊的總和效應。
  - ✓ 在通訊系統中，通道包含了
    - 有線通道，如網路線
    - 無線通道，如無線電波傳輸的空間



通道 (Channel)

* 但因物理環境因素，這些通道對於訊號的傳遞造成或多或少的限制，假設存在一個完美的通道能將訊號從傳送端完整無暇的傳遞到訊號的接收端，相較之下，現實生活中的**通道**就是一種對於訊號傳遞的干擾。



https://www.slideshare.net/forckren/3-models-of-communication

Effect of Noise

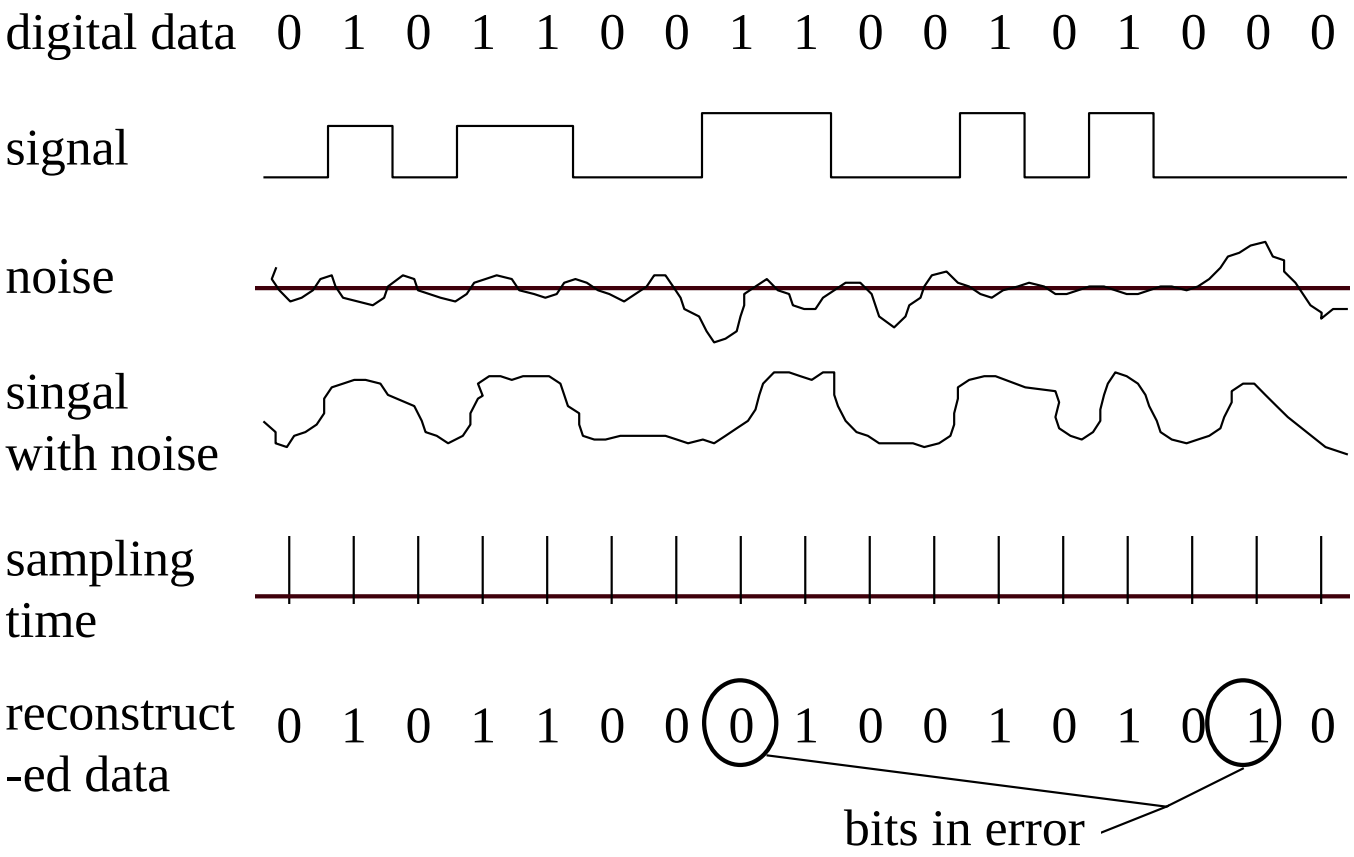


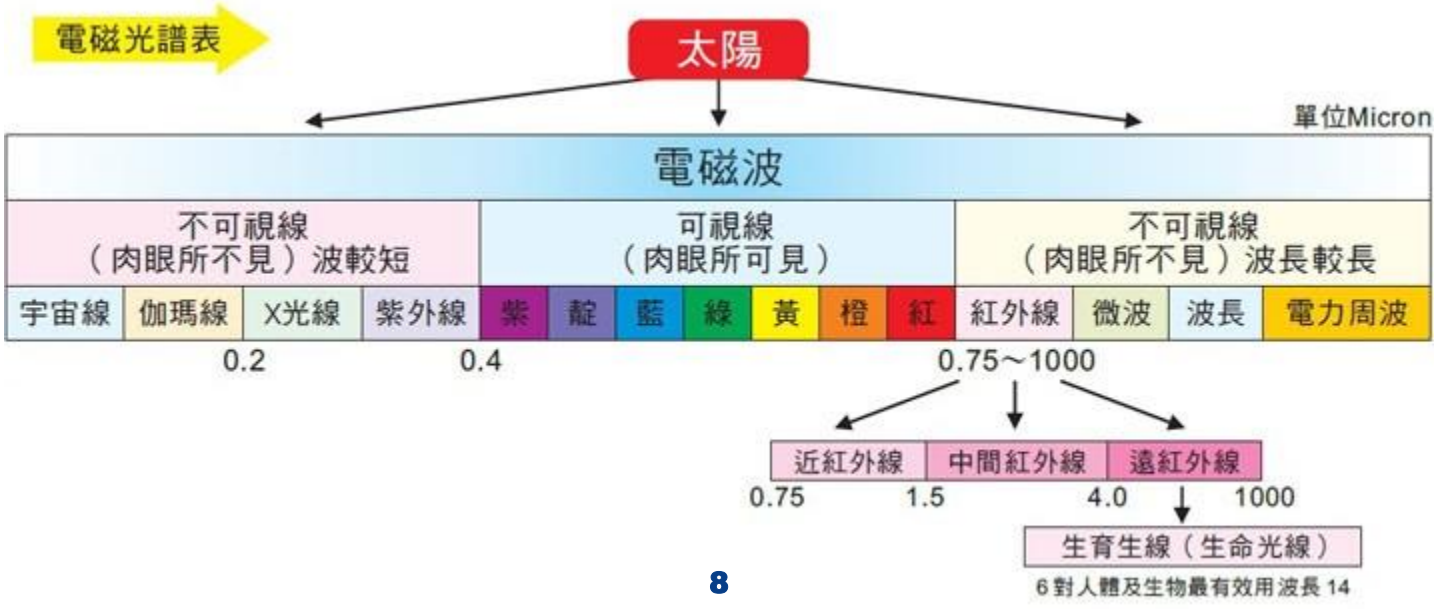
Figure-1. Effect of noise on a digital signal

無線電的傳輸

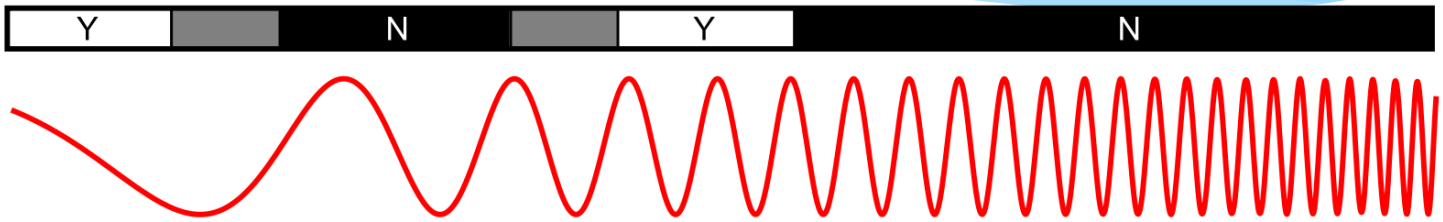
* 無線電傳輸的原理在於藉由變動的電場及磁場交互感應而產生的電磁波，傳遞至遠端。
* 電磁波的傳遞無需介質，真空中也可以傳送。
* 在行動通訊系統中，無線電傳輸所扮演的功能便是手機與基地台之間的資訊傳遞。
* 無線電波的傳遞可能會暫時受障礙物的屏蔽，導致通訊品質不良。為了能在傳輸情況難以控制的大自然環境中，確保傳遞內容的正確性與有效性，許多相關技術被發展出來以解決問題。

電磁波頻譜

* 宇宙裡自然存在的所有電磁波，我們稱為「電磁波頻譜（Spectrum）」，由圖中可以看出中間的部分是光（Light），包括：紅外光（Infrared，IR）、可見光（人類肉眼可以看見的光）、紫外光（Ultraviolet，UV），因此光是一種電磁波；右邊為頻率更高（能量更高）的電磁波；左邊為頻率更低（能量更低）的電磁波，由於頻率較低的電磁波比較安全，而且具有良好的繞射特性，因此適合用來做為無線通訊使用。



電磁波頻譜


能否穿透
地球的大氣層
Y
N
Y
N
輻射種類
波長（m）
無線電
10^{3}
微波
10^{-2}
紅外線
10^{-5}
可見光
0.5×10^{-6}
紫外線
10^{-8}
X 射線
10^{-10}
伽馬射線
10^{-12}
波長的尺度大小
約相當於
建築高度
人類的身高
蝴蝶
針尖
原蟲
分子
原子
原子核
頻率（Hz）
10^{4}
10^{8}
10^{12}
10^{15}
10^{16}
10^{18}
10^{20}
此溫度的物體
所發出的輻射中
最强烈部份的波長
1 K
−272 °C
100 K
−173 °C
10,000 K
9,727 °C
10,000,000 K
~10,000,000 °C

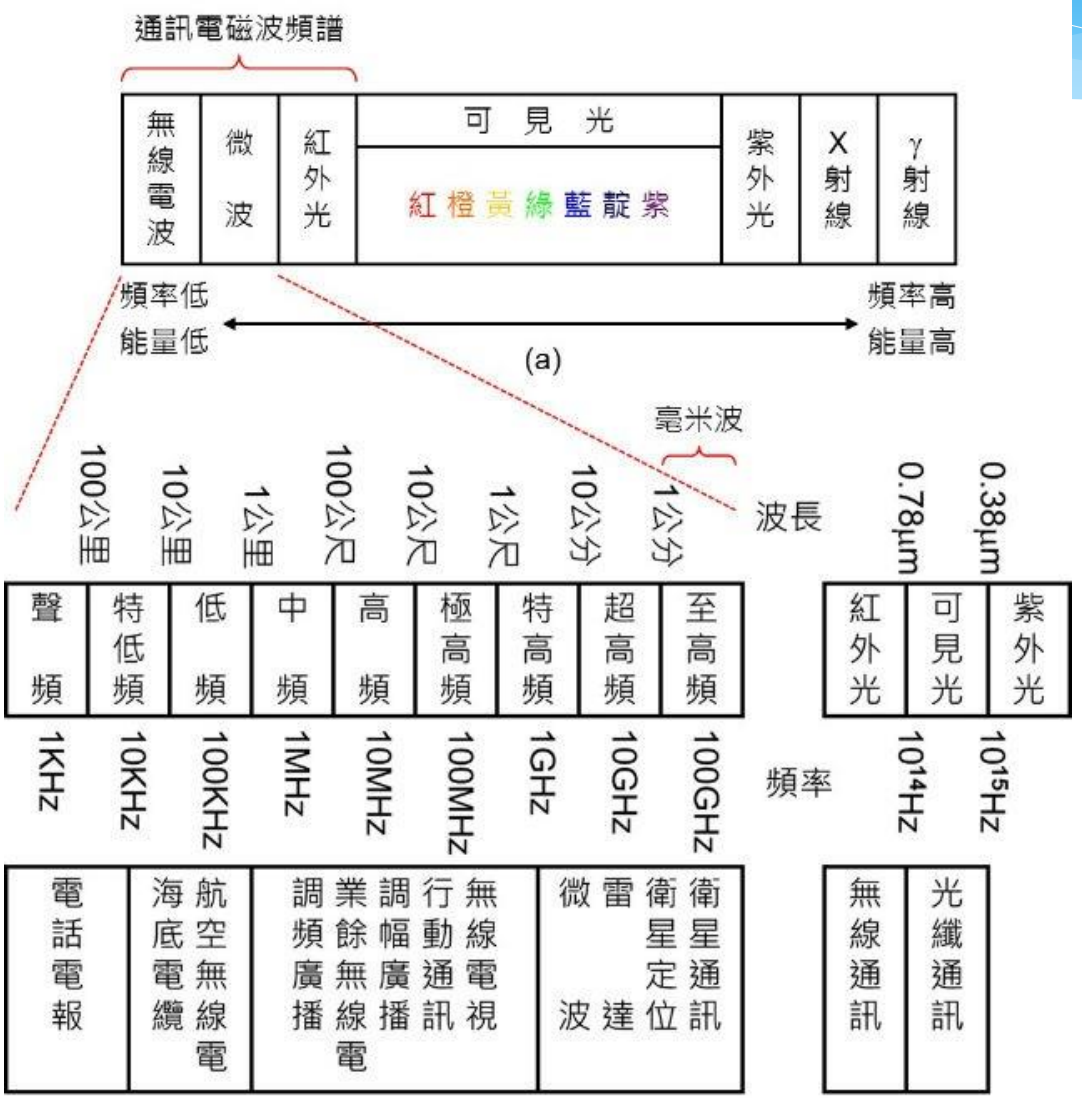
無線通訊的電磁波

* 頻率大約 100G~1GHz 的電磁波：通常應用在衛星通訊、衛星定位、雷達與微波通訊等，而頻率 30GHz 以上（相當於波長 10 毫米以下）的電磁波稱為「毫米波（Millimeter Wave）」，目前有公司計劃應用在 5G 的通訊系統中。

* 頻率大約 100M~1MHz 的電磁波：通常應用在無線電視、行動通訊（GSM / GPRS）、調幅廣播（AM）、業餘無線電、調頻廣播（FM）等。

* 頻率大約 100K~1KHz 的電磁波：通常應用在航空無線電、海底電纜、電話與電報等。

無線通訊的電磁波


通訊電磁波頻譜
無線電波
微波
紅外光
可見光
紅 橙 黃 綠 藍 靛 紫
紫外光
X射線
γ射線
頻率低
能量低
(a)
頻率高
能量高
毫米波
100公里
10公里
1公里
100公尺
10公尺
1公尺
10公分
1公分
波長
0.78μm
0.38μm
聲頻
特低頻
低頻
中頻
高頻
極高頻
特高頻
超高頻
至高頻
紅外光
可見光
紫外光
1KHz
10KHz
100KHz
1MHz
10MHz
100MHz
1GHz
10GHz
100GHz
頻率
10¹⁴Hz
10¹⁵Hz
電話電報
海底電纜
航空無線電
調頻廣播
業餘無線電
調幅廣播
行動通訊
無線電視
微波
雷達
衛星定位
衛星通訊
無線通訊
光纖通訊

無線通訊傳遞通道：頻寬

* 頻寬（Bandwidth）是用來傳遞訊號的「頻率範圍」，單位與頻率相同為「赫茲（Hz）」，而且每一對通訊使用者必須使用「不同的頻率範圍」來通話，假設：

  甲和乙使用頻率 **900~900.2**MHz 的電磁波通話（頻寬 900.2-900=0.2MHz）；
  丙和丁使用頻率 **900.2~900.4**MHz 的電磁波通話（頻寬 900.4-900.2=0.2MHz）；

* 此時我們說這個通訊系統的語音通道頻寬為 0.2MHz。
* 手機並不會分辨到底是誰和誰在通話，而是接收某一個「頻率範圍（頻寬）」的電磁波訊號，因此甲與乙通話時手機都接收頻率 900~900.2MHz 的電磁波，丙與丁通話時手機都接收頻率 900.2~900.4MHz 的電磁波，換句話說，所有的通訊元件都是「只認頻率不認人」，而且相同頻率範圍的電磁波只能使用一次，不能重覆使用，否則會互相干擾。

頻寬與資料傳輸率的差異

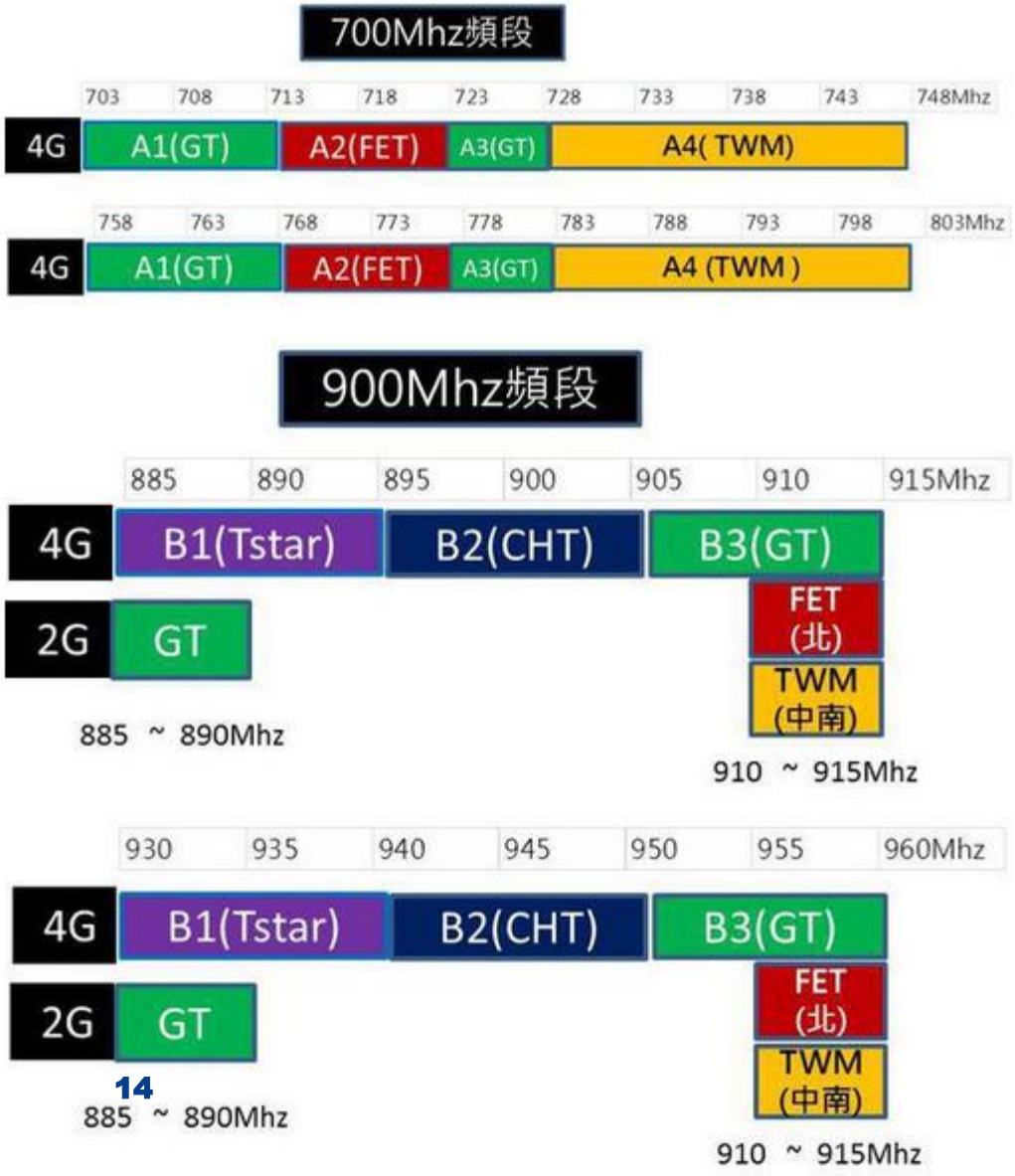
* 「頻寬（Bandwidth）」與「資料傳輸率（Data rate）」的意義很類似，常常讓我們混淆，這裡簡單說明它們之間的差別：

  - 頻寬（Bandwidth）是類比通訊使用的名詞：由圖一可以看出，電磁波是一種連續的波動能量，既然是連續的當然一定是類比訊號，因此「頻寬（Bandwidth）」和它的單位「赫茲（Hz）」指的都是電磁波的物理特性。

  - 資料傳輸率（Data rate）是數位通訊使用的名詞：手機會先將我們講話的聲音（連續的類比訊號）先轉換成不連續的 0 與 1 兩種數位訊號，再經由天線傳送出去。資料傳輸率的單位「每秒位元數（bps：bit per second）」，代表每秒可以傳送幾個位元，也就是每秒可以傳送幾個 0 或 1，例如：1Gbps（1G = 10 億）代表每秒可以傳送 10 億個位元（10 億個 0 或 1）。

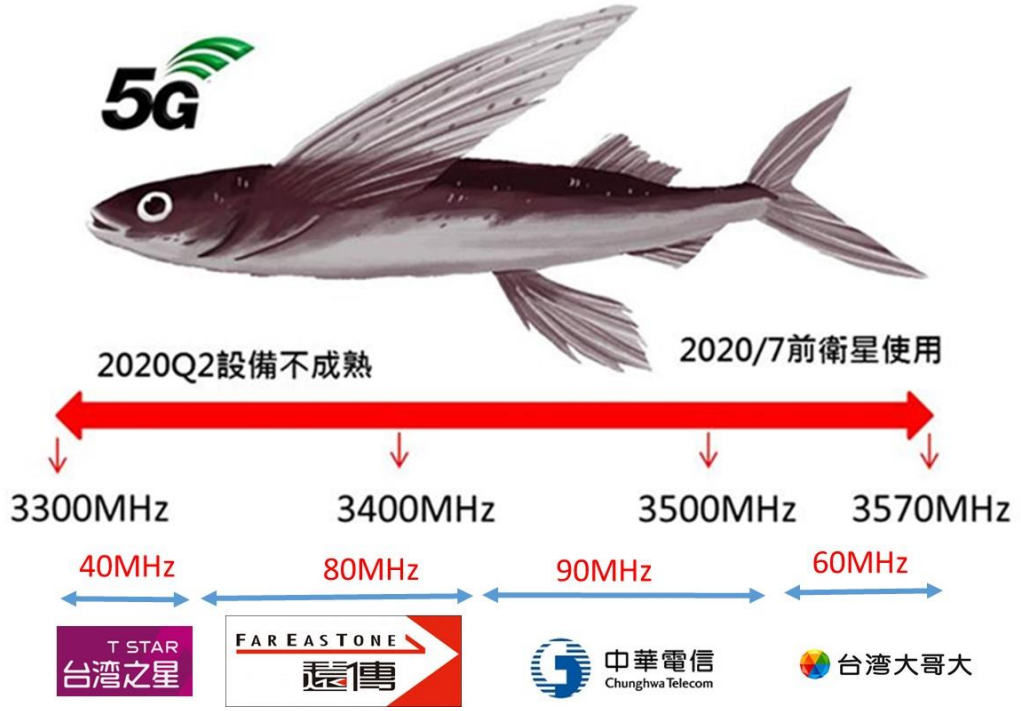
台灣4G頻譜

* 經過各業者自動釋出及交換,目前台灣4G頻譜共有107.4 Mhz頻寬可被使用,其中台灣大哥大擁有31.2Mhz最大,台灣之星5Mhz最小



台灣5G頻譜

* 2020/2/21中華與遠傳合計花了41.1億元取得接近最精華的魚肚位置，台灣首波5G頻譜兩階段總標金總額為1,421.91億元。



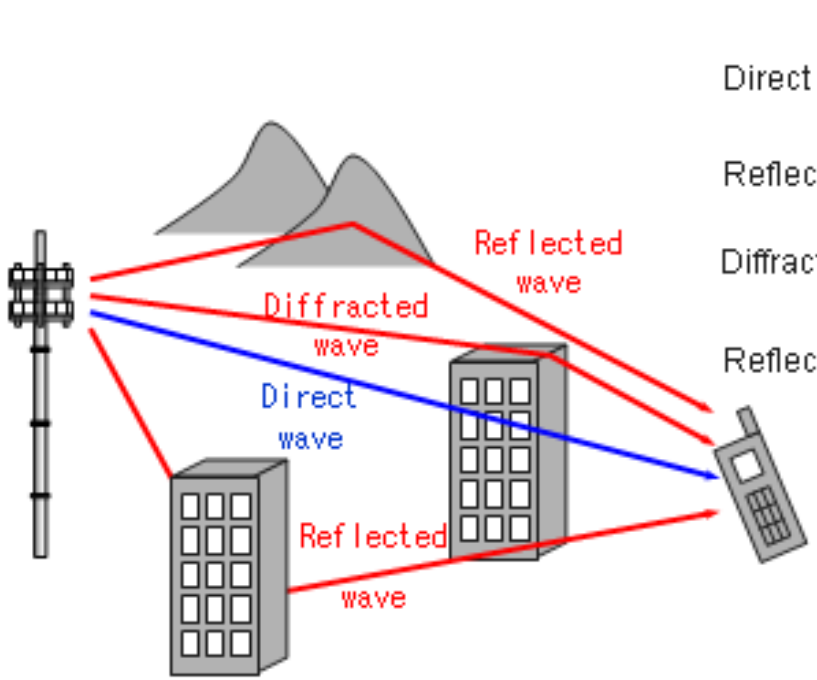
https://tel3c.tw/blog/post/28298

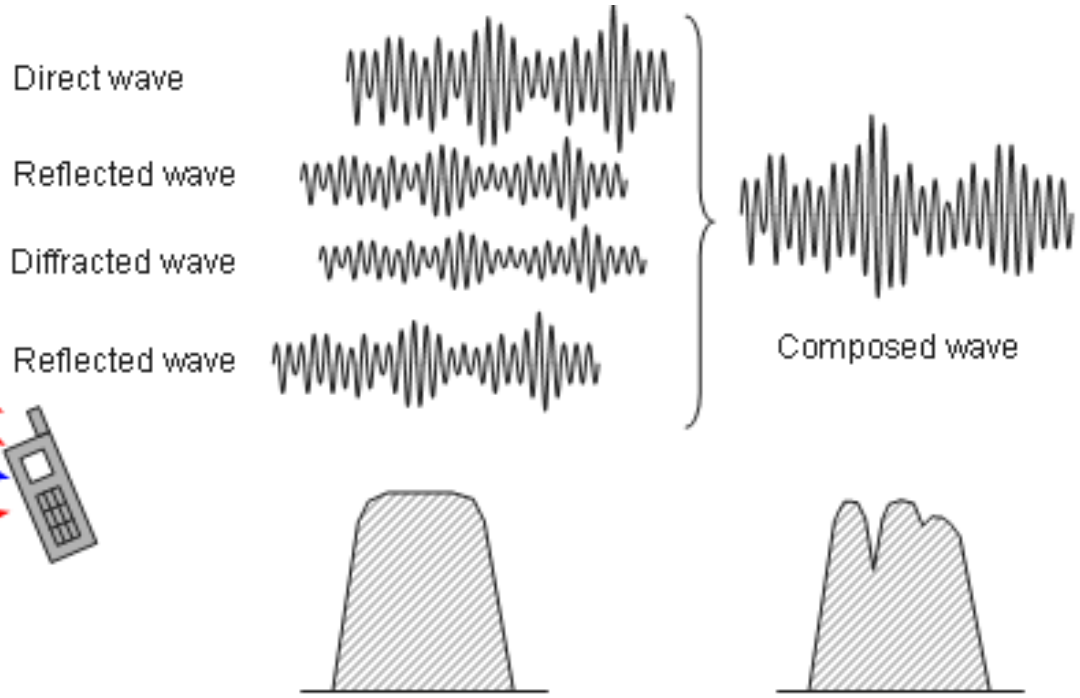
電磁波的傳遞

* 影響電磁波在空氣介質中傳送形態 的三個機制
  - 反射（reflection）
  - 繞射（diffraction）
  - 散射（scattering）
* 為了能猜測環境對電磁波的影響，許多的傳輸模型被提出，希望能預測電磁波傳輸時的特性。

電磁波的傳遞

- 反射（reflection）
- 繞射（diffraction）
- 散射（scattering）

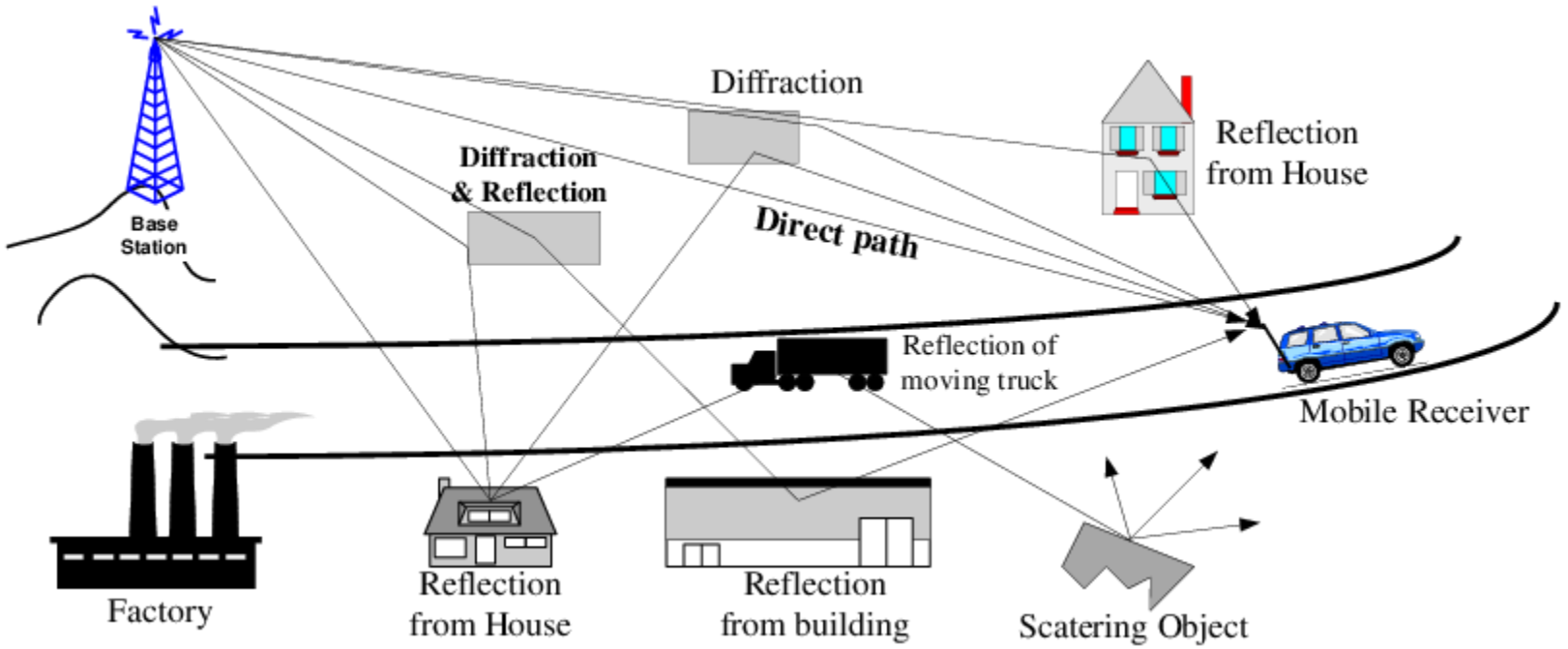




https://www.sharetechnote.com/html/Handbook_LTE_Fading.html

電磁波的傳遞

- 反射（reflection）
- 繞射（diffraction）
- 散射（scattering）



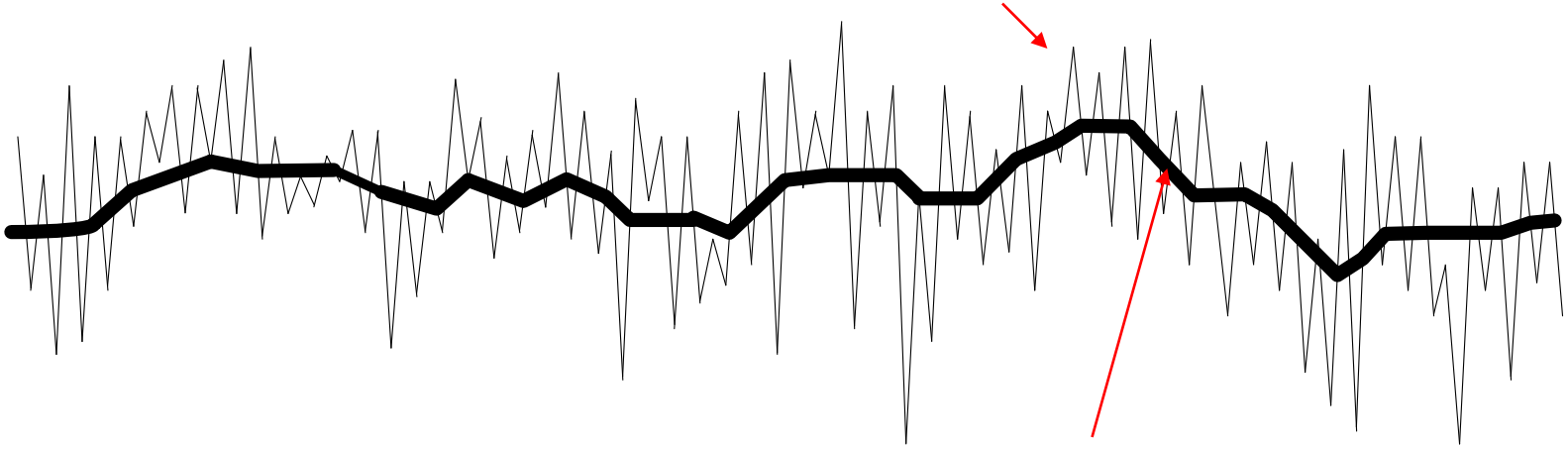
https://www.researchgate.net/figure/Illustration-of-reflection-diffraction-scattering-and-absorption_fig2_228041875

傳播損失（Propagation Loss）

* 當電磁波遇到障礙物（包括空氣與水氣）時，會造成能量消耗。
* 大規模的影響因素
  - 傳播的影響因素是針對收發機之間距離遠近，以及受到外在傳輸環境變動情形下的影響。
  - 主要是預測這個無線電波平均信號的強度，在時間軸上取得一個平均的信號強度進行比較。
* 小規模的信號強度比較
  - 分析在很短暫的時間內，信號強弱改變的情況。
  - 可能是因為週遭物體的移動，或者是如雷雨等自然環境改變所造成的情形。

電磁波信號強弱變化的情形

細線的部分是小規模
信號強弱變化



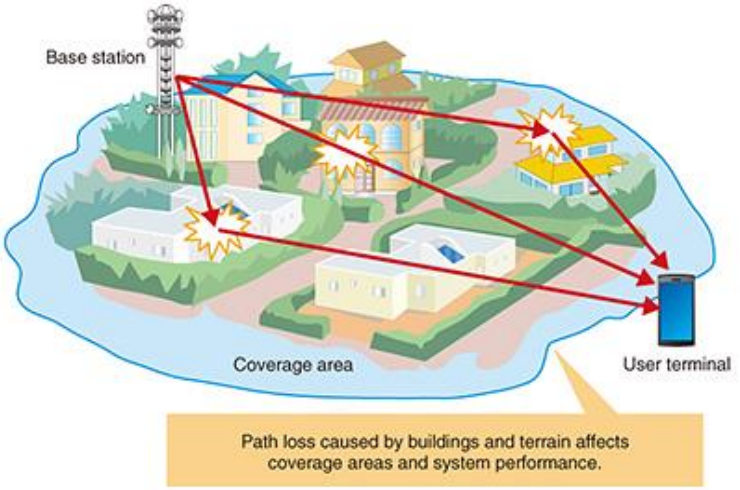
橫向表示距離的移動

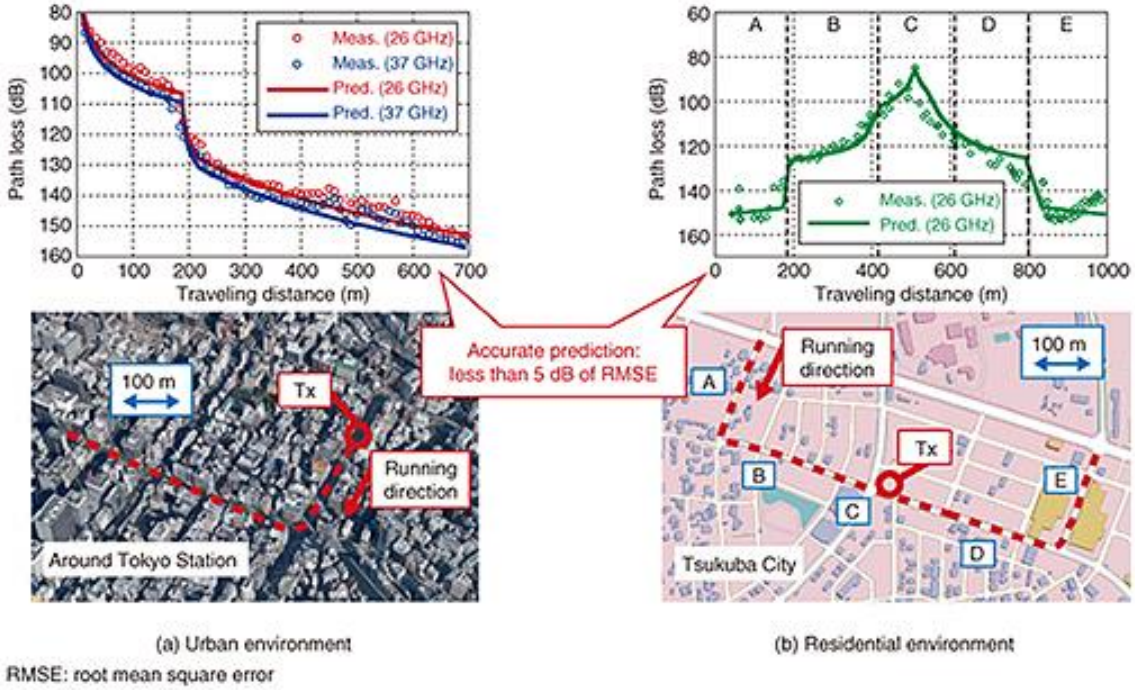
縱向表示信號的強弱大小

粗線的部分是大規模
信號強弱變化

影響無線電波信號強弱的三大類因素

* **路徑衰減（path loss）:**訊號強度隨著傳播路徑增加而衰 減，為發射天線至接收天線之間的功率差值。





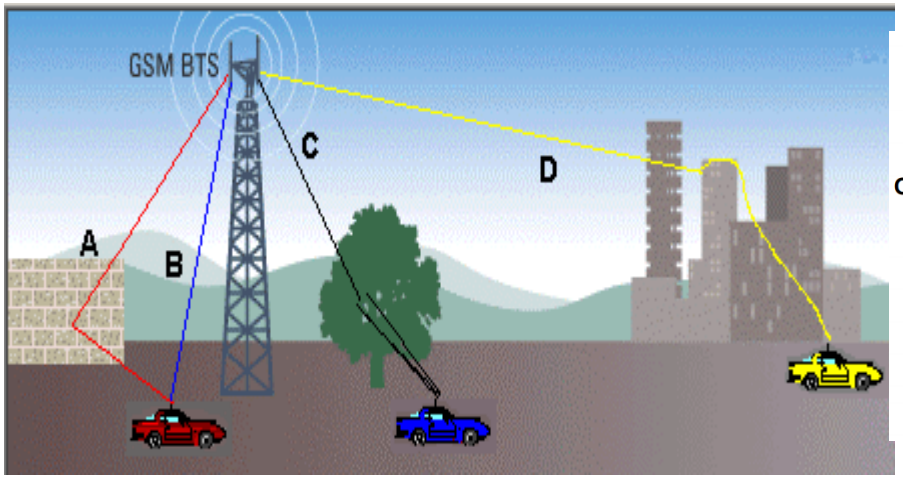
Rx moves along the road about 180 m and then turns the corner.

