ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
KHOA ĐIỆN TỬ VIỄN THÔNG

Báo cáo Thực tập ngành ĐTVT

Công ty Viettel High Tech

Giảng viên hướng dẫn **TS. Nguyễn Hồng Thịnh**
Sinh viên: **Lê Trần Anh Dũng - 22029041**

Hà Nội - Tháng 9,2025

Lời cảm ơn

Em xin gửi lời cảm ơn chân thành đến cô TS.Nguyễn Hồng Thịnh đã dành thời gian để đọc và góp ý cho báo cáo của em. Những nhận xét và hướng dẫn của cô là nguồn động lực và cũng là cơ sở quan trọng giúp em hoàn thiện hơn trong quá trình nghiên cứu, để các báo cáo được hoàn chỉnh và kỳ thực tập được diễn ra hiệu quả.

Bên cạnh đó, em cũng xin bày tỏ lòng biết ơn sâu sắc đến anh Lê Thanh Bằng - Trưởng phòng Phát triển nền tảng FPGA - Trung tâm Nghiên cứu thiết bị Vô tuyến băng rộng - Khối 2 - TCT VHT và anh Trung. Trong suốt thời gian thực tập, em đã nhận được sự chỉ dẫn tận tình, những chia sẻ kinh nghiệm quý báu cũng như sự hỗ trợ nhiệt tình từ anh Bằng và anh Trung. Đây là những hành trang vô cùng quan trọng giúp em tiếp cận thực tế công việc, củng cố kiến thức chuyên môn và định hướng rõ ràng hơn cho con đường nghiên cứu, học tập của mình.

Em xin kính chúc cô sức khỏe dồi dào, hạnh phúc và thành công trong công tác giảng dạy và nghiên cứu. Đồng thời, em cũng xin kính chúc anh Bằng và anh Trung luôn mạnh khỏe, nhiều niềm vui và đạt nhiều thành công trong công việc cũng như cuộc sống.

Hà Nội, ngày 23 tháng 8 năm 2025

Mục lục

Danh sách hình vẽ

Danh sách bảng

Chương 1

Mở đầu

Trong quá trình 6 tuần thực tập tại công ty Vietel High Tech, em đã được giao các đầu việc sau đây:

1. Đọc tổng quan về 5G
   {HYPERLINK "https://www.sharetechnote.com/html/5G/5G_ChannelMapping.html"}
2. Sau 3-4 tuần đọc tiếp cấu trúc kênh PUCCH format 0.
   - Kiến trúc kênh PUCCH hoạt động ở L3.
   - Kiến trúc kênh PUCCH hoạt động ở L2.
   - Kiến trúc kênh PUCCH hoạt động ở L1.
3. Làm bài tập FPGA.
4. Đã có tín hiệu từ UE gửi lên, thiết kế kênh nhiễu, cách L1 decode ra bản tin ban đầu -> gửi lên L2, L2 làm gì-> gửi lên L3.
5. Tìm hiểu sử dụng AI engine trên chip Versal và sử dụng cho việc nhân và nghịch đảo matrix

Đầu việc 1, 2 làm lần lượt; 1, 3, 4 có thể làm song song

Tiến độ các tuần:

Tuần 1

Công việc chính:

- Đọc tài liệu tổng quan về 5G: Tìm hiểu các khái niệm như Frame Structure, Numerology, Waveform, Frequency Band, BWP, Synchronization, Beam Management, CSI Framework, Channel Mapping.
- Làm quen với FPGA, AI Engine: Tìm kiếm các tài liệu của ADM về AI Engine và chip Versal.

Khó khăn: Chưa có đủ tài liệu hướng dẫn chi tiết về AI Engine, chưa có hiểu biết về các thế hệ mạng trước đó nên khó tiếp cận với kiến trúc 5GNR.

Tuần 2

Công việc chính

- Tiếp tục nghiên cứu về 5G, kênh PUCCH format 0 ở các lớp L1, L2, L3
- Tìm hiểu sâu hơn về AI Eingine trên Versal: kiến trúc của AIE tile, streaming interconnect, DMA, programmable logic, đọc tài liệu từ ADM/Xilinx về sử dụng AIE cho nhân ma trận.

Kết quả: Đã hình dung được kiến trúc tổng thể của chip Versal và cách AI Engine hoạt động, các kênh PUCCH hoạt động.

Tuần 3

Công việc chính:

- Ghi chép, tổng hợp tài liệu để viết báo cáo về cách tạo chuỗi của kênh PUCCH format 0 và chuẩn bị triển khai kênh này trên MATLAB.

Khó khăn: không triển khai được FPGA với AIE do chưa có phần mềm Vitis.

Tuần 4

Công việc chính:

- Đọc và ghi chép về PUCCH ở lớp vật lý, xử lý bản tin từ UE đến gNB.
- Mô phỏng tín hiệu PUCCH format 0 trên MATLAB, thêm nhiễu AWGN (SNR = 10dB).
- Phân tích ảnh hưởng của nhiễu đến tín hiệu và chòm sao tín hiệu.
- Tìm tài liệu về điều chế và giải điều chế sóng dựa trên chuỗi Zadoff-Chu.

Tuần 5

Công việc chính:

- Xây dựng bộ sinh chuỗi PUCCH format 0 từ các bit HARQ và SR đầu vào.
- Mô phỏng toàn bộ quá trình sinh chuỗi trên MATLAB:
  - Tính toán các tham số như $Mzc, Tu, v(n), u, \alpha$.
  - Sinh chuỗi Zadoff-Chu và áp dụng xoay pha.
  - Viết lại toàn bộ hàm sinh chuỗi giả ngẫu nhiên theo công thức trong tài liệu.
  - So sánh kết quả với thư viện MATLAB gốc để kiểm chứng độ chính xác.

Kế hoạch tuần tới: Viết hàm decode để trích xuất thông tin HARQ và SR từ tín hiệu nhận được.

Tuần 6

Công việc chính:

- Xây dựng bộ giải mã chuỗi PUCCH format 0 từ tín hiệu nhận được sau khi được truyền qua kênh nhiễu.
- Chỉnh sửa lại code sinh chuỗi trên MATLAB để phù hợp với các điều kiện đầu vào khác nhau.
- Báo cáo tổng kết với người hướng dẫn.

Trong báo cáo này, em sẽ tập trung vào những khía cạnh tổng quan nhất của 5G và thực thi kênh PUCCH (physical uplink channel) format 0 bằng MATLAB với công đoạn mã hóa và giải mã bằng so sánh tương quan theo tài liệu hướng dẫn của 3GPP và website Sharetechnote. Bên cạnh đó, báo cáo này cũng bao gồm những kiến thức cơ bản em tìm hiểu được về kiến trúc của công nghệ AI Engine trên chip Versal theo các tài liệu của nhà sản xuất AMD cung cấp.

Mục tiêu chính của kì thực tập này là củng cố kiến thức nền tảng và tìm hiểu các kiến thức mới về mạng di động 5G, nắm bắt các khái niệm quan trọng trong lớp vật lý của 5GNR, đồng thời rèn luyện kỹ năng triển khai và mô phỏng các kênh vật lý bằng công cụ MATLAB.

Bên cạnh đó, kì thực tập cũng giúp em bước đầu tiếp cận với công nghệ FPGA và đặc biệt là kiến trúc AI Engine trên chip Versal của AMD, từ đó có định hướng nghiên cứu và ứng dụng cho các đề tài sau này liên quan đến xử lý tín hiệu số tốc độ cao và các hệ thống thông tin không dây.

Báo cáo được tổ chức thành 3 phần chính:

- Phần 1: Tổng quan về 5G new radio.
- Phần 2: Thực thi mã hóa và giải mã kênh PUCCH format 0 trên MATLAB.
- Phần 3: Kiến trúc AI Engine trên chip Versal.

Chương 2

Tổng quan về 5G/NR.

Thời gian nghiên cứu: tuần 1, 2, 3.

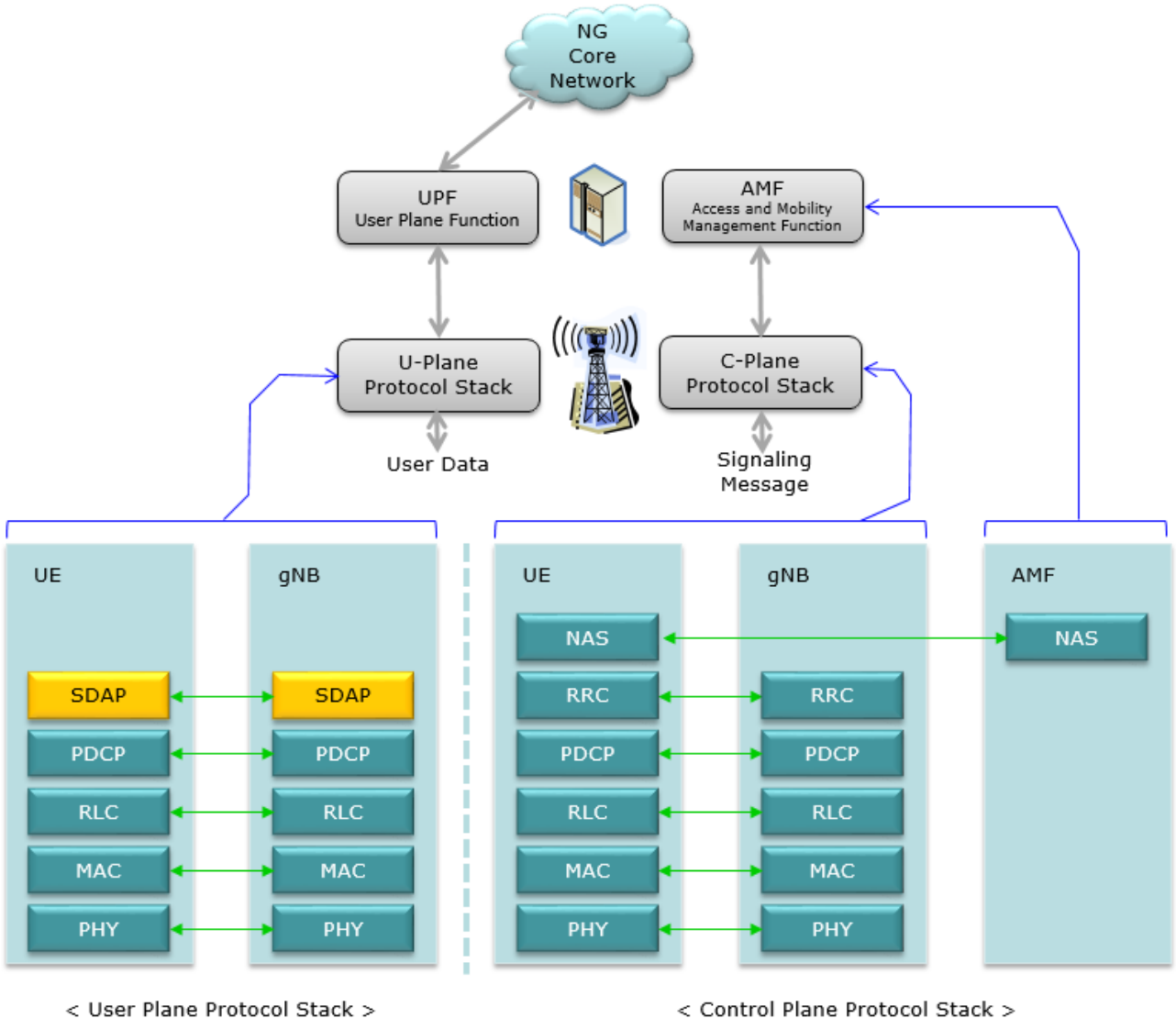
5G/New radio(NR) là thế hệ mạng thứ 5 với tốc độ tối đa được công bố lên tới 20Gbps, so với 1Gbps tối đa của 4GLTE. 5G hứa hẹn sẽ hỗ trợ rất lớn với hệ thông IoT, AI thời gian thực với các xe tự hành, hệ thống thời gian thực....