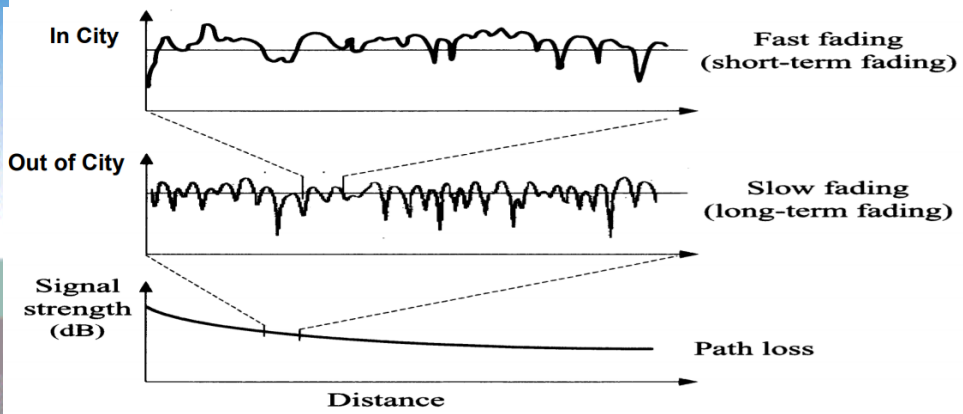
Rx moves from area A to area E.

https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201612ra2.html

影響無線電波信號強弱的三大類因素

* **慢速擾動（slow fading）或遮蔽效應（shadowing）**：遮蔽效應(shadowing effect) 與慢速衰落(slow fading) ，遮蔽效應主要是接收機周圍數十公尺到數百公尺的範圍內有大型障礙物，如大樓、山丘、樹葉及樹木等，對訊號所 造成的遮蔽效應而產生的衰落現象，一般稱為遮蔽效應或 慢速衰落。

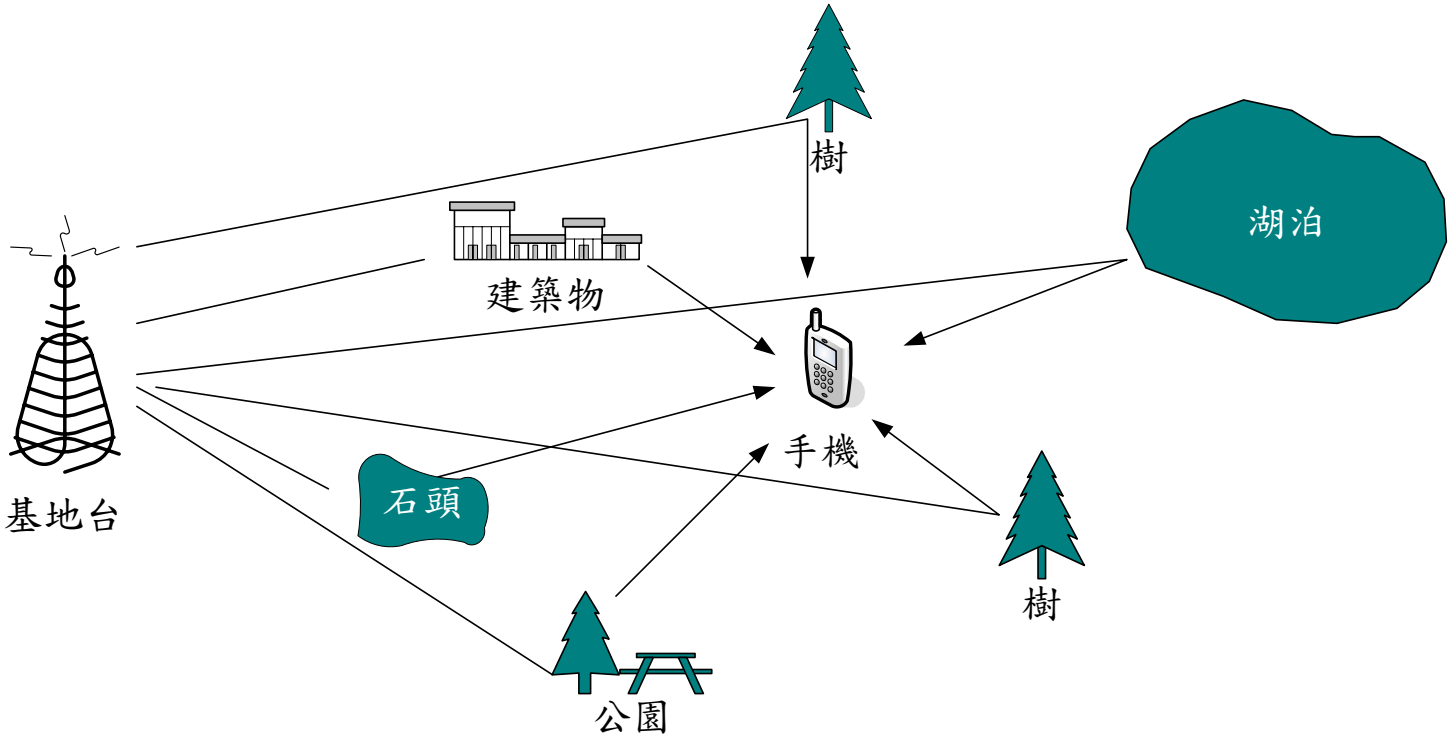




https://rfmw.em.keysight.com//wireless/helpfiles/n5106a/about_fading.htm
http://old.staff.neu.edu.tr/~fahri/wireless_chp2.pdf

無線電傳播影響因素

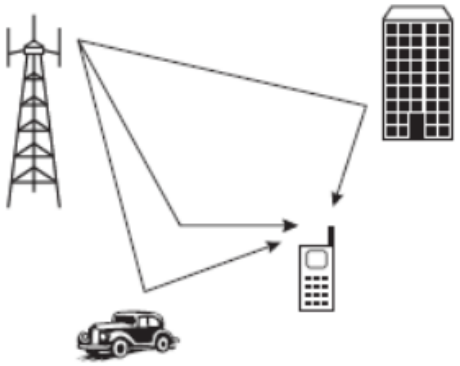
* 當無線電波遇到障礙物時，會產生各種傳導現象，造成許多不同路徑的無線電波被手機所接收，這種現象即所謂多重路徑效應（multi-path fading）。

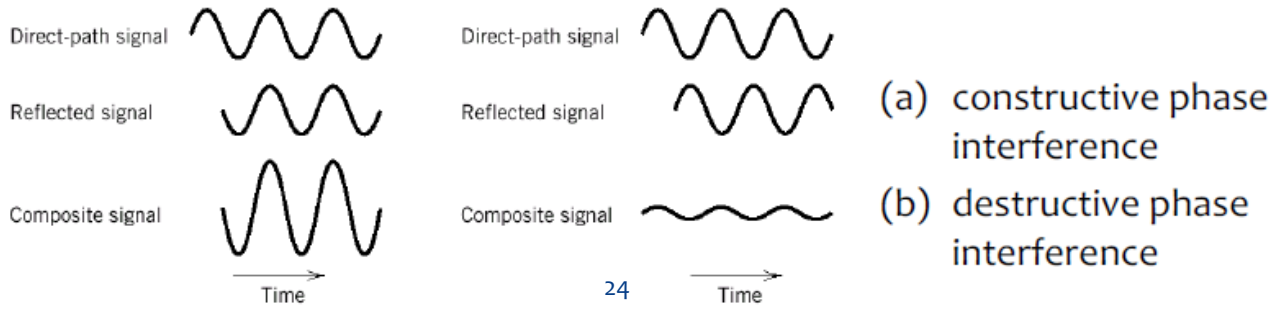


無線電傳播影響因素

* 多重路徑效應（multi-path fading）。

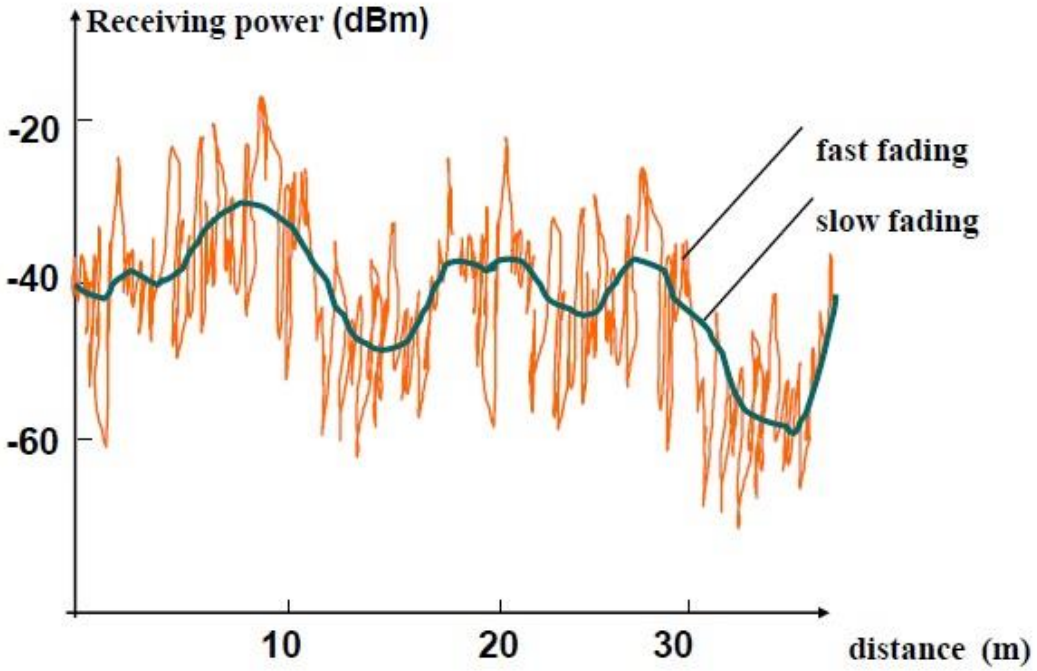
Multipath Propagation





影響無線電波信號強弱的三大類因素

* **快速擾動（fast fading）:**移動台附近的散射體（地形，地物和移動體等）引起的多重路徑傳播信號在接收點相疊加，造成接收信號快速起伏的現象。



http://teletopix.org/3g-wcdma/signal-fading/

無線電傳播影響因素

* 下列哪種通道影響效應，只靠通道無法克服?

(1) 熱雜訊 (2) 路徑損失 (3) 多重路徑效應 (4) 來自別的系統訊訊號的鄰頻干擾

* Ans: 3
  - 多重路徑效應，是指訊號因折射、繞射的效應，在不同時間點到達接收端，造成訊號本身間的干擾，因此，調大訊號功率是無法克服的。

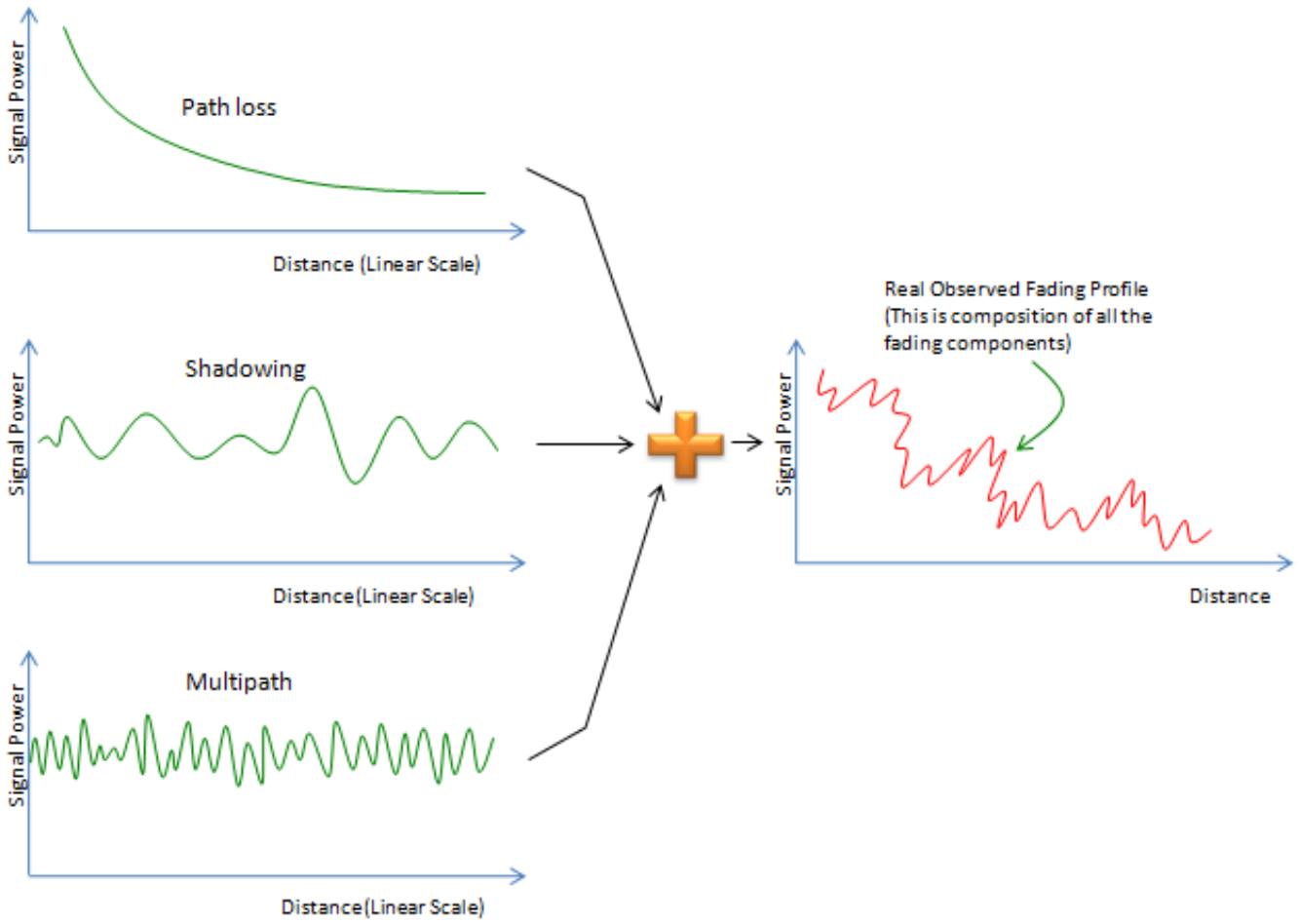
https://sites.google.com/site/yes2comm/kao-ti-xiang-jie/zhong-hua-dian-xin100nian-xing-dong-tong-xun-er

無線電傳播影響因素

Path loss is just gradually decreases but not much of fluctuation.

Shadowing, some fluctuation of the signal strength but the fluctuation frequency over the distance is not that high.

Multipath, fluctuations of signal strength and the fluctuation frequency is also pretty high.



無線電波傳輸模型（Propagation Model）

* 在複雜的無線電波傳輸環境下，若能對無線電波傳輸的情況做個模型化的描述，將有助於系統業者規劃與建置基地台。
* 無線電波傳輸模型是用於“在已知無線電發射機與接收機之間的距離時，預測收到無線電波的信號強度”。
* 但對於短時間內信號改變的情形並不考慮，所以它取的是一個平均信號強度值的動作。
* 建立模型之後，便可進一步利用此模型去預測類似傳輸環境下無線電波信號的強弱。

Okumura-Hata Model

* 用於模擬郊區（urban）大涵蓋範圍基地台的無線電波發射特性

$$L(urban)(dB) = 69.55 + 26.16 \log f_c - 13.82 \log h_{te} - a(h_{re}) + (44.9 - 6.55 \log h_{te}) \log d$$

- L表示傳輸路程中電波強度衰減的情形
- f_c是指這個無線電傳輸的頻率，傳輸的頻率範圍要求在150MHz到1500MHz範圍之內。
- h_{te}是發射機天線的高度，在35～250公尺之間。
- h_{re}是接收機的高度，1～10公尺之間。
- d是發射機與接收機之間的距離。
- a是表示一個校正的因素。

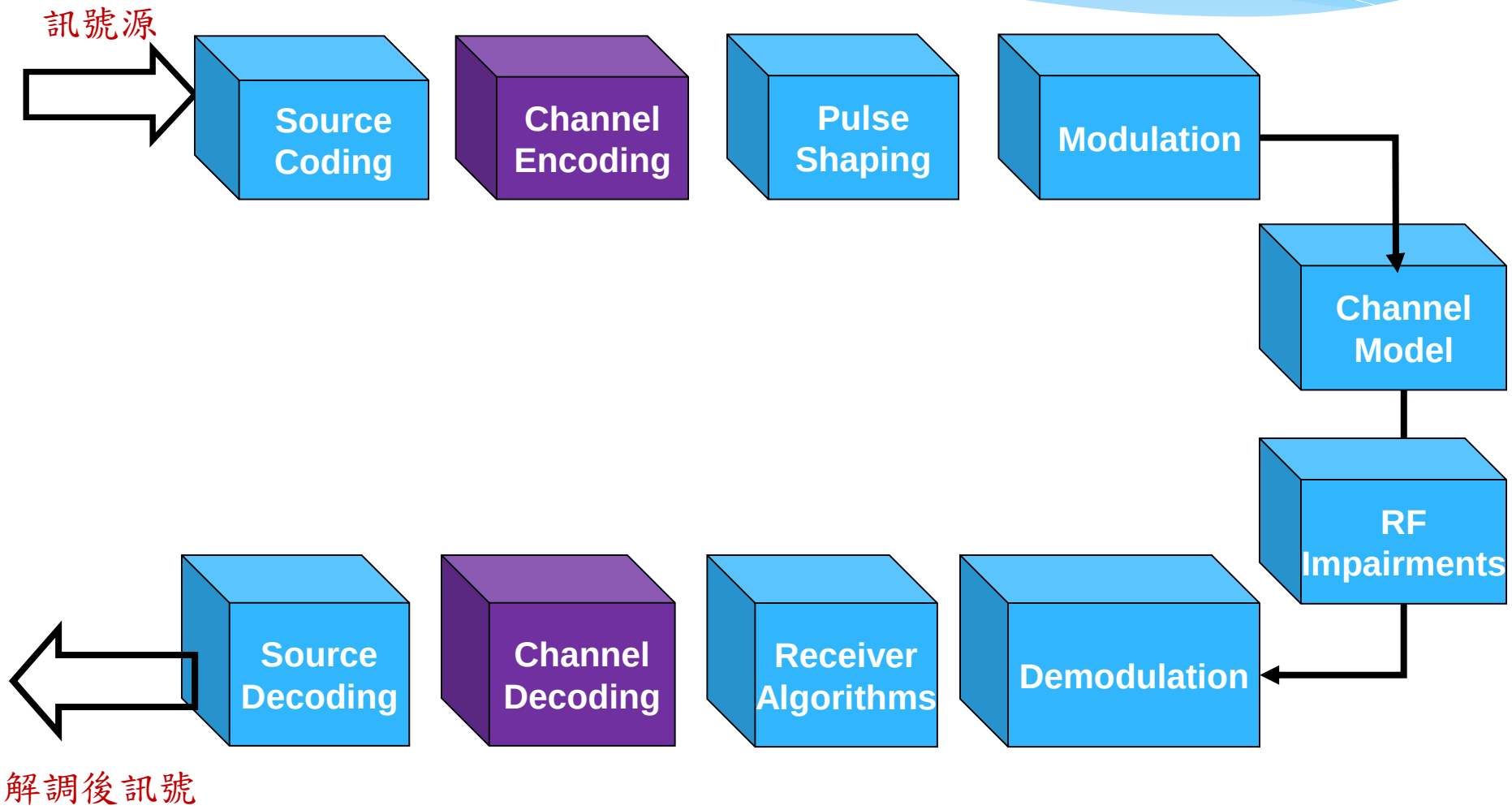
影響無線電波傳輸的因素(1/2)

* 所在地區整體特性，是城市、郊區，或是空曠的開放地區。
* 發送端傳送天線與接收端接收天線的高度。
* 無線電載波頻率。
* 無線電發射機與接收機之間相對的距離（基地台與手機間的距離）。
* 量測地區內建築物平均的高度、街道寬度、建築物之間疏密的程度等。

影響無線電波傳輸的因素(2/2)

* 道路的走向，是否會阻礙電磁波的傳送，或者是一個通道的效益讓信號更容易地傳送。
* 室內量測的房間隔間性質（固定水泥牆或活動隔間板），會影響電磁波信號衰減。
* 發送及接收端之間隔樓層（牆）數、樓層面積、隔間窗戶所佔比例、隔間材質、樓層功能（辦公室、教室或商店）等考慮因素。
* 建築物建材（水泥、木造屋或鐵皮屋）。譬如在台灣有許多鐵皮屋，它是在一些做電磁波信號量測的先進國家，較少用到的建材。

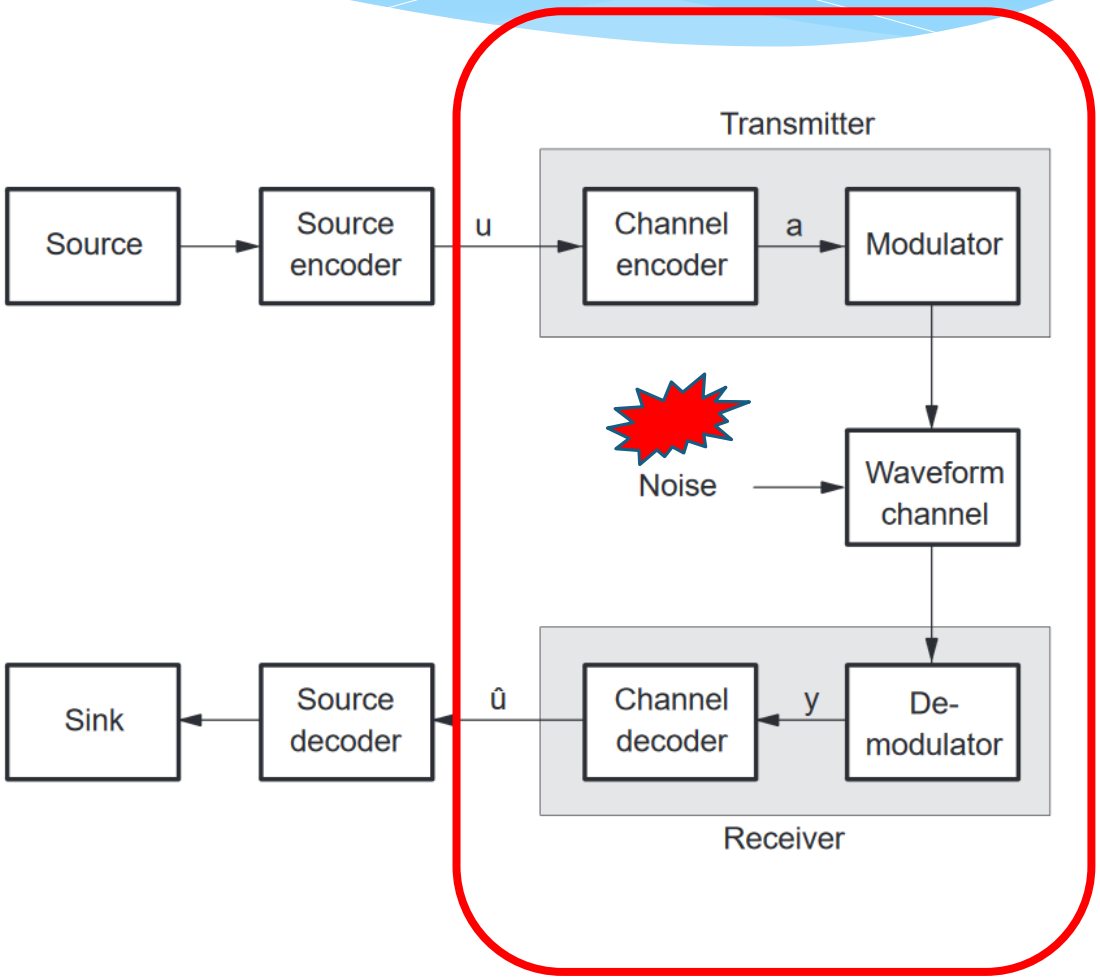
無線通訊系統之通道編碼技術



通道編碼技術

* **Source coding**:
  The messages are compressed in the transmitter side, such that the recovery is possible in the receiver side.

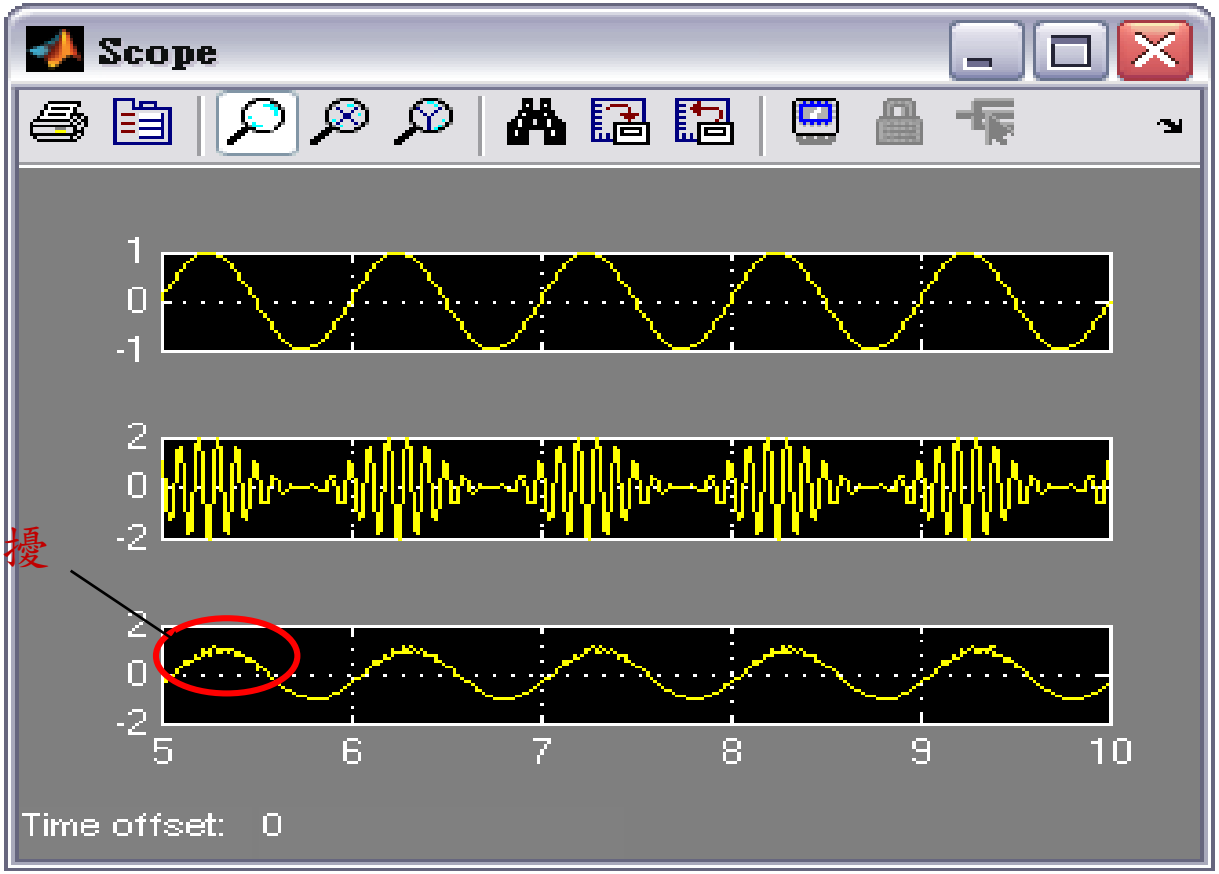
* **Channel coding**:
  Redundancy is added to the information in the transmitter side, such that the receiver can correct errors due to transmission over a noisy channel. Thus reliable communication is possible between the transmitter and receiver.



https://webdemo.inue.uni-stuttgart.de/webdemos/03_theses/chCoding/index.php?id=1

無線通訊系統之訊號示意圖

- 以Matleb Scope顯示執行模擬結果，此範例是DSB AM 的調變和解調後的時域波形。



訊號源

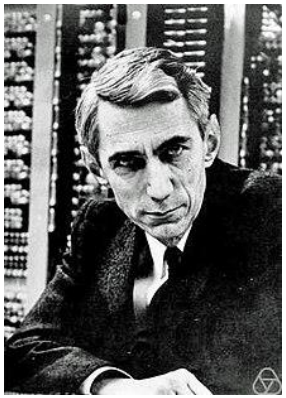
DSB AM調變後訊號

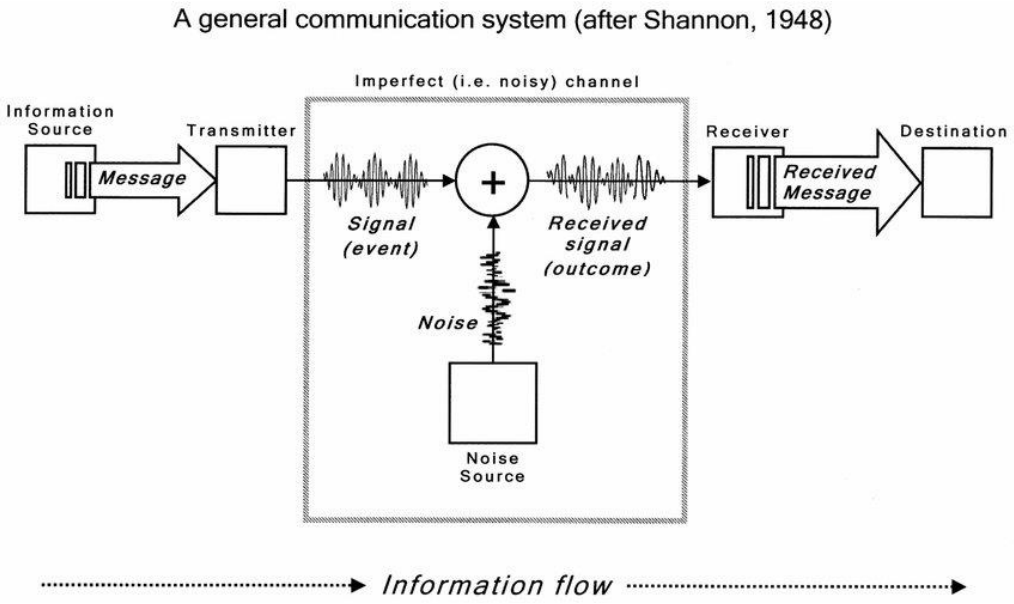
雜訊的干擾

解調後訊號

通訊編碼歷史(1/2)

* 1948年10月Shannon於《貝爾系統技術學報》發表論文
  - 通訊的數學理論：A Mathematical Theory of Communication 作為現代資訊理論研究的開端。
  - 日後被稱為「**資訊理論之父**」。





https://en.wikipedia.org/wiki/Claude_Shannon
https://www.researchgate.net/figure/A-general-communication-system-after-8-to-which-Shannon-applied-his-Information_fig1_317436420

通訊編碼歷史(1/2)

* 1949 年， Golay首先提出 (23, 12, 7) **Golay code** 。
* 1950 年，漢明(Hamming)首先發表漢明碼**Hamming Code**。
* 1950 年代末期到1960年代初期， Bose、Ray-Chaudhuri 與 Hocquenghem三人發展出 **BCH碼**，是編碼理論尤其是糾錯碼中研究得比較多的一種編碼方法。
* 1955年，Elias首先提出**Convolutional code**。
* 1960年， Irving S. Reed and Gustave Solomon發明**Reed-Solomon code**，簡稱RS code。

通訊編碼歷史(2/2)

* 1962年， Gallager首先提出**LDPC (Low-Density Parity Check) code**，在1996年，MacKay and Neal，將LDPC code發揚光大。 LDPC碼因其趨近理論極限的編碼效能與硬體實作的可行性而受到關注，已經被一些通訊標準納入其中，如DVB-S2，802.11n 與802.16e (Mobile WiMAX) 等。

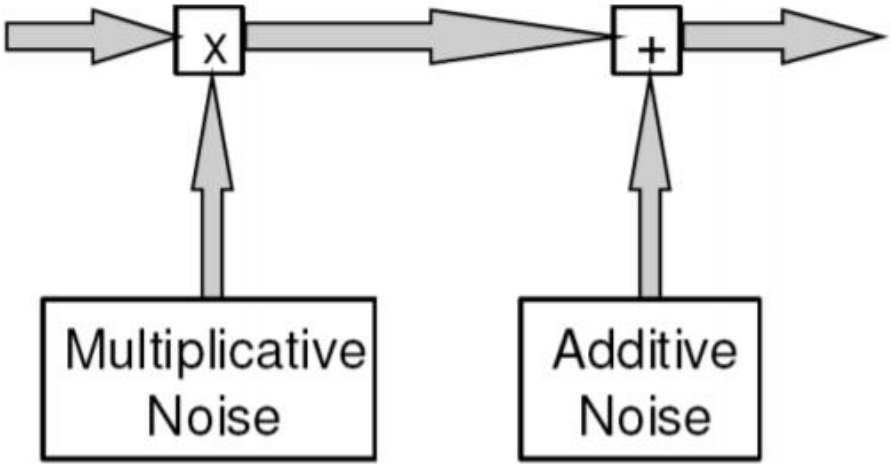
* 1976年－格子編碼調變(**Trellis Coded Modulation, TCM**)、魏特比碼(**Viterbi Code**)。

* 1993年， Berrou, Glavieux, and Thitimajshima三人首先提出**Turbo code**。

* 2001年， Flarion Technologies在IEEE Communications Letter Vol.5上證實LDPC碼**接近Shannon限界**的0.0045dB理論編碼增益界線。

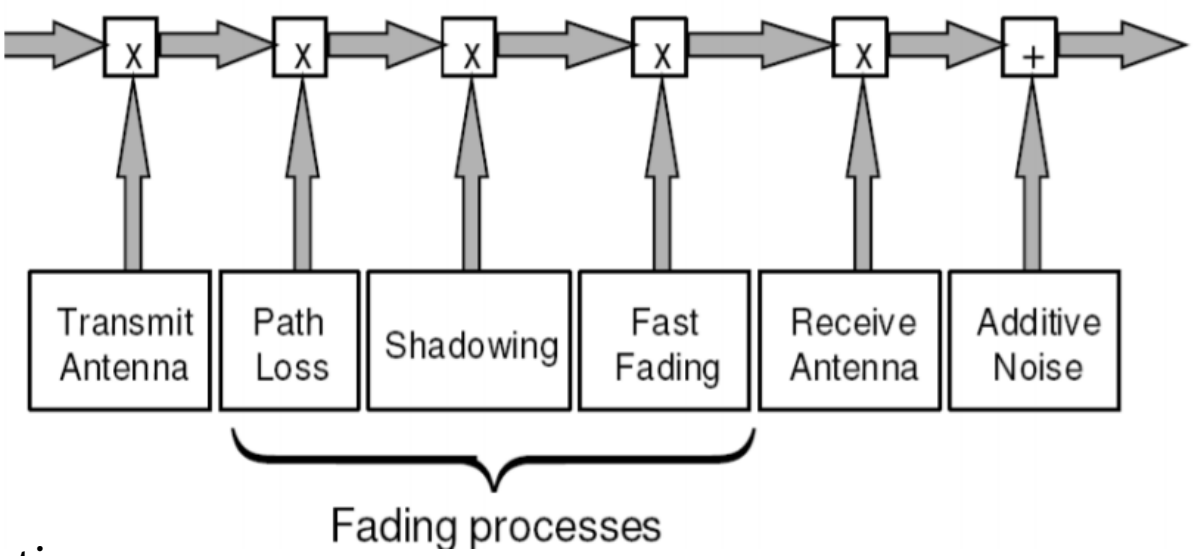
雜訊（Noise）

* 雜訊（Noise），指當訊號在傳輸過程中，會受到一些外在能量所產生訊號（如雜散電磁場）干擾，這些能量即雜訊。
* 在無線通道中，雜訊可分為相乘性雜訊與相加性雜訊。



https://www.inf.ed.ac.uk/teaching/courses/cn/wireless_comms_overview.pdf

雜訊（Noise）



- **Multiplicative**

– Antenna directionality
– Attenuation from absorption (walls, trees, atmosphere)
– Shadowing
– Reflection (smooth surfaces)
– Scattering (rough surfaces and small objects)
– Diffraction (edges of buildings and hills)
– Refraction (atmospheric layers, layered/graded materials)

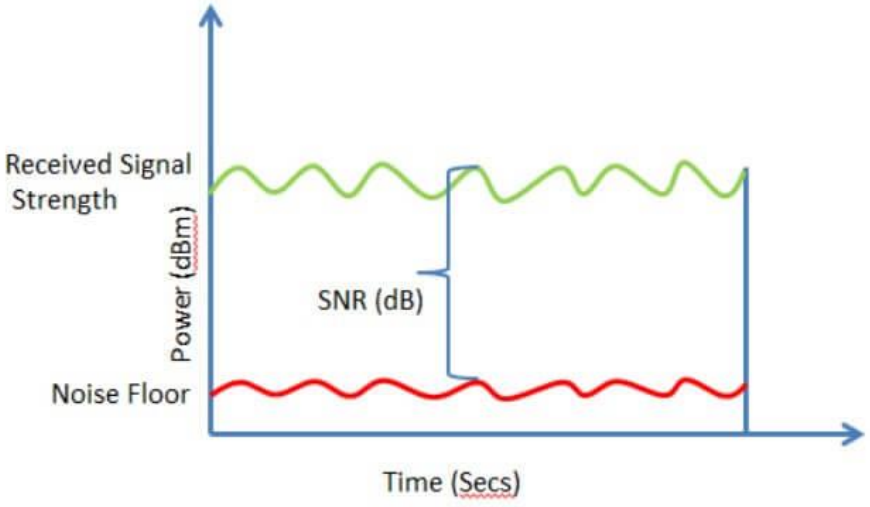
- **Additive**

– Internal sources within the receiver (e.g., thermal noise)
– External sources (e.g., interference from other transmitters and appliances)

https://www.inf.ed.ac.uk/teaching/courses/cn/wireless_comms_overview.pdf

雜訊（Noise）

* 雜訊通常會造成訊號的失真。其來源除了來自系統外部，亦有可能由接收系統本身產生。
* 雜訊的強度通常都是與訊號頻寬成正比，所以當訊號頻寬越寬，雜訊的干擾也會越大。
* 在評估雜訊強度或是系統抵抗雜訊能力的數據，是以訊號強度對雜訊強度的比例為依據，此即訊號雜訊比（Signal-to-noise ratio，SNR）。

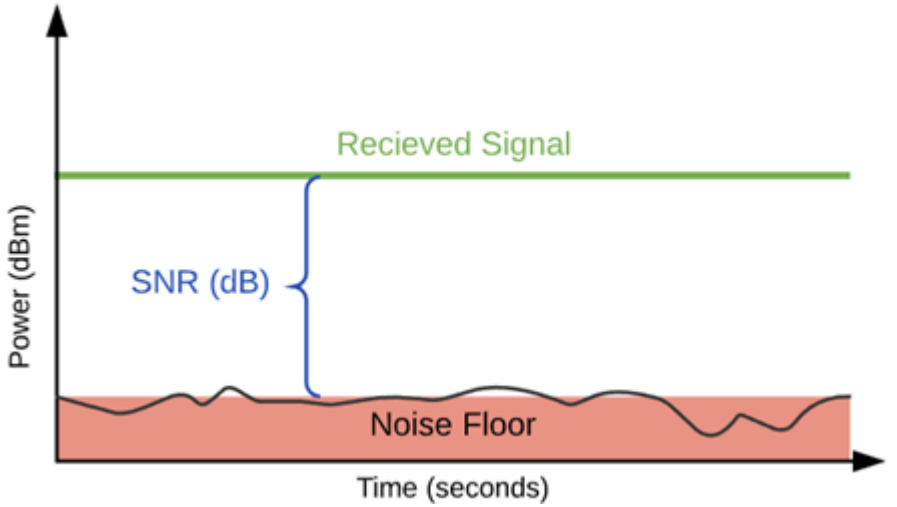


訊號雜訊比（SNR）

* 訊號雜訊比 (Signal-to-noise ratio，SNR) 。

$$\mathrm{SNR} = \frac{P_{\mathrm{signal}}}{P_{\mathrm{noise}}} = \frac{A_{\mathrm{signal}}^2}{A_{\mathrm{noise}}^2}$$

$$\mathrm{SNR(dB)} = 10\log_{10}\left(\frac{P_{\mathrm{signal}}}{P_{\mathrm{noise}}}\right) = 20\log_{10}\left(\frac{A_{\mathrm{signal}}}{A_{\mathrm{noise}}}\right)$$

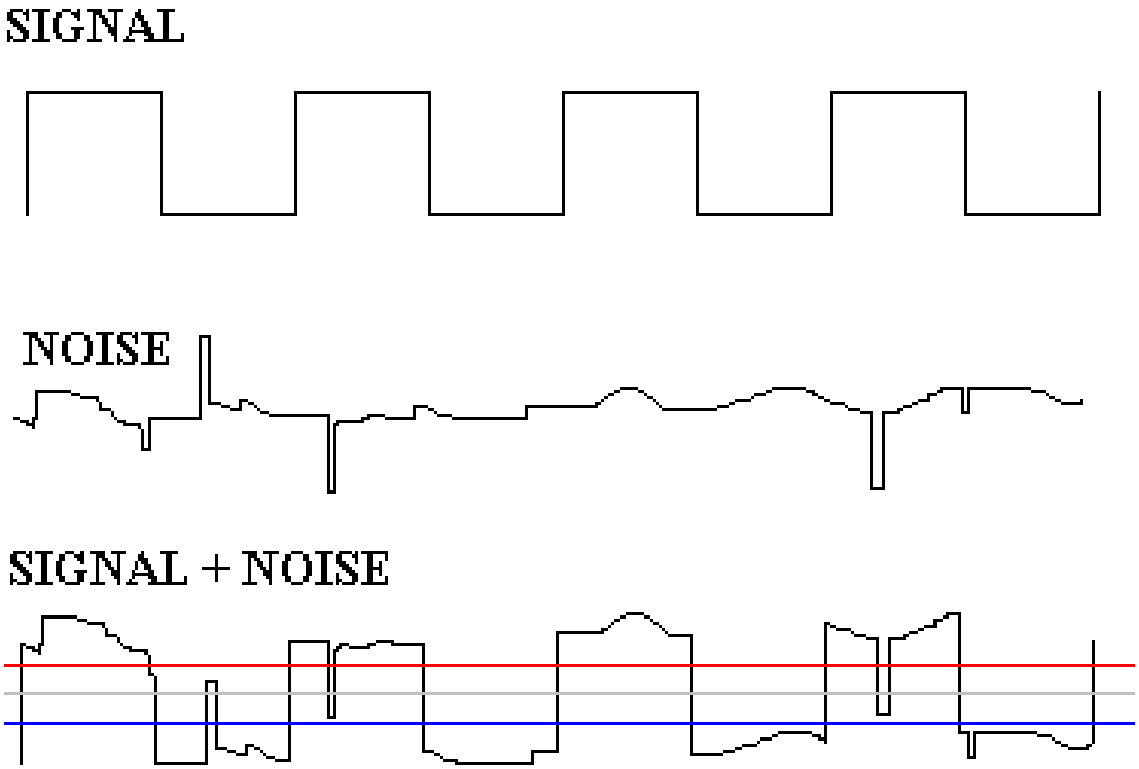


P_{signal} 為訊號功率（Power of Signal）
P_{noise} 為雜訊功率（Power of Noise）
A_{signal} 為訊號振幅（Amplitude of Signal）
A_{noise} 為雜訊振幅（Amplitude of Noise）

https://zh.wikipedia.org/wiki/%E4%BF%A1%E5%99%AA%E6%AF%94

https://documentation.meraki.com/MR/WiFi_Basics_and_Best_Practices/Signal-to-Noise_Ratio_(SNR)_and_Wireless_Signal_Strength

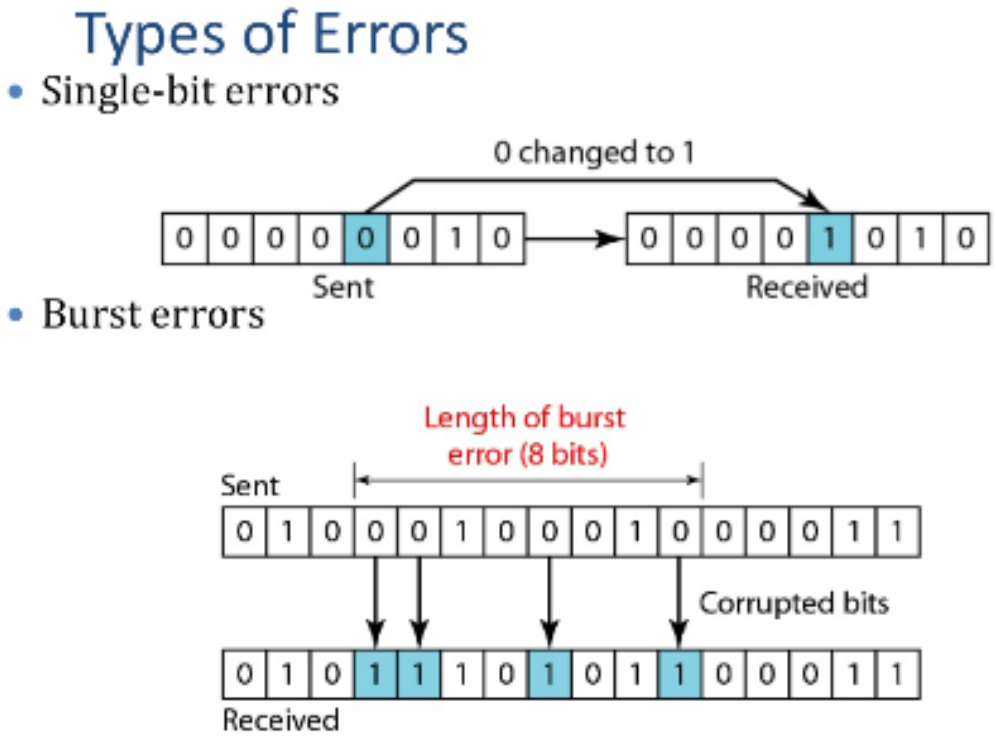
雜訊（Noise）



https://networkengineering.stackexchange.com/questions/43791/what-is-snr-margin-and-line-attenuation

通道編碼的理論

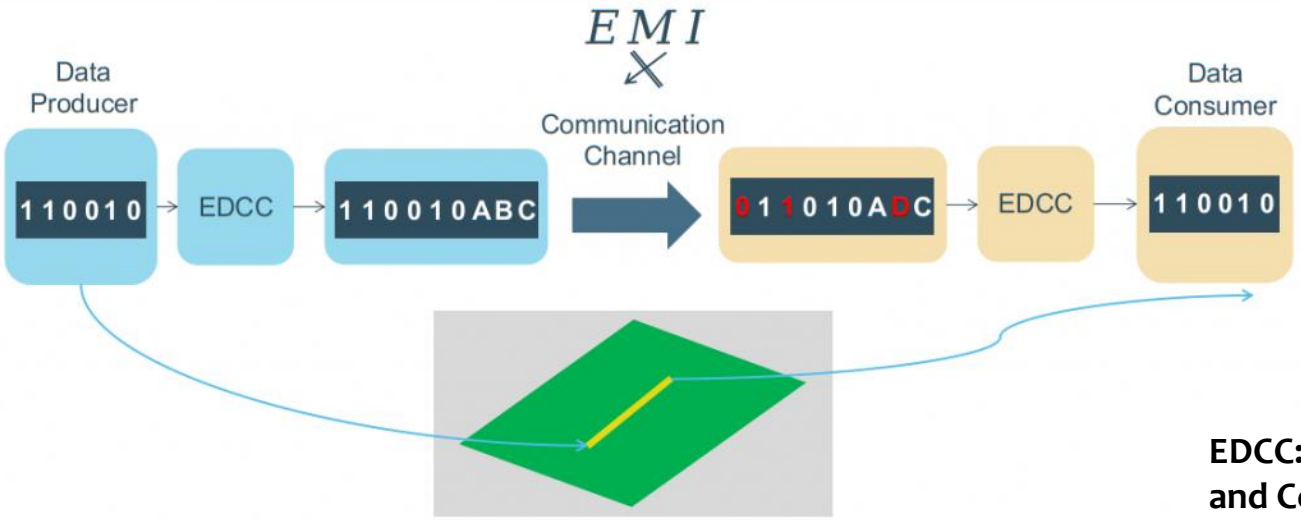
* 通道編碼技術 - reliable communication errors must be detected and corrected.



https://www.slideshare.net/saikrishnatanguturu/computer-networks-34379994

通道編碼的理論

* 通道編碼技術依照一定的規則加入冗餘（redundancy）位元，使得接收端可以透過一定的規則去偵測或更正錯誤，完成後再將冗餘位元去掉，還原傳送資訊。如此一來便可降低誤碼率。



EDCC: Error Detection and Correction Codes

https://etn-peter.eu/2020/05/18/simulation-setup-for-investigating-the-error-detection-and-correction-codes-effectiveness-under-harsh-electromagnetic-interference/

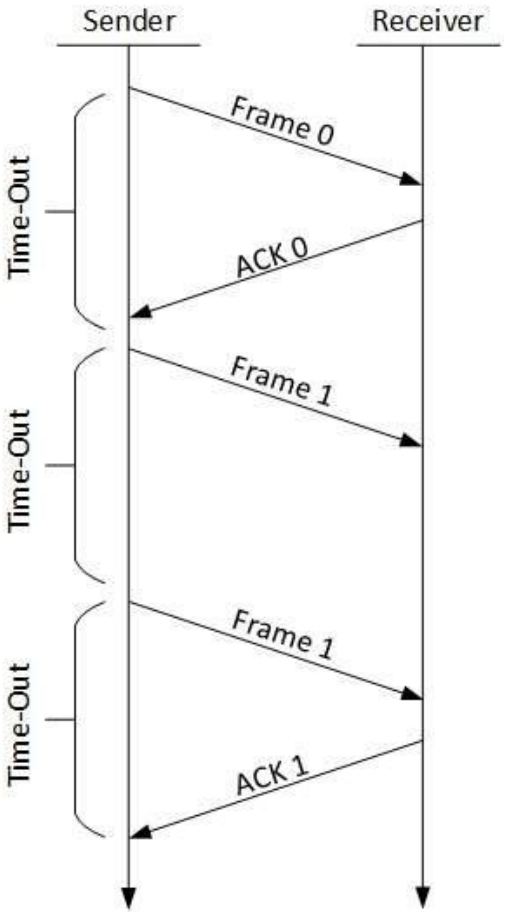
處理資料傳輸錯誤的方法

* 錯誤檢測碼
  - 錯誤檢測碼僅能偵測是否有錯誤發生，例如同位元檢查、循環冗餘碼檢查。
* 自動重傳要求 **(Automatic repeat request, ARQ)**
  - 檢測到錯誤時將資料區塊丟棄
  - 要求發射機重送此資料區塊
* 錯誤更正碼，也稱為前向錯誤更正 **(Forward error correction, FEC)**碼
  - 偵測錯誤還能更正錯誤
  - 大多數的系統都是此系統

ARQ (Automatic repeat request,自動重傳要求)

* ARQ是一種用於資料通訊的錯誤處理機制。這種技術有三種方案：

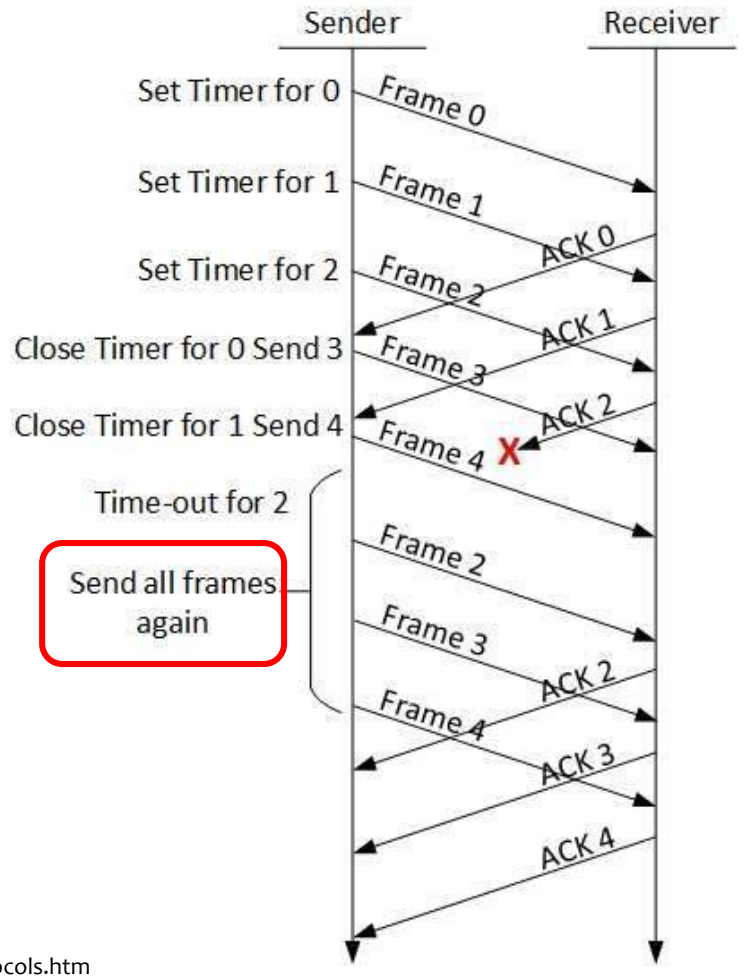
1. Stop-and-Wait ARQ：即是傳送封包後，必須等待固定時間來接收ACK，如果時間一過，系統會判別遺失並且重傳，重傳次數是由系統設定。



http://kevin.hwai.edu.tw/~kevin/material/ComputerNetwork/part2/ControlProtocols.htm

ARQ (Automatic repeat request,自動重傳要求)

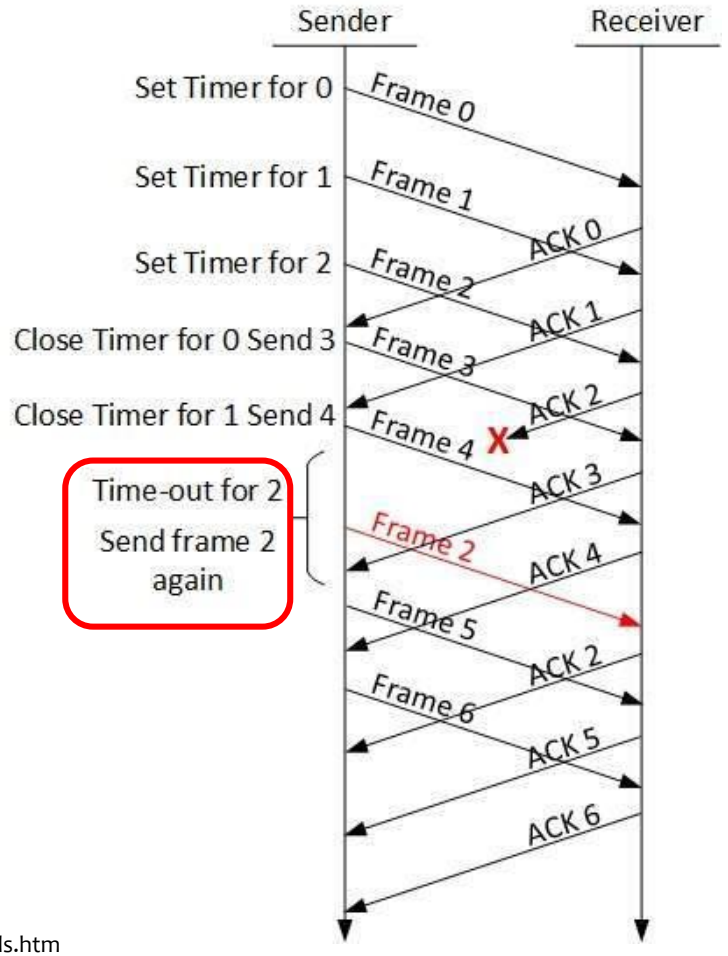
2. Go-Back-N ARQ：這個方法與上述類似，不過是一次傳送數個連續封包，並且等待數個ACK，若有其他一個ACK沒收到，在下一次傳送的數個封包中，會重新傳送符合沒接收到的ACK之對應封包與其後續連續封包。



http://kevin.hwai.edu.tw/~kevin/material/ComputerNetwork/part2/ControlProtocols.htm

ARQ (Automatic repeat request,自動重傳要求)

3. Selective-Repeat ARQ：
此方法為一次傳送數個連續封包，並且等待數個ACK，若有其他一個ACK沒收到，在下一次傳送的數個封包中，僅只會重新傳送符合沒接收到的ACK之對應封包與接收者收過的後續封包。



http://kevin.hwai.edu.tw/~kevin/material/ComputerNetwork/part2/ControlProtocols.htm

錯誤檢測碼 -同位元檢查(Parity Check)

* 同位元檢查可分為偶同位或奇同位。
  - 偶同位(even parity)：全部傳輸字元有偶數個1 。若原資料區塊有偶數個1，則同位元必須為0。若原資料區塊有奇數個1，則同位元必須為1。
  - 奇同位(odd parity) ：全部傳輸字元有奇數個1。若原資料區塊有偶數個1，則同位元必須為1。若原資料區塊有奇數個1，則同位元必須為0。
* 例如，7-bit的資料為1010101，若同位元附加於資料後，則
  - 偶同位傳輸，同位元=0，全部傳輸字元=10101010
  - 奇同位傳輸，同位元=1，全部傳輸字元=10101011

錯誤檢測碼 -同位元檢查(Parity Check)

* 範例

7 位資料區塊	帶有同位元的資料區塊	
	偶同位	奇同位
0000000	00000000	00000001
1010001	10100011	10100010
1101001	11010010	11010011
1111111	11111111	11111110

錯誤檢測碼 -循環冗餘碼檢驗 (Cyclic Redundancy Check, CRC)

* 循環冗餘檢查CRC (Cyclic Redundancy Check)是網路通訊中一種常見的傳輸錯誤檢查技術。

* CRC的計算方式是將待傳輸的資料區塊視為一堆連續位元所構成的一整個數值，並將此數值除以一特定的除數，兩數相除後得到的餘數即為循環冗餘檢查碼 (CRC code) 。

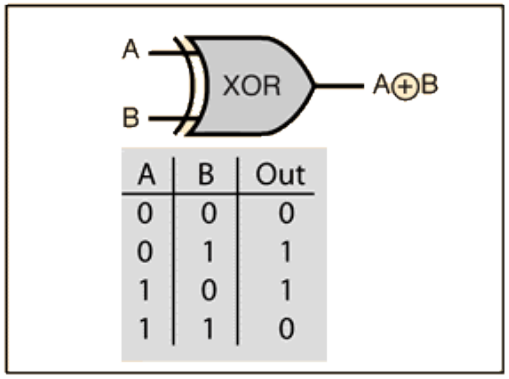
* 在接收端，將接收到的二進制序列數 (包括訊息碼和CRC碼) 除以該生成多項式 ，如果餘數為0，則表示傳輸中無錯誤發生，否則說明傳輸有誤。

* 目前較常使用的 CRC 位元數目有 8、16 或 32，一般簡寫為 CRC-8、CRC-16、CRC-32 。

CRC範例（1/2）

* 設資料位元為110010101110，生成多項式為X3 + 1 (1001)，試求取 CRC碼及加上CRC碼後的完整訊息。

使用長除法求取110010101110000除以生成多項式X3 + 1 (1001) 的餘數，但必須以XOR運算取代此過程中的減法運算。



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

```
          110100001111
      ┌──────────────
1001 ) 110010101110000
       1001
       ----
        1011
        1001
        ----
          1001
          1001
          ----
            1110
            1001
            ----
             1110
             1001
             ----
              1110
              1001
              ----
               1110
               1001
               ----
                111   ← 餘數
```

CRC範例（2/2）

* CRC 碼為步驟 2. 所求出的餘數111，而完整訊息則是在原始的資料位元11001010111 0後面加上CRC碼，得到11001010111 0111，只要收訊端有正確地接收到訊息，該訊息將能被生成多項式整除。
* 理論上，CRC碼的長度愈長，偵錯能力就愈佳，常見的有CRC-8、CRC-16、CRC-12、CRC-32等，不同的長度有不同的生成多項式。
  - 例如CRC-16的生成多項式有X16 + X12 + X5 + 1、X16 +X15 + X2 + 1、X16 + X15 + X10 + X3等。

常用的CRC (ITU-IEEE 規範)(1/2)

名稱	生成多項式
CRC-1	$x + 1$ (用途：硬體，也稱為奇偶校驗位)
CRC-5-CCITT	$x^5 + x^3 + x + 1$ (ITU G.704 標準)
CRC-5-USB	$x^5 + x^2 + 1$ (用途：USB 信令包)
CRC-7	$x^7 + x^3 + 1$ (用途：通信系統)
CRC-8-ATM	$x^8 + x^2 + x + 1$ (用途：ATM HEC)
CRC-8-CCITT	$x^8 + x^7 + x^3 + x^2 + 1$ (用途：1-Wire Bus)
CRC-8	$x^8 + x^7 + x^6 + x^4 + x^2 + 1$
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x + 1$

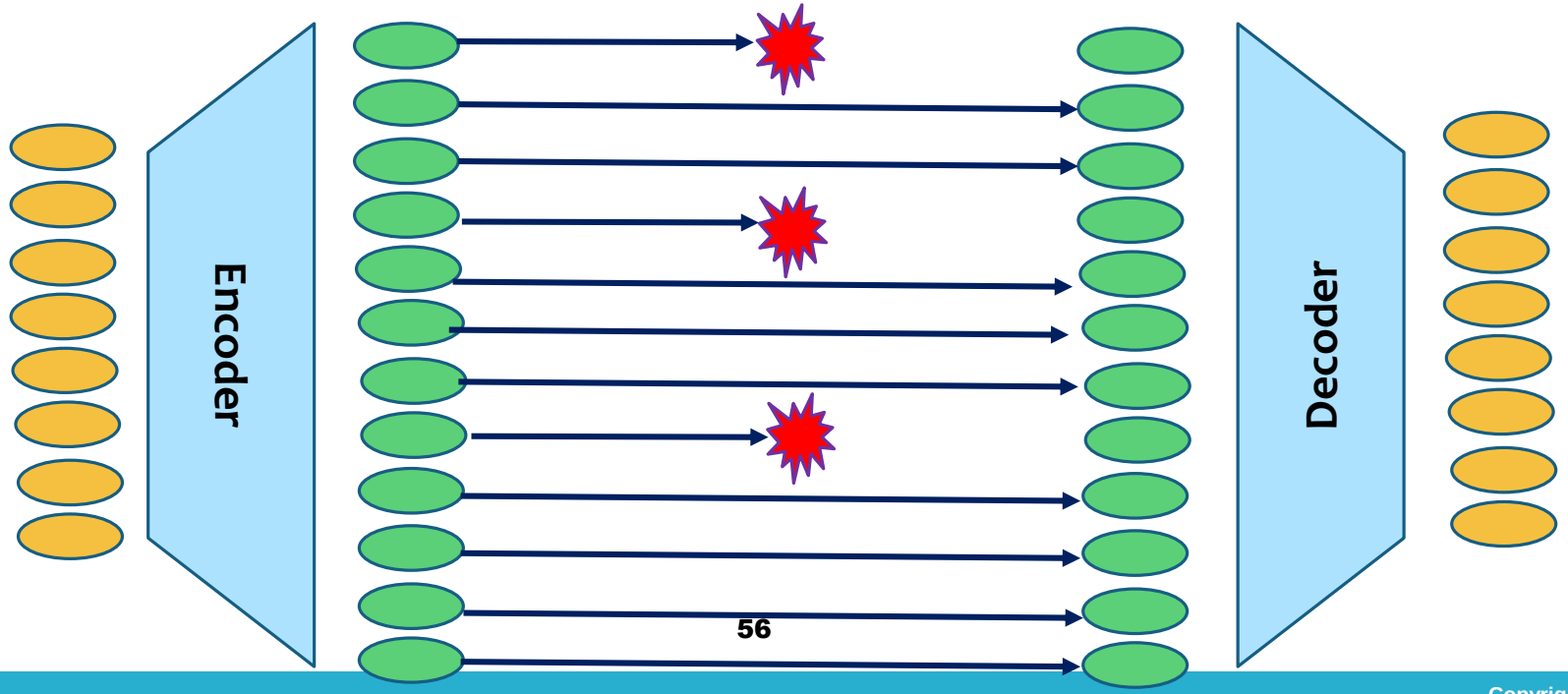
常用的CRC (ITU-IEEE 規範)(2/2)

名稱	生成多項式
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + x + 1$ (用途：通信系統)
CRC-16-CCITT	$x^{16} + x^{12} + x^5 + 1$ (X25, V.41, Bluetooth, PPP, IrDA)
CRC-16-IBM	$x^{16} + x^{15} + x^2 + 1$
CRC-16-BBS	$x^{16} + x^{15} + x^{10} + x^3$ (XMODEM 協議)
CRC-32-MPEG2	IEEE 802.3
CRC-32-IEEE 802.3	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

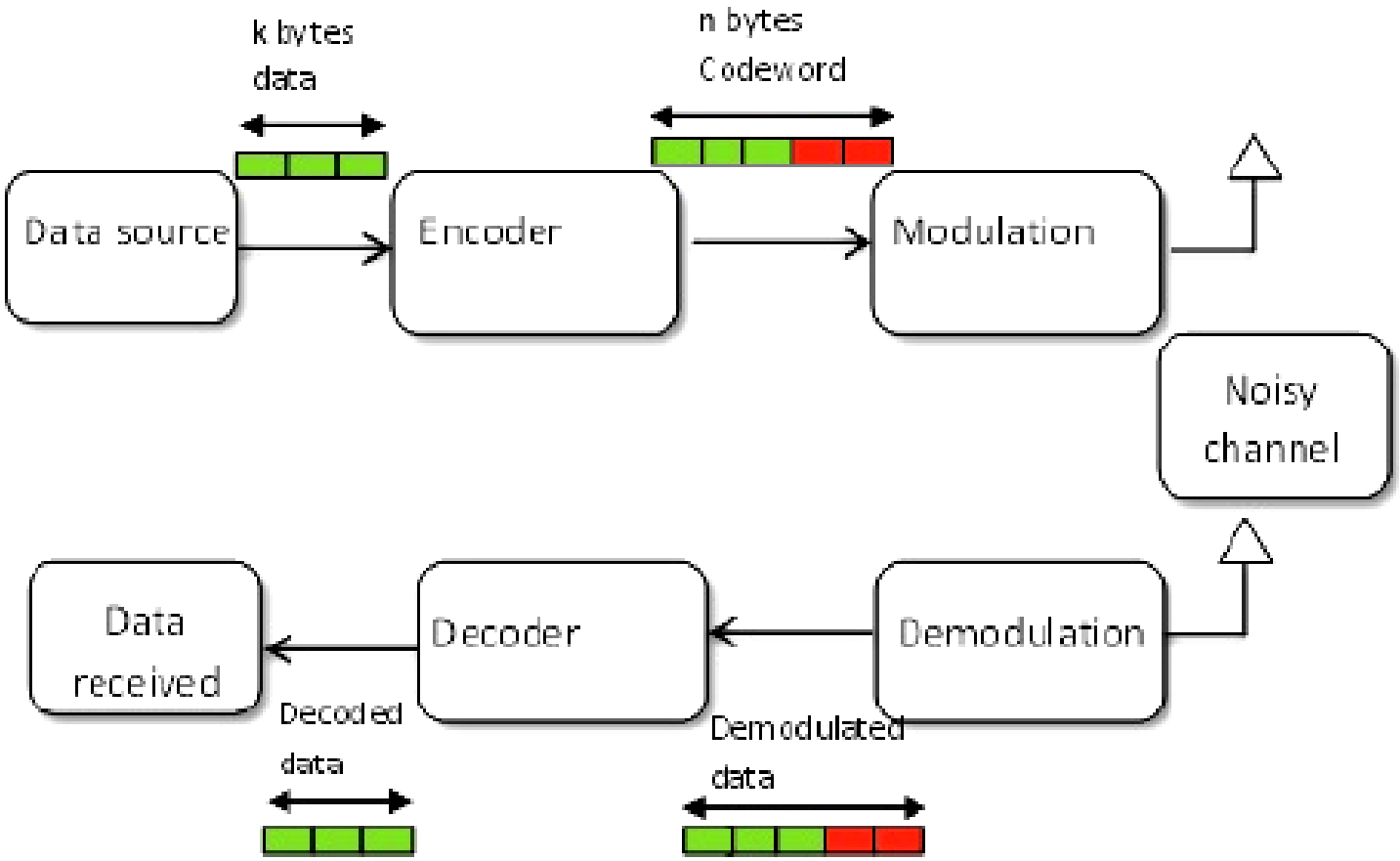
FEC (Forward error correction, 前向錯誤更正)

* FEC 也稱為碼錯誤更正碼，也就是在區塊碼的錯誤更正能力內，不但能偵測(detect)錯誤還能更正(correct)錯誤。

* FEC 的核心目標是傳送足夠量的冗餘資料以確保接收者能夠**自己主動復原資料，而不需要發送者再次重傳資料！**



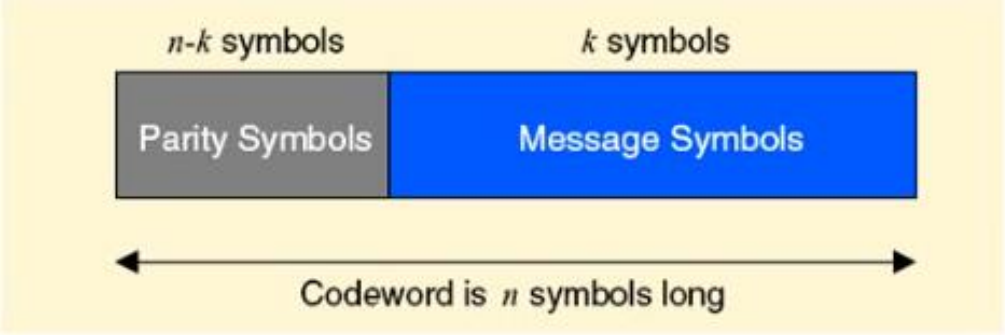
Introduction to Linear Block Codes



https://www.researchgate.net/figure/Structure-of-linear-block-code_fig2_272174911

Introduction to Linear Block Codes

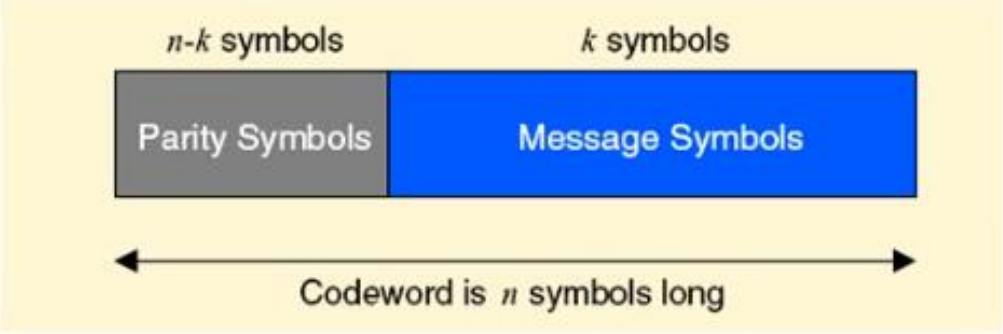
* 線性區塊碼 (Linear Block Code)：
  - A (n,k) block code, where $n > k$



  - Block: encoder accepts a block of message symbols and generates a block of codeword.
  - Linear: addition of any two valid codeword results in another valid codeword.

Introduction to Linear Block Codes

* 區塊碼 (Block codes) 通常以 (*n*, *k*)區塊碼來表示。
  - 所謂的 (*n*, *k*)區塊碼是將*k*位元資料 (Message) 編碼 (Encode)，加入 (*n* – *k*) 個冗餘 (Redundancy) 位元，所組成一個*n*位元的有效碼字 (Codeword)。
  - 一個區塊碼共有2^k個有效的碼字，及2^n個可能的碼字。



* 向量表示　：$\boldsymbol{m} = (m_0, m_1, \ldots, m_{k-1})$, $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ 。
* 多項式表示：$\boldsymbol{m}(x) = m_0 + m_1 x + \ldots + m_{k-1} x^{k-1}$, $\boldsymbol{c}(x) = c_0 + c_1 x + \ldots + c_{n-1} x^{n-1}$ 。

Linear Block Codes

* 若$\boldsymbol{m}(x)$的係數只有0和1，也就是$m_i \in GF(2)$，則此區塊碼稱為二元(Binary)區塊碼，可用模-2運算作計算。

Messages	Code words
(0 0 0 0)	(0 0 0 0 0 0 0)
(1 0 0 0)	(1 1 0 1 0 0 0)
(0 1 0 0)	(0 1 1 0 1 0 0)
(1 1 0 0)	(1 0 1 1 1 0 0)
(0 0 1 0)	(1 1 1 0 0 1 0)
(1 0 1 0)	(0 0 1 1 0 1 0)
(0 1 1 0)	(1 0 0 0 1 1 0)
(1 1 1 0)	(0 1 0 1 1 1 0)
(0 0 0 1)	(1 0 1 0 0 0 1)
(1 0 0 1)	(0 1 1 1 0 0 1)

Messages	Code words
(0 1 0 1)	(1 1 0 0 1 0 1)
(1 1 0 1)	(0 0 0 1 1 0 1)
(0 0 1 1)	(0 1 0 0 0 1 1)
(1 0 1 1)	(1 0 0 1 0 1 1)
(0 1 1 1)	(0 0 1 0 1 1 1)

Linear Block Code with k=4 , n= 7

Linear Block Codes

* Since an (n, k) linear code C is a k-dimensional subspace of the vector space V_n of all the binary n- tuple, it is possible to find k linearly independent code word, $\mathbf{g}_0$, $\mathbf{g}_1$,…, $\mathbf{g}_{k-1}$ in C

$$\mathbf{v} = u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + \cdots + u_{k-1}\mathbf{g}_{k-1}$$

where $u_i = 0$ or 1 for $0 \leq i < k$

Linear Block Codes

* Let us arrange these ***k*** linearly independent code words as the rows of a $k \times n$ matrix as follows:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ . \\ . \\ . \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{00} & g_{01} & g_{02} & \cdot & \cdot & g_{0,n-1} \\ g_{10} & g_{11} & g_{12} & \cdot & \cdot & g_{1,n-1} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ g_{k-1,0} & g_{k-1,1} & g_{k-1,2} & \cdot & \cdot & g_{k-1,n-1} \end{bmatrix}$$

Because the rows of **G** generate the (n, k) linear code C, the matrix **G** is called a *generator matrix* for C

where $\mathbf{g}_i = (g_{i0}, g_{i1}, \ldots, g_{i,n-1})$ for $0 \leq i < k$

Linear Block Codes

* If $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ is the message to be encoded, the corresponding code word can be given as follows:

$$\mathbf{v} = \mathbf{u} \cdot \mathbf{G}$$

$$= (u_0, u_1, \ldots, u_{k-1}) \bullet \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ . \\ . \\ . \\ \mathbf{g}_{k-1} \end{bmatrix}$$

$$= u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + \cdots + u_{k-1}\mathbf{g}_{k-1}$$

Example of Linear Block Codes

- **Example**
  - the (7, 4) linear code given in Table 3.1 has the following matrix as a generator matrix :

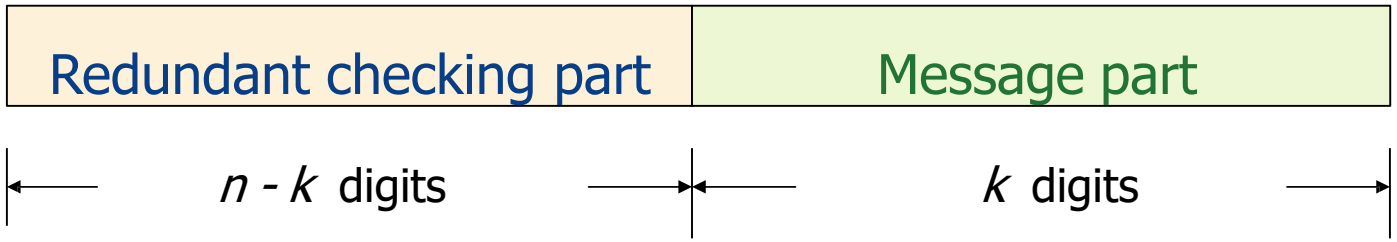
$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \mathbf{g}_2 \\ \mathbf{g}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

  - If u = (1 1 0 1) is the message to be encoded, its corresponding code word, according to (3.3), would be

$$\begin{aligned} \mathbf{v} &= 1 \cdot \mathbf{g}_0 + 1 \cdot \mathbf{g}_1 + 0 \cdot \mathbf{g}_2 + 1 \cdot \mathbf{g}_3 \\ &= (1101000) + (0110100) + (1010001) \\ &= (0001101) \end{aligned}$$

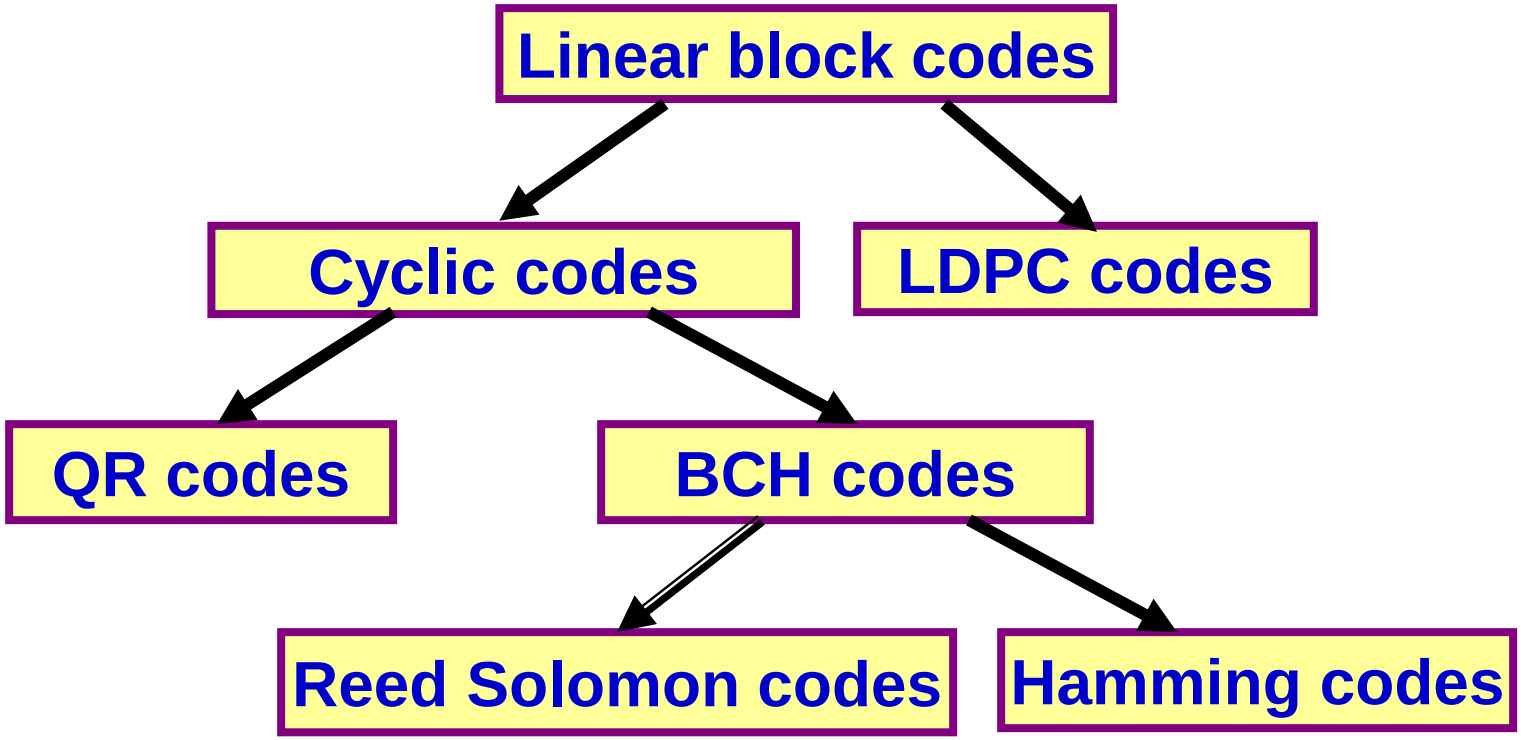
Linear Block Codes

- A desirable property for a linear block code is the *systematic structure* of the code words
  - where a code word is divided into two parts
    - The message part consists of k information digits
    - The redundant checking part consists of $n - k$ parity-check digits
- A linear block code with this structure is referred to as a *linear systematic block code*



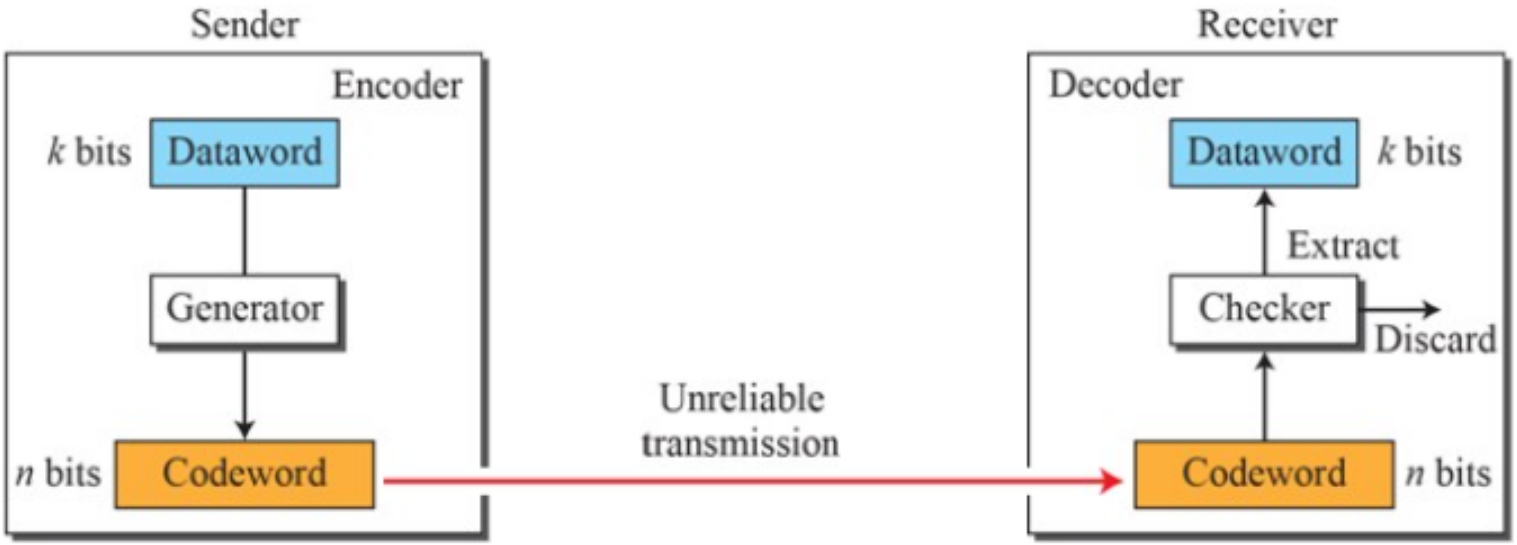
Systematic format of a code word

線性區塊碼家族關係



線性區塊碼-Code Rate

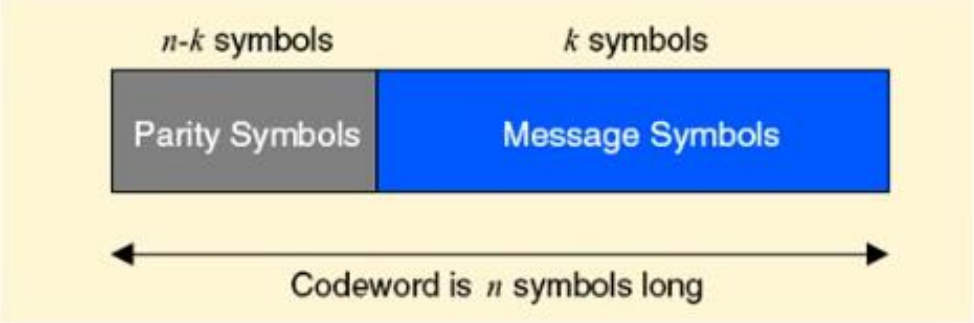
Figure: Process of error detection in block coding



https://slideplayer.com/slide/9860871/

線性區塊碼-Code Rate

* 編碼率(Code Rate)：指未經編碼前資料的長度與經過編碼後的長度比
* *k*/*n* 的比值稱為碼率(code rate)

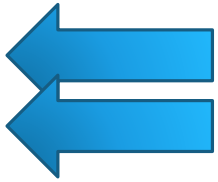


https://zh.wikipedia.org/wiki/%E4%BD%8E%E5%AF%86%E5%BA%A6%E5%A5%87%E5%81%B6%E6%AA%A2%E6%9F%A5%E7%A2%BC
https://www.2cm.com.tw/2cm/zh-tw/tech/CFA2CF3F97934D73825179497DC7B5A3

線性區塊碼-Code Rate

* 碼率越小，則表示冗餘越多，即浪費的位元越多。

FEC	Code Rate	Error Correction capability
Golay(23,12,7)	0.522	3/23 = 0.130
RS (15, 11)	0.733	2/15 = 0.133
RS (128, 96)	0.75	16/128 = 0.125
BCH(31,21)	0.677	2/31 = 0.065
BCH(48,36)	0.75	2/48 = 0.042



資料來源: Kan Yu, Mikael Gidlund, Johan Akerberg, Mats Björkman, Reliable and Low Latency Transmission in Industrial Wireless Sensor Networks, DOI: 10.1016/j.procs.2011.07.120

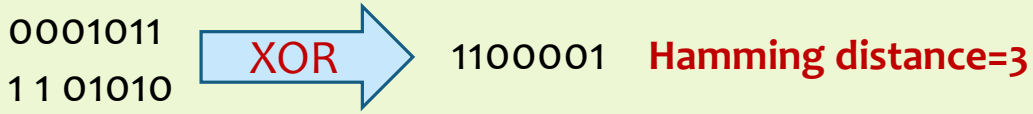
區塊碼的一些術語

* **漢明距離 (Hamming distance)**：表示兩個*n*位元之二進制序列間，不同位元的總數。

* **漢明權重(Hamming weight)**：在碼字裡面非0值的數目。

例如: 兩個向量x= (0001011) 和x′=(1101010)

- Hamming distance就是它們不一樣的地方有幾個



- Hamming weight,是指一個向量不是0的的地方有幾個,所以wH(x) = 3, wH(x′) =4。

區塊碼的一些術語

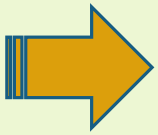
* **最小漢明距離 (Minimum Hamming distance)**：某一種區塊碼由{$w_1, w_2, \ldots, w_s$}個碼字所組成，$s=2^k$，最小距離d_{min}定義為

$$d_{\min} = \min_{i \neq j}[d(\mathbf{w}_i, \mathbf{w}_j)], \quad for\, i \neq j$$

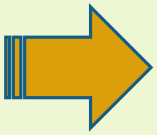
例如:

Given the following coding scheme, let's find the Minimum Hamming distance

00000
0 101 1
1 01 01
1 1 1 1 0



d(00000, 01011) =3
d(00000, 10101) =3
d(00000, 11110) =4
d(01 01 1 , 10101) =4
d(01 01 1 , 11110) =3
d(10 10 1 , 11110) =3



Minimum Hamming distance d= 3

- The smallest Hamming distance between all possible pairs in a set of words.