2.1 Kiến trúc.

- gNB: trạm gốc 5G, chịu trách nhiệm truyền và nhận sóng tới các thiết bị, bao gồm cả C-plane (Control plane) và U-plane (User-plane).
- UE: thiết bị người dùng.

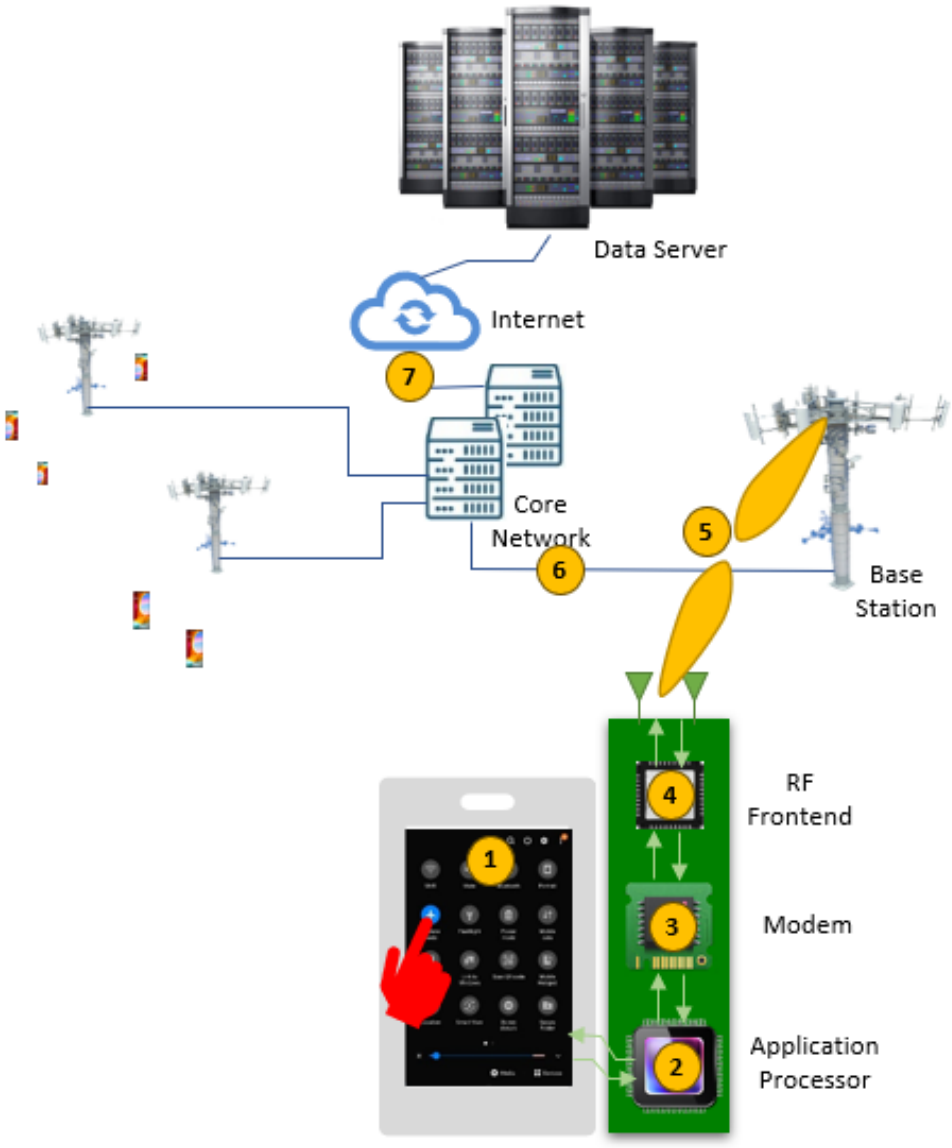
Kiến trúc hệ thống của 5G/NR cũng gần tương đương với 4G/LTE được mô tả ở 2.1, bao gồm 2 stack đảm nhiệm tùy theo loại dữ liệu được xử lý: C-plane: đảm nhận signaling message trong khi U-plane: đảm nhận dữ liệu người dùng. Cả U-plane và C-plane đều bao gồm những thành phần chung: PHY MAC, RLC và PDCP nhưng các thành phần ơ trên cùng có sự khác nhau.

- C-Plane: có thêm 2 layer RRC (radio resource control: thành phần điều khiển trung tâm, giúp UE và mạng cấu hình và đồng bộ các lớp giao thức thấp để truyền thông) và NAS (Non-Access Stratum).
- U-plane: có thêm 1 lớp SDAP(Service Data Adaptation Protocol để phân loại luồng dữ liệu người dùng) ở trên cùng.



Hình 2.1: Kiến trúc mức cao.

2.2 Luồng dữ liệu trong hệ thống.



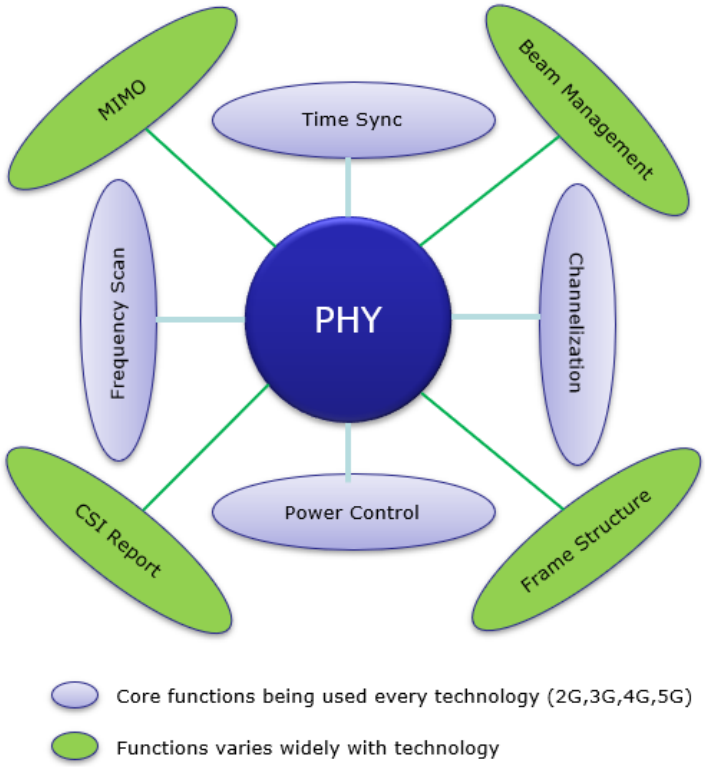
Hình 2.2: Luồng dữ liệu trong hệ thống.

Quá trình truyền và nhận dữ liệu trong mạng 5G, như minh họa, có thể được mô tả qua các bước sau:

1. Người dùng tương tác với ứng dụng trên điện thoại thông minh, tạo ra dữ liệu cần được truyền tải (ví dụ: gửi tin nhắn, truy cập web, xem video).
2. Bộ xử lý ứng dụng (Application Processor) và modem trong điện thoại phối hợp để xử lý, mã hóa và đóng gói dữ liệu này theo các giao thức truyền thông phù hợp (bao gồm các tầng như PDCP, RLC, MAC...).

3. Dữ liệu đã mã hóa được chuyển tới frontend tần số vô tuyến (RF Frontend) của thiết bị, nơi nó được chuyển đổi thành tín hiệu vô tuyến và phát sóng qua ăng-ten.
4. Trạm gốc (gNB) thuộc mạng truy nhập vô tuyến 5G (5G RAN) sẽ tiếp nhận tín hiệu vô tuyến từ thiết bị người dùng.
5. Trạm gốc gNB xử lý tín hiệu, sau đó chuyển tiếp dữ liệu tới mạng lõi (Core Network), nơi chịu trách nhiệm điều phối tổng thể mạng và kết nối với Internet.
6. Mạng lõi định tuyến dữ liệu đến mạng Internet công cộng để đến được máy chủ dữ liệu đích (ví dụ: máy chủ của ứng dụng hoặc website mà người dùng đang truy cập).
7. Máy chủ dữ liệu tiếp nhận yêu cầu, xử lý và tạo ra phản hồi phù hợp (ví dụ: kết quả tìm kiếm, nội dung video, phản hồi từ ứng dụng...).
8. Luồng phản hồi đi ngược lại theo trình tự ban đầu: từ máy chủ → qua Internet → đến mạng lõi → đến trạm gốc gNB → phát sóng trở lại đến thiết bị người dùng.
9. Modem và bộ xử lý ứng dụng trên điện thoại tiếp nhận phản hồi, giải mã và xử lý dữ liệu để hiển thị kết quả cuối cùng cho người dùng thông qua giao diện ứng dụng.

2.3 Lớp vật lý



Lớp vật lý (PHY) trong 5G NR đóng vai trò trung tâm trong việc xử lý tín hiệu vô tuyến, đảm bảo truyền và nhận dữ liệu hiệu quả giữa thiết bị người dùng (UE) và trạm gốc (gNB).

Theo hình minh họa, lớp PHY thực hiện các chức năng cốt lõi như đồng bộ thời gian (Time Sync), quét tần số (Frequency Scan), điều khiển công suất (Power Control), báo cáo trạng thái kênh (CSI Report) và phân kênh (Channelization). Ngoài ra, các chức năng như MIMO, quản lý chùm tia (Beam Management) và cấu trúc khung (Frame Structure) thể hiện sự linh hoạt và khác biệt của 5G so với các thế hệ trước, giúp tối ưu hóa hiệu suất truyền dẫn, mở rộng vùng phủ sóng và đáp ứng đa dạng nhu cầu dịch vụ hiện đại.

2.3.1 Frame Structure.

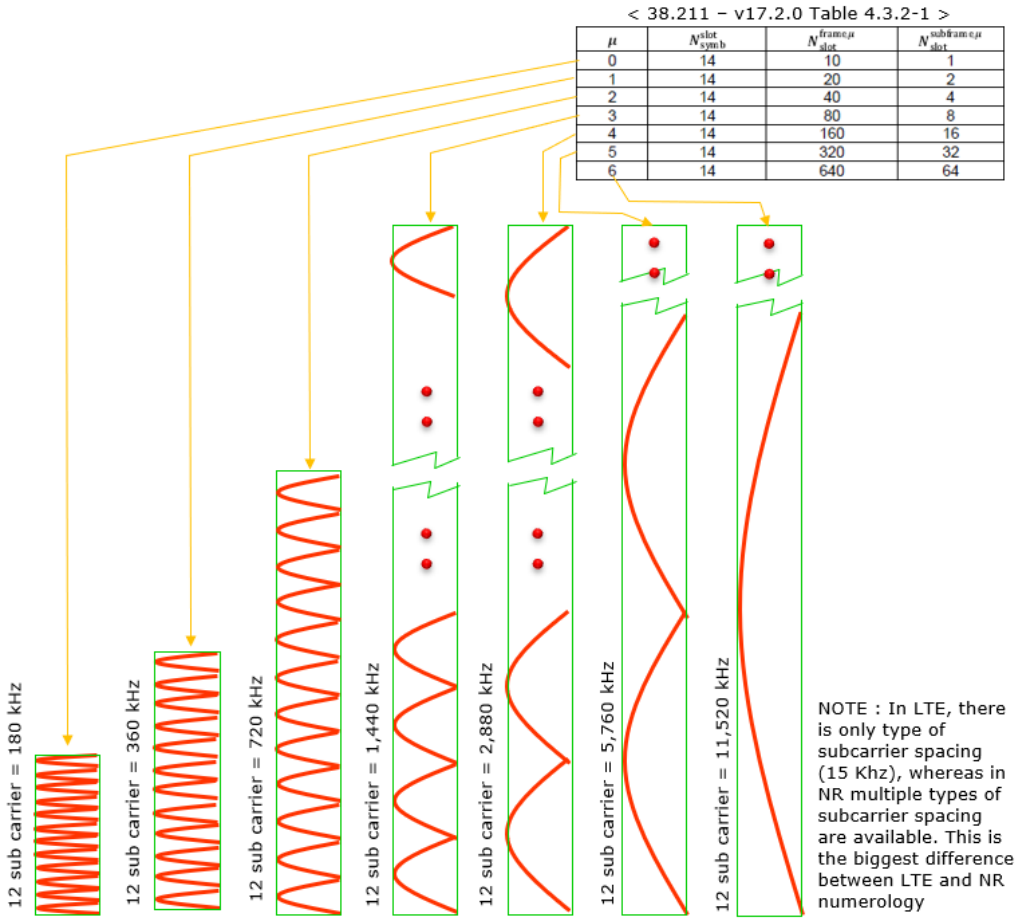
Frame structure đã được thiết kế linh hoạt hơn so với LTE nhằm phục vụ các tác vụ hiện tại như xe tự hành, IoT, truyền video tốc độ cao. Tài nguyên được chia nhỏ thành các resource block sau đó được chia nhỏ thành các element trên lưới tài nguyên (resource grid). Điều này giúp phân bổ tài nguyên hợp lý tùy theo nhu cầu sử dụng. Bên cạnh đó, khái niệm Numerology là cốt lõi để tạo ra sự linh hoạt này.