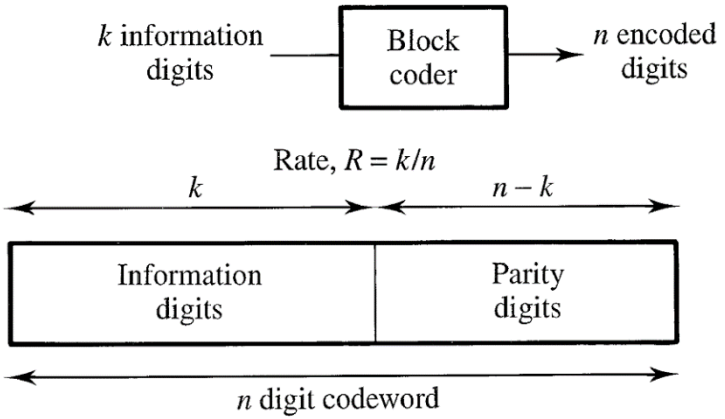
設計區塊碼的考量

* 給定n和k值，我們希望d_{min}值儘可能最大
  * 如此才能偵測及更正較多的位元。

* 編碼和解碼要相對地容易
  * 其所需記憶體要小和解碼處理的時間要快。

* 附加的冗餘位元數目$(n-k)$要儘可能小
  * 即**碼率要大**，以減少頻寬。

* 附加的冗餘位元數目$(n-k)$要儘可能大
  * 即**碼率要小**，以減少錯誤率。



線性區塊碼

* 什麼是線性區塊碼(Linear block codes)？線性區塊碼有下列的性質：
  - 在某一個區塊碼的所有碼字集合中，任何碼字的線性組合仍是此區塊碼中的另一個碼字
  - 線性區塊碼包含全0的碼字
  - 線性區塊碼的最小距離等於非0碼字中權重最小者
* 線性區塊碼常用(n, k)或(n, k, d)來表示，n是碼長，k是訊息長，d是最小距離，例如(7, 4, 3)漢明碼。

線性區塊碼種類

* 線性區塊碼包含有下列常見的碼：
  - 漢明碼(Hamming codes)
  - 循環碼(Cyclic codes)
  - 平方剩餘碼(Quadratic Residue codes)
  - BCH碼(BCH codes)
  - 里德所羅門碼(Reed-Solomon codes)
  - 低密度同位檢查碼 (LDPC codes)

漢明碼

* Richard Hamming在1950年提出漢明碼，它是設計來更正1位元錯誤，及偵測2位元錯誤，它也是線性區塊碼家族的一員。
* 若 $m \geq 3$，漢明碼有下列的特性：
  - 長度：$n = 2^m - 1$
  - 訊息位元長度：$k = 2^m - m - 1$
  - 查核位元或冗餘長度：$n - k = m$
  - 錯誤更正能力：t = 1, (d_{min} = 3)

漢明碼的 General Algorithm

* 從1開始給數字的數據位（從左向右）標上序號, 1，2，3，4，5...
* (1) 校驗位1覆蓋了所有數據位位置序號的二進制表示倒數第一位是1的數據：1（校驗位自身，這裡都是二進制，下同），11，101，111，1001，等
* (2) 校驗位2覆蓋了所有數據位位置序號的二進制表示倒數第二位是1的數據：10（校驗位自身），11，110，111，1010，1011，等
* (3) 校驗位4覆蓋了所有數據位位置序號的二進制表示倒數第三位是1的數據：100（校驗位自身），101，110，111，1100，1101，1110，1111，等
* (4) 校驗位8覆蓋了所有數據位位置序號的二進制表示倒數第四位是1的數據：1000（校驗位自身），1001，1010，1011，1100，1101，1110，1111，等

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗		
	p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗		...
	p4				✗	✗	✗	✗					✗	✗	✗	✗					✗	
	p8								✗	✗	✗	✗	✗	✗	✗	✗						
	p16																✗	✗	✗	✗	✗	

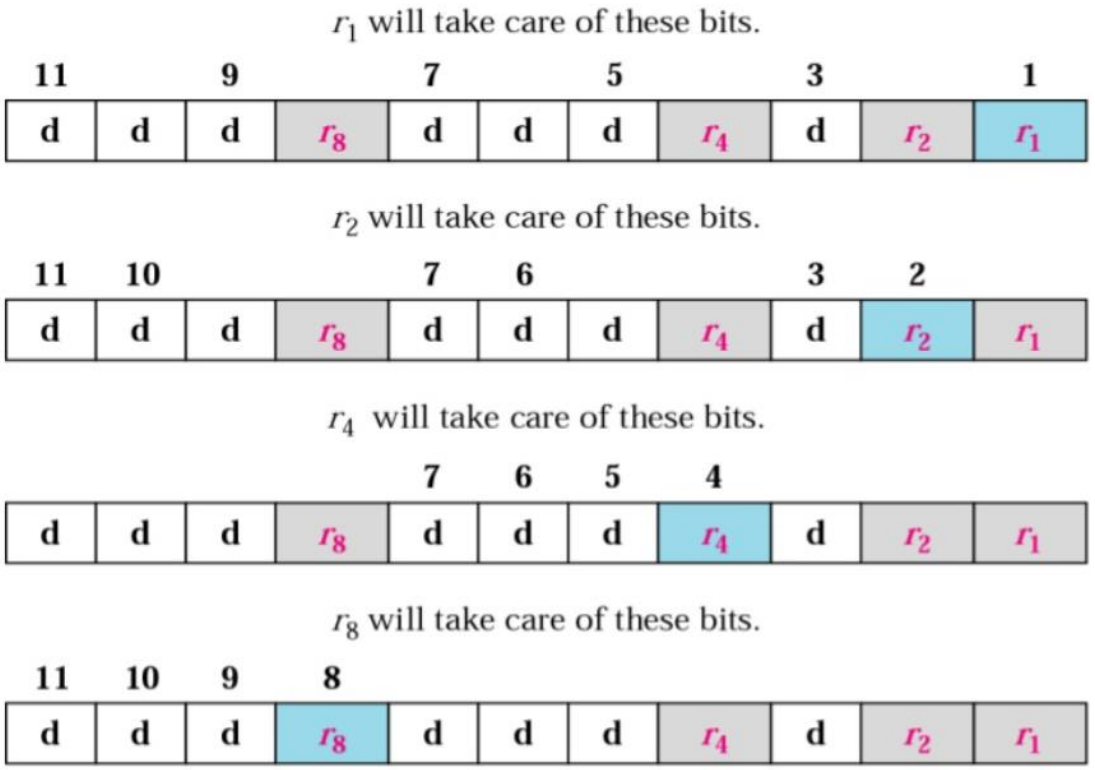
https://www.newton.com.tw/wiki/%E6%BC%A2%E6%98%8E%E7%A2%BC

https://en.bitcoinwiki.org/wiki/Amp/Hamming_code

漢明碼的 General Algorithm

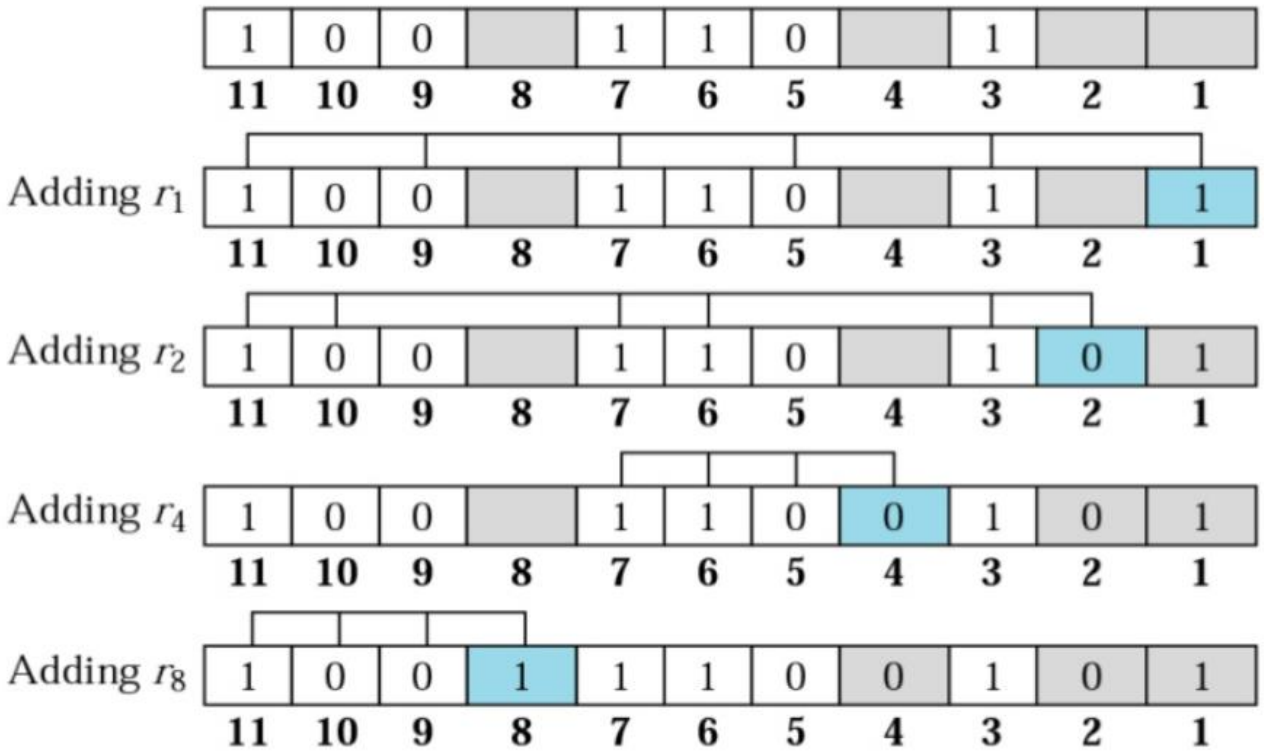
Redundant Bit Calculation

r個冗餘位分佈在2的冪次位置，即1、2、4、8、...
位置1對應編碼後的位元序號2^0，
位置2對應編碼後的位元2^1，
依此類推。



漢明碼的 General Algorithm

Example: *Hamming Code*



Data:
1 0 0 1 1 0 1

Code:
1 0 0 1 1 1 0 0 1 0 1

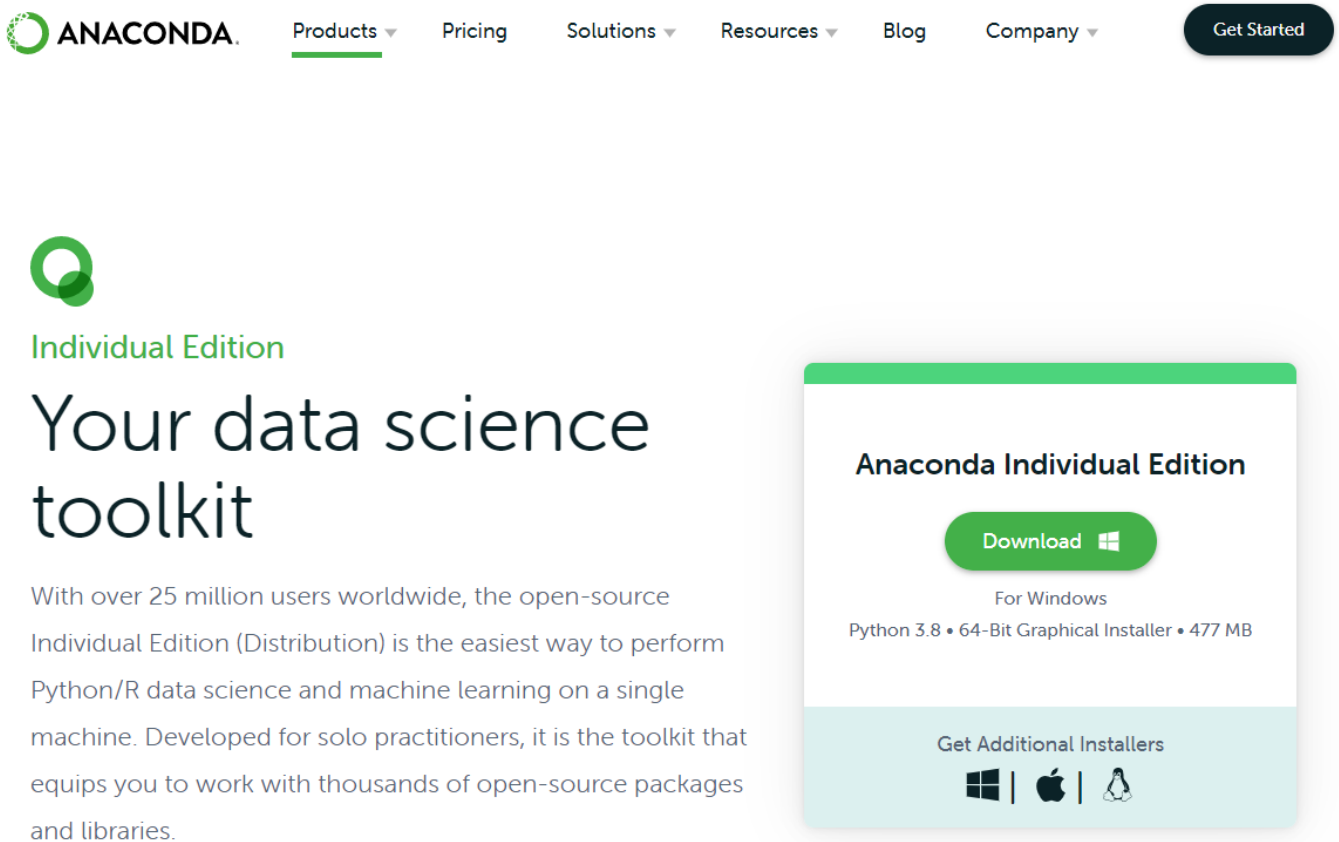
漢明碼的 Code Rate

* Code Rate：

Parity bits	Total bits	Data bits	Name	Rate
2	3	1	Hamming(3,1) (Triple repetition code)	1/3 ≈ 0.333
3	7	4	Hamming(7,4)	4/7 ≈ 0.571
4	15	11	Hamming(15,11)	11/15 ≈ 0.733
5	31	26	Hamming(31,26)	26/31 ≈ 0.839
6	63	57	Hamming(63,57)	57/63 ≈ 0.905
7	127	120	Hamming(127,120)	120/127 ≈ 0.945
8	255	247	Hamming(255,247)	247/255 ≈ 0.96

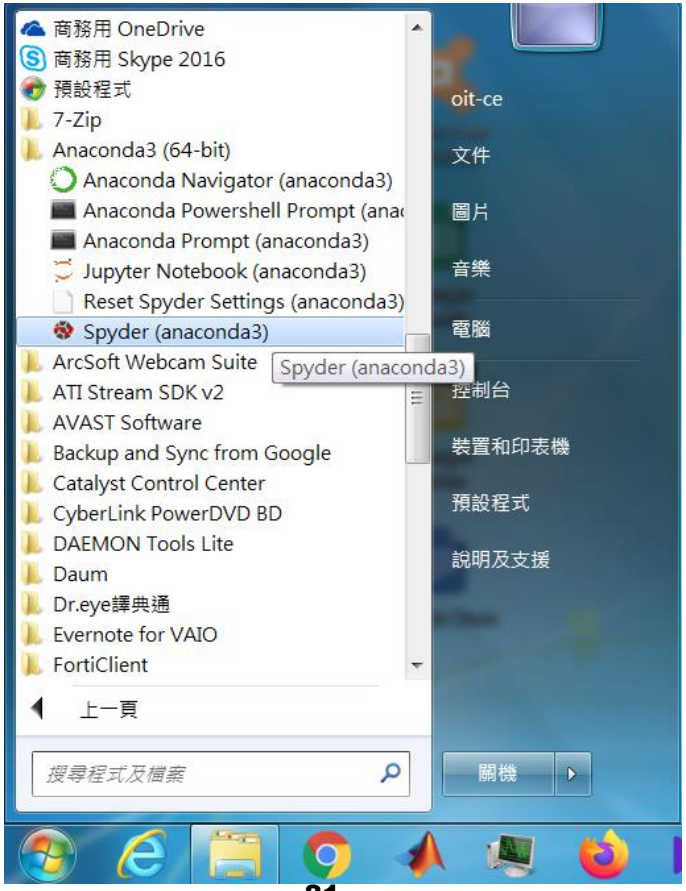
漢明碼 python 範例

* 安裝 Anaconda：
https://www.anaconda.com/products/individual#windows



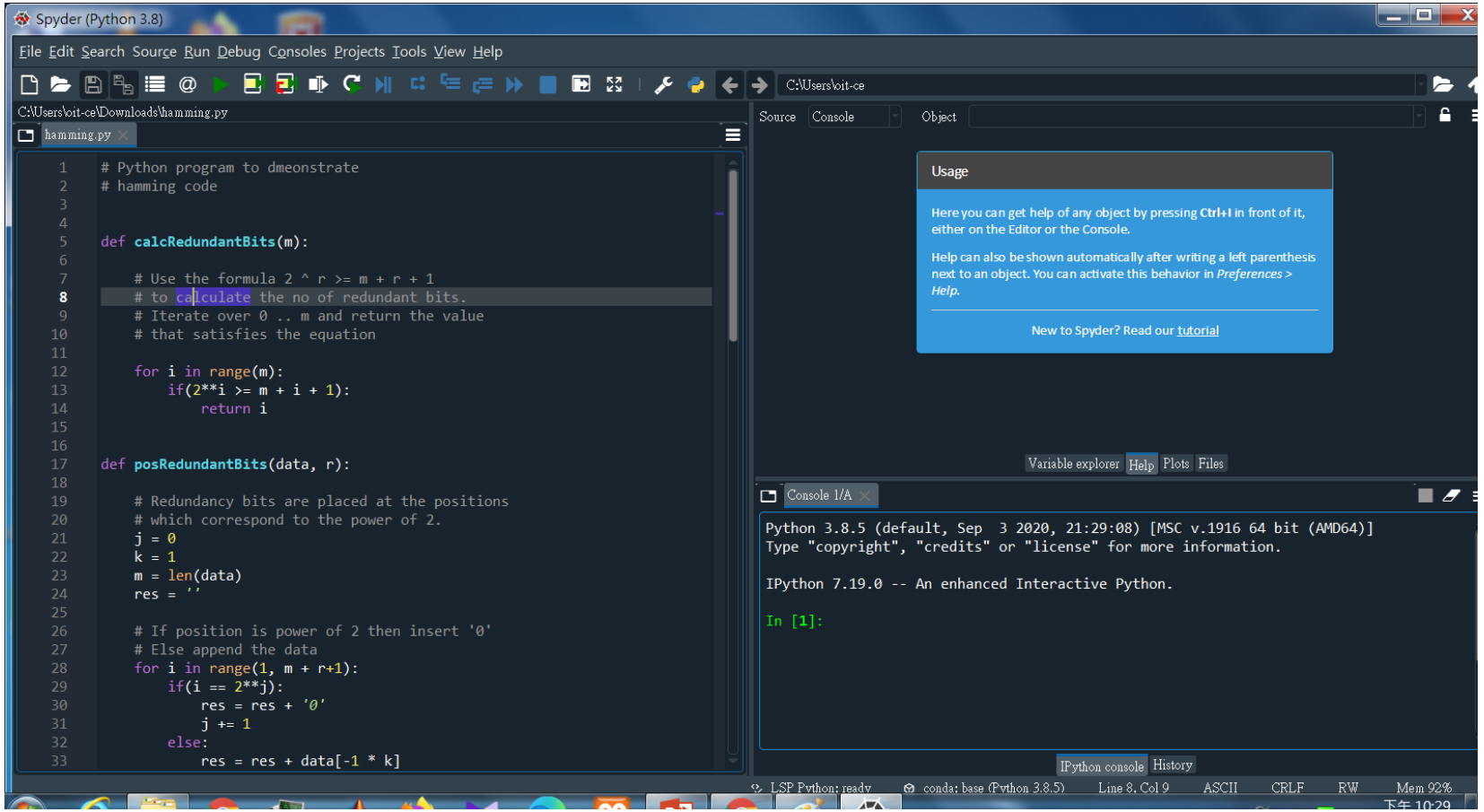
漢明碼 python 範例

* 進入Anaconda 的 Spyder 整合環境



漢明碼 python 範例

* 漢明碼 python 範例：hamming.py



漢明碼 python 範例

* 漢明碼 python 範例：

```
IPython 7.19.0 -- An enhanced Interactive Python.

runfile('C:/Users/oit-ce/Downloads/hamming.py',
wdir='C:/Users/oit-ce/Downloads')
Data is 1001101
Data transferred is 10011100101
Error Data is 10111100101
The position of error is 9
```

Parity Check/Hamming碼/BCH碼-1

* 先回顧 同位碼 和 Hamming碼
  - 簡單的同位碼只能檢查出錯誤而不知道具體是哪裡出錯。
  - Hamming碼只能糾正一位元錯誤。因此我們想能不能有一種方法可以糾正多位元錯誤呢？
* 此時，多項式的威力就發揮了出來。
  - 假如我們的真正的訊息是$m(x)$
  - 然後乘上一個編碼多項式$p(x)$，得到$m(x)p(x)$
  - 將其發送過去。發送過程中會會受到很多干擾，於是多項式被加上錯誤多項式$e(x)$

Parity Check/Hamming碼/BCH碼-2

- 接收者接收者的訊息為$c(x)$，就有

$$c(x) = m(x)p(x) + e(x)$$

- 接收者怎麼知道消息有沒錯呢？接收者事先就會和發送者商量好$p(x)$，然後就很簡單，只要讓$c(x)$除以$p(x)$，如果餘項是**0**，即沒有$e(x)$，沒有受到干擾，那麼商就是$m(x)$，就是正確的訊息。
- 如果餘項不是零，我們的商就變成了$m(x) + e(x)/p(x)$，無法將真正的訊息分離出來。
- 上述方法有個前提，在$m(x) + e(x)/p(x)$，$p(x)$必須是不可約的本原多項式(primitive polynomials)，這樣才能讓$e(x)$不被除盡，我們才能意識到有干擾的存在。

本原多項式(Primitive Polynomials)

* **Primitive Polynomials原則：**

  - *r*次多項式，最高次項*x*^r的係數必須是1。
  - 多項式如果不包含常數項1，就會被x整除。
  - 多項式項數不能是偶數的，比如 $1+x+x^2+x^4$ 有四項，很容易將其分解為 $(1+x)(1+x^2)$ 。
  - 每一項的次數也不能都是偶數，否則將每一項次數減半就能得到因數。例如 $1+x^2+x^4=(1+x+x^2)^2$ 。

本原多項式(Primitive Polynomials)

* **Primitive Polynomials ：**

degree (n)	$p(x)$
1	$x+1$
2	x^2+x+1
3	x^3+x+1
4	x^4+x+1
5	x^5+x^2+1
6	x^6+x+1
7	x^7+x+1
8	$x^8+x^6+x^5+x+1$
9	x^9+x^4+1
10	$x^{10}+x^3+1$
11	$x^{11}+x^2+1$
12	$x^{12}+x^7+x^4+x^3+1$
13	$x^{13}+x^4+x^3+x+1$
14	$x^{14}+x^{12}+x^{11}+x+1$
15	$x^{15}+x+1$

degree (n)	$p(x)$
15	$x^{15}+x+1$
16	$x^{16}+x^5+x^3+x^2+1$
17	$x^{17}+x^3+1$
18	$x^{18}+x^7+1$
19	$x^{19}+x^6+x^5+x+1$
20	$x^{20}+x^3+1$
21	$x^{21}+x^2+1$
22	$x^{22}+x+1$
23	$x^{23}+x^5+1$
24	$x^{24}+x^4+x^3+x+1$
25	$x^{25}+x^3+1$
26	$x^{26}+x^8+x^7+x+1$
27	$x^{27}+x^8+x^7+x+1$
28	$x^{28}+x^3+1$
29	$x^{29}+x^2+1$
30	$x^{30}+x^{16}+x^{15}+x+1$

二進位域與多項式

* 把二進位數字表示為多項式
  - 例如11011，可以表示成為 $1 + x + x^3 + x^4$
  - 二進位1+1=2=0，所以我們每次計算後相當於模上 2。例如計算11*11，在多項式表示就是

$$(1 + x) * (1 + x) = 1 + 2 * x + x^2 = 1 + x^2$$

  - 這的x的係數2在二進位中等於0。另外很重要的概念就是二進位域中，加等於減，例如

$$x^2 + 1 = 0 \quad \text{相當於} \quad x^2 = 1$$

BCH碼

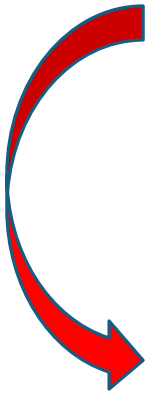
* BCH碼廣泛應用於無線通訊。
* BCH碼（BCH codes、Bose–Chaudhuri–Hocquenghem codes）為取自Bose、Ray-Chaudhuri與Hocquenghem的縮寫。
* BCH碼一般是指binary BCH碼，RS碼一般是指nonbinary BCH碼。
* Let $c(x) = c_0 + c_1x + \ldots + c_{n-1}x^{n-1}$ 是一個BCH碼的碼字，係數$c_i \in GF(2)$。
* 任何兩個整數m和t，二進制(n, k)BCH碼的參數如下：
  - 碼長度：$n = 2^m - 1$
  - 同位查核位元數：$n - k \leq mt$
  - 最小距離：$d_{\min} \geq 2t + 1$

BCH碼

* 循環碼之中BCH碼功能最強大且廣泛應用於無線通訊，任何兩個整數m和t，二進制(n, k)BCH碼的參數如下：
  - 碼長度：$n = 2^m - 1$
  - 同位查核位元數：$n - k \leq mt$
  - 最小距離：$d_{\min} \geq 2t + 1$
* BCH碼可以更正t個位元以內的錯誤，其產生多項式由$(x^{(2^m - 1)} + 1)$的因式建構得到
* BCH碼的參數選擇具彈性
* BCH碼參數及產生多項式通常可查表得知

BCH範例

* Example :
  - 發送的訊息$m(x)$是1001
  - 編碼多項式$p(x)$是1101
  - 要發送出去的訊息就是1100101



- 假如接收者也接收到了這個訊息，用$p(x)$除後余項是 0，沒有錯誤存在，商是1001，就是發送者發送的訊息，自然皆大歡喜。
- 但如果發送過程中，有一個位元被改變了，變成了1100111，接收者同樣用$p(x)$去除，得到了餘項1110000，這下不好了，接收者知道出錯了。但現在怎麼去找出錯誤多項式$e(x)$呢？

BCH範例

* Example :
  - 發送的訊息$m(x)$是1001
  - 編碼多項式$p(x)$是1101
  - 要發送出去的訊息就是1100101

變成了1100111，接收者同樣用$p(x)$去除，得到了餘項1110000

e(x)	e(x)/p(x) 的餘項
1000000	1000000
0100000	0100000
0010000	0010000
0001000	1100000
0000100	0110000
0000010	1110000
0000001	1010000

BCH碼

* 但如果想要糾正多個錯誤呢？此時僅一個編碼多項式$p(x)$就不夠了，還需要更多的編碼多項式來讓我們能找出多個錯誤。

* BCH 編碼多項式標準形式是：

$$Q(x) = p(x)p_3(x)p_5(x)\cdots p_{2t-1}(x)$$

* 其中$p(x)$是本原多項式，而 $p_3(x^3)$，$p_5(x^5)\ldots$ 都是可以被$p(x)$除餘0的多項式。即

$$p_3(x^3) = p_5(x^5)\cdots p_{2t-1}(x^{2t-1}) = 0 \pmod{p(x)}$$

BCH碼

* 訊息$m(x)$是11010
* 選擇的本原多項式是 $1+x+x^4$　　則

$$p_3(x)=1+x+x^2+x^3+x^4 \quad , \quad p_5(x)=1+x+x^2$$

* 編碼多項式就是

$$Q(x)=p(x)p_3(x)p_5(x)=(1+x+x^4)(1+x+x^2+x^3+x^4)(1+x+x^2)$$
$$=1+x^2+x^3+x^4+x^5+x^8+x^{10}$$

* 表示成二進位是11101100101，再乘上訊息$m(x)$就是100001110110010，就是我們發送的出去的。

$$r(x)=m(x)p(x)p_3(x)p_5(x)+x^{e_1}+x^{e_2}+x^{e_3}$$

BCH碼

* 訊息$m(x)$ 是100001110110010
* 但接收者收到的$r(x)$卻是 1010000110110010。然後接收者開始計算：

$$r(x) = 1 + x^2 + x^6 + x^7 + x^9 + x^{10} + x^{13} = x$$

$$r(x^3) = 1 + x^6 + x^{18} + x^{21} + x^{27} + x^{30} + x^{39} = x^{13}$$

$$r(x^5) = 1 + x^{10} + x^{30} + x^{35} + x^{45} + x^{50} + x^{65} = 0$$

* 前兩項餘項不為0，而第三項為0說明接到的只有兩處錯誤。
* 可計算得出第一個錯誤在第三個位，第二個錯誤在第六個位，沒有第三個錯誤。計算方式請參考:https://zhuanlan.zhihu.com/p/95909150

Channel coding techniques for 5G wireless networks

* 3rd Generation Partnership Project (3GPP)
  - 成立於1998年12月的標準化機構
  - 成員包括歐洲的ETSI、日本的ARIB和TTC、中國的CCSA、韓國的TTA、北美洲的ATIS和印度的電信標準開發協會
  - best known work:
    - GSM and related **2G** and **2.5G** standards, including GPRS and EDGE
    - UMTS and related **3G** standards, including HSPA and HSPA+
    - LTE and related **4G** standards, including LTE Advanced and LTE Advanced Pro
    - 5G NR and related **5G** standards

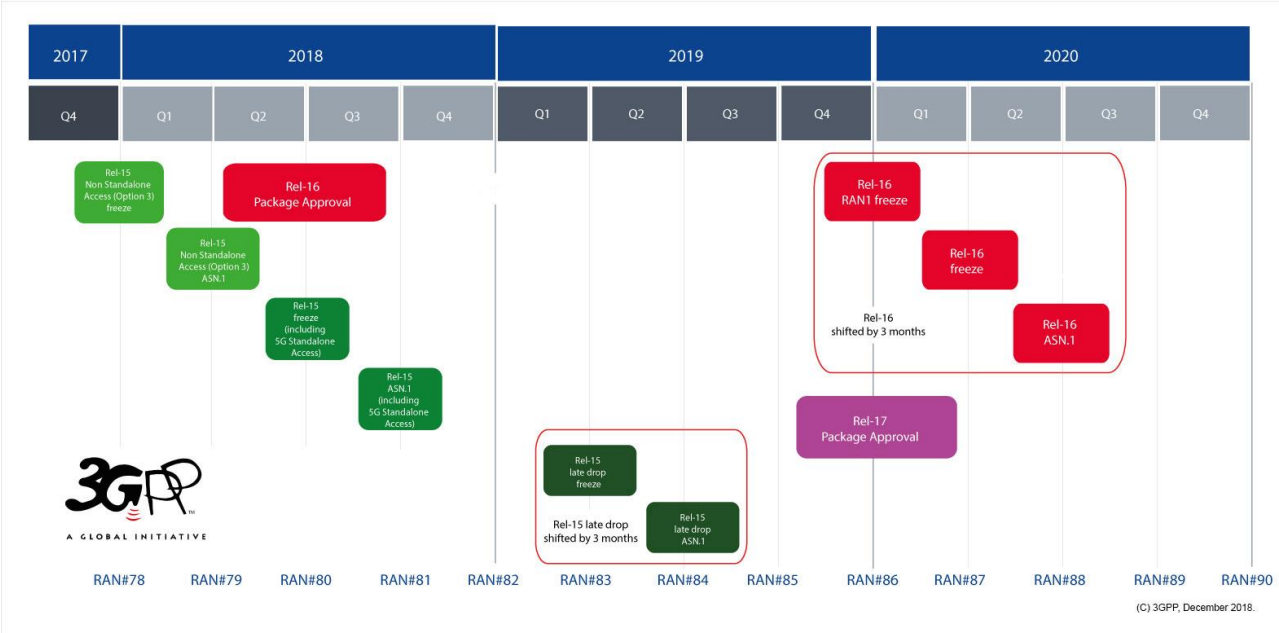
https://zh.wikipedia.org/wiki/3GPP
https://www.3gpp.org/

A GLOBAL INITIATIVE

Channel coding techniques for 5G wireless networks

* 3rd Generation Partnership Project (3GPP)
  * release-15 5G new radio (NR) access technology

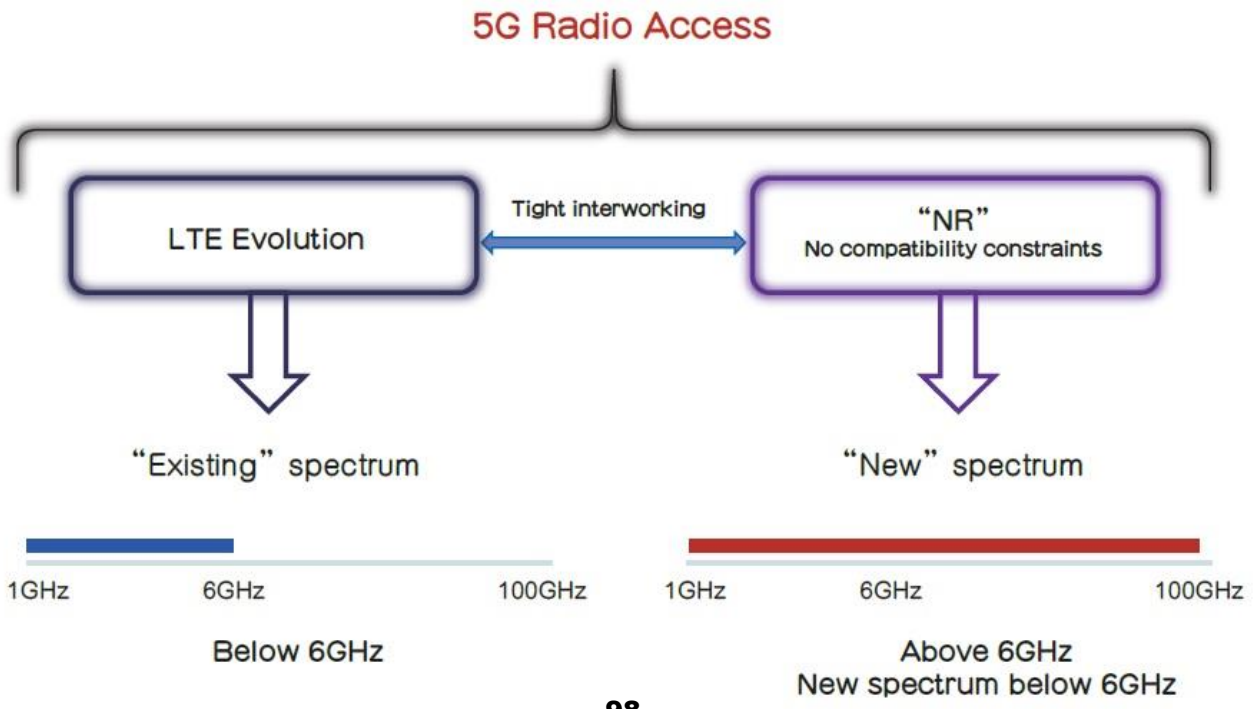
    standards by 3rd Generation Partnership Project (3GPP)



https://zh.wikipedia.org/wiki/3GPP
https://www.3gpp.org/release-15

Channel coding techniques for 5G wireless networks

* 5G無線電接取(Radio Access)架構由LTE Evolution和新無線電接取技術(New Radio Access Technology，RAT)，也就是NR組成。NR工作在1GHz~100GHz。



https://www.edntaiwan.com/20170717-5g-new-radio/

Channel coding techniques for 5G wireless networks

* 5G NR uses low density parity check (LDPC) codes for the data transmission for mobile broadband (MBB) services and polar codes for the control signaling.

註: 5G NR，它指的是5G新空中介面(New Radio)

https://moniem-tech.com/questions/what-is-the-channel-coding-in-5g-nr/

Low-density parity-check (LDPC)

* 低密度奇偶檢查碼(**Low-Density Parity-Check, LDPC**) 於1960 年代被提出

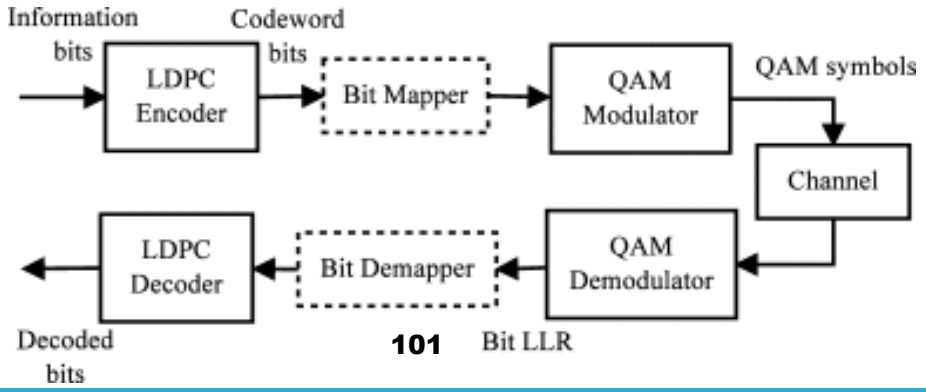
* LDPC code 是線性分組碼（linear block code）的一種，用於更正傳輸過程中發生錯誤的編碼方式，錯誤修正碼(Error Correcting Coding)，也有人稱糾錯編碼

* LDPC碼是目前既存錯誤修正碼當中編碼增益最高的方式

https://zh.wikipedia.org/wiki/%E4%BD%8E%E5%AF%86%E5%BA%A6%E5%A5%87%E5%81%B6%E6%AA%A2%E6%9F%A5%E7%A2%BC
https://www.2cm.com.tw/2cm/zh-tw/tech/CFA2CF3F97934D73825179497DC7B5A3

Application for LDPC Codes

* Wireless, Wired, and Optical Communications.
* Different throughput requirement
* Need to design codes that work with multi-level modulation (e.g. QAM or M-PSK)
* LDPC Application for next generation communication systems (Wireless, OFDM, ADSL).



https://www.sciencedirect.com/science/article/abs/pii/S1051200417301690

Low-density parity-check (LDPC)

* 定義:
  - LDPC Code（低密度同位碼）: Linear block code with a sparse parity-check matrix. LDPC碼的校驗矩陣具有非常強的稀疏性，也就是校驗矩陣裡面“0”佔大多數，“1”的數量極少。“1”元素的分佈非常稀疏，所以是低密度的。
  - Sparse parity-check matrix also called an LDPC matrix.
  - H: LDPC matrix
    - Number of 1s << n(n-k)
    - 收到的資訊位元和校驗矩陣的每一行的每一個元素對應相乘再相加，最後得到（行數）個結果，如果各個結果都是0那麼就是沒錯誤。

https://www.youtube.com/watch?v=rBorhQKyV34

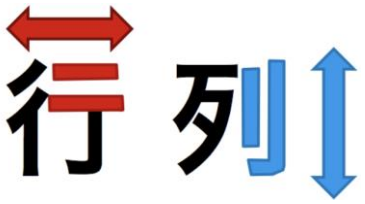
Low-density parity-check (LDPC)

* H: LDPC matrix ，H矩陣（校驗矩陣）構造
  - 每行的“1”元素數量一致，均為k，k為行重。
  - 每列的“1”元素數量一致，均為j，j為列重。
  - 在該矩陣中每兩列的相同位置均為“1”的個數不超過1 。

發送碼字長度為 N，資訊位元長度為 K。所以校驗資訊長度為 $M=N-K$。

碼率就是 $R=K/N$，我們需要的LDPC碼校驗矩陣H 大小為 $M * N$。

$$H=\begin{bmatrix}1&1&0&1&0&0\\0&1&1&0&1&0\\1&0&0&0&1&1\\0&0&1&1&0&1\end{bmatrix}$$

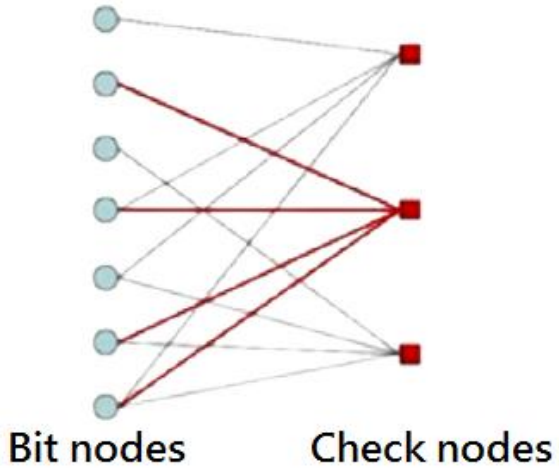


https://www.youtube.com/watch?v=rBorhQKyV34

Low-density parity-check (LDPC)

* Tanner Graph
  - Graphical representation of a parity-check matrix.
  - Tanner Graph 包含兩類頂點：***n*** 個碼字比特頂點（Bit nodes），分別與校驗矩陣的各列對應；***m*** 個校驗方程頂點（Check nodes）

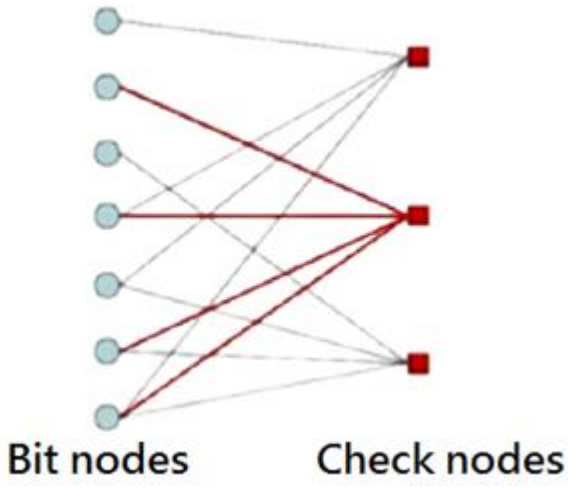
$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$



Tanner Graph

- Bit nodes <-> Columns
- Check nodes <-> Rows
- Column weight (w_r) - number of '1's in a column Number of times a symbol taking part in parity checks
- Row weight (w_c) - number of '1's in a row Number of Number of symbols taking part in a parity check
- If the parity check matrix has uniform row weight and uniform column weight (same number of '1' in a column and same number of '1' in a row) we call that a regular parity check matrix.

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$



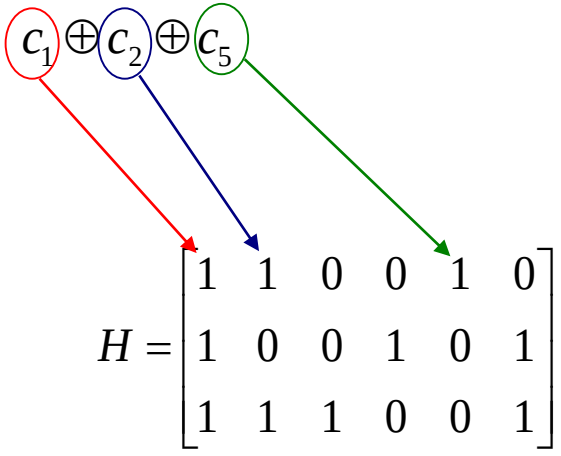
Each check node

* Consider the 6 bit long codeword in the form $c = [c_1, c_2, c_3, c_4, c_5, c_6]$ which satisfies 3 parity check equations as shown below.

$$c_1 \oplus c_2 \oplus c_5 = 0$$
$$c_1 \oplus c_4 \oplus c_6 = 0$$
$$c_1 \oplus c_2 \oplus c_3 \oplus c_6 = 0$$

w_r and w_c changes, therefore this is an **irregular** parity check matrix

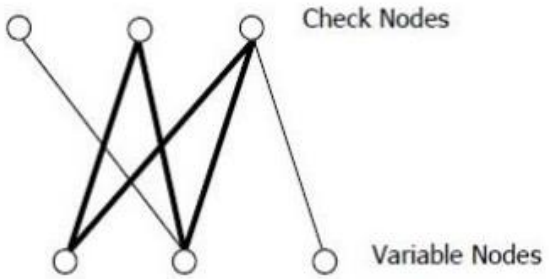
We can now define 3x6 parity check matrix as,



Low-density parity-check (LDPC)

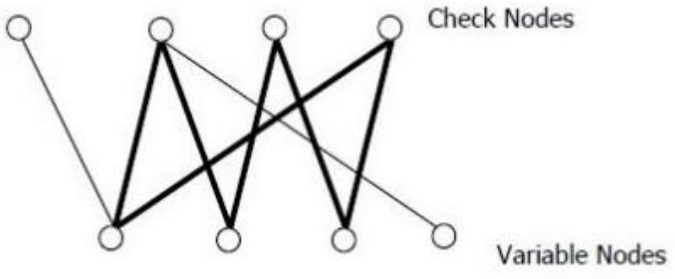
* Cyclic Length of LDPC Codes

Length 4 Cycle



$$H = \begin{bmatrix} .. & .. & .. & .. & .. \\ .. & 1 & .. & 1 & .. \\ .. & .. & .. & .. & .. \\ .. & 1 & .. & 1 & .. \\ .. & .. & .. & .. & .. \end{bmatrix}$$

Length 6 Cycle

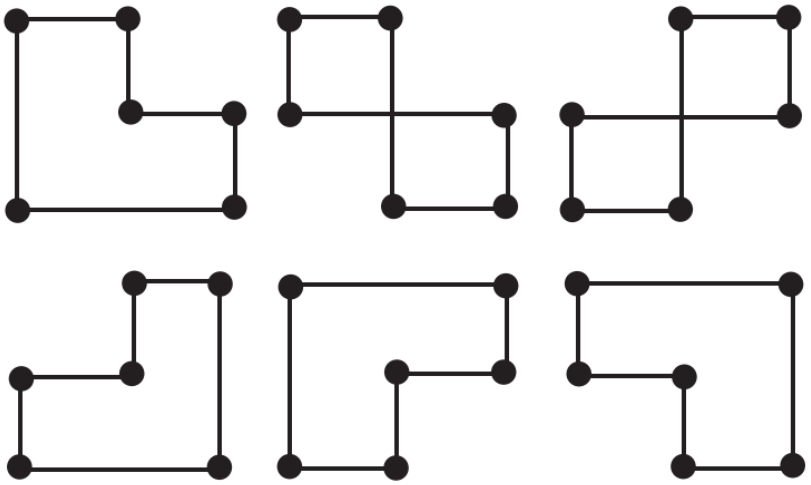


$$H = \begin{bmatrix} .. & .. & .. & .. & .. & .. & .. \\ .. & 1 & .. & .. & .. & 1 & .. \\ .. & .. & .. & .. & .. & .. & .. \\ .. & .. & .. & 1 & .. & 1 & .. \\ .. & .. & .. & .. & .. & .. & .. \\ .. & 1 & .. & 1 & .. & .. & .. \\ .. & .. & .. & .. & .. & .. & .. \end{bmatrix}$$

https://www.slideshare.net/bhagwatsinghmahecha/itc-final-report

Low-density parity-check (LDPC)

* H: LDPC matrix ，**短迴圈與消除短迴圈**
  - 由於短迴圈的存在會嚴重削弱 LDPC 碼的性能，因此構造時需儘量減少短迴圈。
  - The length of the smallest cycle in Tanner graph is called the **girth** of the Tanner graph.



Six kinds of 6-cycles with different figures

108

Lingjun Kong and Yang Xiao, Design of good QC-LDPC codes without small girth in the p-plane, Journal of Systems Engineering and Electronics Vol. 22, No. 2, April 2011, pp.183–187

LDPC Encoder

* LDPC Encoder
  - message ***m*** 為原始資料，經過Encoder後，編碼成codeword ***c***，對應的運算式為：***c = m · G***
  - 其中***G*** 為Encdoer的generator matrix。假設message的長度為***k***, ***m*** = [m_1, m_2,…,m_k] , codeword的長度為***n***， ***c*** = [c_1, c_2,…,c_n] 。則***G***為***k*** x ***n***的矩陣

$$G = \begin{pmatrix} \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_k \end{pmatrix} = \begin{pmatrix} g_{11} & \cdots & g_{1n} \\ \vdots & \ddots & \vdots \\ g_{k1} & \cdots & g_{kn} \end{pmatrix}$$

同時，可將此簡記 *c*為(*n*, *k*) block code.

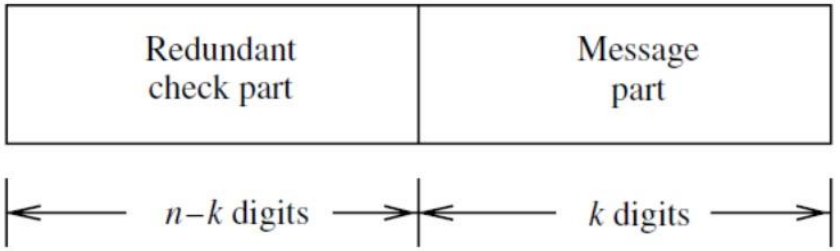
LDPC Encoder

* LDPC Encoder
  * 如 ***G*** 矩陣為如下形式:

$$G = [P\ I_k] = \begin{pmatrix} p_{1,1} & \cdots & p_{1,n-k} & 1 & 0 & \cdots & 0 \\ p_{2,1} & \cdots & p_{2,n-k} & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{k,1} & \cdots & g_{k,n-k} & 0 & 0 & \cdots & 1 \end{pmatrix}$$

其中，*P*為*k* x (*n*-*k*)矩陣，I_k為*k* x *k*矩陣。此encoder為systematic encoder，且 $C = \mathbf{m}[P\ I_k] = [\mathbf{m}P\ \mathbf{m}]$

codeword由兩部分組成：message ***m*** 和parity check ***mP***



LDPC Encoder

$$H=\begin{bmatrix}1&1&0&0&1&0\\1&0&0&1&0&1\\1&1&1&0&0&1\end{bmatrix}$$

The $H_{M\times N}$ parity check matrix defines a rate $R=K/N$, (N,K) code where $K=N-M$

Codeword is said to be **valid** if it satisfies the syndrome calculation

$$z=c\bullet H^T=0$$

We can generate the codeword in by multiplying message m with generator matrix G

$$c=m\bullet G$$

We can obtain the generator matrix G from parity check matrix H by,

1.) Arranging the parity check matrix in systematic form using row and column operations $H_{sys}=\left[I_M\middle|P_{M\times K}\right]$ $H_{sys}=\begin{bmatrix}1&0&0&1&0&1\\0&1&0&1&1&1\\0&0&1&0&1&1\end{bmatrix}$

2.) Rearranging the systematic parity check matrix $G=\left[P_{K\times M}^T\middle|I_K\right]$ $G=\begin{bmatrix}1&1&0&1&0&0\\0&1&1&0&1&0\\1&1&1&0&0&1\end{bmatrix}$

3.) 此$(N-K)\times N$ 矩陣為Encdoer的parity check matrix，且 $G\bullet H^T=0$

LDPC Encoder

1. Suppose we have codeword c as follows: $c = [c_1, c_2, c_3, c_4, c_5, c_6]$ where each c_i is either '0' or '1' and codeword now has three parity-check equations.

$$c_1 \oplus c_2 \oplus c_5 = 0$$
$$c_1 \oplus c_4 \oplus c_6 = 0$$
$$c_1 \oplus c_2 \oplus c_3 \oplus c_6 = 0$$

a.) Determine the parity check matrix ***H*** by using the above equation

b.) Show the systematic form of ***H*** by applying Gauss Jordan elimination

c.) Determine Generator matrix ***G*** from ***H*** and prove $\mathbf{G} \bullet \mathbf{H}^T = \mathbf{0}$

d.) Find out the dimension of the ***H, G***

e.) State whether the matrix is regular or irregular

LDPC Encoder

$$c_1 \oplus c_2 \oplus c_5 = 0$$
$$c_1 \oplus c_4 \oplus c_6 = 0$$
$$c_1 \oplus c_2 \oplus c_3 \oplus c_6 = 0$$

$$H = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

6 columns and 3 rows

Convert H into systematic form H_{sys} using basic row and operations (try to avoid column operations)

$$H = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \xrightarrow{R_2 \leftrightarrow R_3} \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \xrightarrow{R_2 = R_3 + R_2} \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \xrightarrow{R_3 = R_3 + R_2 + R_1} \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

LDPC Encoder

$$H=\begin{bmatrix}1&1&0&0&1&0\\0&1&1&1&0&0\\0&0&1&0&1&1\end{bmatrix}\xrightarrow{R_1=R_1+R_2+R_3}\begin{bmatrix}1&0&0&1&0&1\\0&1&1&1&0&0\\0&0&1&0&1&1\end{bmatrix}$$

$$H=\begin{bmatrix}1&0&0&1&0&1\\0&1&1&1&0&0\\0&0&1&0&1&1\end{bmatrix}\xrightarrow{R_2=R_2+R_3}\begin{bmatrix}1&0&0&1&0&1\\0&1&0&1&1&1\\0&0&1&0&1&1\end{bmatrix}$$

$$H_{sys}=\begin{bmatrix}1&0&0&1&0&1\\0&1&0&1&1&1\\0&0&1&0&1&1\end{bmatrix}$$

$$H_{sys}=\left[I_M\middle|P_{M\times K}\right]\qquad G=\left[P^T_{K\times M}\middle|I_K\right]$$

$$G=\begin{bmatrix}1&1&0&1&0&0\\0&1&1&0&1&0\\1&1&1&0&0&1\end{bmatrix}$$

$$G\bullet H^T=\begin{bmatrix}1&1&0&1&0&0\\0&1&1&0&1&0\\1&1&1&0&0&1\end{bmatrix}\cdot\begin{bmatrix}1&1&1\\1&0&1\\0&0&1\\0&1&0\\1&0&0\\0&1&1\end{bmatrix}=\begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix}$$

$$G\bullet H^T=0$$

Dimensions of H = 3x6

Dimensions of G = 3x6

Since the column weight and row weight changes, this is an **irregular parity check matrix**

LDPC Encoder

2. Consider parity check matrix **H** generated

a.) Determine message bits length *K*, parity bits length *M*, codeword length *N*

b.) Use the generator matrix **G** obtained in *question 1* to generate all possible codewords *c*.

$$H = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \qquad G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Dimensions of the H matrix = 3x6

$$M = 3, \quad N = 6, \quad K = N - M = 3$$

All possible message set

$$m = \begin{bmatrix} msg\_1 \\ msg\_2 \\ msg\_3 \\ msg\_4 \\ msg\_5 \\ msg\_6 \\ msg\_7 \\ msg\_8 \end{bmatrix} = \begin{bmatrix} 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{bmatrix}$$

All possible codeword set

$$c = \begin{bmatrix} codeword\_1 \\ codeword\_2 \\ codeword\_3 \\ codeword\_4 \\ codeword\_5 \\ codeword\_6 \\ codeword\_7 \\ codeword\_8 \end{bmatrix} = \begin{bmatrix} msg\_1.G \\ msg\_2.G \\ msg\_3.G \\ msg\_4.G \\ msg\_5.G \\ msg\_6.G \\ msg\_7.G \\ msg\_8.G \end{bmatrix} = \begin{bmatrix} 000000 \\ 111001 \\ 011010 \\ 100011 \\ 110100 \\ 001101 \\ 101110 \\ 010111 \end{bmatrix}$$

LDPC Encoder

Question

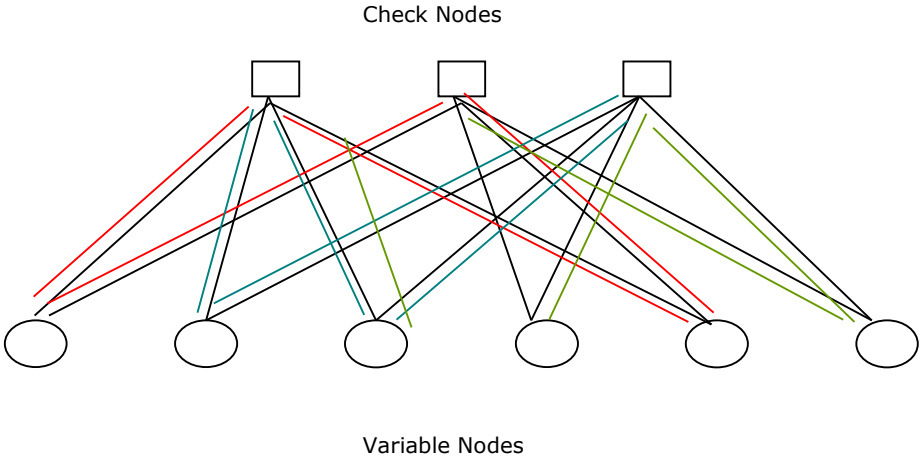
a.) What is the difference between regular and irregular LDPC codes?

b.) What is the importance of cycles in parity check matrix?

c.) Identify the cycles of 4 in the following tanner graph.

If the parity check matrix has uniform row weight and uniform column weight (same number of '1' in a column and same number of '1' in a row) we call that a regular parity check matrix.

The decoding algorithm assumes that the LDPC code is cycle free (large girth sizes). The short cycles in the code (cycles with a girth of 4 and cycles of girth of 6) weakens the code. Therefore the codes must be carefully constructed to be free of short cycles.

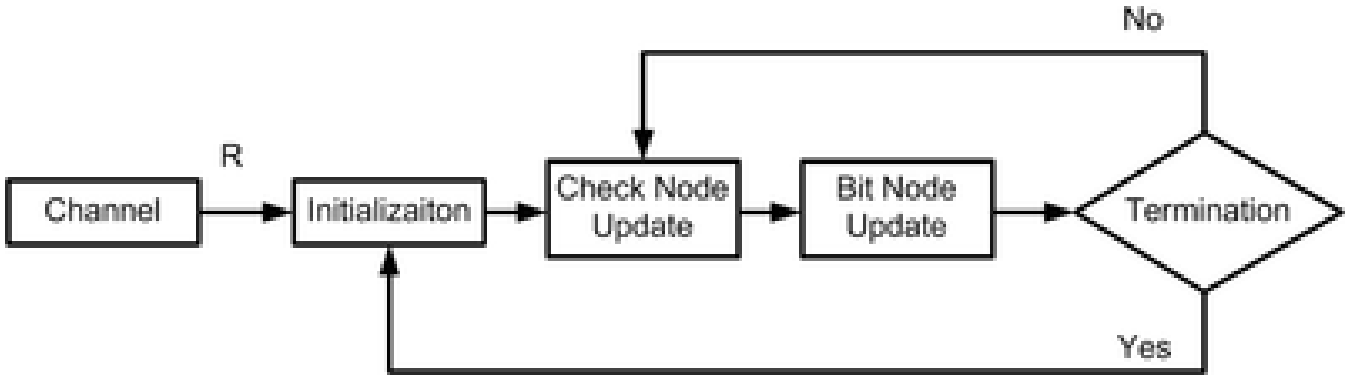


Richardson 提出一種有效率的編碼適用於任何矩陣，且減少矩陣運算的複雜度及運算量。

- Efficient Encoding of Low-Density Parity-Check Codes, IEEE Transactions on Information, 2001

LDPC Decoder

* LDPC Decoder: LDPC 的解碼方式採用遞迴運算(Iterative Decoding)，利用多次重覆傳送位元節點與檢查節點之間的訊息使解碼結果逼近原始資料，通常遞迴的次數越多則解碼值越正確。



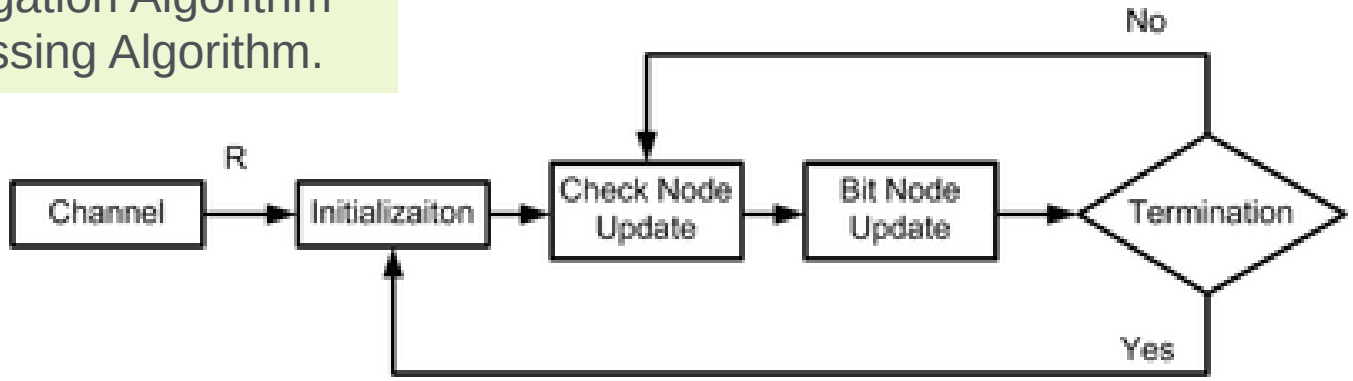
基於有疊代性(iterative)的置信傳播(belief propagation)

https://zhuanlan.zhihu.com/p/20258157

https://zh.wikipedia.org/wiki/%E4%BD%8E%E5%AF%86%E5%BA%A6%E5%A5%87%E5%81%B6%E6%AA%A2%E6%9F%A5%E7%A2%BC

LDPC Decoder

LDPC codes can be decoded by an iterative decoding algorithm as Belief Propagation Algorithm or Message Passing Algorithm.



- 當接收資料**R**從通訊頻道(channel)進入低密度奇偶檢查碼的解碼器，解碼器會對訊息作初始化(initialization)。
- **check node**根據互相連接的多個**bit node**內的資料做更新運算(update)。
- **bit node**對相連接的多個**check node**內的資料做更新運算。
- 觀察終止(termination)條件，來決定是否繼續疊代計算。

https://zhuanlan.zhihu.com/p/20258157

https://zh.wikipedia.org/wiki/%E4%BD%8E%E5%AF%86%E5%BA%A6%E5%A5%87%E5%81%B6%E6%AA%A2%E6%9F%A5%E7%A2%BC

LDPC Decoder

* LDPC Decoder:
  - message-passing decoding: message-passing是指運用資訊在相連的check nodes和bit nodes間傳遞，簡單但存在糾錯的缺陷。
  - bit-flipping decoding :位元翻轉解碼法有較低的複雜度，但其解碼效能相對於和積演算法(sum-product algorithm)較不理想。
  - sum-product decoding :總和-乘積演算法具有較佳的錯誤更正能力，卻具較高的計算複雜度。
  - min-sum decoding :最小值-總和演算法在稍微減低的錯誤更正能力下，換取較低的計算複雜度。

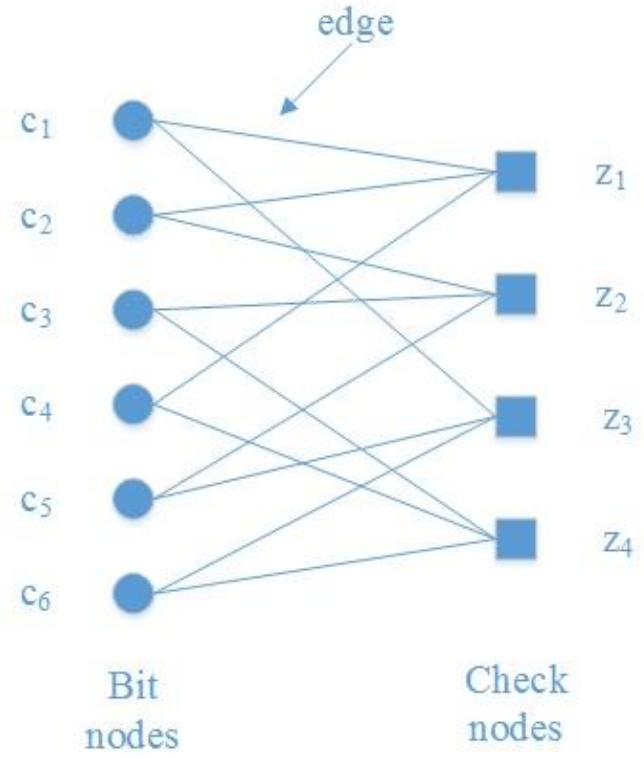
https://zhuanlan.zhihu.com/p/20258157

https://zh.wikipedia.org/wiki/%E4%BD%8E%E5%AF%86%E5%BA%A6%E5%A5%87%E5%81%B6%E6%AA%A2%E6%9F%A5%E7%A2%BC

LDPC Decoder

* bit-flipping decoding:
  * 由H矩陣，編碼得到一個codeword *c*=[0 0 1 0 1 1]。經過通道後，接收到的信號 *r*=[1 0 1 0 1 1]. Bit-flipping decoding 的糾錯過程如下：

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

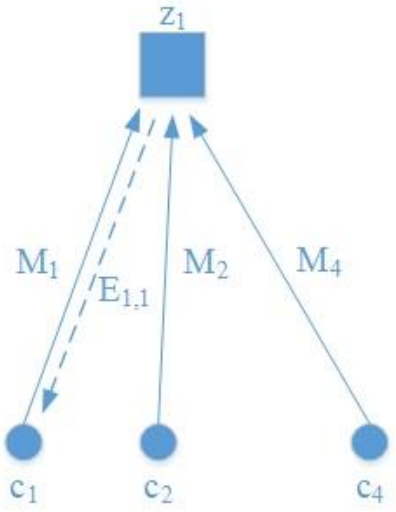


LDPC Decoder

* bit-flipping decoding:
  - 初始化：bit nodes傳遞給check nodes的初始訊息 M=[1 0 1 0 1 1].
  - Check nodes update: check node $z1$接收來自bit node $c1$, $c2$, $c4$的訊息，則

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

$$E_{1,1} = \sum_{n'=N_{1,1}} M_{n'} = M_2 \oplus M_4 = 0$$



即回饋給c1的資訊 $E_{1,1} = 0$, 如圖所示。

LDPC Decoder

* bit-flipping decoding:

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

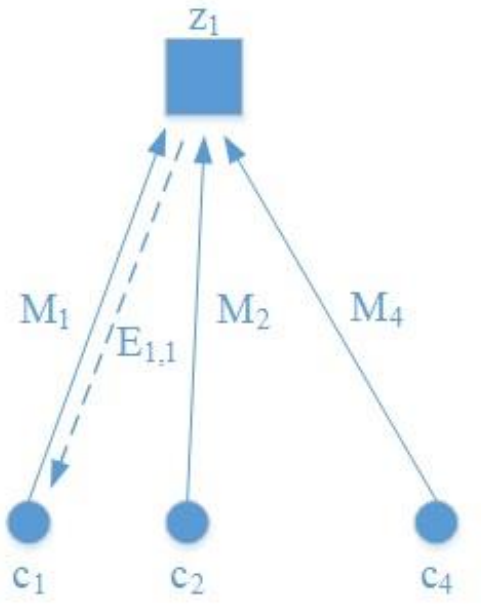
回饋給c2的資訊 $E_{1,2} = \sum_{n'=N_{1,2}} M_{n'} = M_1 \oplus M_4 = 1$

回饋給c4的資訊 $E_{1,4} = \sum_{n'=N_{1,4}} M_{n'} = M_1 \oplus M_2 = 1$

同理依次計算出 $E_{2,2} = 0,\ E_{2,3} = 1,\ E_{2,5} = 1$

$E_{3,1} = 0,\ E_{3,5} = 0,\ E_{3,6} = 0$

$E_{4,3} = 1,\ E_{4,4} = 0,\ E_{4,6} = 1$

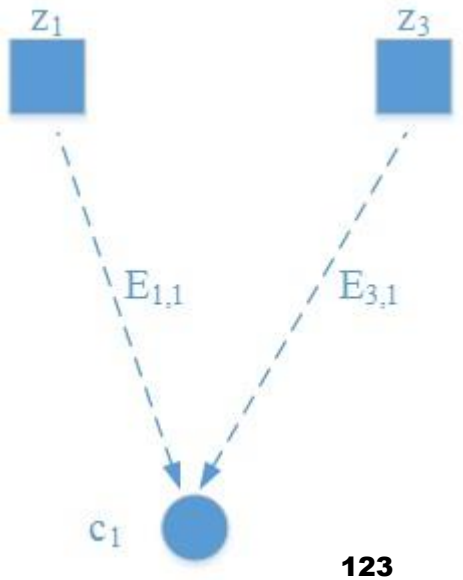


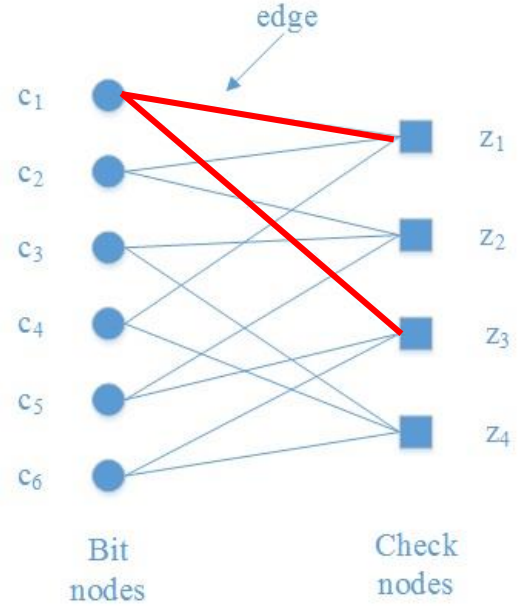
LDPC Decoder

* bit-flipping decoding:
  - Bit nodes update：bit node c1與check nodes z1，z3相連，如圖所示。$E_{1,1}=0, E_{3,1}=0$，而 $M_1=1$ 。根據服從大多數的原則，c1的資訊要翻轉，則 $M_1=0$ ；

codeword
c=[**0** 0 1 0 1 1]

M=[**1** 0 1 0 1 1]





LDPC Decoder

* bit-flipping decoding:

bit node c2與check nodes z1，z2相連，不滿足大多數的原則，M_2 保持不變；

$$E_{1,2}=1,\ E_{2,2}=0,\ M_2=0$$

bit node c3與check nodes z2，z4相連，則 M_3保持不變；

$$E_{2,3}=1,\ E_{4,3}=1,\ M_3=1$$

bit node c4與check nodes z1，z4相連，M_4保持不變；

$$E_{1,4}=1,\ E_{4,4}=0,\ M_4=0$$

bit node c5與check nodes z2，z3相連，M_5保持不變；

$$E_{2,5}=1,\ E_{3,5}=0,\ M_5=1$$

bit node c6與check nodes z3，z4相連，M_6保持不變。

$$E_{3,6}=0,\ E_{4,6}=1,\ M_6=1$$

Bit nodes update完成後，M=[0 0 1 0 1 1].

LDPC Decoder

* bit-flipping decoding:
  - 終止，驗證M是否滿足syndrome decoding:

$$s_1 = M_1 \oplus M_2 \oplus M_4 = 0$$

$$s_2 = M_2 \oplus M_3 \oplus M_5 = 0$$

同理， $s_3 = s_4 = 0$ 。因此糾錯成功，
c=[0 0 1 0 1 1] 為發送codeword.

LDPC Decoder

* sum-product decoding :
  * 假設在一通訊系統的頻道有AWGN屬性，而傳送訊號: $\mathbf{U}(u_1, u_2, \ldots, u_n)$，接收訊號是 $\mathbf{Y}(y_1, y_2, \ldots, y_n)$ ，bit node有*n*個，check node有*m*個。而總和-乘積演算法在解碼流程如下：

➢ 初始化：

假設AWGN的方差(variance)是 σ^2

- **bit node** *n*會被初始化成： $\lambda_{n\to m}(u_n) = \log\frac{P(u_n = 0/y_n)}{P(u_n = 1/y_n)} = \frac{2y_n}{\sigma^2}$。
- **check node** *m*會被初始化成： $\Lambda_{m\to n}(u_n) = 0$。

➢ Check nodes update:

- **check node** *m*更新為： $\Lambda_{m\to n}(u_n) = 2\tanh^{-1}\left\{\prod_{n'\in N(m)\backslash n}\tanh\left[\frac{\lambda_{n'\to m}(u_{n'})}{2}\right]\right\}$;

其中 $n' \in N(m)\backslash n$ 是連接到check node *m*的bit node組合，但不包含第*n*個bit node。

https://zhuanlan.zhihu.com/p/20258157

https://zh.wikipedia.org/wiki/%E4%BD%8E%E5%AF%86%E5%BA%A6%E5%A5%87%E5%81%B6%E6%AA%A2%E6%9F%A5%E7%A2%BC

LDPC Decoder

* sum-product decoding :

  - Bit nodes update ·

    • **bit node** *n*更新為： $\lambda_{n\to m}(u_n)=\frac{2y_n}{\sigma^2}+\sum_{m'\in M(n)\backslash m}\Lambda_{m'\to n}(u_n)$;

    其中 $m'\in M(n)\backslash m$ 是連接到bit node *n*的check node組合，但不包含第*m*個check node。

  - 終止

    • 解碼位元計算：假設解碼後訊號為 $\hat{\mathbf{U}}(\hat{u}_1,\hat{u}_2,\ldots,\hat{u}_n)$ 。

    • Hard-decision解碼位元可由右列兩式求出：

$$\lambda_n(u_n)=\frac{2y_n}{\sigma^2}+\sum_{m\in M(n)}\Lambda_{m\to n}(u_n)$$

$$\hat{u}_i=\begin{cases}0, & \text{if } \lambda_i(u_i)\geq 0\ \forall i\in\{1,2,\ldots,n\}\\ 1, & \text{if } \lambda_i(u_i)\leq 0\ \forall i\in\{1,2,\ldots,n\}\end{cases}$$

只要解碼後的碼字 *v* 在恆等式 $\mathbf{H}\mathbf{v}^T=0$ 成立，即終止疊代計算。

https://zhuanlan.zhihu.com/p/20258157

https://zh.wikipedia.org/wiki/%E4%BD%8E%E5%AF%86%E5%BA%A6%E5%A5%87%E5%81%B6%E6%AA%A2%E6%9F%A5%E7%A2%BC

LDPC code in standards

* Protograph construction
  * Base graphs for different rates
  * Expanded by right shift permutation matrices
  * Optimized for performance vs complexity
* 5G
  * Two base graphs - LDPC codes with a structured H matrix, which can be generated by the expansion of a $Z \times Z$ base matrix
  * Several expansions
  * Shortening and puncturing for multiple rates

Protograph example

* Protograph construction

$$B = \begin{bmatrix} 1 & -1 & 3 & 1 & 0 & -1 \\ 2 & 0 & -1 & 0 & 0 & 0 \\ -1 & 4 & 2 & 1 & -1 & 0 \end{bmatrix}$$

Expansion factor: 5

$$H = \left[\begin{array}{cccccc|cccccc|cccccc|cccccc|cccccc|cccccc} 0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\ \hline 0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0\\ 0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0\\ 0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0\\ 1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&1\\ 0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0\\ \hline 0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0\\ 0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1\\ 0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0 \end{array}\right]$$

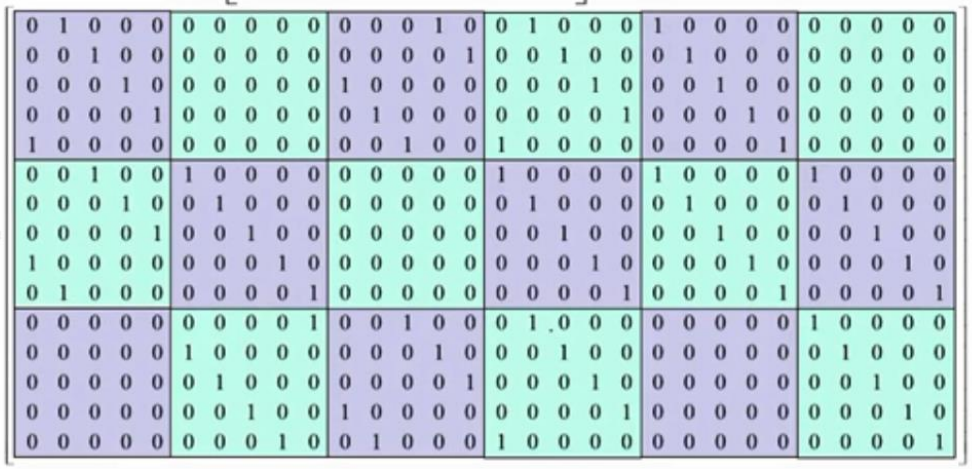
Protograph example

* Expansion factor: 5
* The base matrix will have entries: -1, 0, 1, 2, 3,4
* Base matrix -> expanded matrix (binary matrix)
* -1 => replace it with a 5 x 5 all zero binary matrix
* 0 => replaced by 5 x 5 identity matrix
* others => right shift permutation matrix

$$B = \begin{bmatrix} 1 & -1 & 3 & 1 & 0 & -1 \\ 2 & 0 & -1 & 0 & 0 & 0 \\ -1 & 4 & 2 & 1 & -1 & 0 \end{bmatrix}$$

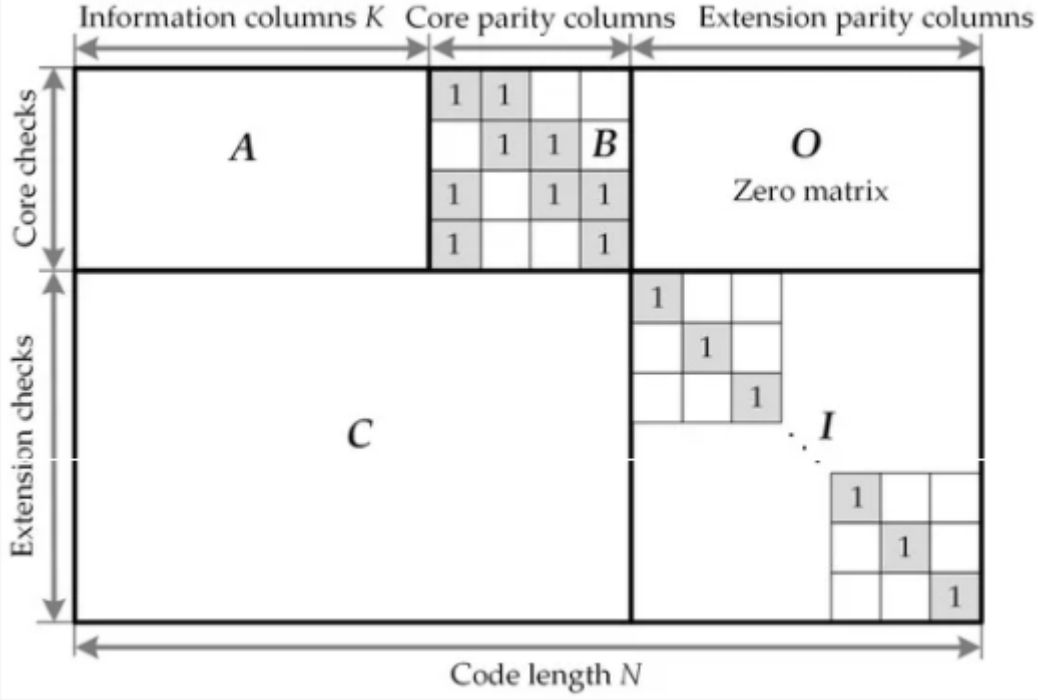
Expansion factor: 5

$H =$



5G NR base matrices

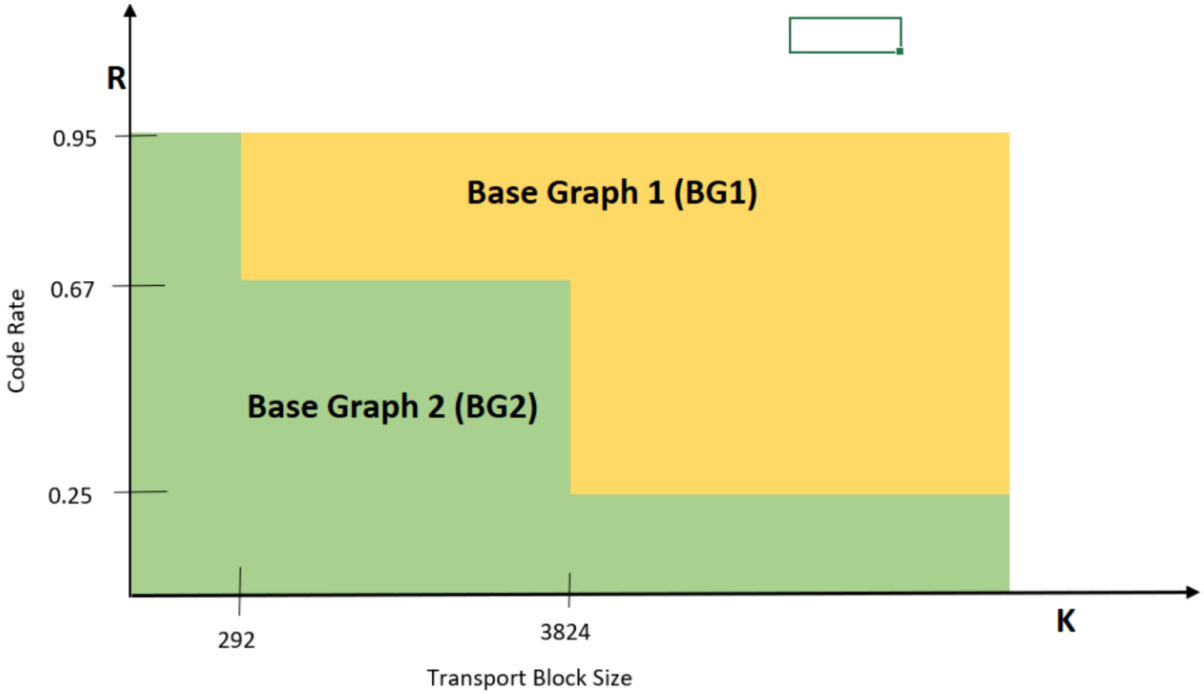
* Two base matrices in the specification, 38.212 (Multiplexing and channel coding)
  * BG1: 46 x 68 and BG2: 42 x 52



https://www.linkedin.com/pulse/5g-nr-dl-sch-ldpc-channel-coding-base-graph-selection-chelikani

5G NR base matrices

- BG1: 46 x 68
  - High rates
  - Long blocklengths
  - Code rate between 1/3 & 8/9
- BG2: 42 x 52
  - Smaller rates
  - Shorter blocklengths
  - Code rate between 1/5 & 2/3



https://www.linkedin.com/pulse/5g-nr-dl-sch-ldpc-channel-coding-base-graph-selection-chelikani

https://www.slideshare.net/hady_20021/5g-mmwaves-problems-and-solutions-graduation-project

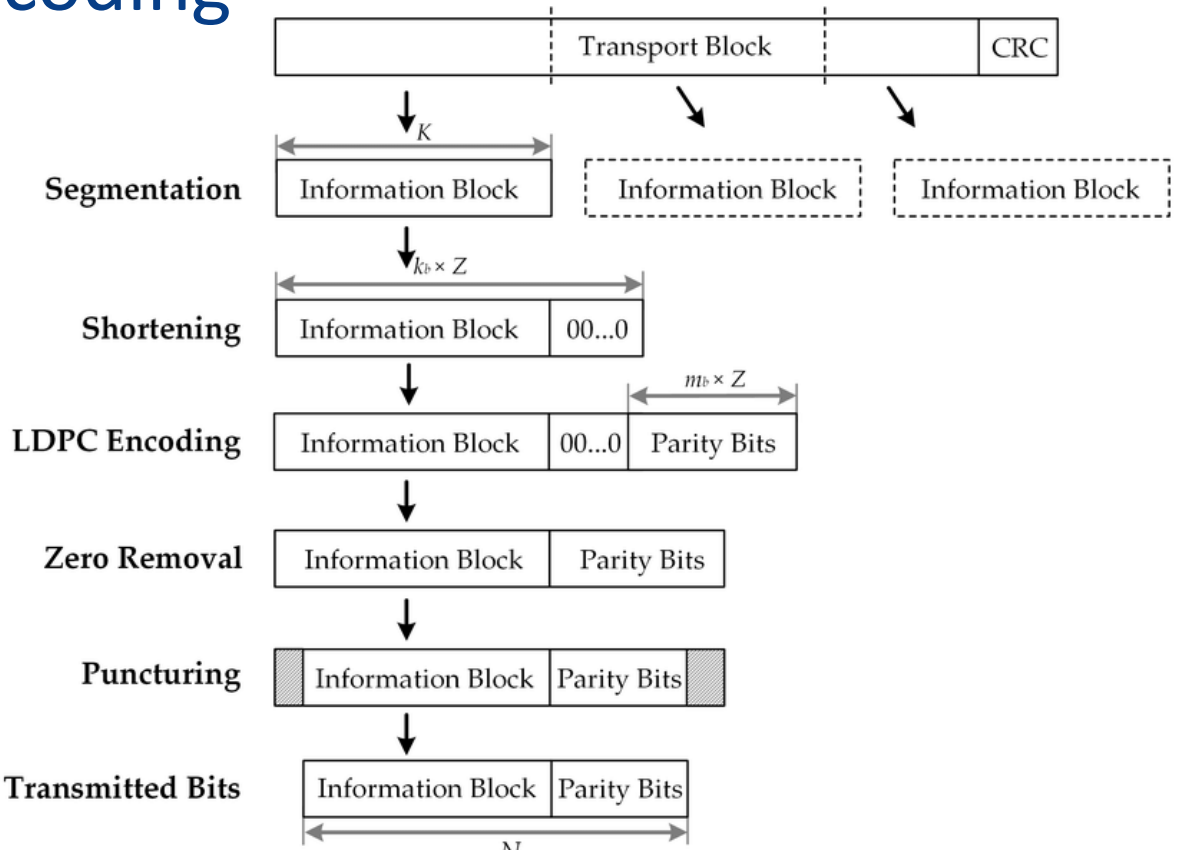
5G NR LDPC

* 5G-NR DL-SCH Low Density Parity Code (LDPC) Channel coding
  - For transmission of a DL transport block , a transport block CRC is first appended to provide error detection, followed by a LDPC base graph selection.
  - NR supports two LDPC base graphs, one for small transport blocks and one for larger transport blocks.
  - Then transport block is segmented into code blocks and code block CRC attachment is performed.
  - Each code block is individually LDPC encoded. The LDPC coded blocks are then individually rate matched.
  - Finally, code block concatenation is performed to create a codeword for transmission on the Physical Downlink Shared Channel (PDSCH) . Up to 2 code words can be transmitted simultaneously on the PDSCH.