Numerology.

Một số khái niệm:

- **Numerology** (kí hiệu là μ): trong 5G là tập hợp các thông số vật lý định nghĩa cấu trúc sóng mang OFDM
- Subcarrier spacing (SCS): Khoảng cách giữa các sóng con (tính bằng kHz).
- Cyclic Prefix (CP): tiền tố vòng để tránh giao thoa đa đường
- Slot length: nhỏ hơn khi subcarrier spacing rộng hơn (μ nhỏ).

Khác với LTE chỉ hỗ trợ một loại subcarrier spacing ở 15kHz, 5GNR hỗ trợ nhiều loại subcarrier spacing tùy theo tham số μ.



Hình 2.3: Các thông số vật lý trong numerology.[1]

Với giá trị μ càng tăng thì khoảng cách giữa các sóng con được nhân đôi theo cấp số nhân 2, thời gian slot giảm, tốc độ truyền nhanh hơn, làm cho băng thông 12 sóng con tăng theo (với μ= 0 thì 12 sub carier là $15 * 12 * 2^0 = 180$ kHz, với $\mu = 6$ thì 12 sub carrier là $12 * 15 * 2^6 = 11520$ kHz).

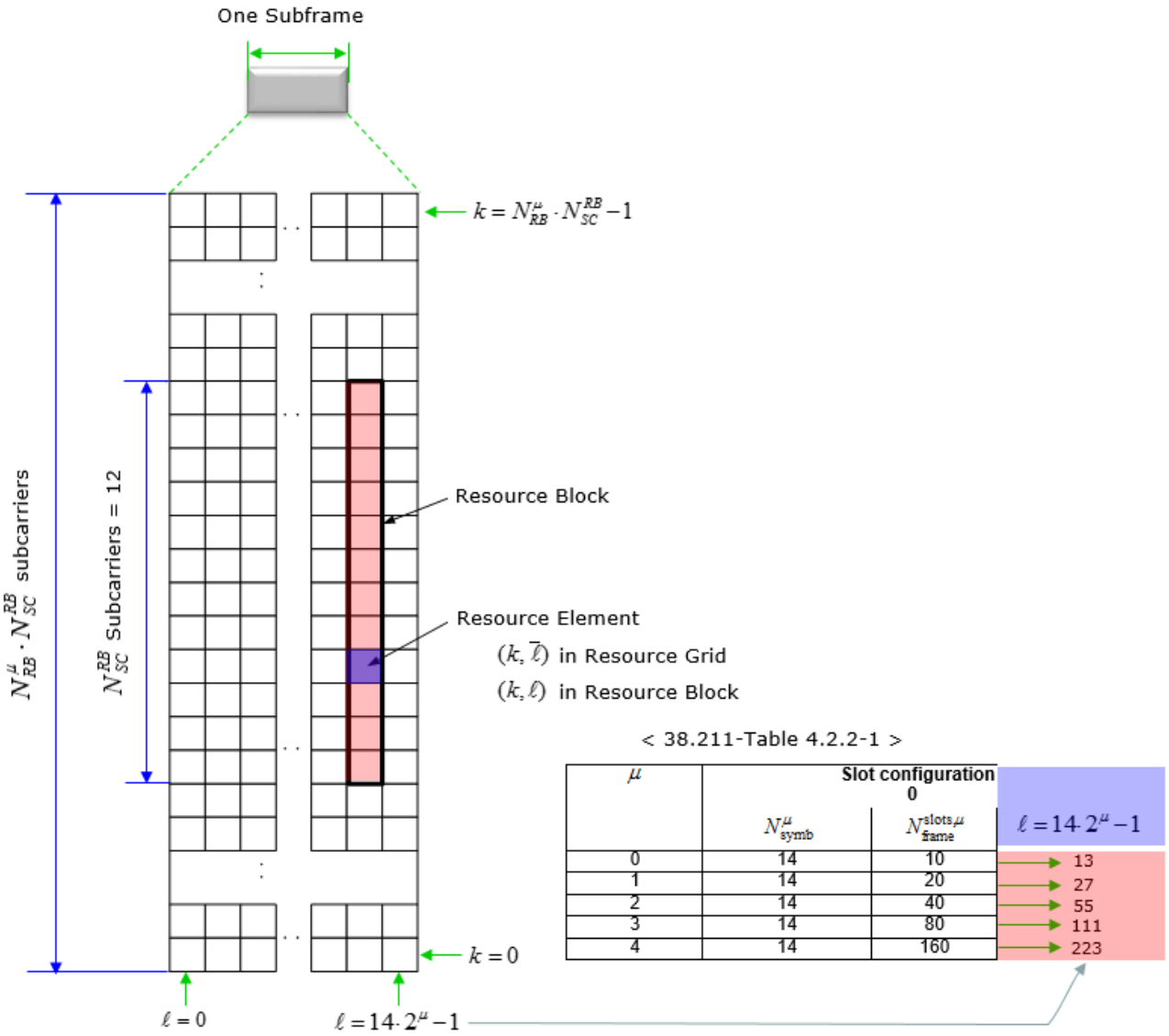
Tại sao lại cần nhiều loại subcarrier spacing?

- Subcarrier spacing lớn (μ cao):
  - Symbols ngắn hơn -> truyền nhah hơn.
  - Phù hợp với tần số cao.
  - Dùng cho truyền tin siêu nhanh (URLLC), ví dụ: xe tự lái.
- Subcarrier spacing lớn (μ nhỏ):
  - Symbol dài hơn -> chống nhiễu tốt hơn.
  - Phù hợp cho môi trường rộng, nhiều nhiễu.
  - Dùng cho truyền tải dữ liệu lớn: xem video 4K, tải dữ liệu lớn.

Lưới tài nguyên (Resource Grid).

Như được mô tả trong hình2.4, 1 subframe = 1 ms, chứa $14 * 2^{\mu}$ symbols.

- Trục hoành (l): thời gian chia thành các OFDM symbols, gồm $14 * 2^{\mu}$ được đánh số từ 0 đến $14 * 2^{\mu} - 1$.
- Trục tung (k): tần số chia thành các subcarriers.
- Resource element(RE): đơn vị nhỏ nhất trong resouce grid, ứng với 1 subcarrier tại 1 OFDM symbol.
- Resource block (RB): Ứng với $12N_{SC}^{RB}$ subcarrier liên tiếp trong miền tần số và độ dài tùy chỉnh trong miền thời gian. [38.211-4.4.4.1]



< 38.211-Table 4.2.2-1 >

μ	Slot configuration 0		$l = 14 \cdot 2^{\mu} - 1$
	N_{symb}^{μ}	$N_{frame}^{slots,\mu}$	
0	14	10	13
1	14	20	27
2	14	40	55
3	14	80	111
4	14	160	223

Hình 2.4: Mô tả trực quan của khối dữ liệu trong 1 subframe.[1]

2.4 Dạng sóng.

Khi bắt đầu thiết kế 5G, nhiều waveform mới được đề cập (FBMC, UFMC). Tuy nhiên sau khi đánh giá, OFDM vẫn được sử dụng với tính linh hoạt trong cấu hình. [2] Khái niệm cơ bản về OFDM:

- Các sóng trực giao với nhau, tại thời điểm lấy mẫu của sóng này thì các sóng khác bằng 0.
- Nhằm truyền đa kênh.
- Tuy nhiên trong không gian cũng có nhiễu làm trôi tần số, gây ra ảnh hưởng của các kênh lên nhau.
- Cách xử lý: thêm cyclic prefix và để xử lý lỗi chồng lấn.

Chương 3

Sinh và phát hiện chuỗi dựa trên tương quan để giải mã UCI trong PUCCH format 0.

Thời gian nghiên cứu: tuần 3, 4, 5, 6.

3.1 Khái quát về kênh PUCCH và các format.

Kênh PUCCH (Physical Uplink Control Channel) là kênh vật lý đường lên trong hệ thống 4G LTE và 5G NR, được thiết kế để truyền các thông tin điều khiển ngắn gọn (**UCI**) từ thiết bị đầu cuối (UE) về trạm gốc (gNB/eNB). Thông tin này thường bao gồm các chỉ số phản hồi (Hybrid Automatic Repeat Request Acknowledgment - **HARQ-ACK**), báo hiệu yêu cầu tài nguyên đường lên (Scheduling Request - **SR**) và các giá trị chỉ thị chất lượng kênh (Channel State Information - **CSI**).[3]

Khác với kênh PUSCH, vốn mang dữ liệu người dùng, PUCCH chỉ tập trung vào dữ liệu điều khiển nhằm đảm bảo kết nối hiệu quả và duy trì liên lạc ổn định. Tùy theo dung lượng thông tin và yêu cầu độ tin cậy, PUCCH được chuẩn hóa thành nhiều định dạng khác nhau, trong đó mỗi định dạng có cơ chế ánh xạ tài nguyên và phương pháp điều chế riêng.

Có 5 format để lựa chọn (0, 1, 2, 3, 4) tùy vào các đặc điểm:

- 0, 1 dùng để mang số lượng UCI bit <= 2.
- 0, 2 là short PUCCH vì nó chỉ dài 1-2 symbols.
- 1, 3, 4 là long PUCCH vì dài 4 -> 14 symbols.
- 0, 1, 2 được sử dụng nhiều hơn.

Báo cáo này tập trung nghiên cứu kênh PUCCH Format 0, với trọng tâm là các công thức mã hóa (encode) và giải mã (decode) tín hiệu.

Format 0:

- Loại: short PUCCH.
- Độ dài: 1-2 OFDM symbols.

- UE multiplexing: có, tối đa 12 UE.
- Dạng tín hiệu: dựa trên Zadoff-Chu sequence và cyclic shift.

3.2 Công thức liên quan.

3.2.1 Tính tương quan.

Tương quan giữa hai chuỗi tín hiệu là một phép đo mức độ giống nhau giữa chúng khi một chuỗi được dịch chuyển tương đối so với chuỗi còn lại. Trong xử lý tín hiệu, tương quan thường được sử dụng để phát hiện mẫu, trích xuất đặc trưng hoặc đo độ trễ giữa các tín hiệu.

Cho hai chuỗi tín hiệu rời rạc $x[n]$ và $y[n]$, hàm tương quan chéo (cross-correlation) được định nghĩa như sau:

$$R_{xy}[k] = \sum_{n=-\infty}^{\infty} x[n] \cdot y[n+k] \tag{3.1}$$

Trong thực tế, với tín hiệu hữu hạn chiều dài N, công thức được viết lại như sau:

$$R_{xy}[k] = \sum_{n=0}^{N-1} x[n] \cdot y[n+k] \tag{3.2}$$

Khi $x[n] = y[n]$, ta thu được hàm tương quan tự thân (auto-correlation):

$$R_{xx}[k] = \sum_{n=0}^{N-1} x[n] \cdot x[n+k] \tag{3.3}$$

Giá trị $R_{xy}[k]$ đạt cực đại khi hai chuỗi giống nhau nhất tại độ dịch k, từ đó có thể xác định độ trễ hoặc mức độ đồng bộ giữa hai tín hiệu.