https://www.linkedin.com/pulse/5g-nr-dl-sch-ldpc-channel-coding-base-graph-selection-chelikani

5G NR LDPC

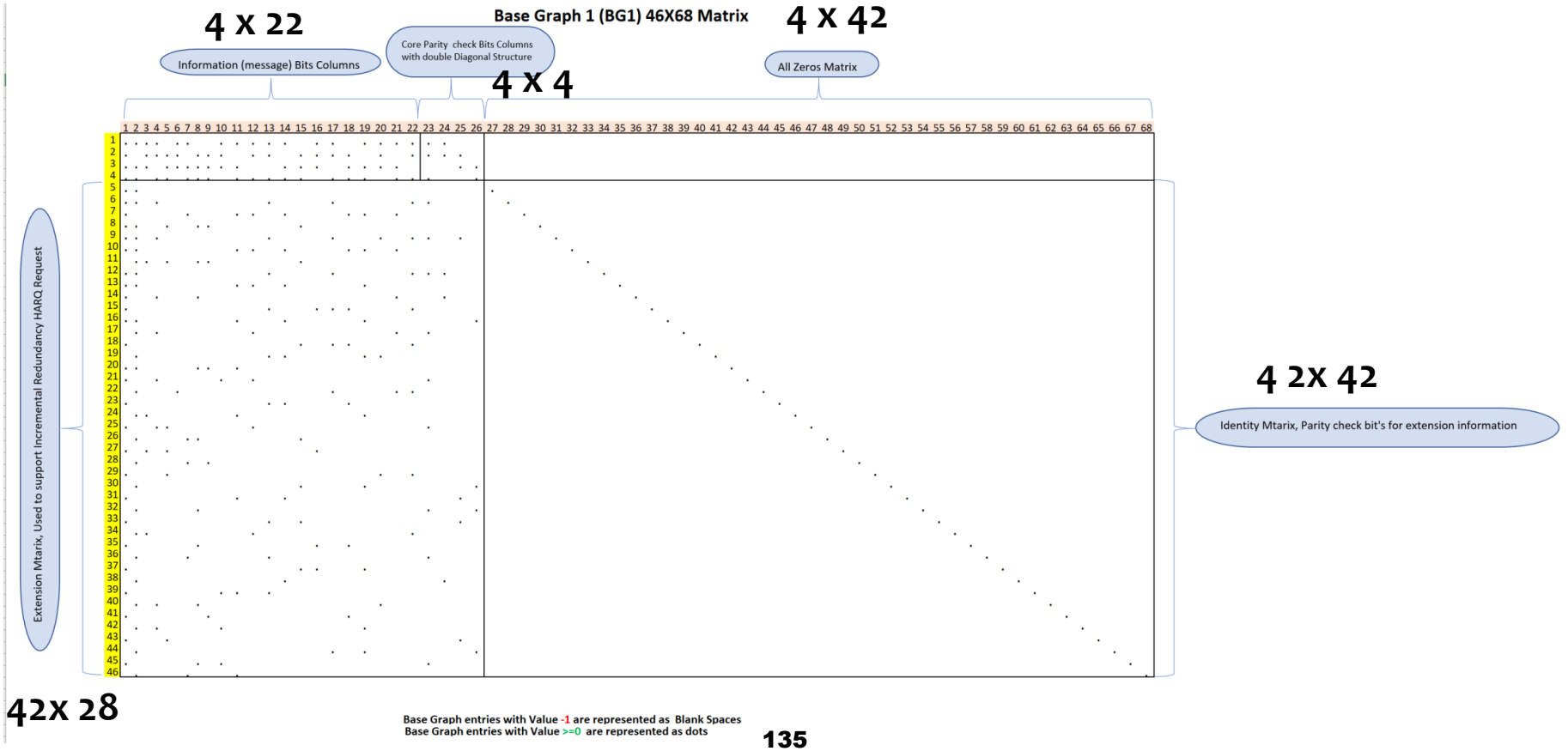
* 5G-NR DL-SCH Low Density Parity Code (LDPC) Channel coding



https://www.linkedin.com/pulse/5g-nr-dl-sch-ldpc-channel-coding-base-graph-selection-chelikani

5G NR base matrices – BG1

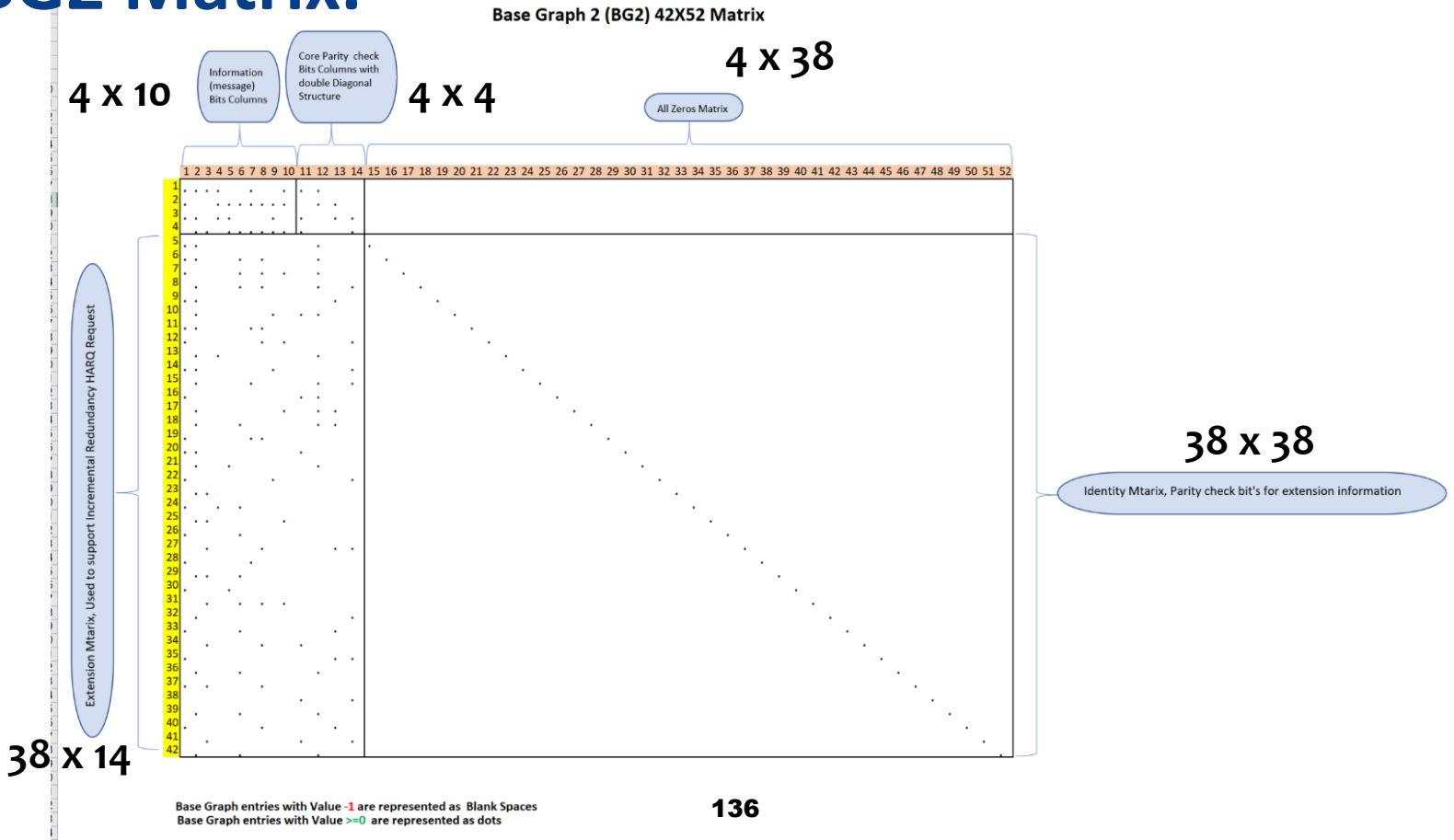
* **High Level visualization of fully populated BG1 Matrix.**



https://www.linkedin.com/pulse/5g-nr-dl-sch-ldpc-channel-coding-base-graph-selection-chelikani

5G NR base matrices – BG2

* **High Level visualization of fully populated BG2 Matrix.**



Sample Code for LDPC

* https://shubhamchandak94.github.io/LDPC-codes/

Software for Low Density Parity Check Codes

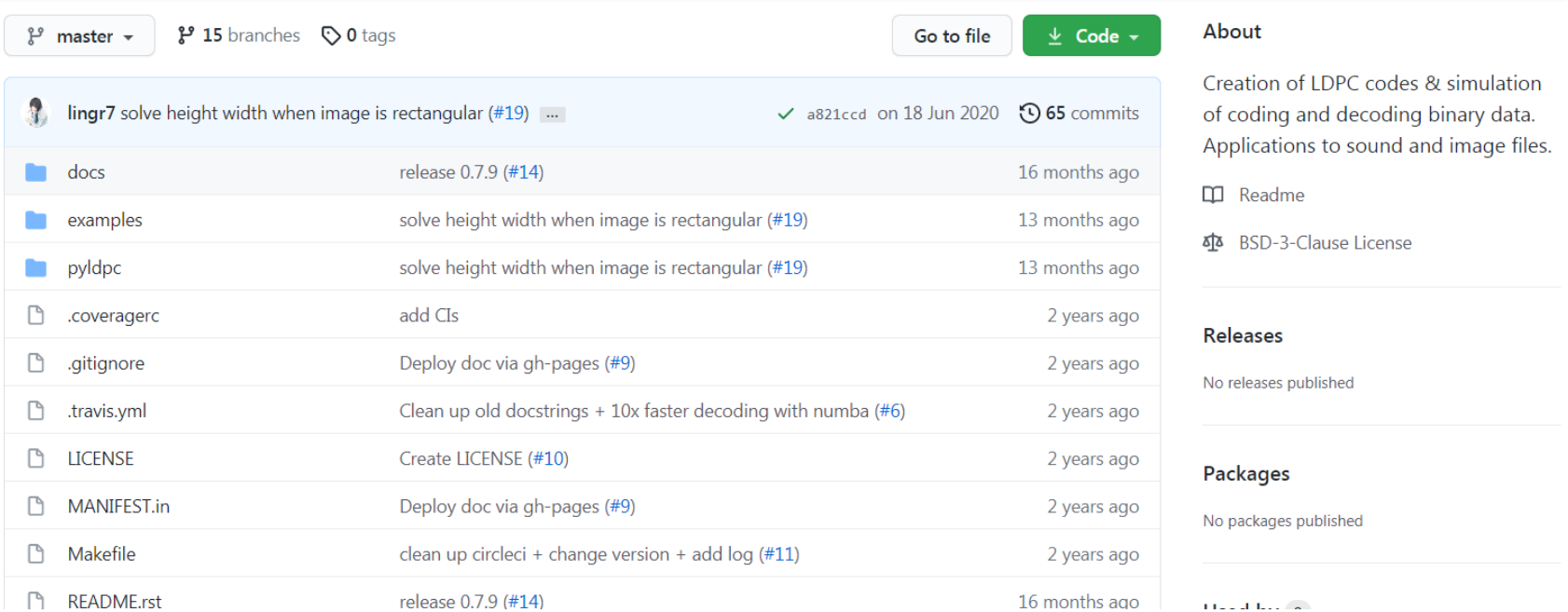
This collection of programs and modules, written in C, is intended to support research and education concerning Low Density Parity Check (LDPC) codes. (Note, however, that the copyright notice no longer restricts use to these purposes). These error-correcting codes were invented by Robert Gallager in the early 1960's, and re-invented and shown to have very good performance by David MacKay and myself in the mid-1990's. The decoding algorithm for LDPC codes is related to that used for Turbo codes, and to probabilistic inference methods used in other fields. Variations on LDPC and Turbo codes are currently the best practical codes known, in terms of their ability to transmit data at rates approaching channel capacity with very low error probability.

Sample Code for LDPC

* https://pypi.org/project/pyldpc/
  * https://github.com/hichamjanati/pyldpc

Sample Code for LDPC

* https://pypi.org/project/pyldpc/
  * https://github.com/hichamjanati/pyldpc



master | 15 branches | 0 tags | Go to file | Code

lingr7 solve height width when image is rectangular (#19) ... a821ccd on 18 Jun 2020 65 commits

docs	release 0.7.9 (#14)	16 months ago
examples	solve height width when image is rectangular (#19)	13 months ago
pyldpc	solve height width when image is rectangular (#19)	13 months ago
.coveragerc	add CIs	2 years ago
.gitignore	Deploy doc via gh-pages (#9)	2 years ago
.travis.yml	Clean up old docstrings + 10x faster decoding with numba (#6)	2 years ago
LICENSE	Create LICENSE (#10)	2 years ago
MANIFEST.in	Deploy doc via gh-pages (#9)	2 years ago
Makefile	clean up circleci + change version + add log (#11)	2 years ago
README.rst	release 0.7.9 (#14)	16 months ago

About

Creation of LDPC codes & simulation of coding and decoding binary data. Applications to sound and image files.

Readme

BSD-3-Clause License

Releases

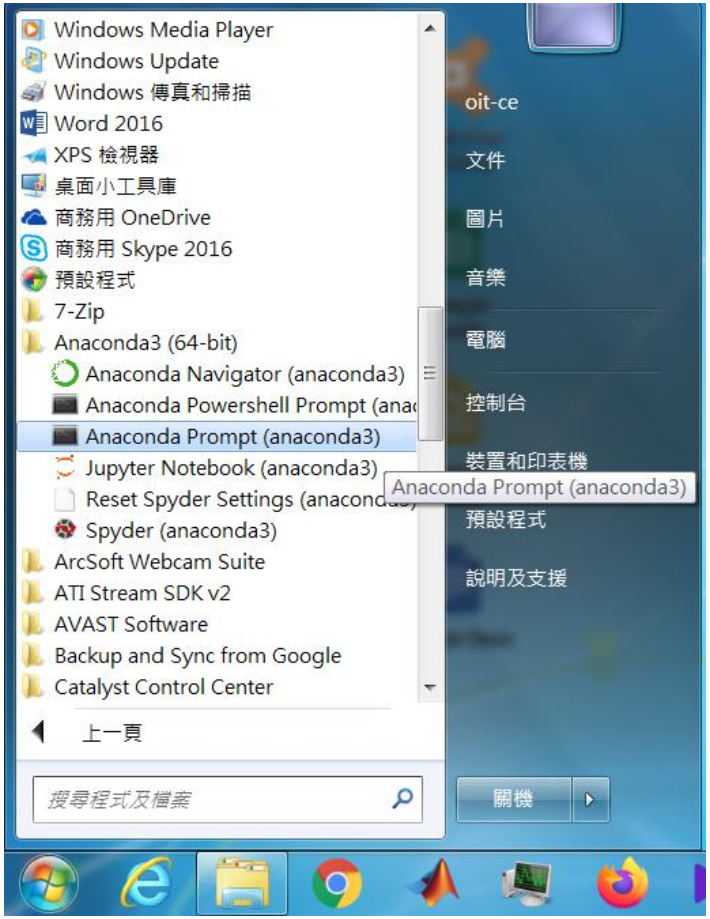
No releases published

Packages

No packages published

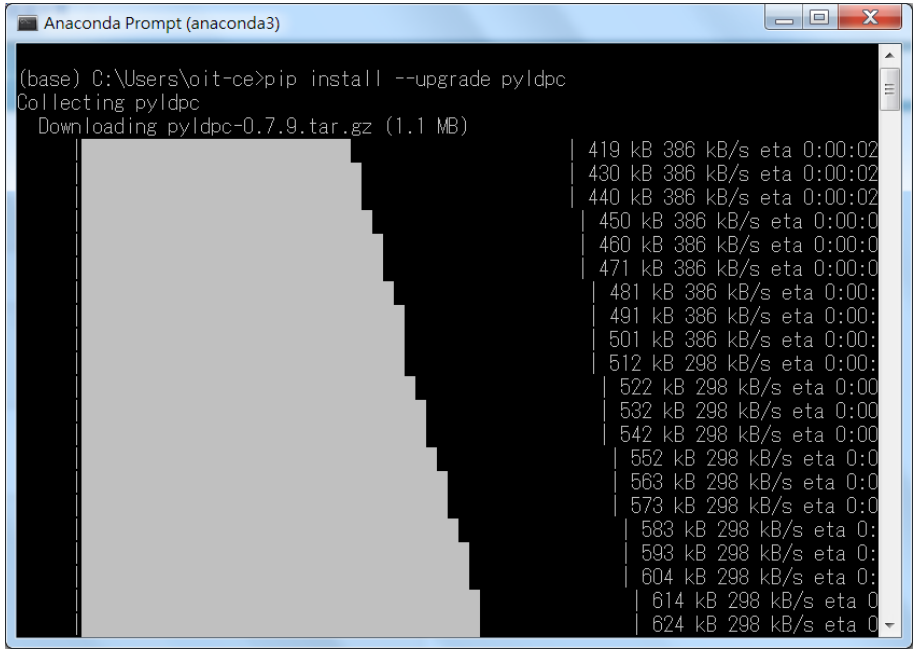
Sample Code for LDPC

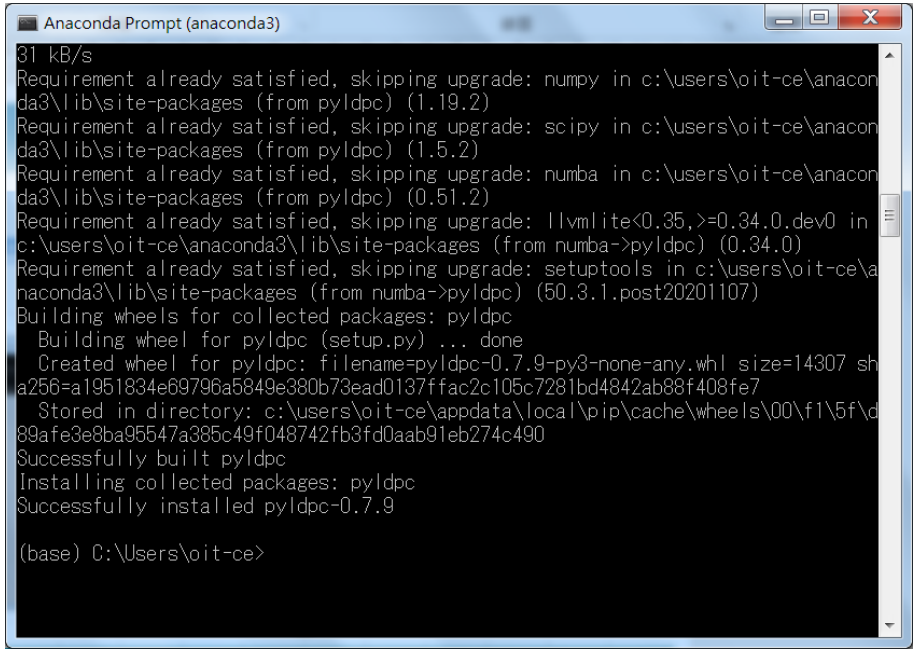
* 進入 Anaconda Prompt(anaconda3)



Sample Code for LDPC

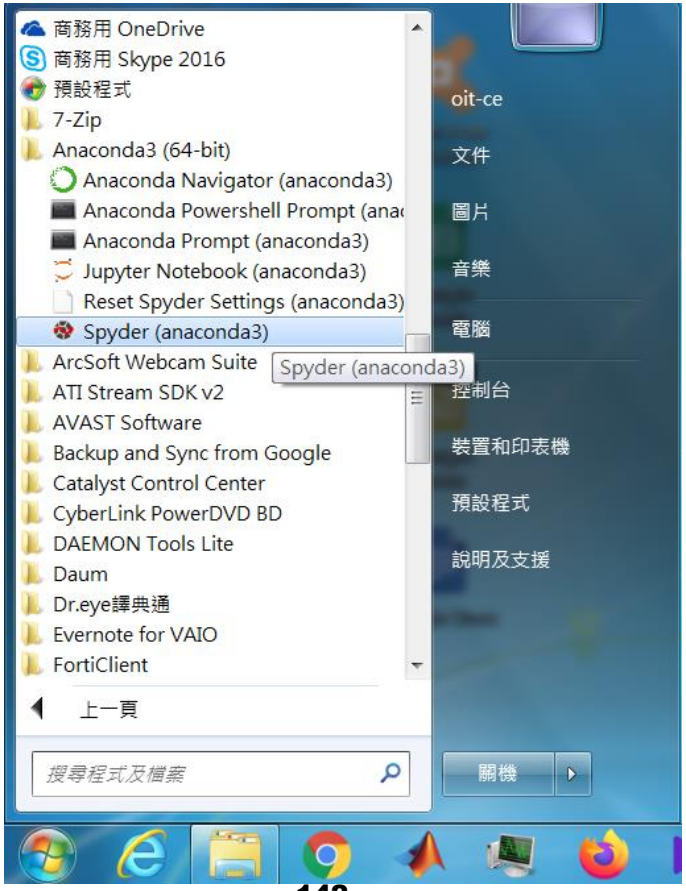
* pip install --upgrade pyldpc





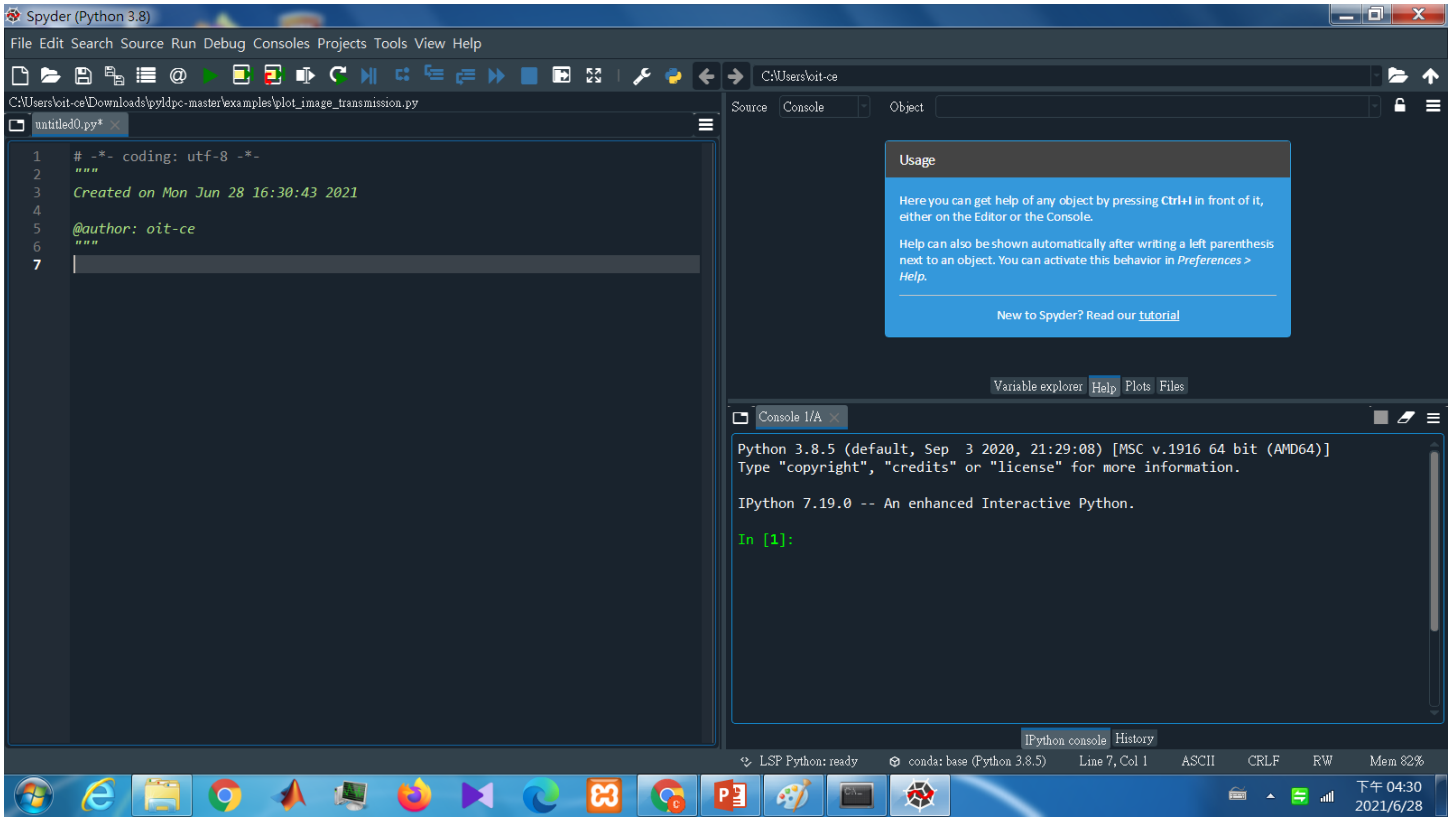
Sample Code for LDPC

* 進入Anaconda 的 Spyder 整合環境



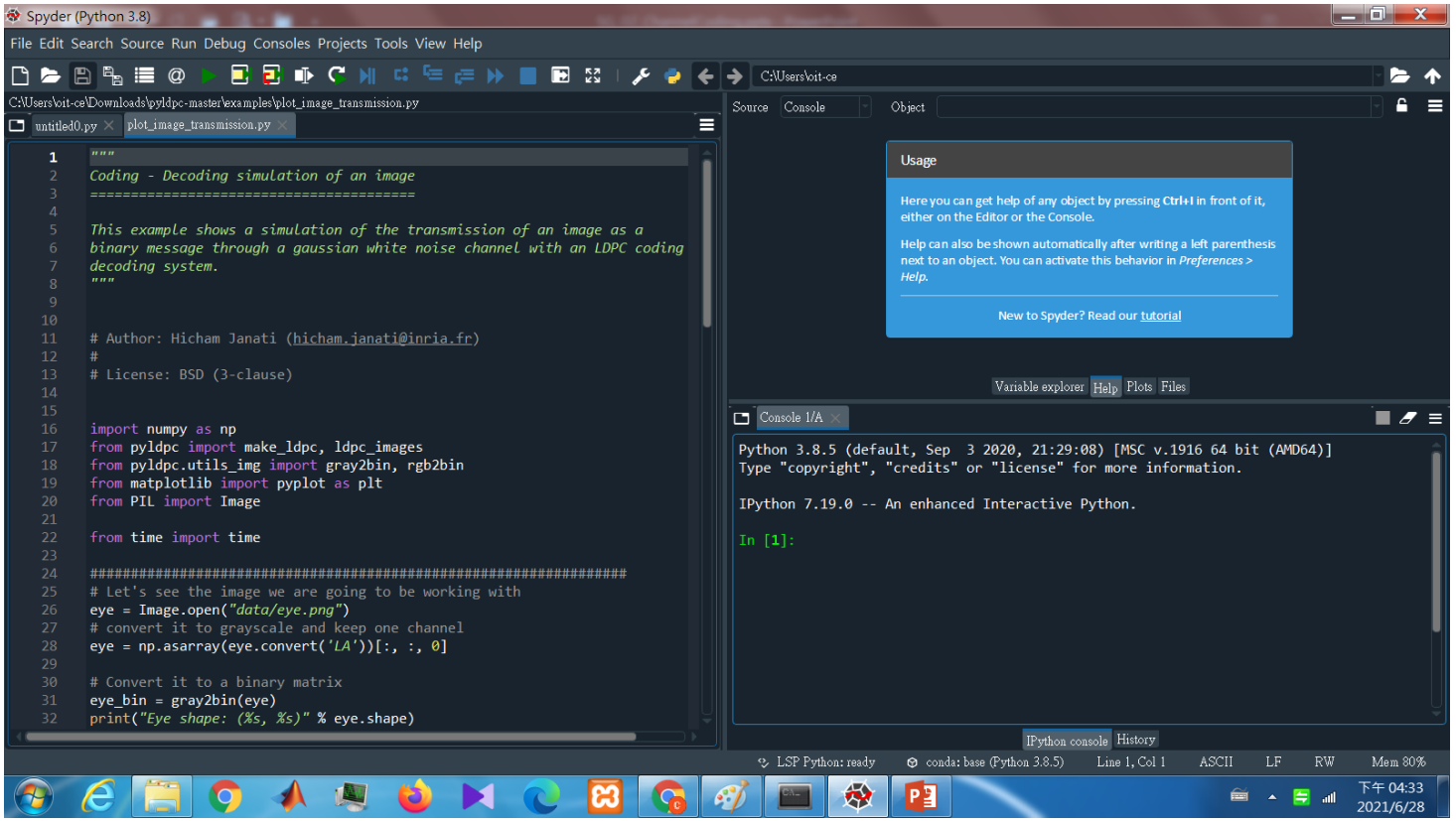
Sample Code for LDPC

* 進入Anaconda 的 Spyder 整合環境



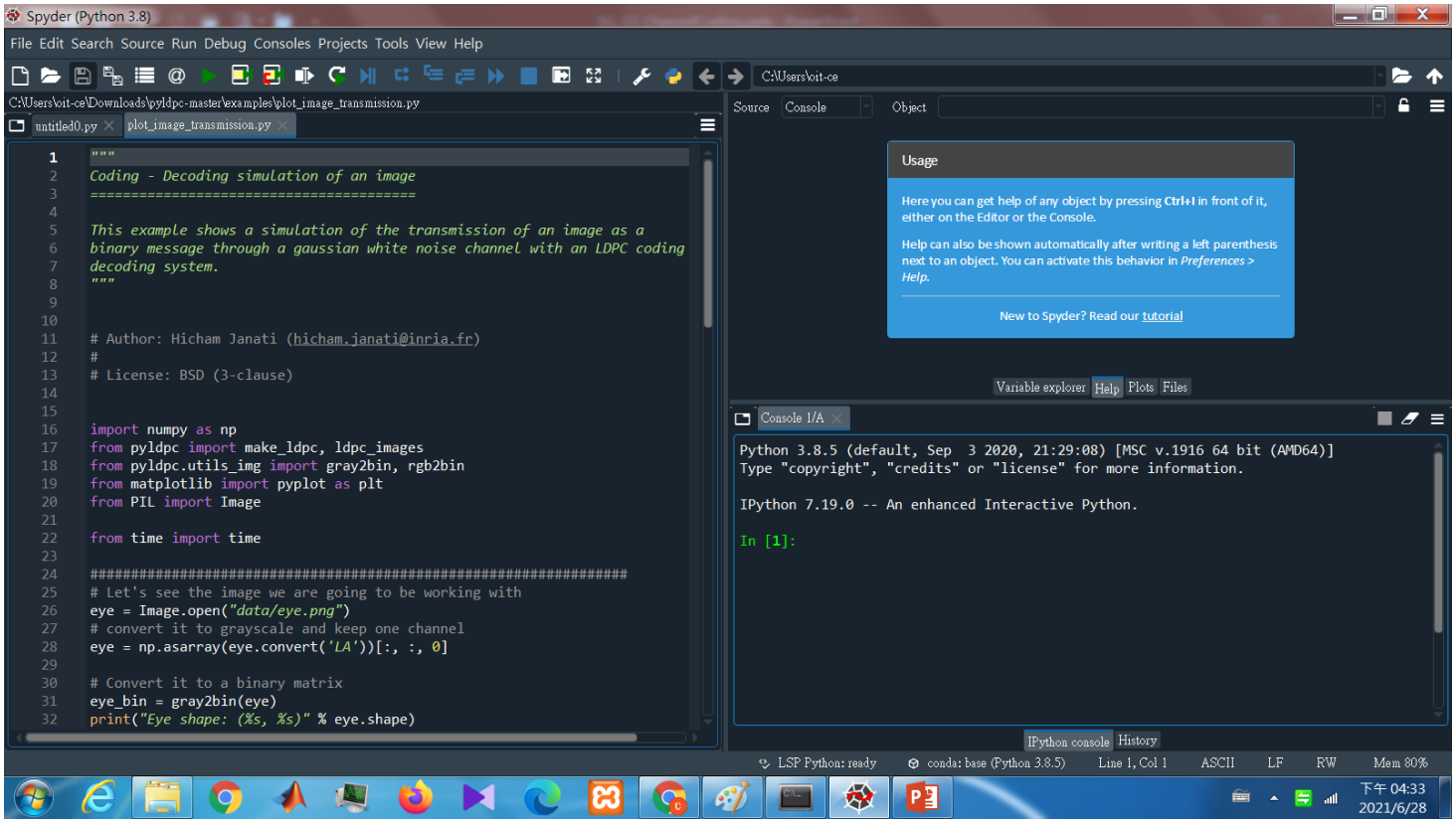
Sample Code for LDPC

* 開啟 C:\pyldpc-master\examples\plot_image_transmission.py



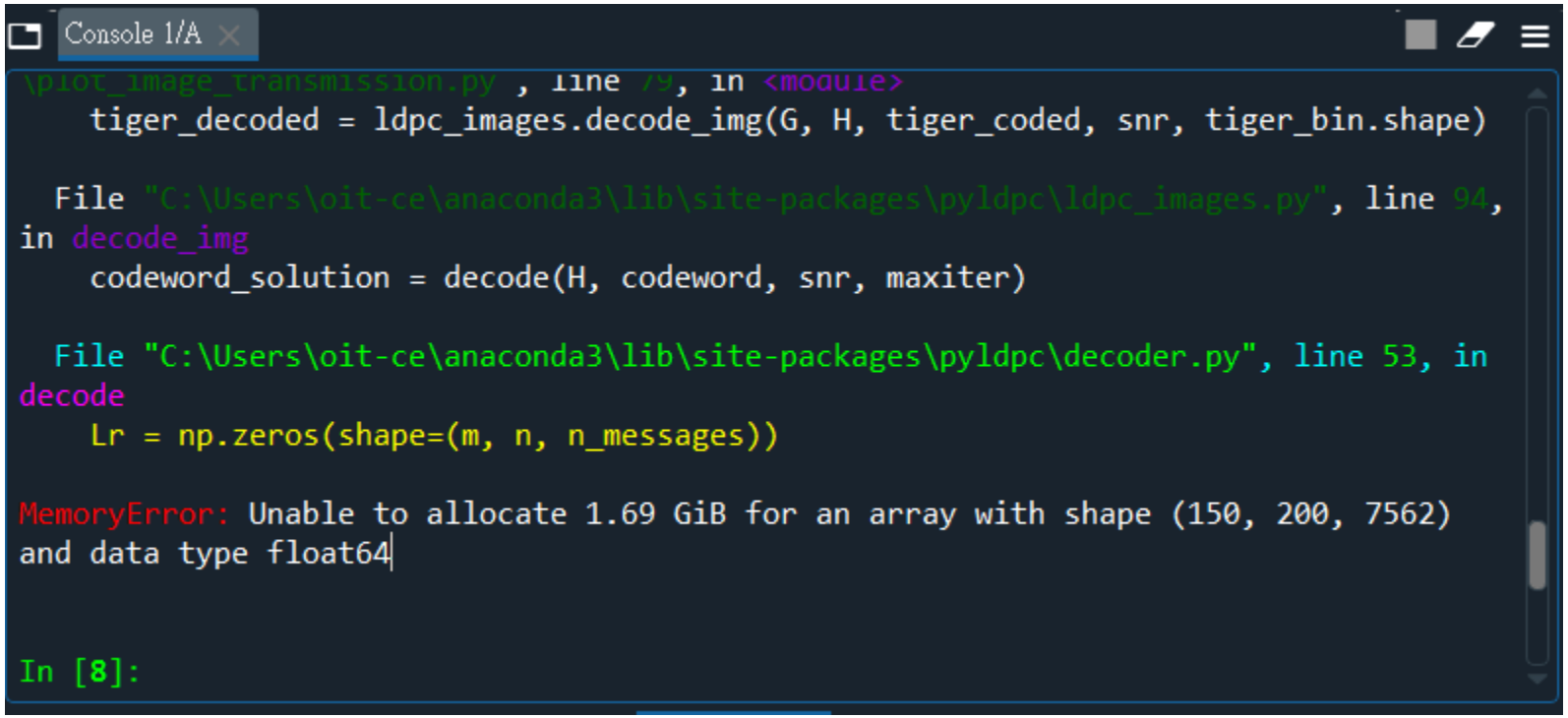
Sample Code for LDPC

* 執行



Sample Code for LDPC

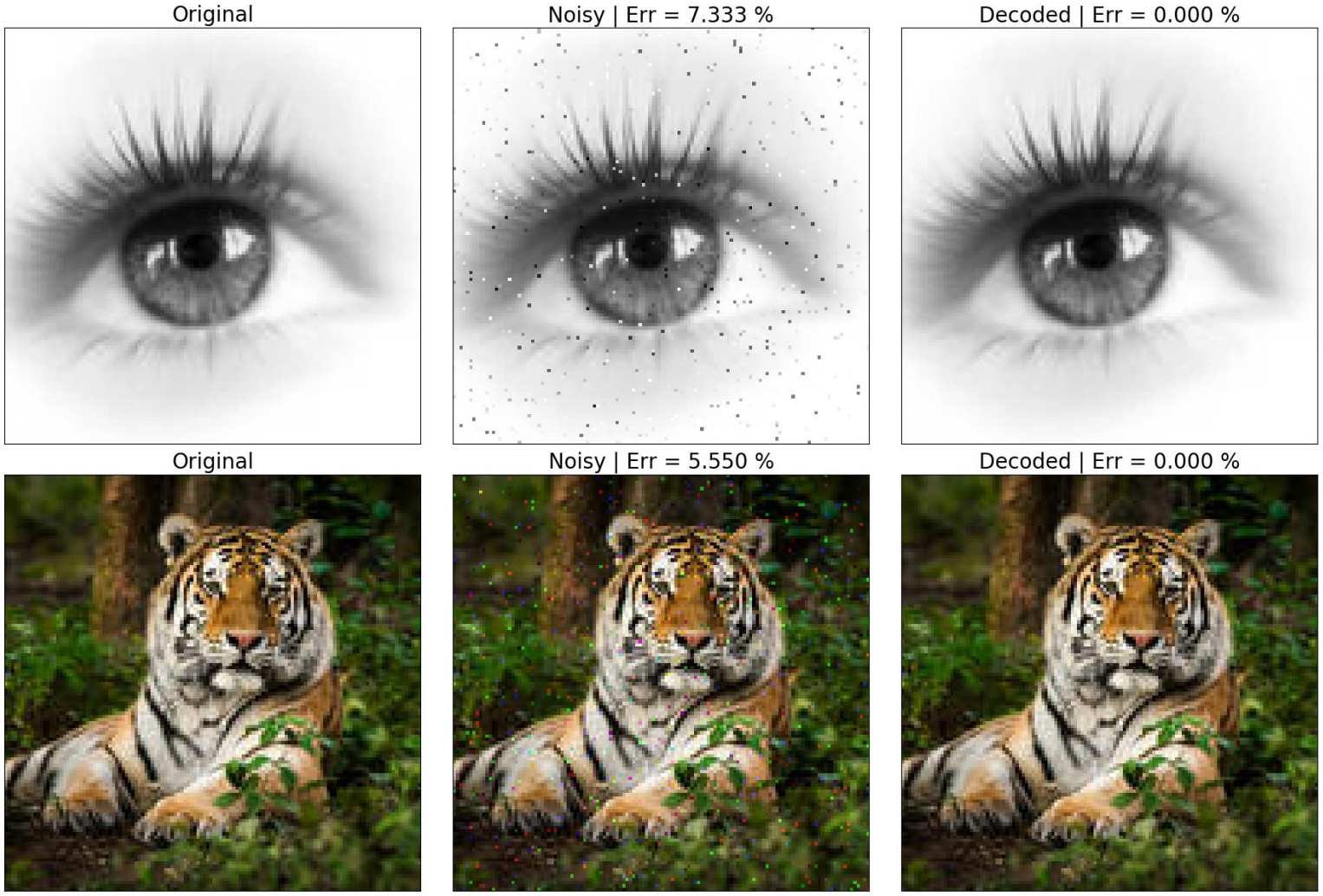
* MemoryError: Unable to allocate 1.69 GiB for an array with shape (150, 200, 7562) and data type float64



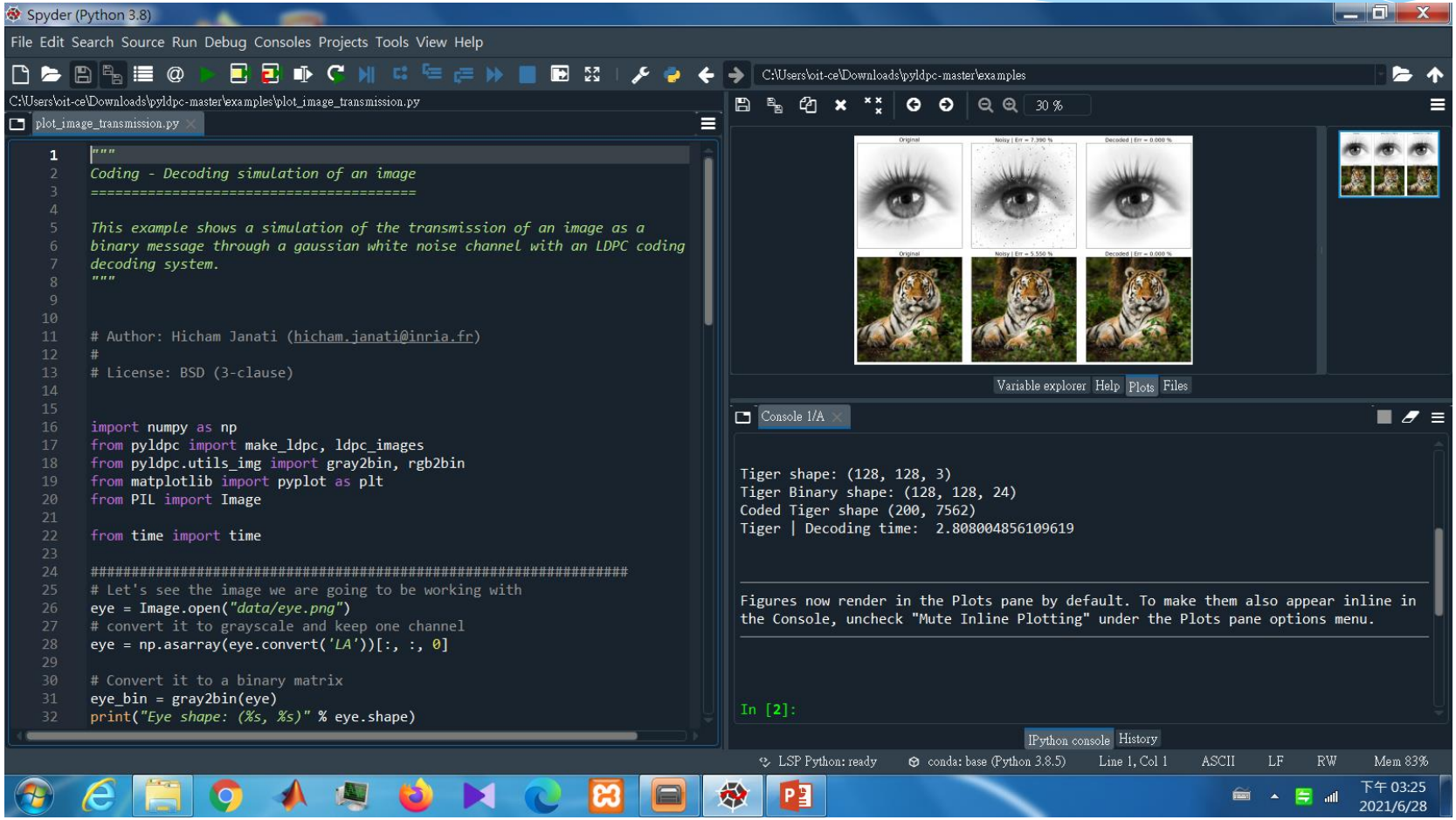
Sample Code for LDPC

* Unable to allocate array with shape and data type
  - Windows 10
    - Press the Windows key
    - Type SystemPropertiesAdvanced
    - Click Run as administrator
    - Under Performance, click Settings
    - Select the Advanced tab
    - Select Change...
    - Uncheck Automatically managing paging file size for all drives
    - Then select Custom size and fill in the appropriate size
    - Press Set then press OK then exit from the Virtual Memory, Performance Options, and System Properties Dialog
    - Reboot your system

Sample Code for LDPC



Sample Code for LDPC



Polar Code

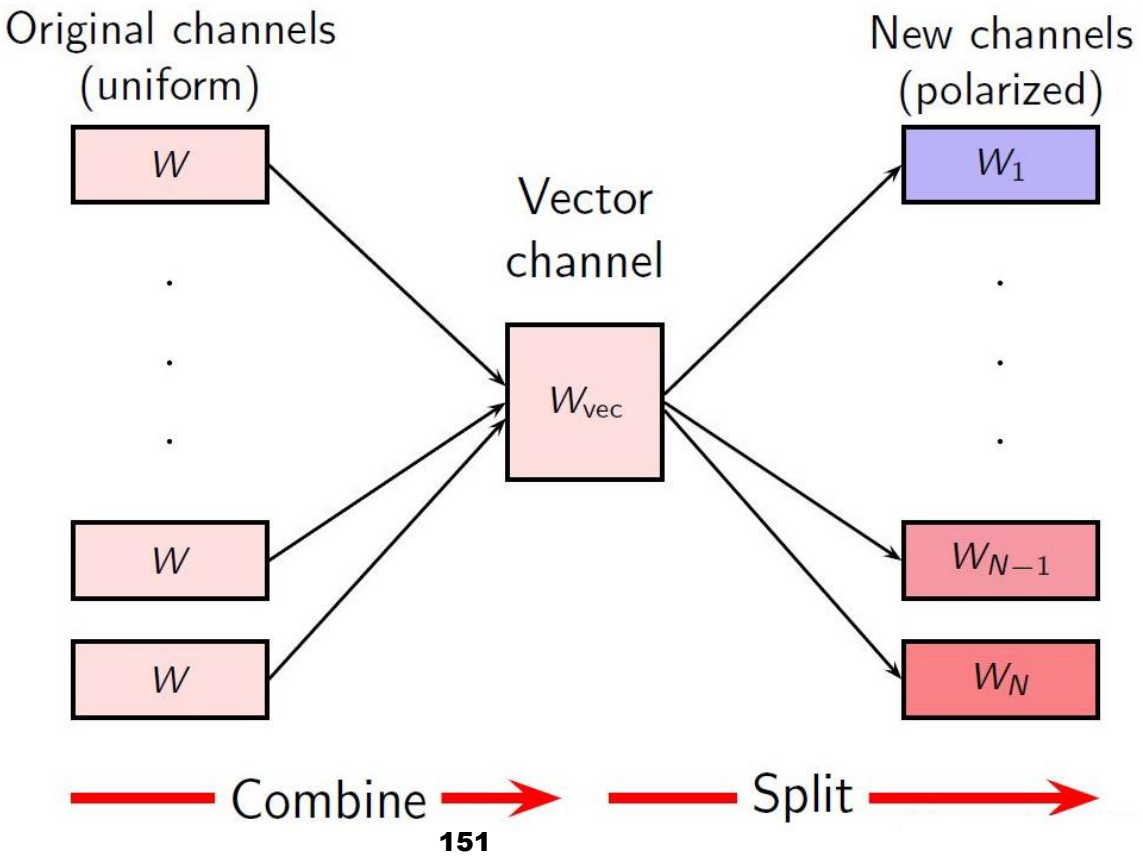
* **極化碼**（Polar code）是一種前向錯誤更正編碼方式。
* **通道極化**（Channel Polarization）的概念 invented by Erdal Arikan 2008.
* 2016/11/18，在美國內華達州里諾召開的 3GPP RAN1 #87次會議，確定 Polar Code 作為 5G eMBB（增強移動寬頻）下控制通道編碼方案。
* 極化碼構造的核心是透過通道極化（channel polarization）處理
  - 在編碼側採用方法使各個子通道呈現出不同的可靠性。
  - 當碼長持續增加時，部分通道將趨向於容量近於1的完美通道（無誤碼），另一部分通道趨向於容量接近於0的純噪聲通道。
  - 選擇在容量接近於1的通道上直接傳輸訊息以逼近通道容量，能夠被證明可以達到香農極限。

https://www.zhihu.com/question/31656512

Polar Code

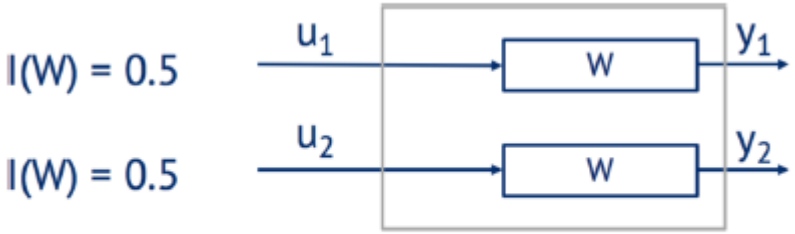
* Polar Codes – Channel polarization

把通道複製 N 份後組合到一起，經過了規律的線性變換，最後產生了分裂split，出現了極化效果polarization：這複製的 N 份通道一部分容量趨於 1，另一部分趨於 0，所以叫做Polar碼。

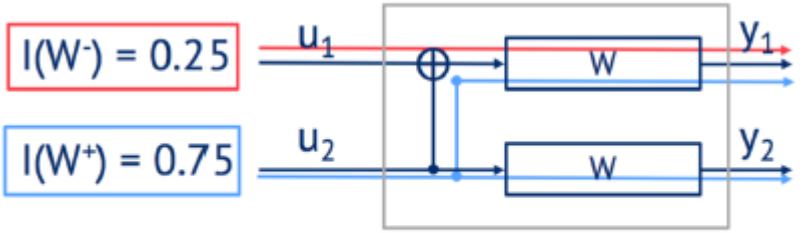


Polar Code

* Polar Codes – Channel polarization
  - An example for a set of two channels before and after the polarization.
    - The channel with decreased capacity is called W^- (red)
    - The other with the improved capacity is called W^+(blue).



(a) Two independent copies of channel W



(b) Two new channels (W^- in red and W^+ in blue) after polarization

the smallest possible Polar Code with **N** = 2

Polar Code

* Polar transform (極座標轉換): G_2 ， 2 bits to 2 bits

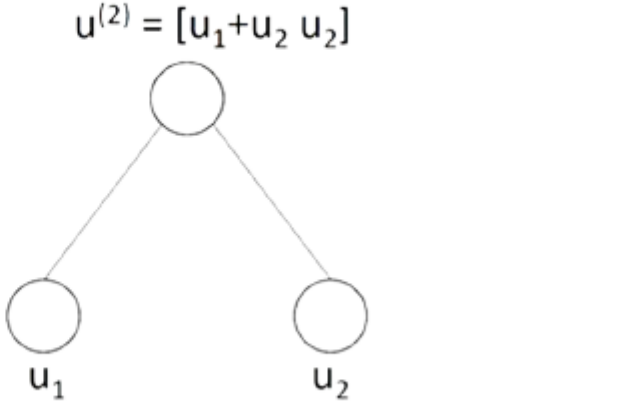
$$G_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

u_1 , u_2 is the input to the transform

• $[u_1\ u_2]\ G_2 = [u_1+u_2\ u_2]$

input	output
00	00
01	11
10	10
11	01

Binary tree representation

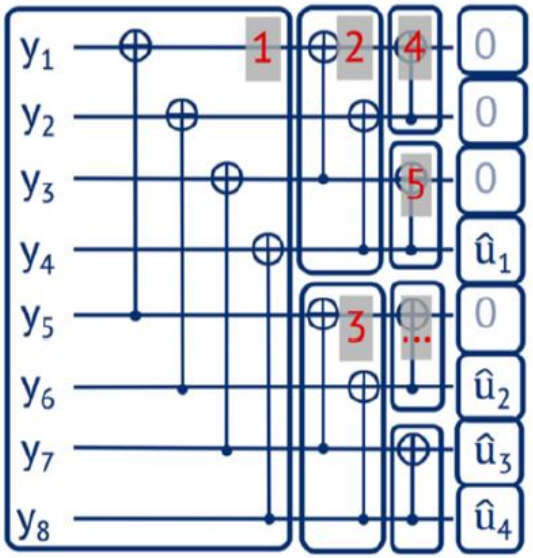


u(2): length-2 vector

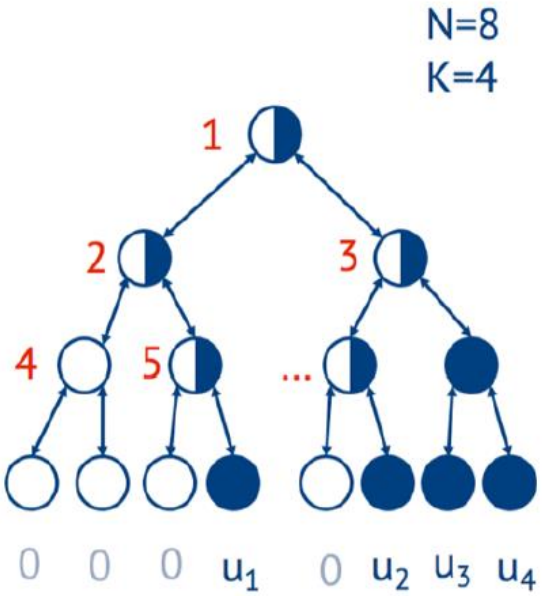
https://www.youtube.com/watch?v=rBorhQKyV34

Polar Code

* Polar Code Representations



(a) Polar Factor Graph

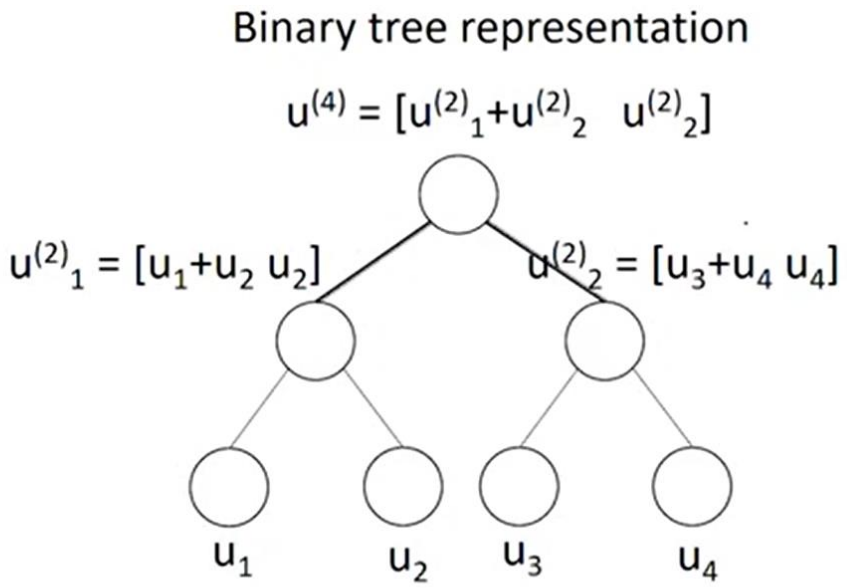


(b) Polar Factor Tree

Polar Code

* Polar transform: G_4 , 4 bits to 4 bits

$$G_4 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$



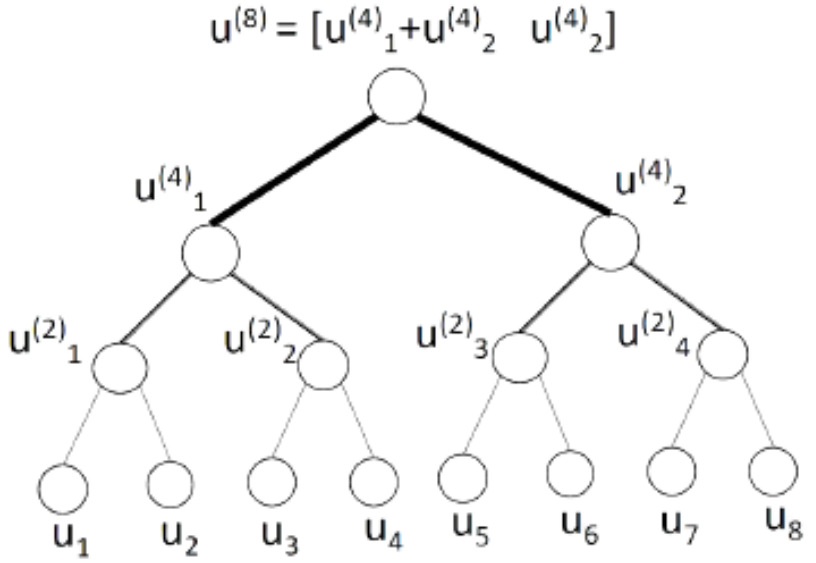
• $[u_1\ u_2\ u_3\ u_4]\ G_4 = [u_1+u_2+u_3+u_4\ u_2+u_4\ u_3+u_4\ u_4]$

https://www.youtube.com/watch?v=rBorhQKyV34

Polar Code

* Polar transform: G_8 , 8 bits to 8 bits

$$G_8 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes 3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



- $[u_1\ u_2\ u_3\ u_4\ u_5\ u_6\ u_7\ u_8]\ G_8 = [u_1+u_2+u_3+u_4+u_5+u_6+u_7+u_8 \quad u_2+u_4+u_6+u_8 \quad u_3+u_4+u_7+u_8 \quad u_4+u_8]$
  $u_5+u_6+u_7+u_8 \qquad u_6+u_8 \qquad u_7+u_8 \qquad u_8$

https://www.youtube.com/watch?v=rBorhQKyV34

Polar Code

* Polar transform: General

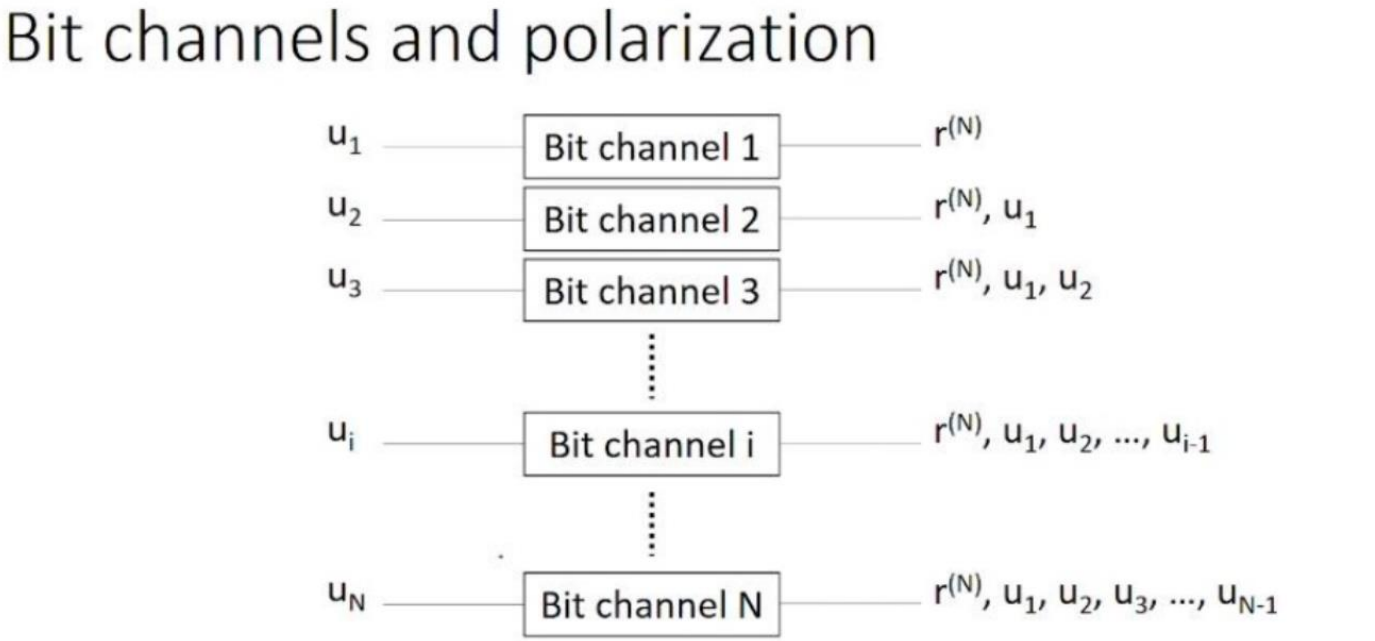
$$G_{2^n} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes n}$$

- $N = 2^n$
- G_N : N x N matrix, Kronecker product of 2 x 2 kernel
- Binary tree representation
  - Depth n
  - $u^{(N)} = u\, G_N$: evaluated on tree with u at bottom and $u^{(N)}$ at top
- 5G: uses up to $n = 10$

https://www.youtube.com/watch?v=rBorhQKyV34

Polar Code

* Channel Polarization



Polar Code

* Reliability sequence

the transmitted bit is either received correctly or lost with probability δ.

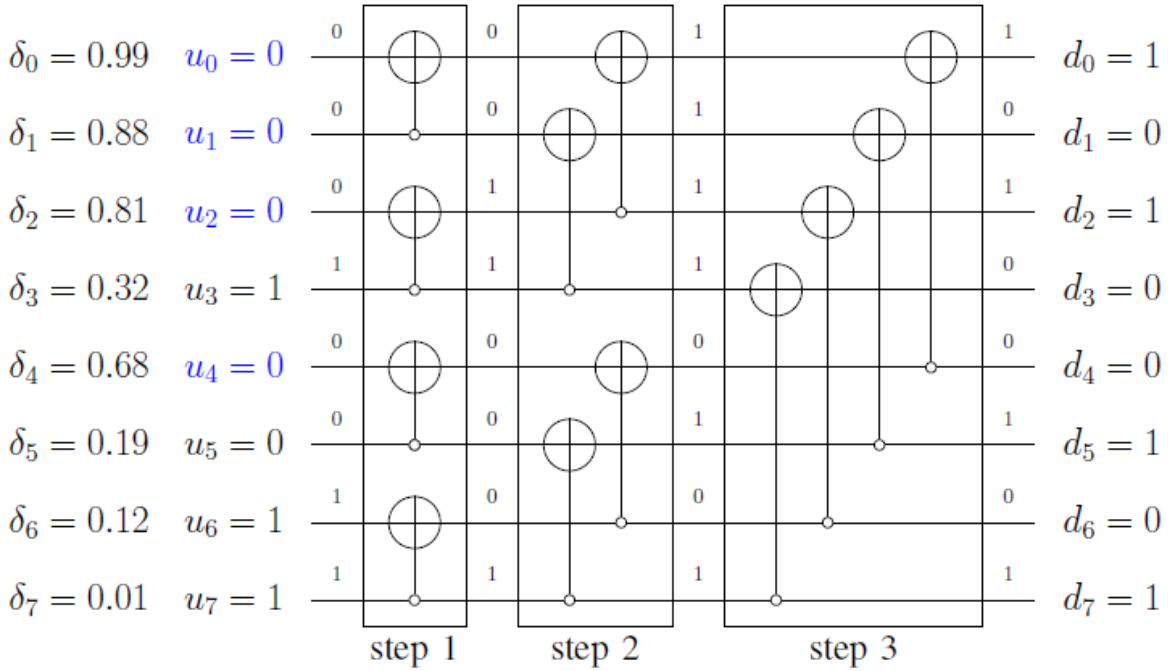


Fig. : Encoder of an $(8, 4)$ polar code.

Polar Code

* Reliability sequence

Reliability sequence: worst to best (5G)

- N = 8
  - 1 2 3 5 4 6 7 8
- N = 16
  - 1 2 3 5 9 4 6 10 7 11 13 8 12 14 15 16
- N = 32
  - 1 2 3 5 9 17 4 6 10 7 18 11 19 13 21 25 8 12 20 14 15 22 27 26 23 29 16 24 28 30 31 32
- Specified for N = 1024 in the 5G standard
  - Sequence for smaller N derived from the sequence for N = 1024

Polar Code

* Reliability sequence

Reliability sequence: N = 1024

1 2 3 5 9 17 33 4 6 65 10 7 18 11 19 129 13 34 66 21 257 35 25 37 8 130 67 513 12 41 69 131 20 14 49 15 73 258 22 133 36 259 27 514 81 38 26 23
137 261 265 39 515 97 68 42 145 29 70 43 517 50 75 273 161 521 289 529 193 545 71 45 132 82 51 74 16 321 134 53 24 135 385 77 138 83 57 28 98
40 260 85 139 146 262 30 44 99 516 89 141 31 147 72 263 266 162 577 46 101 641 52 149 47 76 267 274 518 105 163 54 194 153 78 165 769 269
275 519 55 84 58 522 113 136 79 290 195 86 277 523 59 169 140 100 87 61 281 90 291 530 525 197 142 102 148 177 143 531 322 32 201 91 546 293
323 533 264 150 103 106 305 297 164 93 48 268 386 547 325 209 387 151 154 166 107 56 329 537 578 549 114 155 80 270 109 579 225 167 520 553
196 271 642 524 276 581 292 60 170 561 115 278 157 88 198 117 171 62 532 526 643 282 279 527 178 294 389 92 585 770 199 173 121 202 337 63
283 144 104 179 295 94 645 203 593 324 393 298 771 108 181 152 210 285 649 95 205 299 401 609 353 326 534 156 211 306 548 301 110 185 535
538 116 168 226 327 307 773 158 657 330 111 118 213 172 777 331 227 550 539 388 309 217 417 272 280 159 338 551 673 119 333 580 541 390
174 122 554 200 785 180 229 339 313 705 391 175 555 582 394 284 123 449 354 562 204 64 341 395 528 583 557 182 296 286 233 125 206 183 644
563 287 586 300 355 212 402 186 397 345 587 646 594 536 241 207 96 328 565 801 403 357 308 302 418 214 569 833 589 187 647 405 228 897 595
419 303 650 772 361 540 112 332 215 310 189 450 218 409 610 597 552 651 230 160 421 311 542 774 611 658 334 120 601 340 219 369 653 231
392 314 451 543 335 234 556 775 176 124 659 613 342 778 221 315 425 396 674 584 356 288 184 235 126 558 661 617 343 317 242 779 564 346
453 398 404 208 675 559 786 433 358 188 237 665 625 588 781 706 127 243 566 399 347 457 359 406 304 570 245 596 190 567 677 362 707 590
216 787 648 349 420 407 465 681 802 363 591 410 571 789 598 573 220 312 709 599 602 652 422 793 803 612 603 411 232 689 654 249 370 191
365 655 660 336 481 316 222 371 614 423 426 452 615 544 236 413 344 373 776 318 223 427 454 238 560 834 805 713 835 662 809 780 618 605
434 721 817 837 348 898 244 663 455 319 676 619 899 782 377 429 666 737 568 841 626 239 360 458 400 788 592 679 435 678 350 246 459 667
621 364 128 192 783 408 437 627 572 466 682 247 708 351 600 669 791 461 250 683 574 412 804 790 710 366 441 629 690 375 424 467 794 251
372 482 575 414 604 367 469 656 901 806 616 685 711 430 795 253 374 606 849 691 714 633 483 807 428 905 415 224 664 693 836 620 473 456
797 810 715 722 838 717 865 811 607 913 723 697 378 436 818 320 622 813 485 431 839 668 489 240 379 460 623 628 438 381 819 462 497 670
680 725 842 630 352 468 439 738 252 463 443 442 470 248 684 843 739 900 671 784 850 821 729 929 792 368 902 631 686 845 634 712 254 692
825 903 687 741 851 376 445 471 484 416 486 906 796 474 635 745 853 961 866 694 798 907 716 808 475 637 695 255 718 576 914 799 812 380
698 432 608 490 867 724 487 909 719 814 477 857 840 726 699 915 753 869 820 815 440 930 491 624 672 740 917 464 844 382 498 931 822 727
962 873 493 632 730 701 444 742 846 921 383 823 852 731 499 881 743 446 472 636 933 688 904 826 501 847 746 827 733 447 963 937 476 854
868 638 908 488 696 747 829 754 855 858 505 800 256 965 910 720 478 916 639 749 945 870 492 700 755 859 479 969 384 911 816 977 871 918
728 494 874 702 932 757 861 500 732 824 923 875 919 503 934 744 761 882 495 703 922 502 877 848 993 448 734 828 935 883 938 964 748 506
856 925 735 830 966 939 885 507 750 946 967 756 860 941 831 912 872 640 889 480 947 751 970 509 862 758 971 920 876 863 759 949 978 924
973 762 878 953 496 704 936 979 884 763 504 926 879 736 994 886 940 995 981 927 765 942 968 887 832 948 508 890 985 752 943 997 972 891
510 950 974 1001 893 951 864 760 1009 511 980 954 764 975 955 880 982 983 928 996 766 957 888 986 998 987 944 892 999 767 512 989 1002 952
1003 894 976 895 1010 956 1005 1011 958 984 959 988 1013 1000 1017 768 990 1004 991 1006 960 1012 1014 896 1007 1015 1018 1019 992 1021
1008 1016 1020 1022 1023 1024

Polar Code

* Polar Code (Encoding)

(N,K) Polar Code

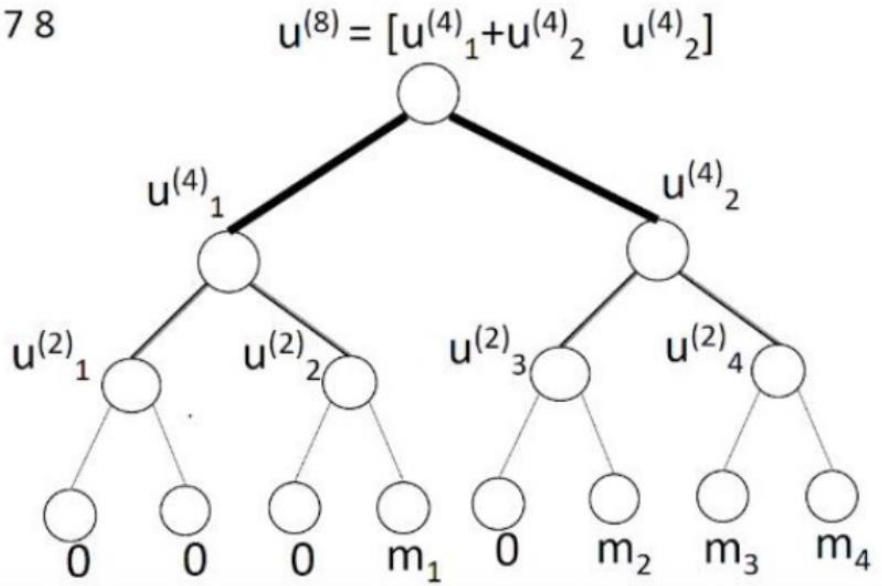
- $N = 2^n$
- Message: m of length K bits
- Form a vector u of length N bits as follows
  - Find N – K least reliable (worst) channels from reliability sequence
  - Set u_i for those N – K channels to zero (called frozen positions)
  - m: remaining K bits of u (called message positions)
- Codeword: u G_N

Polar Code

* Polar Code (Encoding)

Polar code example: (8,4)

- Reliability sequence: 1 2 3 5 4 6 7 8
  - Frozen: 1 2 3 5
  - Message: 4 6 7 8



Polar Code

* Polar Code (Encoding Example)

- 設碼長為 $N = 8$，information bits 數 $K = 4$。列出所有使用到的公式：

1. u_1^N is information bits

2. a generator matrix G_N that is built by using the Kronecker power

3. the N × N bit reversal permutation matrix B_N

$$c_1^N = u_1^N G_N$$

$$G_N = B_N F^{\otimes n}$$

$$F^{\otimes n} = F \otimes F^{\otimes (n-1)}$$

$$F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$B_N = R_N \left(I_2 \otimes B_{N/2} \right)$$

$$B_2 = I_2$$

Example (Bit Reversal)

- 000 → 000
- 001 → 100
- 010 → 010
- 011 → 110
- 100 → 001
- 101 → 101
- 110 → 011
- 111 → 111

$$\begin{bmatrix} x_0 \\ x_4 \\ x_2 \\ x_6 \\ x_1 \\ x_5 \\ x_3 \\ x_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}$$

https://marshallcomm.cn/2017/03/05/polar-code-3-encoding-example/

Polar Code - Encoder

* Polar Code (Encoding Example)
  - 極化通道的可靠性估計

假設對於各 Binary Erasure Channel, 已計算出各個分裂通道的巴氏參數（Bhattacharyya Parameter）$Z\left(W_N^{(i)}\right)$ 。

作為每個分裂通道(Channel Splitting)的可靠性度量，越大表示通道的可靠程度越低。

  - Bits 混合

假設通過第一步得到巴氏參數最小的4個子通道序號為“4, 6, 7, 8”，則information bits集合記為

$$A = \{4, 6, 7, 8\}$$

則frozen bits 集合為

$$A^c = \{1, 2, 3, 5\}$$

https://marshallcomm.cn/2017/03/05/polar-code-3-encoding-example/

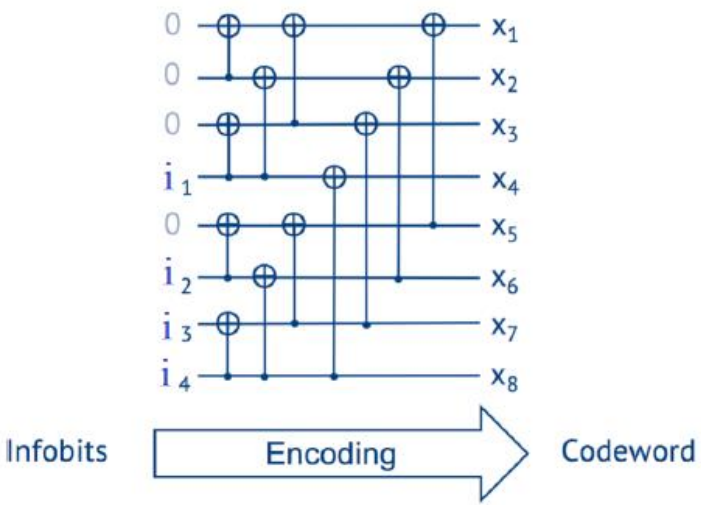
Polar Code - Encoder

* Polar Code (Encoding Example)

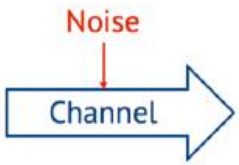
設information bits 集合為 $(i_1, i_2, i_3, i_4) = (1, 1, 1, 1)$ ，

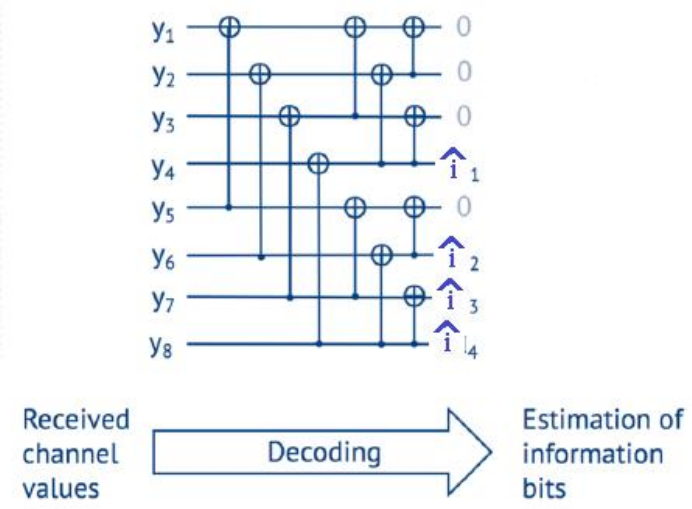
frozen bits集合為 (0, 0, 0, 0)，則最終得到

$$u_1^8 = [0, 0, 0, i_1, 0, i_2, i_3, i_4] = [0, 0, 0, 1, 0, 1, 1, 1]$$



(a) Encoding of Polar Codes





(b) Decoding of Polar Codes

Polar Code- Encoder

* Polar Code (Encoding Example)
  * 構造生成矩陣
    求排序矩陣BN

遞迴式：

$$B_8 = R_8 (I_2 \otimes B_4)$$

$$B_4 = R_4 (I_2 \otimes B_2)$$

計算：

$$B_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$I_2 \otimes B_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

https://marshallcomm.cn/2017/03/05/polar-code-3-encoding-example/

Polar Code- Encoder

* Polar Code (Encoding Example)

R_4由I_4變換得來，先排I_4的奇數列，再排I_4的偶數列：

矩陣R_4為置換矩陣，
$$R_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Leftarrow I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B_4 = R_4 (I_2 \otimes B_2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$I_2 \otimes B_4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

https://marshallcomm.cn/2017/03/05/polar-code-3-encoding-example/

Polar Code- Encoder

* Polar Code (Encoding Example)

  R_8由I_8變換得來，先排I_8的奇數列，再排I_8的偶數列：

$$R_8 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix} \Leftarrow I_8 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix}$$

$$B_8 = R_8\,(I_2 \otimes B_4) = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix} \cdot \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix}$$

https://marshallcomm.cn/2017/03/05/polar-code-3-encoding-example/

Polar Code- Encoder

* Polar Code (Encoding Example)
  - 求F的n次克羅內克積
  - 遞迴式：

$$F^{\otimes 3} = F \otimes F^{\otimes 2}$$

$$F^{\otimes 2} = F \otimes F^{\otimes 1}$$

  - 計算：

$$F^{\otimes 1} = F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$F^{\otimes 2} = F \otimes F^{\otimes 1} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes F^{\otimes 1} = \begin{bmatrix} F^{\otimes 1} & 0 \\ F^{\otimes 1} & F^{\otimes 1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$F^{\otimes 3} = F \otimes F^{\otimes 2} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes F^{\otimes 2} = \begin{bmatrix} F^{\otimes 2} & 0 \\ F^{\otimes 2} & F^{\otimes 2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

https://marshallcomm.cn/2017/03/05/polar-code-3-encoding-example/

Polar Code- Encoder

* Polar Code (Encoding Example)
  * 求生成矩陣G_N

$$G_8 = B_8 F^{\otimes 3} = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix} \cdot \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0 \\ 1&1&1&1&0&0&0&0 \\ 1&0&0&0&1&0&0&0 \\ 1&1&0&0&1&1&0&0 \\ 1&0&1&0&1&0&1&0 \\ 1&1&1&1&1&1&1&1 \end{bmatrix}$$

$$= \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&0&0&0&1&0&0&0 \\ 1&0&1&0&0&0&0&0 \\ 1&0&1&0&1&0&1&0 \\ 1&1&0&0&0&0&0&0 \\ 1&1&0&0&1&1&0&0 \\ 1&1&1&1&0&0&0&0 \\ 1&1&1&1&1&1&1&1 \end{bmatrix}$$

Polar Code- Encoder

* Polar Code (Encoding Example)
  * 生成Polar Code

$$c_1^8 = u_1^8 G_8 = [0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

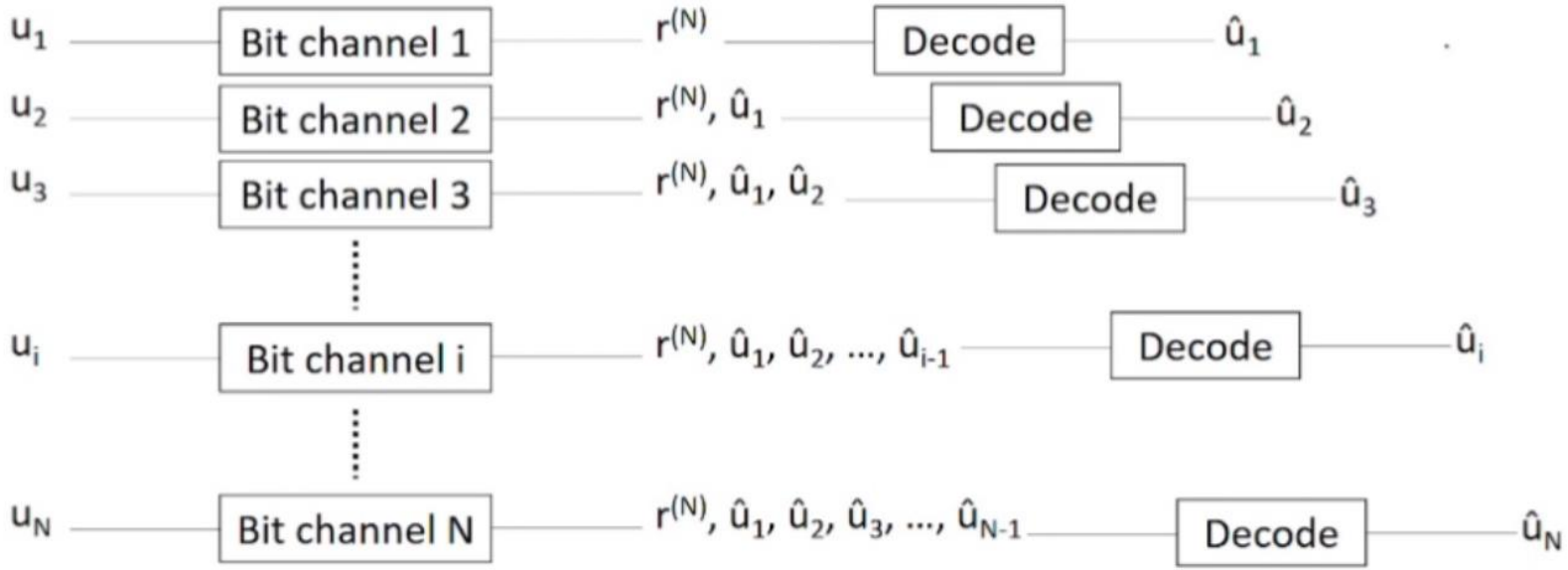
$$= [0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1]$$

$$u_1^8 = [0, 0, 0, i_1, 0, i_2, i_3, i_4] = [0, 0, 0, 1, 0, 1, 1, 1]$$

Polar Code

* Polar Code (Decoding)

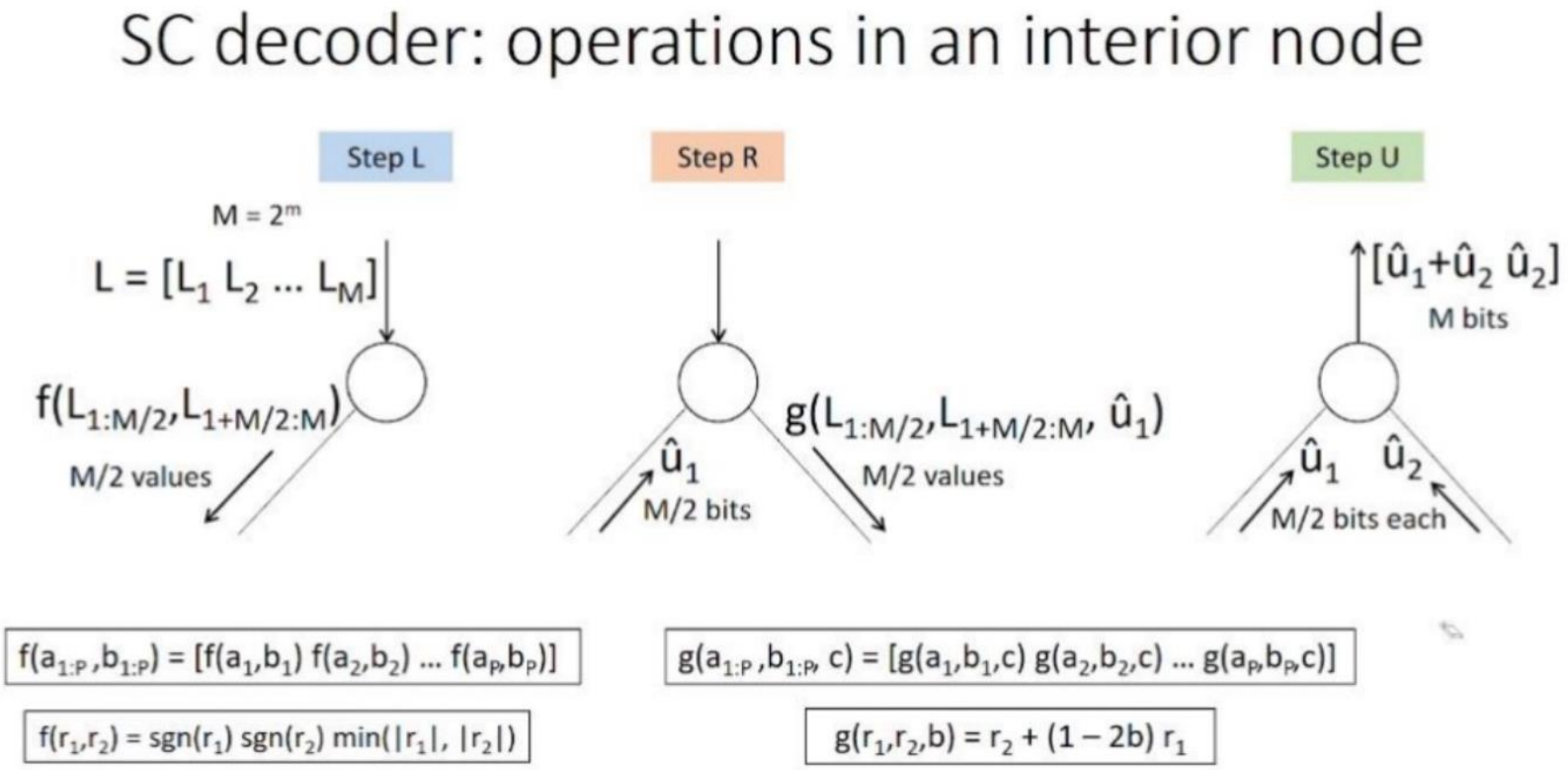
Successive Cancellation (SC) Decoding



- If bit i is frozen, $\hat{u}_i = 0$
- If bit i is a message bit, $\hat{u}_i$ is found using a SISO decoder

Polar Code

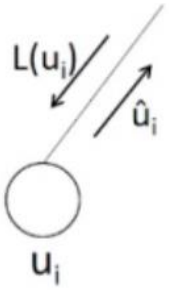
* Polar Code (Decoding)



Polar Code

* Polar Code (Decoding)

SC decoder: decision in a leaf node



If `i' is a frozen position: $\hat{u}_i = 0$

If `i' is a message position: $\hat{u}_i = 0$, if $L(u_i) \geq 0$; $\hat{u}_i = 1$, if $L(u_i) < 0$

Polar Code

* Polar Code (Decoding)

SC decoder: sequence of operations

- Start at root

At every node…

- If not leaf, do the following in sequence
  - Do Step L and go to left child
  - When decision is received from left child, do Step R and go to right child
  - When decision is received form right child, do Step U and go to parent
- If leaf, make decision and go to parent

Polar Code

Polar Code (Decoding)

- This structure is used to encode message **c** = [1, 0, 1, 1] in n = 3 steps with the assistance of input vector u = [0, 0, 0, 1, 0, 0, 1, 1] to obtain codeword d = [1, 0, 1, 0, 0, 1, 0, 1].