3.3 Công thức điều chế kênh PUCCH format 0.

3.3.1 Chuỗi Low-PAPR (Low peak-to-average-power ratio).

Format 0 sử dụng phương trình chuỗi LPAPR loại 1, và cấu trúc của nó được minh họa trong Hình 14. Phương trình:

$$r_{u,v}^{\alpha,\delta}(n) = e^{j\alpha n}\bar{\mathrm{r}}_{u,v}(n), 0 \leq n < M_{ZC}$$

Chuỗi bao gồm hai thành phần: phần quay pha $e^{j\alpha n}$ và phần chuỗi gốc $\bar{\mathrm{r}}_{u,v}(n)$. Phần chuỗi gốc là chuỗi Zadoff-Chu. Thành phần quay pha được sử dụng để ghép kênh nhiều UE cùng chia sẻ các tài nguyên vật lý giống nhau và mang dữ liệu UCI trong trường hợp format 0.

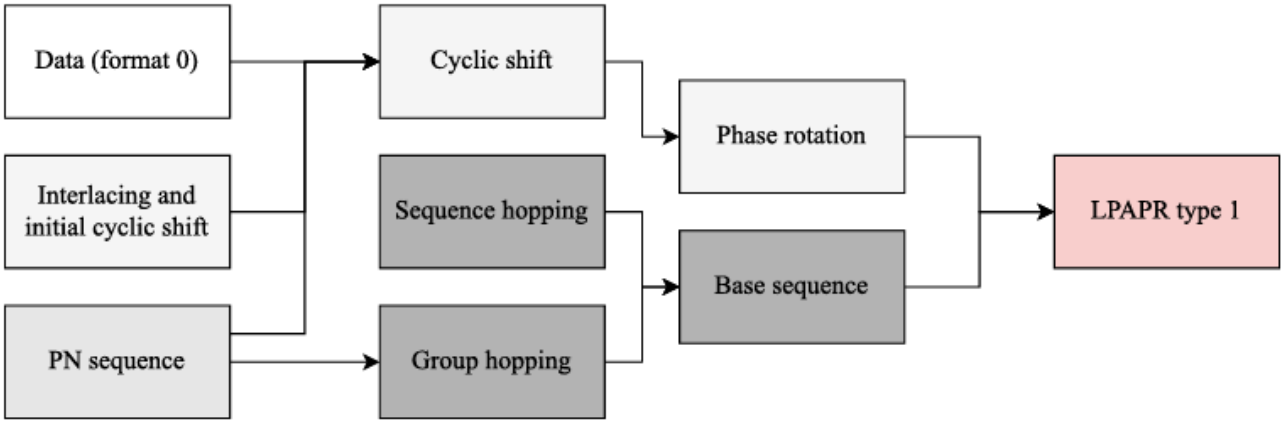


Figure 14. Low PAPR sequence type 1 processing chain.

Hình 3.1: Quá trình tạo chuỗi Low-PAPR.[3]

3.3.2 Độ dài chuỗi ZC: M_{ZC}

Được tính theo

$$M_{ZC} = mN_{sc}^{RB}/2^{\delta}$$

với các giá trị:

- m là số RB được sử dụng, ở đây format 0 thì $m = 1$.
- N_{sc}^{RB}: số subcarrier trên 1 resource block, ở đây là 12 (chiều rộng tần số là 12).
- $\delta = 0$.

Bằng việc có được độ dài chuỗi, ta tìm chuỗi ZC.

3.3.3 Chuỗi cơ sở $\bar{r}_{u,v}(n)$

$$\bar{r}_{u,v}(n) = e^{j\phi(n)\pi/4}, 0 \leq n \leq M_{ZC} - 1$$

Với việc đối chiếu giá trị u với bảng giá trị đã được tính toán trước của chuỗi ZC 3.2, ta tìm được chuỗi $\phi(n)$ với $M_{ZC} = 12$. Kết quả của chuỗi thu được sau phương trình 3.3.3 sẽ là 1 chuỗi phức gồm M_{ZC} phần tử.

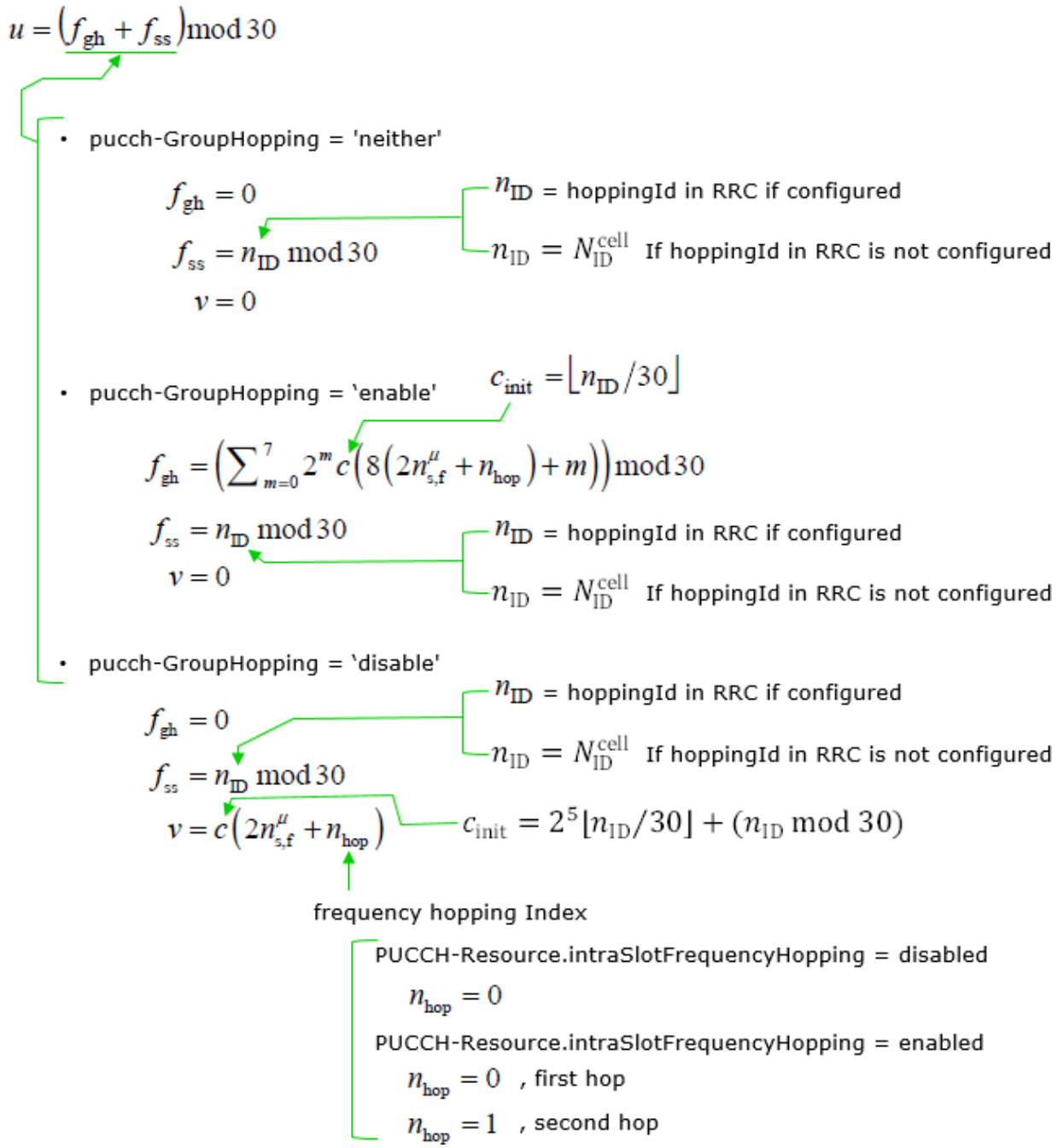
u	$\phi(0)$	$\phi(1)$	$\phi(2)$	$\phi(3)$	$\phi(4)$	$\phi(5)$	$\phi(6)$	$\phi(7)$	$\phi(8)$	$\phi(9)$	$\phi(10)$	$\phi(11)$
0	-3	1	-3	-3	-3	3	-3	-1	1	1	1	-3
1	-3	3	1	-3	1	3	-1	-1	1	3	3	3
2	-3	3	3	1	-3	3	-1	1	3	-3	3	-3
3	-3	-3	-1	3	3	3	-3	3	-3	1	-1	-3
4	-3	-1	-1	1	3	1	1	-1	1	-1	-3	1
5	-3	-3	3	1	-3	-3	-3	-1	3	-1	1	3
6	1	-1	3	-1	-1	-1	-3	-1	1	1	1	-3
7	-1	-3	3	-1	-3	-3	-3	-1	1	-1	1	-3
8	-3	-1	3	1	-3	-1	-3	3	1	3	3	1
9	-3	-1	-1	-3	-3	-1	-3	3	1	3	-1	-3
10	-3	3	-3	3	3	-3	-1	-1	3	3	1	-3
11	-3	-1	-3	-1	-1	-3	3	3	-1	-1	1	-3
12	-3	-1	3	-3	-3	-1	-3	1	-1	-3	3	3
13	-3	1	-1	-1	3	3	-3	-1	-1	-3	-1	-3
14	1	3	-3	1	3	3	3	1	-1	1	-1	3
15	-3	1	3	-1	-1	-3	-3	-1	-1	3	1	-3
16	-1	-1	-1	-1	1	-3	-1	3	3	-1	-3	1
17	-1	1	1	-1	1	3	3	-1	-1	-3	1	-3
18	-3	1	3	3	-1	-1	-3	3	3	-3	3	-3
19	-3	-3	3	-3	-1	3	3	3	-1	-3	1	-3
20	3	1	3	1	3	-3	-1	1	3	1	-1	-3
21	-3	3	1	3	-3	1	1	1	1	3	-3	3
22	-3	3	3	3	-1	-3	-3	-1	-3	1	3	-3
23	3	-1	-3	3	-3	-1	3	3	3	-3	-1	-3
24	-3	-1	1	-3	1	3	3	3	-1	-3	3	3
25	-3	3	1	-1	3	3	-3	1	-1	1	-1	1
26	-1	1	3	-3	1	-1	1	-1	-1	-3	1	-1
27	-3	-3	3	3	3	-3	-1	1	-3	3	1	-3
28	1	-1	3	1	1	-1	-1	-1	1	3	-3	1
29	-3	3	-3	3	-3	-3	3	-1	-1	1	3	-3

Hình 3.2: Bảng giá trị cho $\phi(n)$ với $M_{ZC} = 12$ [4]

3.3.4 Tham số u

$$u = (f_{gh} + f_{ss}) \bmod 30$$

Với tham số *pucch-grouphopping* khác nhau, các giá trị của f_{gh} và f_{ss} khác nhau



Hình 3.3: Công thức giá trị của *u* ứng với giá trị *GroupHopping* khác nhau. [2]

- n_{ID}: như trên, là tham số được cung cấp bởi layer cao hơn.
- $n_{s,f}^{\mu}$: định nghĩa bởi slot number trong radio frame hiện tại với giá trị μ được cấu hình.
- n_{hop} = 0 hoặc 1, dựa vào frequency hopping config.
- $c(n) = (x_1(n + N_C) + x_2(n + N_C)) \bmod 2$ dùng để tạo chuỗi giả ngẫu nhiên từ 2 chuỗi x1, x2. Đọc thêm ở [5].