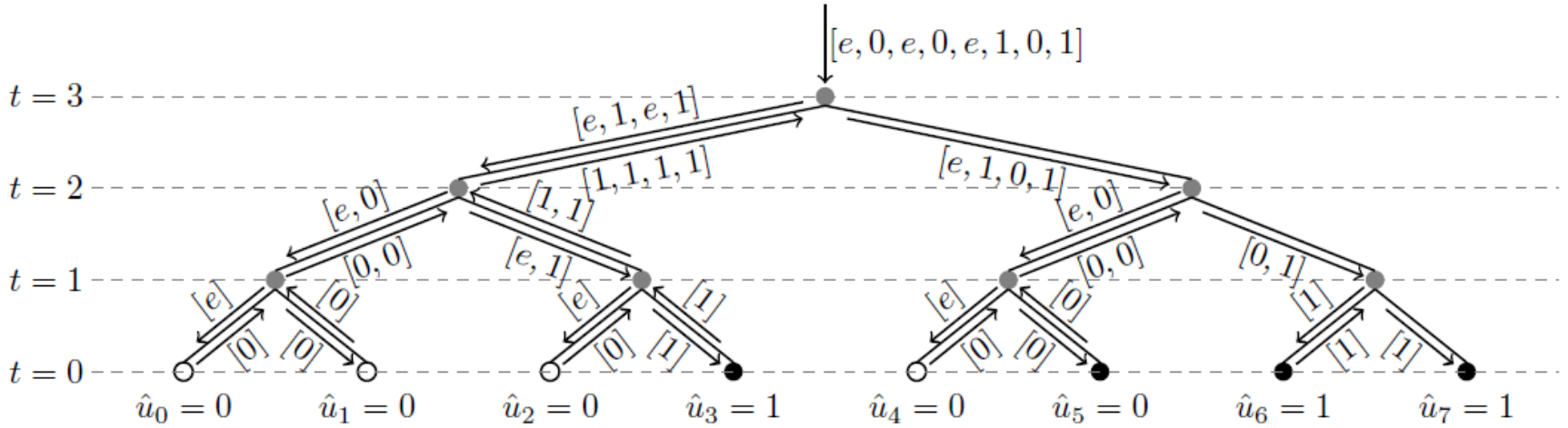
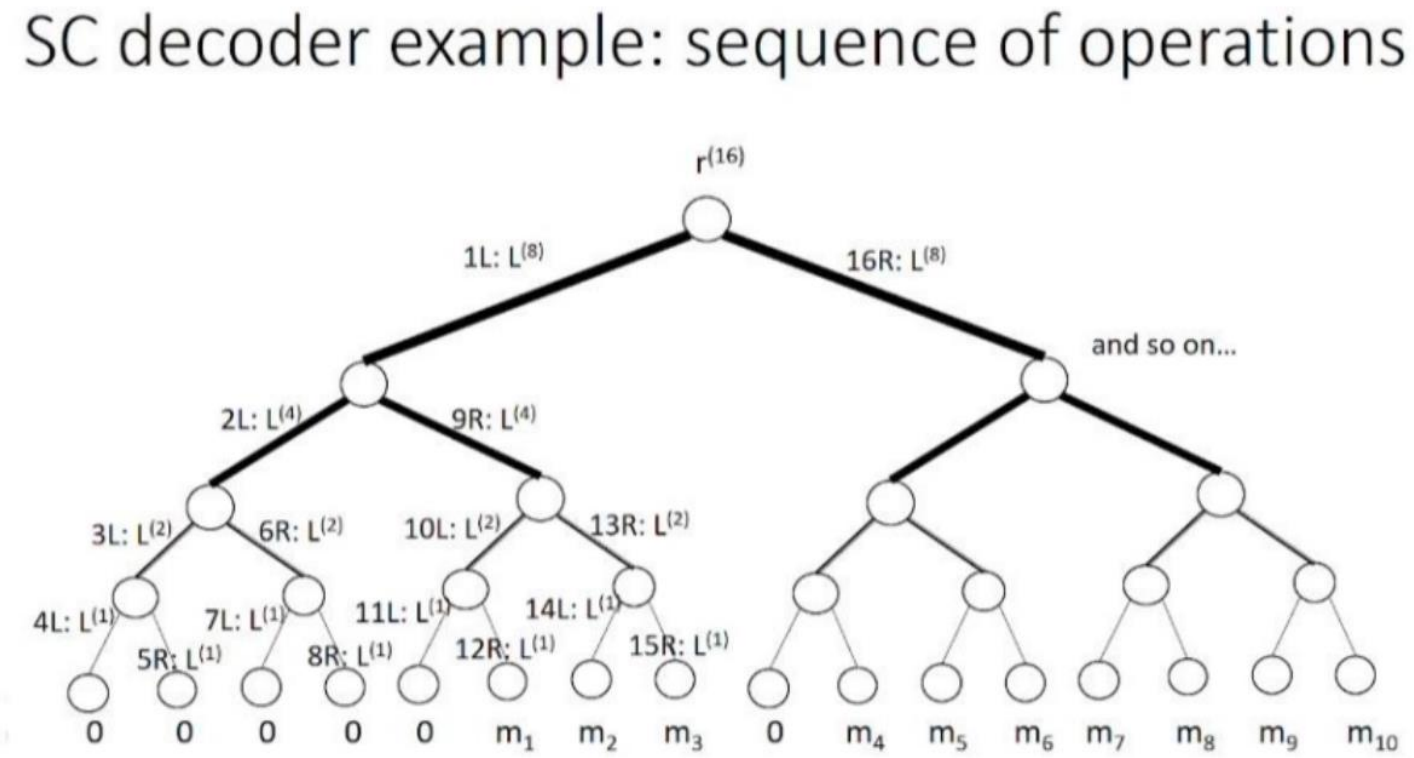


Fig. : SC decoding of an $(8, 4)$ polar code over a BEC; white and black dots represent frozen and information bits.

- black leaf nodes representing information bits and white ones frozen bits.

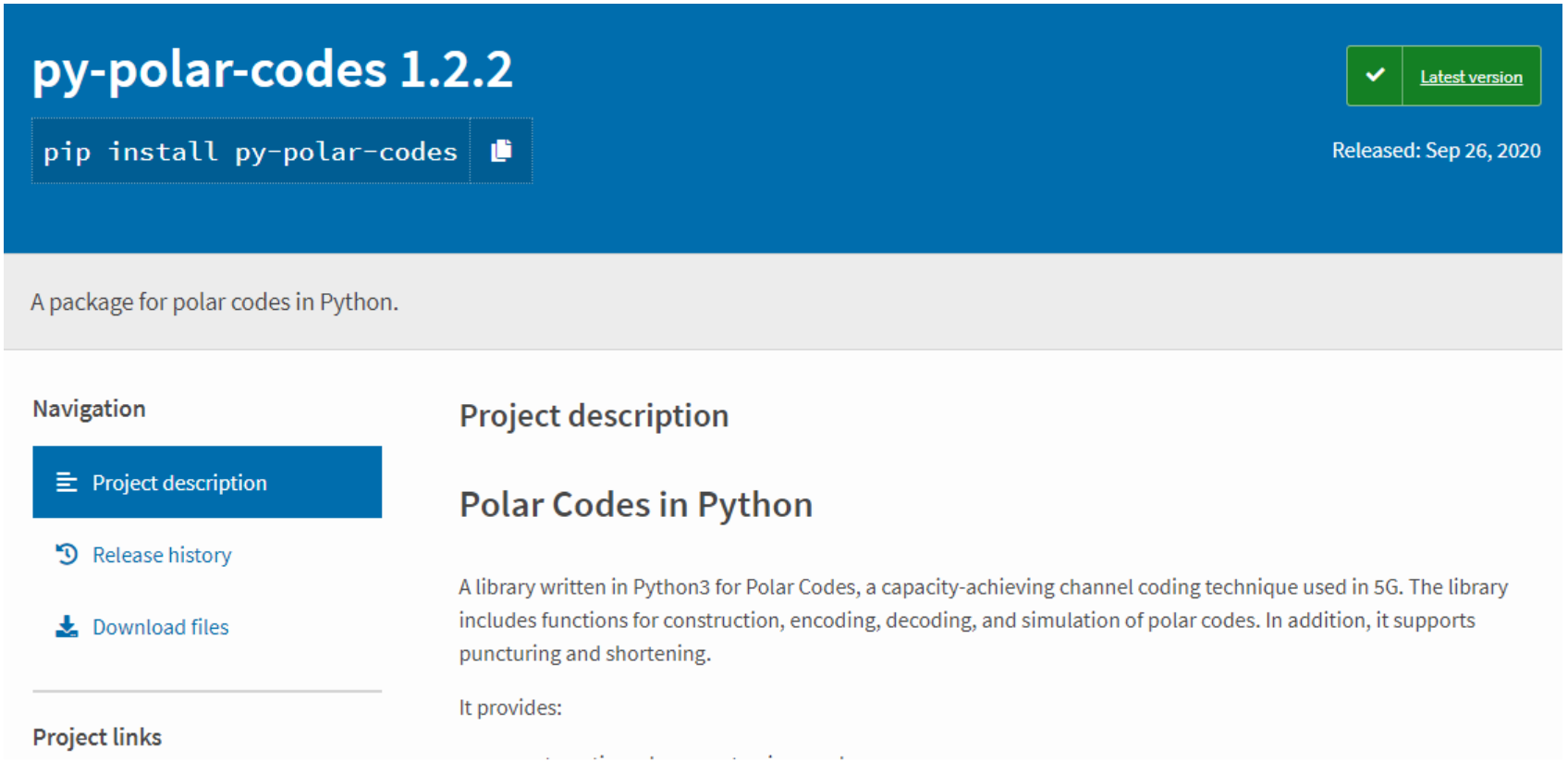
Polar Code

* Polar Code (Decoding)



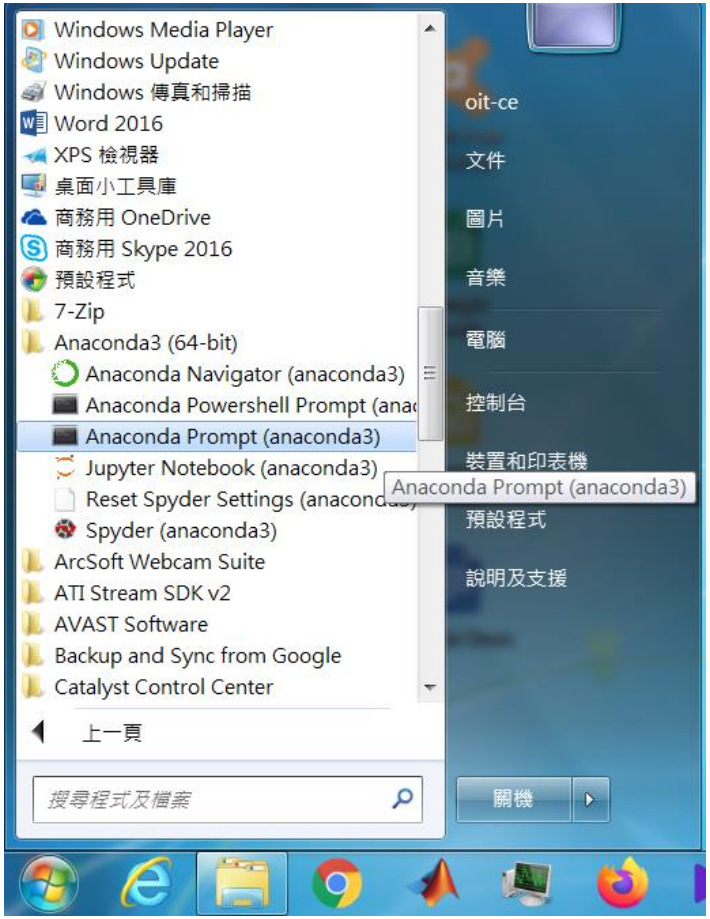
Sample Code for Polar Code

* https://pypi.org/project/py-polar-codes/



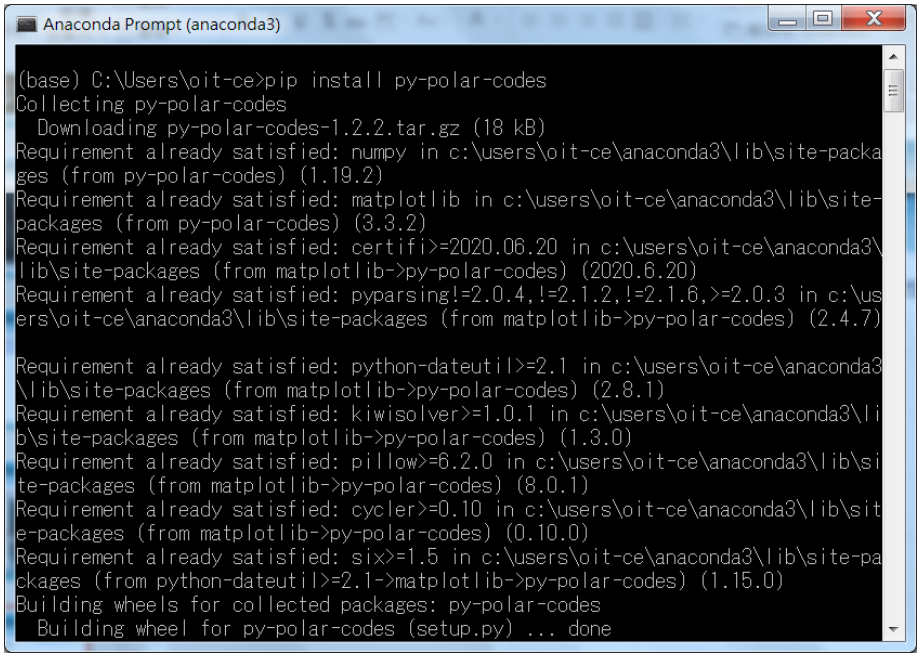
Sample Code for Polar Code

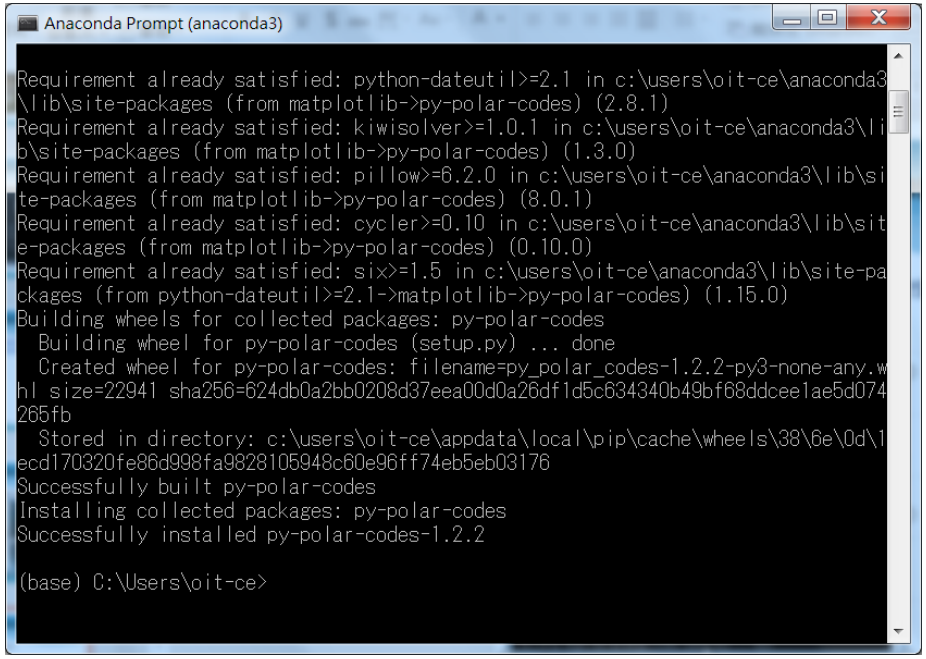
* 進入 Anaconda Prompt(anaconda3)



Sample Code for Polar Code

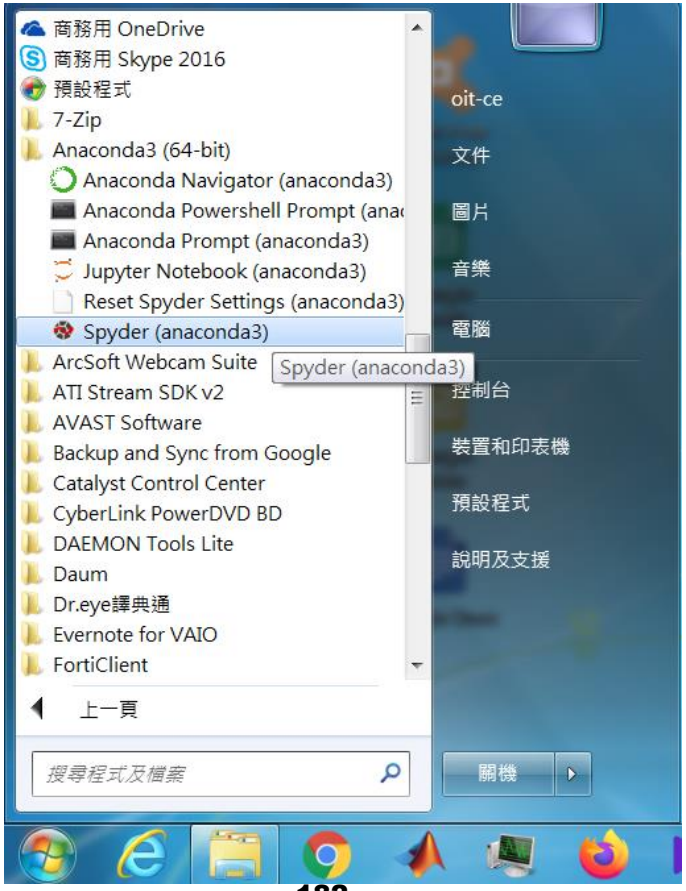
* pip install py-polar-codes





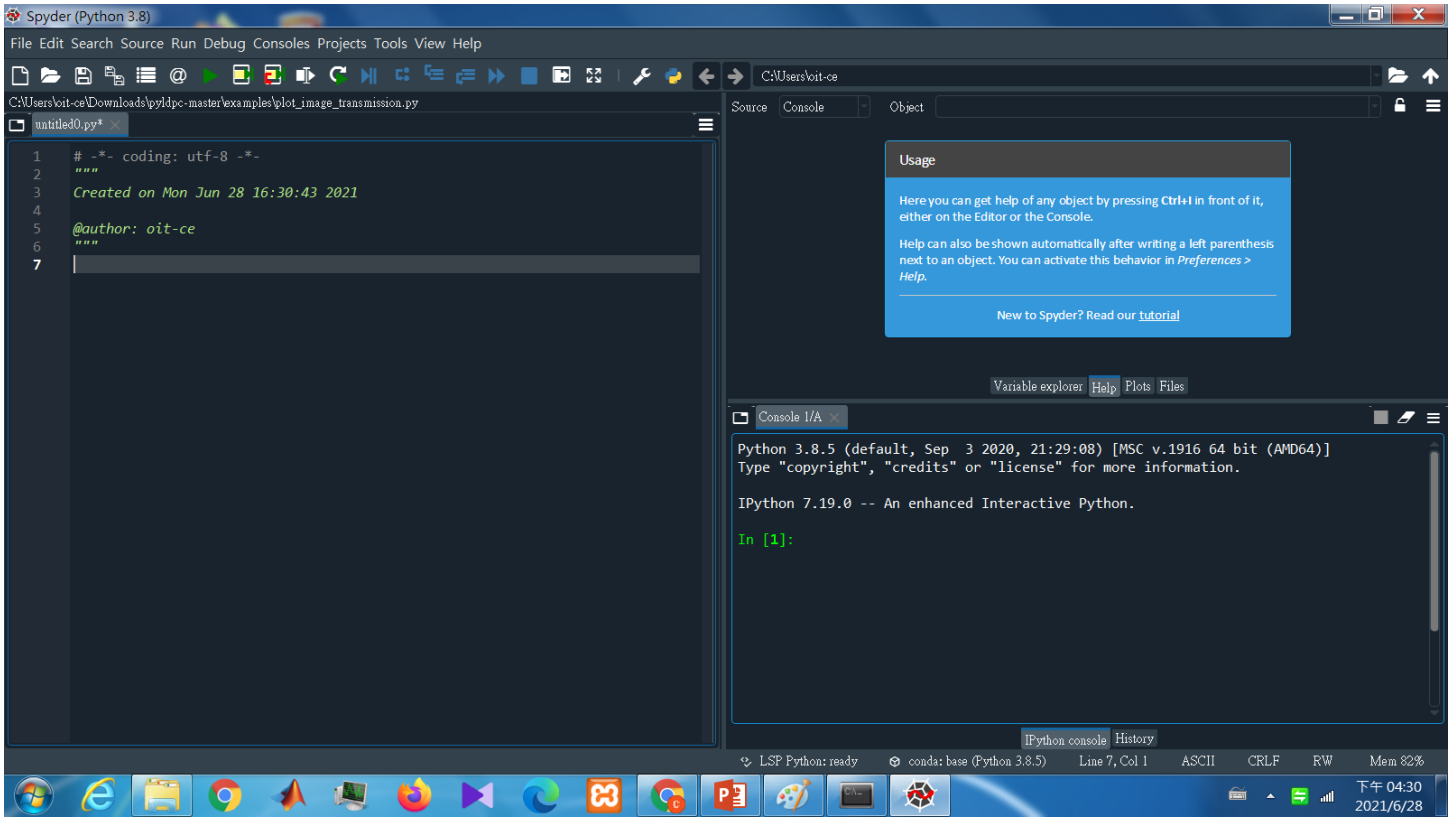
Sample Code for Polar Code

* 進入Anaconda 的 Spyder 整合環境



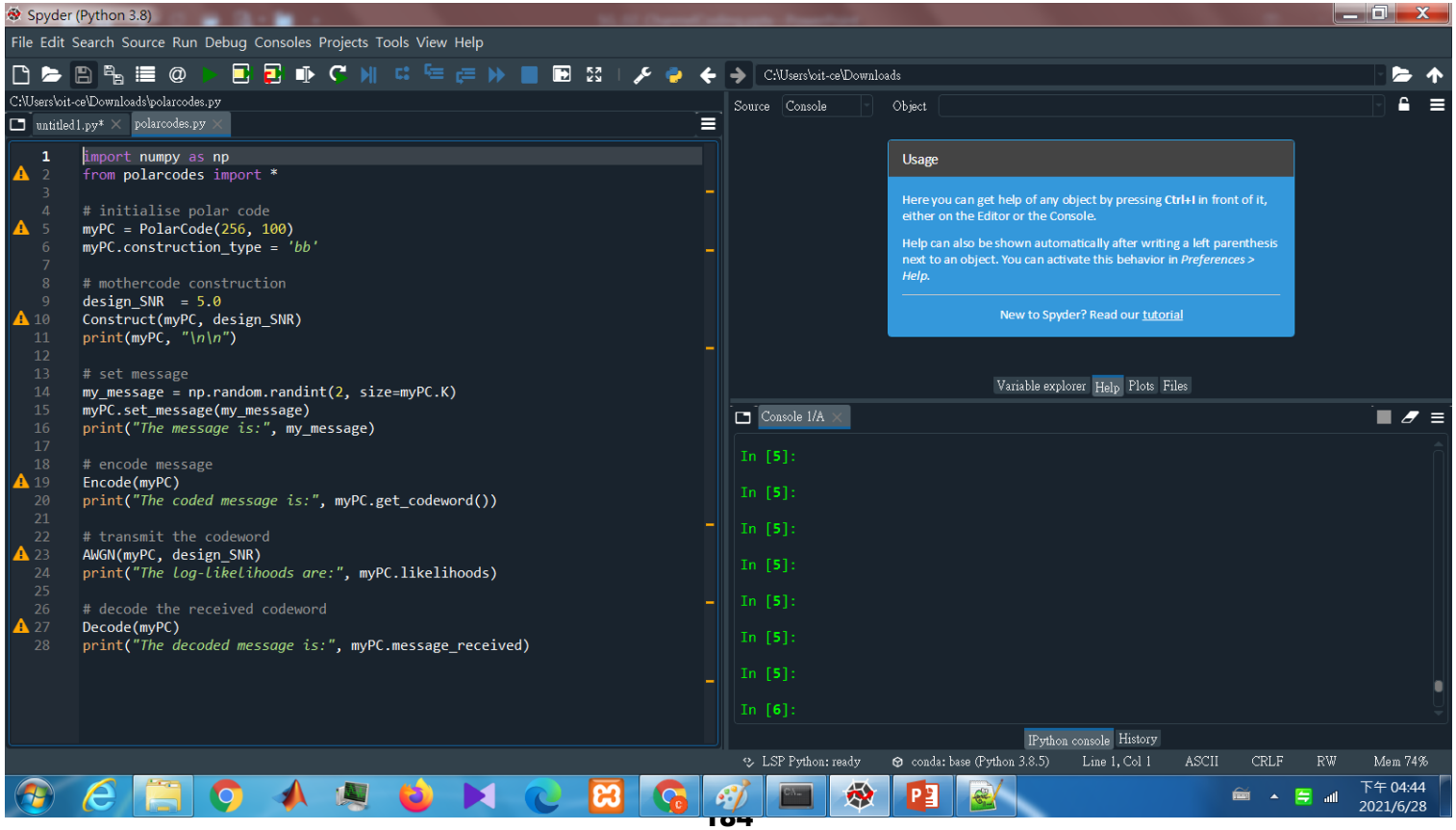
Sample Code for Polar Code

* 進入Anaconda 的 Spyder 整合環境



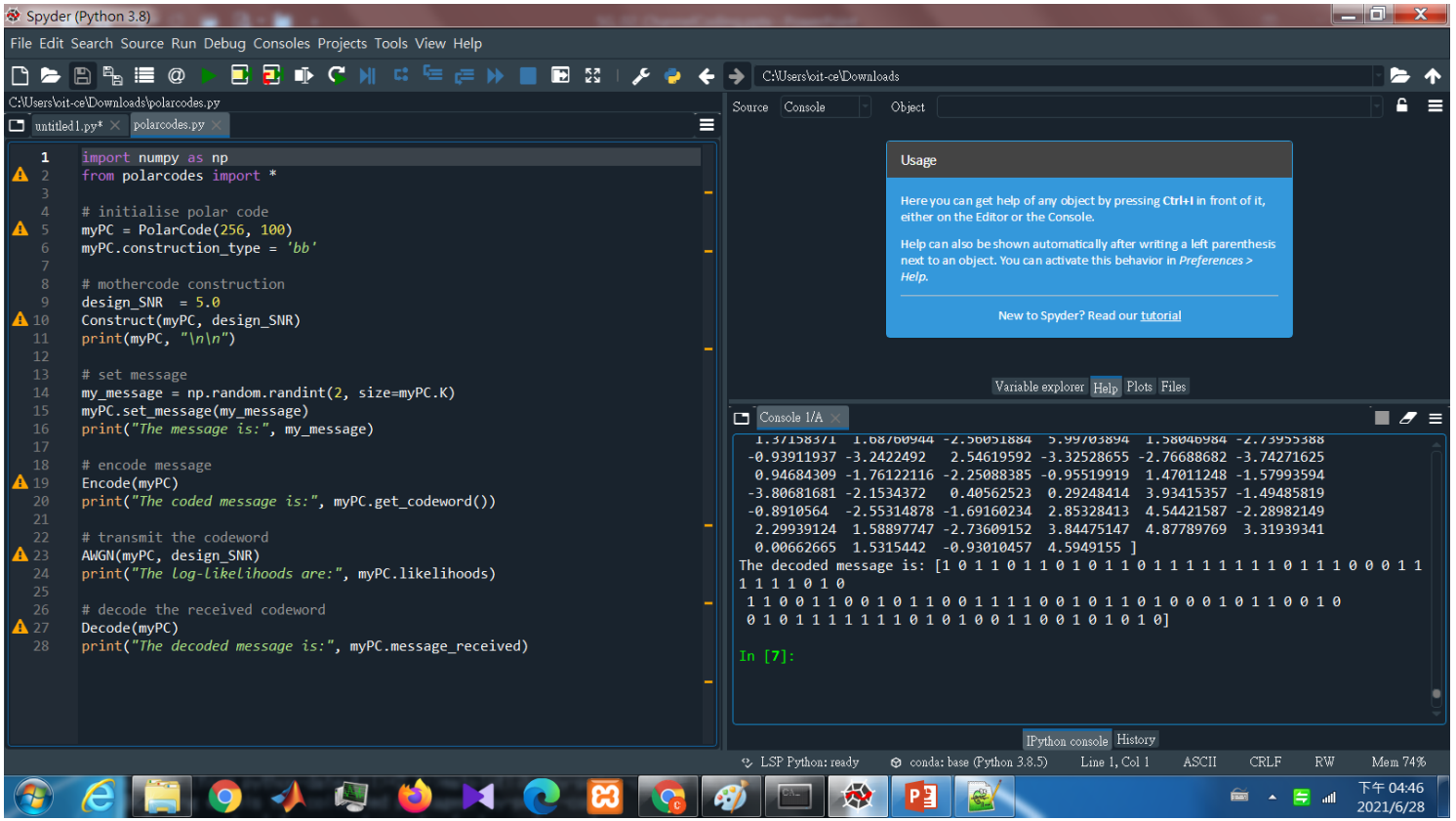
Sample Code for Polar Code

* 開啟 polarcodes.py , (https://pypi.org/project/py-polar-codes/)



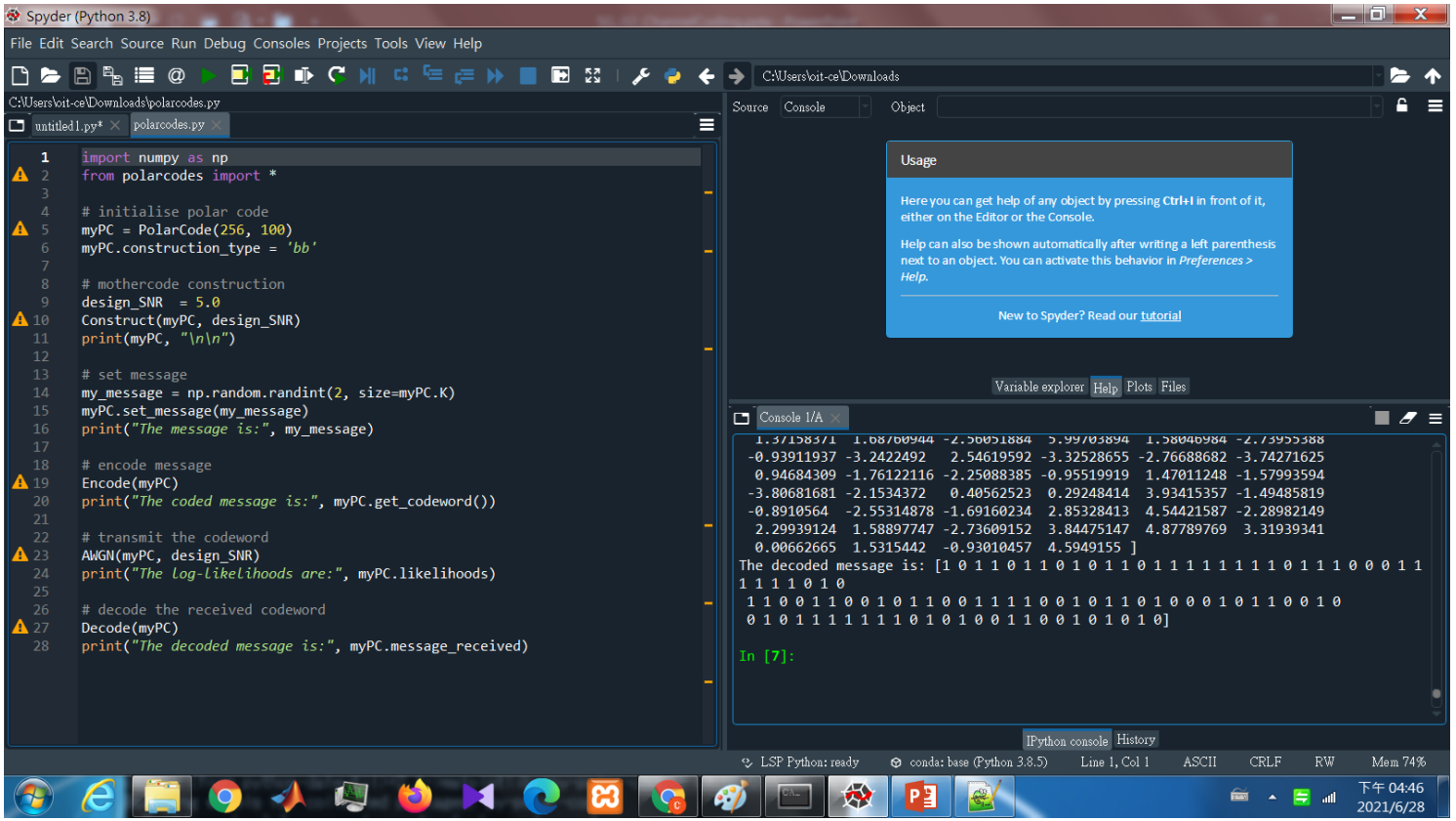
Sample Code for LDPC

* 執行



Sample Code for Polar Code

* 執行



Sample Code for Polar Code

* 執行

```
runfile('C:/Users/oit-ce/Downloads/polarcodes.py', wdir='C:/Users/oit-ce/Downloads')
========== Polar Code ==========
N: 256
M: 256
K: 100
Mothercode Construction: bb
Ordered Bits (least reliable to most reliable): [
   0  16  32  64 128   8   4 192 160  96 144   2  80  48 136  72  40 132
  68  24   1  36 224 130  20 208  66 176 200 112  34 168  12 104 196 152
 129  18  88 164  65 100  56 148  33  10 194  84 240 162  17 140  52 232
  98   6  76 146 216 193   9  82 228  44 184 161 138  50 212 120   5  28
  97  74 145 180 226 134  42   3 204  81 116  70 137 210  26  49 172  38
  73 108 178 248 133 156  41 225  22 202 114  69  92  25  14 170 131 209
  37  60 244 106 198  67  21 154 177  35  13  90 166 201 113  19 236  58
 102  11 169 150 242 105   7  86 197 220 153 142  54  89  78 165 188  57
  46 101 234 124 195 149  30  85 163 141  53 218  99 241  77 147  45 186
 230  83  29 122 139  51  75 233 135  43 214  71  27  39 182  23 118 217
 206  15 185 174 229 252 121 110 158  94 213  62 181 117 227 205 173 109
 211 157 250  93 179  61 115 203 171 107 199 155  91 167  59 103 151 246
```

Sample Code for Polar Code

* 執行

```
87 143  55  79  47  31 249 238 222 190 126 245 237 221 189 243 125 235
 219 187 123 231 215 183 119 207 175 111 159  95  63 254 253 251 247 239
 223 191 127 255]
Frozen Bits: [218  53 141 163  85  30 149 195 124 234 101  46  57 188 165  78  89  54
 142 153 220 197  86   7 105 242 150 169  11 102  58 236  19 113 201 166
  90  13  35 177 154  21  67 198 106 244  60  37 209 131 170  14  25  92
  69 114 202  22 225  41 156 133 248 178 108  73  38 172  49  26 210 137
  70 116  81 204   3  42 134 226 180 145  74  97  28   5 120 212  50 138
 161 184  44 228  82   9 193 216 146  76   6  98 232  52 140  17 162 240
  84 194  10  33 148  56 100  65 164  88  18 129 152 196 104  12 168  34
 112 200 176  66 208  20 130 224  36   1  24  68 132  40  72 136  48  80
   2 144  96 160 192   4   8   0  16  32  64 128]
Puncturing Flag: False
Puncturing Parameters: {punct_type:
                        punct_algorithm:
                        punct_set: []
                        source_set: []
                        update_frozen_flag: None}
```

Sample Code for Polar Code

* 執行

The message is: [1 0 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 0 0 1 1 1 1 1 1 0 1 0
1 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 0 1 0 1 1 0 1 0 0 0 1 0 1 1 0 0 1 0
0 1 0 1 1 1 1 1 1 1 0 1 0 1 0 0 1 1 0 0 1 0 1 0 1 0]
The coded message is: [0 0 0 1 1 1 0 1 0 1 1 1 0 1 0 0 1 1 1 0 1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 1
0 0 0 0 1 1 1 1 0 1 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 1 1 1 0 0 0 1 1 0 1 0 1
0 0 1 1 1 0 0 0 0 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 0 1 1 0 1 0 0 0 1 0 1 0
0 1 0 0 0 1 1 1 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 1 1
0 1 0 0 1 1 1 0 1 1 0 1 0 0 0 1 0 0 0 1 1 0 1 1 0 1 0 0 1 1 1 0 0 0 1 0 0
0 1 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0 1 1 1 1 0 1 0 1 1 0 0 1 0 0 1
1 1 0 1 1 1 1 1 1 0 0 1 1 1 0 0 0 1 1 1 1 0 0 1 0 0 1 0 0 0 0 0 1 0]
The log-likelihoods are: [4.54579816 0.84510999 2.47211405 0.1410955 -3.24070706 -3.22661411
 1.93141327 -2.30829225 0.7875071 -1.02742401 -3.01269199 -4.25501337
 5.52868545 -2.69222501 2.08167108 0.51424562 -2.96977299 -1.60539454
-1.82284083 1.52462041 -4.0315815 -1.75098736 -2.24688435 2.61428908
 5.18554129 2.29489532 2.48714837 -1.83821368 2.228472   2.36843926
 1.81364615 -2.97552467 4.0981338 -1.45705029 3.20856617 0.4857722
-1.8951422   4.56053433 0.48827241 4.04473014 1.91551152 -1.74997061
-2.0149377 -3.26957676 -4.4532108   2.43183176 -3.34017274 -2.98285351
-0.42279319 -0.45115553 1.95227061 -1.58822632 4.14547936 4.12320043

Sample Code for Polar Code

* 執行

```
3.02997729 -2.6589483  -3.73557695 -1.20224163 -4.38875659  3.49601738
 0.85787144 -1.89563114 -2.39882108  0.48975782  2.6268727  -3.56062563
 1.95873979 -2.20548505  1.30074328  2.23006664 -2.44867282 -1.12361783
-1.71665512  3.33347855  1.5876199   1.6163265   3.0650642  -3.58743797
 3.29448164 -5.36348443 -1.62439926 -2.03576334  1.34684324 -3.19268308
-0.43330356  5.13401455  2.83791388  0.24196617 -1.42660075  3.07527499
-1.05149755 -1.44960967  1.44689618  2.76371963 -4.36904907 -2.87155452
-0.03331589 -3.86504934 -0.1797062   1.85154746  1.05190385 -1.10551273
 3.76053968 -0.97976713  1.38190682  3.21377435 -2.69389016  0.86350452
 2.46885766  1.58463027 -3.0017898  -2.78381474 -1.58517568  1.09509605
 2.99451258 -1.0056052  -1.67484451 -2.36935128 -0.62075633 -2.87505612
-2.60726488  0.81197344  2.88543917  2.95347319 -3.19911845  0.49438083
-2.25079077 -0.15062368  1.60117073  0.09262061  0.75645586  2.18200445
-4.15755233  0.20451622  5.1067657   3.78231415 -0.79812523 -0.74913243
 2.46126446 -4.5859401  -3.73115528 -0.19422885  3.53863718 -0.91280216
 1.16873734  1.80352642  0.57883512 -0.71625816 -1.16949441  3.16042976
-2.35806594 -0.52190546  3.96738652  0.1664273   1.86045233  2.98465567
 3.10684447 -2.9566001   2.52396792  2.33194856  2.275132   -2.21466766
-2.26989452  3.21700696 -2.59176612 -2.63817734  3.42912185 -3.80367323
```

Sample Code for Polar Code

* 執行

1.01362694 -1.22176062 -2.41614726 -1.66552393 -2.56923911 2.16566691
4.44156978 2.69523958 -0.19773657 2.4624747 2.26919638 0.06084258
-1.79607941 3.30902861 -1.10703565 -3.15448261 -0.5333973 4.12017835
-3.69826186 -0.81784511 1.58581359 -0.21895719 3.35520846 -0.69244565
2.72182585 2.29318614 -0.93490478 -1.39119539 -1.28227049 -4.13406488
1.72047047 2.7587242 3.39557326 -1.35138984 -2.2157346 0.1176055
-1.67636667 2.77350741 -3.22128086 2.62704658 -3.57517748 -2.75328088
1.37158371 1.68760944 -2.56051884 5.99703894 1.58046984 -2.73955388
-0.93911937 -3.2422492 2.54619592 -3.32528655 -2.76688682 -3.74271625
0.94684309 -1.76122116 -2.25088385 -0.95519919 1.47011248 -1.57993594
-3.80681681 -2.1534372 0.40562523 0.29248414 3.93415357 -1.49485819
-0.8910564 -2.55314878 -1.69160234 2.85328413 4.54421587 -2.28982149
2.29939124 1.58897747 -2.73609152 3.84475147 4.87789769 3.31939341
0.00662665 1.5315442 -0.93010457 4.5949155]
The decoded message is: [1 0 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 0 0 1 1 1 1 1 1 0 1 0
1 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 0 1 0 1 1 0 1 0 0 0 1 0 1 1 0 0 1 0
0 1 0 1 1 1 1 1 1 1 0 1 0 1 0 0 1 1 0 0 1 0 1 0 1 0]