- $N_C = 1600$: gọi là "warm-up phase"– nghĩa là 1600 bit đầu tiên được bỏ qua để tránh tính tuần hoàn ban đầu và đảm bảo tính ngẫu nhiên.
- $x_1(n+31) = (x_1(n+3) + x_1(n)) \bmod 2$ với $x_1(0) = 1$ và $x_1(k) = 0$ với k từ 1 đến 30.
- $x_2(n+31) = (x_2(n+3) + x_2(n+2) + x_2(n+1) + x_2(n)) \bmod 2$ với 30 phần tử đầu được khởi tạo với giá trị nhị phân là biểu diễn của thành phần c_{init} dưới dạng số nhị phân 31 bit với

$$c_{\text{init}} = \left\lfloor \frac{n_{\text{ID}}}{30} \right\rfloor$$

3.3.5 Giá trị nhảy dịch vòng α

Là nơi ghi giá trị data được truyền, biểu thị độ xoay pha.

$$\alpha = \frac{2\pi}{N_{\text{sc}}^{\text{RB}}} \cdot ((m_0 + m_{\text{cs}} + m_{\text{int}} + n_{\text{cs}}(n_{s,f}^{\mu},\ l + l')) \bmod N_{\text{sc}}^{\text{RB}})$$

Phương trình chia 2π ra làm 12 góc quay, khi được truyền vào giá trị $e^{j\alpha n}$, sẽ có 12 giá trị đầu ra.

- m_0: lấy từ chỉ số quay ban đầu (Initial cyclic shift) từ lớp RRC hoặc mặc định.
- $m_{int} = 5n_{IRB}^{\mu}$ nếu PUCCH sử dụng ánh xạ interlaced theo tham số *userInterlacePUCCH-PUSCH*, trong đó n_{IRB}^{μ} là số resource block trong interlace.
- m_{cs}: Được xác định theo giá trị của các UCI bit (bao gồm ACK và SR):

HARQ Value	0	1	0, 0	0, 1	1, 1	1, 0
Negative SR	0	6	0	3	6	9
Positive SR	3	9	1	4	7	10

Bảng 3.1: Giá trị m_{cs} với HARQ, SR thay đổi

- $n_{\text{cs}}(n_{s,f}^{\mu}, l) = \sum_{m=0}^{7} 2^m\, c\left(8\, n_{s,f}^{\mu} + 8l + m\right)$
  - $n_{\text{cs}}(n_{s,f}^{\mu}, l)$:
    - Chỉ số cyclic shift (độ dịch chuyển vòng) dùng để xác định góc quay pha trong tín hiệu PUCCH Format 0.
    - Giá trị phụ thuộc vào slot và symbol hiện tại, đảm bảo pha quay thay đổi theo thời gian.
  - $n_{s,f}^{\mu}$:
    - Chỉ số slot trong khung truyền, với μ là tham số định nghĩa loại subcarrier spacing (ví dụ $\mu = 0$ cho 15 kHz, $\mu = 1$ cho 30 kHz,...).
    - Biểu thị slot số n_s trong frame f, tính theo tham số μ.
  - l:

    - Chỉ số của OFDM symbol trong slot.
    - Kết hợp với slot giúp xác định chính xác thời điểm trong khung truyền.
  - m:
    - Biến chạy từ 0 đến 7, đại diện cho 8 bit liên tiếp được sử dụng để tạo ra n_{cs}.
    - Tổng 8 bit được ghép lại thành một số nguyên từ 0 đến 255.
  - $c(\cdot)$:
    - Phần tử thứ $\left(8n_{s,f}^{\mu} + 8l + m\right)$ trong chuỗi bit pseudo-random.
    - Chuỗi $c(\cdot)$ được tạo ra từ bộ tạo bit giả ngẫu nhiên (ví dụ Gold sequence) để đảm bảo tính ngẫu nhiên và giảm can nhiễu.
    - Mỗi bit $c(\cdot)$ có giá trị 0 hoặc 1.
  - Ý nghĩa tổng:

$$n_{\text{cs}}(n_{s,f}^{\mu}, l) = \sum_{m=0}^{7} 2^m \cdot c\left(8n_{s,f}^{\mu} + 8l + m\right)$$

    - Ghép 8 bit liên tiếp từ chuỗi $c(\cdot)$ thành một số nguyên từ 0 đến 255.
    - Số nguyên này được dùng làm chỉ số dịch vòng (cyclic shift) để thay đổi pha quay.
    - Giúp pha quay thay đổi theo thời gian, tránh giao thoa và tăng tính ngẫu nhiên của tín hiệu.

3.3.6 Công thức của format 0 được tổng hợp thành:

$$x\left(lM_{\text{RB}}^{\text{PUCCH},0}N_{\text{sc}}^{\text{RB}} + n\right) = r_{u,v}^{\alpha,\delta}(n) \tag{3.4}$$

$$n = 0, 1, \ldots, M_{\text{RB}}^{\text{PUCCH},0}N_{\text{sc}}^{\text{RB}} - 1$$

$$l = \begin{cases} 0 & \text{Chỉ dùng 1 symbol} \\ 0, 1 & \text{Dùng 2 symbol} \end{cases}$$

3.4 Quá trình phát hiện chuỗi (sequence detection) dựa trên tương quan để giải mã UCI của PUCCH format 0.

Quá trình giải mã UCI của PUCCH format 0 được thực hiện bằng so khớp tương quan chuẩn hóa:

- Tín hiệu sau OFDM được so sánh với tất cả các chuỗi tham chiếu (sinh ra từ các tổ hợp của ACK và SR).
- Tính giá trị tương quan trung bình trên các anten và chọn tổ hợp có giá trị cao nhất.
- Tổ hợp này được ánh xạ ngược lại thành các bit ACK/SR, với điều kiện giá trị tương quan vượt ngưỡng phát hiện.

3.4.1 Tương quan giữa 2 chuỗi.

$$R_{xy} = \frac{\left|\sum_{n=0}^{N-1} x[n] \cdot y^*[n]\right|}{\sqrt{\left(\sum_{n=0}^{N-1} |x[n]|^2\right) \cdot \left(\sum_{n=0}^{N-1} |y[n]|^2\right)}} \tag{3.5}$$

trong đó:

- $x[n]$: mẫu tín hiệu thu được sau giải điều chế OFDM.
- $y[n]$: mẫu chuỗi tham chiếu (reference sequence) ứng với một tổ hợp bit UCI giả định.
- $(\cdot)^*$: liên hợp phức (complex conjugate).
- N: số phần tử trong chuỗi (bao gồm toàn bộ subcarrier và symbol).
- $|\cdot|$: giá trị tuyệt đối (độ lớn) của số phức.

Giá trị R nằm trong khoảng $[0, 1]$, càng gần 1 thì mức độ giống nhau giữa hai chuỗi càng cao.

3.5 Triển khai trên MATLAB.

Em đã đọc code mẫu của MATLAB kết hợp với lý thuyết trên, em đã triển khai lại theo ý hiểu toàn bộ phần code để sinh chuỗi giả ngẫu nhiên (PRBS), tính alpha, và sinh ra chuỗi cơ sở với kết quả tương đương như hàm của MATLAB với cùng các tham số đầu vào. Code được đính kèm ở cuối báo cáo.

Code và các hàm liên quan được đăng trên github PUCCHF0_MATLAB – GitHub Repository.

Chương 4

Tìm hiểu kiến trúc AI Engine trên chip Versal.

Thời gian nghiên cứu: tuần 1, tuần 2, tuần 3.

4.1 Giới thiệu về AI Engine và Versal

Versal là dòng chip mới của AMD/Xilinx, thuộc nhóm ACAP (Adaptive Compute Acceleration Platform), kết hợp nhiều loại tài nguyên tính toán: CPU, FPGA, và đặc biệt là AI Engine (AIE). AI Engine là bộ xử lý vector hiệu năng cao, tối ưu cho các tác vụ xử lý tín hiệu số (DSP), học máy, và các phép toán ma trận.

Các điểm nổi bật của Versal và AI Engine:

- **Versal ACAP**: Tích hợp CPU ARM, logic lập trình (FPGA), và AI Engine, cho phép xử lý linh hoạt, hiệu năng cao, phù hợp nhiều ứng dụng từ mạng, xử lý ảnh, đến AI.
- **AI Engine**: Là tập hợp các tile xử lý song song, mỗi tile gồm bộ xử lý SIMD VLIW, bộ nhớ riêng, DMA, và giao tiếp streaming. AIE giúp tăng tốc các thuật toán tính toán nặng như nhân ma trận, FFT, inference AI.
- **Kiến trúc linh hoạt**: Versal cho phép kết nối giữa các khối chức năng qua mạng NoC, hỗ trợ truyền dữ liệu tốc độ cao, cấu hình động, và tối ưu hóa tài nguyên cho từng ứng dụng.

Nhờ sự kết hợp này, Versal và AI Engine mở ra khả năng tăng tốc phần cứng cho các ứng dụng hiện đại, đồng thời vẫn giữ được tính linh hoạt của FPGA truyền thống.

4.2 Versal Adaptive Compute Acceleration Platform (Versal ACAPs)

Versal ACAPs có **3 loại Engine** chính:

1. **Scalar Engines (CPU)**
   - Là các lõi ARM:
     - `Cortex-A72`: CPU hiệu năng cao, chạy Linux, xử lý phần mềm ứng dụng.

  - `Cortex-R5F`: CPU thời gian thực, dùng cho các tác vụ điều khiển (real-time).
- $\Rightarrow$ Dùng cho quản lý hệ thống, khởi động, điều phối AI Engine, xử lý điều kiện...

2. **Adaptable Engines (PL - Programmable Logic)**
   - Tương tự như FPGA truyền thống: logic block, LUT, DSP, BRAM.
   - Lập trình bằng Verilog/VHDL/HLS (High-Level Synthesis C++).
   - $\Rightarrow$ Dùng khi cần tuỳ biến thuật toán hoặc xử lý dữ liệu đặc thù, không cố định.
3. **Intelligent Engines (AI Engines – điểm nhấn của Versal)**
   - Là **các tile AI Engine** chứa:
     - SIMD VLIW processor: xử lý vector hiệu năng cao.
     - Bộ nhớ nội bộ (32KB), DMA.
     - Stream kết nối với tile khác.
   - Dùng để xử lý **tín hiệu (DSP), machine learning, nhân ma trận, nhận dạng, inference**...

4.3 AI Engine Array Features

AI Engine Array bao gồm `AIE tiles` và `AIE array interface` (gồm Network on Chip và PL).

AIE Tiles:

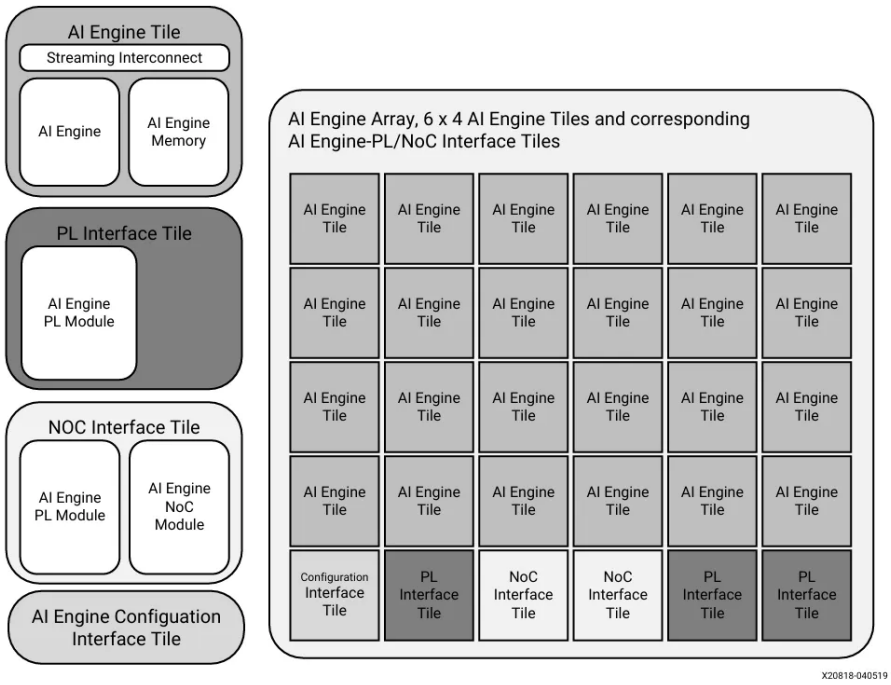
- Mỗi tile là một khối riêng biệt, nằm ngoài PL.
- Bao gồm một bộ xử lý VLIW SIMD tối ưu cho xử lý tín hiệu và học máy.
- Có 8 bank RAM đơn cổng, tổng cộng **32 KB/tile**.
  - Mỗi bank có thể truy cập **độc lập**, phù hợp với stream và pipeline.
- Hỗ trợ luồng dữ liệu tốc độ cao giữa AIE và PL.
- DMA riêng của tile cho phép:
  - Truyền dữ liệu từ incoming stream $\rightarrow$ local memory.
  - Truyền dữ liệu từ local memory $\rightarrow$ outgoing stream.
- CPU (như Cortex-A72) hoặc PL có thể cấu hình AIE tile thông qua bus **AXI4 MM**.
  - Ví dụ: load code, đặt tham số, kích hoạt kernel.
- Có cơ chế đồng bộ phần cứng (giữa 2 tile, giữa AIE với DMA, hoặc với CPU).

AIE Array Interface to NoC and PL Resources:

- DMA quản lý traffic dạng memory-mapped và streams vào/ra AIE array.
- Chức năng cấu hình và điều khiển kết nối (qua AXI4).
- Giao diện AIE đến PL cung cấp chuyển giao clock không đồng bộ giữa `AIE clk` và `PL clk`.
- AI Engine có giao tiếp logic tích hợp để **gửi và nhận dữ liệu** qua **NoC** – một mạng lưới kết nối nội bộ giữa các khối chức năng trên Versal.

4.4 Tổng quan về mảng AI Engine

Figure 2: **Hierarchy of Tiles in a AI Engine Array**



Hình 4.1: Cấu trúc mảng AIE

- **AIE Tiles (phần chính):**
  - Thực hiện tính toán.
  - Mỗi tile chứa: AIMD VLIW processor, 32KB RAM, DMA, streaming interface, AXI config.
  - Ứng dụng: nhân ma trận, xử lý tín hiệu ...
- **Interface Tiles:**

- **PL Interface Tiles**: kết nối với PL, là nơi bắt đầu hoặc kết thúc dòng dữ liệu đến AIE.
- **NoC Interface Tiles**: dùng để load dữ liệu vào, lưu kết quả ra DDR hoặc giao tiếp với hệ thống.

- **Configuration Interface Tile:**
  - Chứa PLL để sinh ra clock.
  - Có các thanh ghi cấu hình cho toàn hệ thống.

Note: Tất cả các tile AIE đều chạy trong 1 clock domain. tần số 1GHz, nguồn 0.7V.

4.5 Cấu trúc chi tiết AI Engine tile.

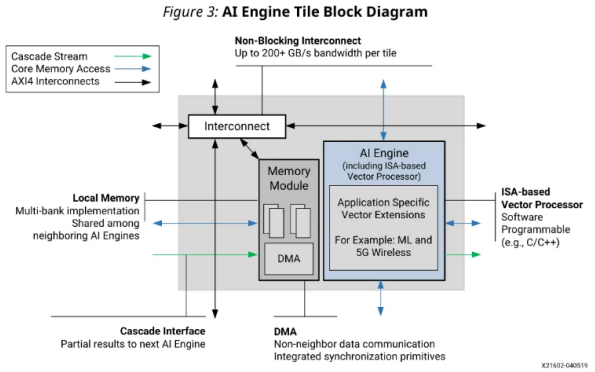


Figure 3: **AI Engine Tile Block Diagram**

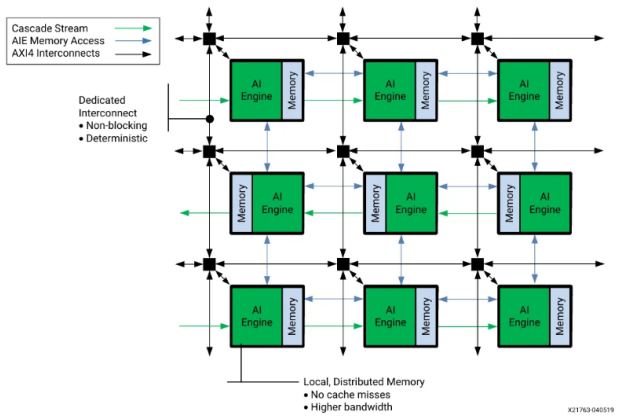


Figure 4: **AI Engine Array**

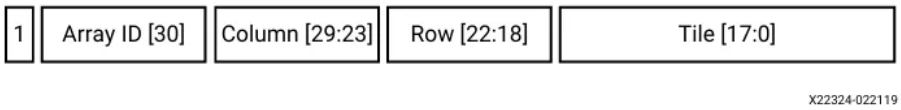
Hình 4.2: Cấu trúc chi tiết của AI Engine Tile

- Tile interconnect: quản lý dữ liệu chạy vào tile qua NoC, PL và được định tuyến tới RAM
- AIE Core: có thể lập trình bằng C/C++,... tối ưu cho tính toán song song, có thể cấu hình cho ứng dụng cụ thể, truyền kết quả trung gian tới tile kế tiếp
- Memory Module:
  - Bao gồm: **32KB RAM** chia thành 8 bank, DMA, cơ chế đồng bộ giữa các tile.
  - Khả năng truy cập chéo: mỗi tile có thể truy cập:
    * RAM của chính nó.
    * RAM của tile phía Bắc (North).
    * RAM của tile phía Nam (South).
    * RAM của tile phía Đông hoặc Tây (East/West) tùy vào vị trí dòng.
- Cascade interface: Các tile được kết nối theo hướng zigzag và hướng lên trên (như minh họa ở Hình 4)

4.6 Truy cập tài nguyên qua AXI4 MM

Master AXI4 kết nối vào NoC có thể: đọc ghi RAM của tile, cấu hình bộ DMA, viết lệnh vào thanh ghi control, debug lỗi,...

Figure 5: **AI Engine Memory-Mapped AXI4 Interface Addresses**



Hình 4.3: Địa chỉ giao diện AXI4

Cấu trúc địa chỉ:

- column: tối đa 128 cột
- row: tối đa 32 hàng
- tile: offset bên trong tile (để ghi vào thanh ghi, RAM, DMA,...)

Ví dụ: Viết từ CPU ARM: ghi 0x42 vào offset 0x100 của tile tại (row=3, col=10):

```
uint32_t* tile_base = (uint32_t*)( (3 << 18) | (10 << 23) | 0x100 );
*tile_base = 0x42;
```

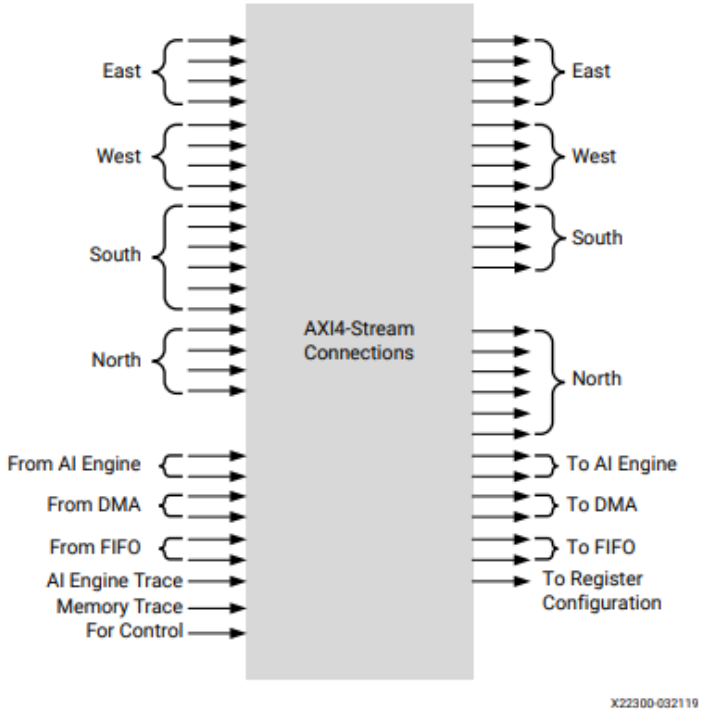
Listing 4.1: Code

4.7 AXI4-Stream Interconnect[6].

Mỗi AIE có 1 AXI4-stream interconnect giúp truyền dữ liệu giữa các tile, cấu hình tĩnh, thông qua giao diện bộ nhớ, có thể lập trình được.
Chức năng: quản lý backpressure (điều tiết dữ liệu khi phía nhận bị nghẽn), đảm bảo băng thông tối đa.
Cấu trúc thành phần:

- Port Handlers: Điều khiển luồng vào/ra tại mỗi cổng. Chọn tuyến đường cho dữ liệu.
- FIFOs: Hai buffer (bộ đệm) 16 hàng, mỗi hàng rộng 32-bit + 1-bit TLAST. Có thể xâu chuỗi (chaining) để tạo độ trễ hoặc buffer dữ liệu.
- Arbiters (bộ chọn): Sáu bộ chọn lập trình được. Dùng trong chế độ packet switching để phân xử khi nhiều luồng tranh tài nguyên.
- Stream Switch Configuration Registers: Các thanh ghi cấu hình dùng để thiết lập đường truyền (circuit hoặc packet-switched).

Figure 6: **AXI4-Stream Switch High-level Block Diagram**



Khối trung tâm chứa các thành phần đã được liệt kê ở trên.
Lối vào:

- 4 hướng dữ liệu từ các phía
- Form AIE: dữ liệu được tile này sinh ra
- From DMA: dữ liệu từ bộ DMA,
- From FIFO: dữ liệu từ bộ FIFO nội bộ
- Còn lại là dùng cho debug,...

Lối ra:

- 4 hướng dữ liệu đến các phía
- To AIE: dữ liệu được gửi đến tile này
- To DMA: dữ liệu chuyển tiếp đến nơi khác
- To FIFO: lưu dữ liệu vào FIFO nội bộ
- To Register Configuration – cấu hình tuyến hoặc điều khiển chuyển mạch

Table 2: **AI Engine AXI4-Stream Tile Interconnect Bandwidth**

Connection Type	Number of Connections	Data Width (bits)	Clock Domain	Bandwidth per Connection (GB/s)	Aggregate Bandwidth (GB/s)
To North/From South	6	32	AI Engine (1 GHz)	4	24
To South/From North	4	32	AI Engine (1 GHz)	4	16
To West/From East	4	32	AI Engine (1 GHz)	4	16
To East/From West	4	32	AI Engine (1 GHz)	4	16

Ví dụ: nếu tile này muốn gửi dữ liệu sang tile phía đông: dữ liệu từ From AIE đến đầu ra East.

4.8 AI Engine Tile Program Memory[6]

AIE có bộ nhớ 16KB để lưu câu lệnh VLIW. Có 2 giao diện để quản lý:

- Memory-mapped AXI4 interface: External master có thể đọc/ghi program memory thông qua Memory-mapped AXI4.
- AIE interface: AIE có giao diện rộng 128bit để nạp lệnh, chỉ có thể đọc mà không thể ghi đến program memory

Để **truy cập đồng thời cả AIE và AXI4**: Bộ nhớ chia thành nhiều bank, mỗi bên truy cập không trùng nhau. Nếu truye cập trùng, cần có arbitration logic để tránh xung đột, ưu tiên luồng nào trước.

4.9 AI Engine Interfaces

Đọc thêm tại trang 17, tài liệu [6].

Bảng 4.1: Tóm tắt nhanh các giao diện của AI Engine

Giao diện	Mục đích chính	Băng thông / Độ rộng
Data Memory	Đọc/ghi dữ liệu nội tile	2×256b load, 1×256b store
Program Memory	Fetch lệnh từ bộ nhớ	128b
AXI4-Stream (Direct)	Gửi/nhận dữ liệu qua các stream chuẩn	4×32b (2 in, 2 out)
Cascade Stream	Truyền accumulator giữa tile liền kề	384b
Debug	Truy cập thanh ghi, hỗ trợ debug	AXI4
Lock Interface	Đồng bộ hóa hoạt động giữa các tile	–
Stall Handling	Điều khiển tạm dừng do xung đột hoặc tranh chấp	–
Event Interface	Giao diện sinh/tạo sự kiện	16b
Tile Timer	Đếm thời gian nội bộ tile	64b

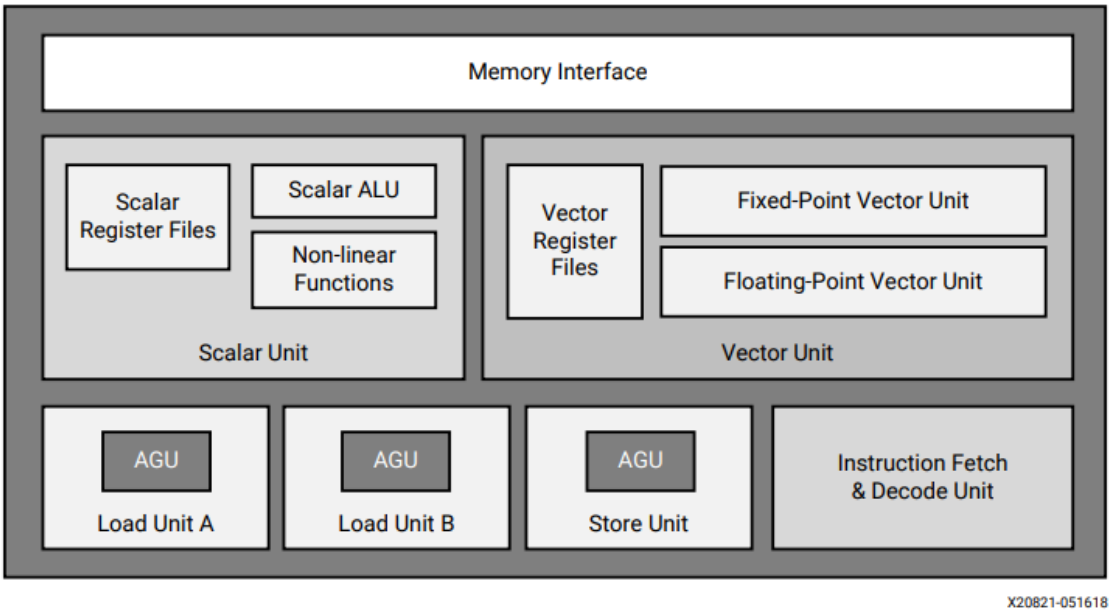
4.10 AIE Architechture

AIE là bộ xử lý được tối ưu cho:

- Single-instruction multiple-data: cho phép thực thi 1 lệnh trên nhiều dư liệu 1 lúc, giúp tăng tốc xử lý các vecto, ví dụ như nhiều phần tử trong ma trận cùng lúc.
- Very-long instruction word: cho phép đóng gói nhieeufleenhj vào 1 từ lệnh dài, giúp xử lý song song nhiều lệnh trên 1 chu kì.
- Hỗ trợ dấy phẩy động và dấu phẩy tĩnh.

Các chức năng của AIE:

Figure 21: **AI Engine**



- **Scalar unit**: Bộ xử lý RISC 32-bit dạng scalar
  - Có thanh ghi chung để lưu trữ con trỏ và thanh ghi cấu hình
  - Hỗ trợ các hàm toán học phi tuyến
  - Scalar ALU: Bộ xử lý logic số học đơn lẻ, có thể thực hiện nhân 2 số 32 bit scalar
  - Có khả năng chuyển đổi giữa số nguyên và phẩy động
- **AGU**: 3 bộ sinh địa chỉ để đọc/ghi dữ liệu từ/to bộ nhớ:
  - 2 AGU để load, 1 AGU để lưu
  - hỗ trợ nhiều chế độ địa chỉ: địa chỉ không đổi, địa chỉ từ thanh ghi,...
  - FFT Address Generation: Có phần cứng riêng để tạo địa chỉ đặc biệt phục vụ biến đổi Fourier nhanh (FFT).
- **Vector fixed-point/integer unit**:
  - 32 phép nhân 16-bit có thể được thực hiện mỗi chu kỳ trên một AI Engine tile.

– Hỗ trợ phép nhân tích lũy (MAC – Multiply and Accumulate) song song trên nhiều "lane" (đường dữ liệu song song).
– Hỗ trợ dữ liệu real và complex với nhiều độ rộng bit khác nhau (8, 16, 32). (Bảng dưới)

Table 6: **Supported Precision Width of the Vector Data Path**

X Operand	Z Operand	Output	Number of MACs
8 real	8 real	48 real	128
16 real	8 real	48 real	64
16 real	16 real	48 real	32
16 real	16 complex	48 complex	16
16 complex	16 real	48 complex	16
16 complex	16 complex	48 complex	8
16 real	32 real	48/80 real	16
16 real	32 complex	48/80 complex	8
16 complex	32 real	48/80 complex	8
16 complex	32 complex	48/80 complex	4
32 real	16 real	48/80 real	16
32 real	16 complex	48/80 complex	8
32 complex	16 real	48/80 complex	8
32 complex	16 complex	48/80 complex	4
32 real	32 real	80 real	8
32 real	32 complex	80 complex	4
32 complex	32 real	80 complex	4
32 complex	32 complex	80 complex	2
32 SPFP	32 SPFP	32 SPFP	8

– X, Z là 2 vector đầu vào, X từ thanh ghi 1024 bit và Z từ thanh ghi 256 bit. Do số lương phần tử của Z ít hơn, nên sẽ được sao chép (broadcast) đến nhiều lane để tận dụng tối đa X. Có 128 bộ nhân 8 bit, và kết qua được cộng, lưu vào accumulator.

- SPFP (Single-Precision Floating-Point) Vector Unit: Hỗ trợ dấu phẩy động đơn 32bit. Có thể xử lý song song trên nhiều lane, thực hiện 8 đơn vị MAC trên 1 chu kỳ(?)
- **3 cổng truy cập dữ liệu**: 2 đọc, 1 ghi, có thể hoạt động đồng thời, cần tối ưu layout để tránh xung đột bank, gây stall toàn bộ pipeline.
- **VLIW Function:** Hỗ trợ phát nhiều lệnh tới tất cả function unit, nhiều định dạng khác nhau, độ dài lệnh khác nhau. Tối đa 7 lệnh thực hiện song song trong 1 chu kì.

4.11 Code Bài tập nhân ma trận AI Engine

```
// matrix mul a[m][k] * b[k][n]
template <unsigned M, unsigned K, unsigned N>
void mmul_blocked(unsigned rowA, unsigned colA, unsigned colB,
                  const int16 * __restrict pA, const int16 * __restrict pB, int16 *
                      __restrict pC) // pA,pb,pc: con tro tro den cac matran
{
   using MMUL = aie::mmul<M, K, N, int16, int16>; // dinh nghia kieu MMUL voi kich
       thuoc M, K, N

   for (unsigned z = 0; z < rowA; z += 2) chess_loop_range(2,) // 2 dong 1 lan
       // tim den dia chi 0 (dau tien) cua 2 dong dang tinh trong C
       int16 * __restrict pC1 = pC + (      z * colB +       0) * MMUL::size_C;
       int16 * __restrict pC2 = pC + ((z + 1) * colB +       0) * MMUL::size_C;

       for (unsigned j = 0; j < colB; j += 2) chess_loop_range(2,) { // 2 cot 1 lan
           // MMUL::size_A: so luong phan tu trong 1 block ma phep nhan ma tran co
               the xu ly cung luc (< so voi kich thuoc cua A)
           const int16 * __restrict pA1 = pA + (      z * colA +       0) * MMUL::
               size_A; // Tinh con tro den vi tri bat dau tinh cua 1 block (1 chieu)
               can xu ly cua A
           const int16 * __restrict pA2 = pA + ((z + 1) * colA +       0) * MMUL::
               size_A; // same nhung la hang +1
           const int16 * __restrict pB1 = pB + (      0 * colB +       j) * MMUL::
               size_B; // same as A but cot
           const int16 * __restrict pB2 = pB + (      0 * colB + (j + 1)) * MMUL::
               size_B; // same as A but cot

           // load du lieu tu pa12, pb12 o tren, cung voi cap nhat vi tri ngay sau do
           aie::vector<int16, MMUL::size_A> A0 = aie::load_v<MMUL::size_A>(pA1); pA1
               += MMUL::size_A;  // load ca block voi do rong size_A vao A0, sau do
               dich pA1 bang 1 block
           aie::vector<int16, MMUL::size_A> A1 = aie::load_v<MMUL::size_A>(pA2); pA2
               += MMUL::size_A;  // same but hang duoi
           aie::vector<int16, MMUL::size_B> B0 = aie::load_v<MMUL::size_B>(pB1); pB1
               += MMUL::size_B * colB; // same but cot
           aie::vector<int16, MMUL::size_B> B1 = aie::load_v<MMUL::size_B>(pB2); pB2
               += MMUL::size_B * colB; // same but cot +1

           // nhan 2 vector 1 chieu => tao ra 1 vector ket qua bang 1 block
           MMUL C00; C00.mul(A0, B0);
           MMUL C01; C01.mul(A0, B1);
           MMUL C10; C10.mul(A1, B0);
           MMUL C11; C11.mul(A1, B1);

           for (unsigned i = 1; i < colA; ++i) chess_prepare_for_pipelining
               chess_loop_range(3,) {
               // tiep tuc load
               A0 = aie::load_v<MMUL::size_A>(pA1); pA1 += MMUL::size_A;
               A1 = aie::load_v<MMUL::size_A>(pA2); pA2 += MMUL::size_A;
               B0 = aie::load_v<MMUL::size_B>(pB1); pB1 += MMUL::size_B * colB;
               B1 = aie::load_v<MMUL::size_B>(pB2); pB2 += MMUL::size_B * colB;

               // nhan a0, b0 roi cong don vao c00
               C00.mac(A0, B0);
               C01.mac(A0, B1);
```

```
                C10.mac(A1, B0);
                C11.mac(A1, B1);
            }
            // Luu ket qua 4 khoi C00, C01, C10, C11 tu accumulator -> vector -> bo
                nho.
            aie::store_v(pC1, C00.template to_vector<int16>()); pC1 += MMUL::size_C;
            aie::store_v(pC1, C01.template to_vector<int16>()); pC1 += MMUL::size_C;
            aie::store_v(pC2, C10.template to_vector<int16>()); pC2 += MMUL::size_C;
            aie::store_v(pC2, C11.template to_vector<int16>()); pC2 += MMUL::size_C;
        }
    }
}
```

Listing 4.2: Code

tài liệu API chính thức của AMD [7], [8] bài viết giải thích từ Xilinx [9], và hướng dẫn phòng lab chi tiết [10].

Chương 5

Kết luận và hướng phát triển.

Qua báo cáo này, em đã hệ thống hóa các kiến thức nền tảng về 5GNR, kiến trúc AI Engine và các thuật toán then chốt trong quá trình mã hóa, giải mã chuỗi PUCCH format 0. Việc triển khai các công thức trên MATLAB đã giúp em hiểu sâu hơn về cách tạo chuỗi giả ngẫu nhiên, tính toán tham số và sinh chuỗi cơ sở, đồng thời kiểm chứng tính chính xác của từng bước thông qua mô phỏng. Tuy nhiên, để mô phỏng sát thực tế hơn, em cần bổ sung các bước truyền chuỗi qua lưới tài nguyên, kênh nhiễu, thực hiện giải mã tín hiệu nhận được nhằm trích xuất thông tin UCI và đánh giá hiệu quả giải điều chế. Đây sẽ là định hướng phát triển tiếp theo của em, đồng thời là tiền đề cho các nghiên cứu chuyên sâu hoặc đề tài tốt nghiệp trong tương lai. Em rất mong nhận được những góp ý quý báu từ cô để hoàn thiện hơn về nội dung và phương pháp nghiên cứu.

Encode PUCCH format 0

PUCCH format 0

Cấu hình carrier

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 60;
carrier.CyclicPrefix = 'normal';
carrier.NSlot = 57;
```

Cấu hình PUCCH

```
pucch = nrPUCCH0Config;
pucch.SymbolAllocation = [13 1];
pucch.FrequencyHopping = 'neither';
pucch.GroupHopping = 'disable';
pucch.HoppingID = 512;
pucch.InitialCyclicShift = 6;
```

Data

```
ack = [0; 0];
sr = [1];
uci = {ack,sr};
```

Bắt đầu

```
lenACK = length(uci{1});
lenSR = length(uci{2});
symStart = pucch.SymbolAllocation(1);
nPUCCHSym = pucch.SymbolAllocation(2);
```

Độ dài chuỗi....

```
m = 1;
nRBSC = 12;
mzc = m*12;
```

Tính chuỗi $\overline{r}_{u,v}(n) = e^{j\phi(n)\pi/4}$

- Tính u

fss

```
if isempty(pucch.HoppingID)
    nid = carrier.NCellID;
else
    nid = pucch.HoppingID;
```

```
end
nid
```

```
nid = 512
```

```
fss = mod(nid,30) ;
fss %print
```

```
fss = 2
```

fgh,v

```
fgh = zeros(1,2);
% fgh có 2 giá trị vì có thể sử dụng đến 2 symbol
v = zeros(1,2);
% slot number
nslot = mod(double(carrier.NSlot),carrier.SlotsPerFrame);
switch(pucch.GroupHopping)
    case 'neither'
        % fgh = 0; v = 0;
    case 'enable'
        cinit = floor(nid/30);
        fgh(1,:) = mod((2.^(0:7))*reshape(PRBSGen(cinit,[8*2*nslot 16]),8,[]),30);
        v = 0;
    case 'disable'
        cinit = 2.^5*floor(nid/30) + mod(nid,30);
        v(1,:) = double(PRBSGen(cinit,[2*nslot 2])');
end
fgh %print
```

```
fgh = 1×2
     0     0
```

```
v
```

```
v = 1×2
     0     1
```

u

```
u = mod(fgh+fss,30)
```

```
u = 1×2
     2     2
```

Chuỗi r

```
phi = returnPhiValue(u(1),mzc)
```

```
phi = 12×1
    -3
     3
     3
     1
    -3
```

```
     3
    -1
     1
     3
    -3
     ⋮
```

```
rSeq = exp(1i*phi*pi/4)
```

```
rSeq = 12×1 complex
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
     ⋮
```

Tính alpha $\alpha = \frac{2\pi}{N_{\text{sc}}^{\text{RB}}} \cdot ((m_0 + m_{\text{cs}} + m_{\text{int}} + n_{\text{cs}}(n_{s,f}^{\mu},\ l + l')) \bmod N_{\text{RB}}^{\text{sc}})$

- Sử dụng cyclic Prefix

```
if strcmpi(carrier.CyclicPrefix,'extended')
    nSlotSymb = 12;
else
    nSlotSymb = 14;
end
nSlotSymb
```

```
nSlotSymb = 14
```

```
nslot
```

```
nslot = 17
```

- Tìm ncs

```
% Giá trị ncs cho tất cả symbol
ncs = (2.^(0:7))*reshape(PRBSGen(nid,nSlotSymb*8*[nslot 1]),8,[])
```

```
ncs = 1×14
   187   136   142   110   130   124   103   254   185     2   198    63   246 ···
```

- Tìm mCS với giá trị của uci

```
cstable = returnMcs(lenACK);
```

```
if lenACK == 0
    mcs = cstable(1,1);
elseif (lenSR == 0) || (sr == 0)
    Uci = comm.internal.utilities.convertBit2Int(ack,lenACK);%% Chuyển chuỗi bit
ACK thành giá trị số nguyên có độ dài lenACK
    mcs = cstable(1,Uci+1);
else
    Uci = comm.internal.utilities.convertBit2Int(ack,lenACK);% same
    mcs = cstable(2,Uci+1);
end

mcs %print
```

```
mcs = 1
```

```
m0 = pucch.InitialCyclicShift;
mint = 0; % default = 0 với interlace = false;
alpha = 2*pi*mod(mint + mcs + m0 + ncs,nRBSC)/nRBSC
```

```
alpha = 1×14
    1.0472    5.7596    2.6180    4.7124    2.6180    5.7596    1.0472    4.7124 ···
```

Chỉ lấy giá trị alpha ứng với Symbol được dùng:

```
realAlpha = alpha(:,symStart+(1:nPUCCHSym))
```

```
realAlpha = 5.2360
```

Output r

```
nIndex = (0:mzc-1)';
lps = exp(1j*nIndex*realAlpha).*repmat(rSeq,1,length(realAlpha))
```

```
lps = 12×1 complex
  -0.7071 - 0.7071i
   0.2588 + 0.9659i
   0.9659 + 0.2588i
  -0.7071 - 0.7071i
   0.9659 - 0.2588i
  -0.9659 - 0.2588i
   0.7071 - 0.7071i
   0.9659 - 0.2588i
   0.9659 + 0.2588i
   0.7071 + 0.7071i
      ⋮
```

Nếu có 2 symbols và có freq hopping

```
if strcmp(pucch.FrequencyHopping,'intraSlot') && (nPUCCHSym ==  2)
    %lấy giá trị u(2) để tìm chuỗi r
    phi2 = returnPhiValue(u(2),mzc);
    rSeq2 =  exp(1i*phi2*pi/4);
    lps2 = exp(1j*nIndex*realAlpha(nPUCCHSym)).*repmat(rSeq2,1,1)
    seq = [lps(:,1);lps2]
else
```

```
    seq = lps(:);
end
seqf = encodePUCCH0dx(carrier,pucch,uci);
% so sánh với hàm có sẵn của MATLAB, có thể bỏ
seqm = nrPUCCH(carrier,pucch,uci);
a = (seqf == seqm)
```

```
a = 12×1 logical array
   1
   1
   1
   1
   1
   1
   1
   1
   1
   1
     ⋮
```

```
figure
% Create stem of seq
stem(seq,"DisplayName","seq");
```

```
Warning: Using only the real component of complex data.
```

```
hold on
% Create stem of seqf
stem(seqf,"DisplayName","seqf");
```

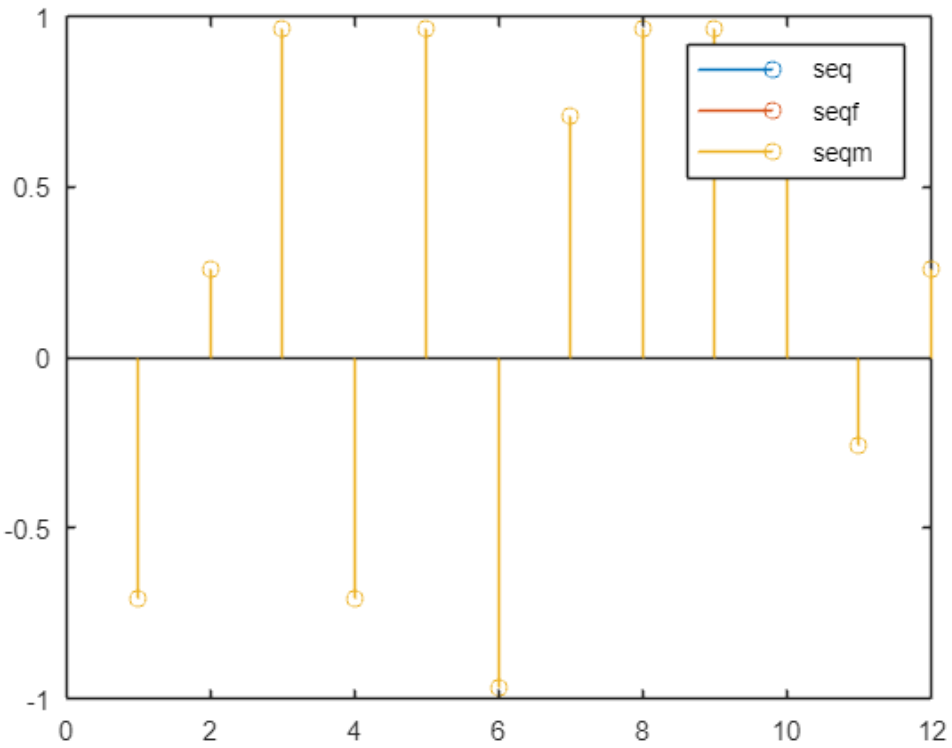
```
Warning: Using only the real component of complex data.
```

```
% Create stem of seqm
stem(seqm,"DisplayName","seqm");
```

```
Warning: Using only the real component of complex data.
```

```
hold off

legend
```



Transmission process....

```
%Đường truyền và giải điều chế OFDM, FFT
```

Demodulation process...

```
% Bao gồm giải điều chế OFDM và hàm nrExtractResources với các chỉ số PUCCH0
% (vị trí trong resource grid) để lấy giá trị của chuỗi.
% Giả sử rằng chúng ta có 5 anten để thu.

numAntenna = 5;
rx_sym = repmat(seq,1,numAntenna);
```

Decoding

Inputs

```
% thông tin về carrier và pucch biết trước
dc_carrier = carrier;
dc_pucch = pucch;
% số bit ack và sr
dc_numOfack = numel(ack)
```

```
dc_numOfack = 2
```

```
dc_numOfsr = numel(sr)
```

```
dc_numOfsr = 1
```

```
% tín hiệu nhận được sau demod
dc_sym = rx_sym;
%tính giá trị ngưỡng, được khuyến nghị bởi MATLAB
thres = 0.49 - 0.07*(pucch.SymbolAllocation(2)==2);
```

Kiểm tra giá trị numofSR

```
nPUCCHsym = pucch.SymbolAllocation(2);
seqlen = 12*nPUCCHsym; % 12 subcarriers.
```

num of sr

```
srOnly = 0;
if (dc_numOfack == 0) && dc_numOfsr
    % SR only transmission
    srOnly = 1; % Flag to indicate if there is only SR transmission
end
```

Main

khởi tạo các tổ hợp

```
allAcks = 2.^dc_numOfack;
allSR = 2.^dc_numOfsr;
c =zeros(allSR,allAcks);
```

năng lượng của sym

```
esym = sum(abs(dc_sym).^2);
```

tạo bảng để so sánh với tất cả trường hợp của uci

```
if ~srOnly
    ackreftable = [0 0 1 1; 0 1 0 1];
else
    ackreftable = false(1,0);
end
srreftable = [0; 1];
```

Kiểm tra các trường hợp

$$R = \frac{\left| \sum_{n=0}^{len} sym[n] * ref[n]^* \right|}{\sqrt{E_{sym} * E_{ref}}}$$

$$E = \sum_{n=0}^{len} |sym[n]^2|$$

```
for srIdx = 1:allSR
    for ackIdx = 1:allAcks
        % sử dụng encode để tìm chuỗi với gía trị ack và sr đang xét
        refsym = encodePUCCH0dx(carrier,pucch,
{ackreftable(:,ackIdx),srreftable(srIdx)});
        if ~isempty(ackreftable)
            mulrefsym = repmat(refsym,1,size(dc_sym,2));
            %power
            emulrefsym = sum(abs(mulrefsym).^2);
            norm = sqrt(emulrefsym.*esym);
            % lưu vào lưới giá trị
            c(srIdx,ackIdx) = mean(abs(sum(dc_sym.*conj(mulrefsym))))./(norm +
eps));% eps for 0 value case
        end
    end
end
```

Lấy giá trị UCI dựa trên max(c) > thres

```
maxval = max(c,[],'all');
if (maxval >= thres)
    [ridx, cidx] = find(c == repmat(maxval,allSR,allAcks));
    resAck = ackreftable(:,cidx);
    % Có truyền SR
    if dc_numOfsr
        resSr = (ridx == 2);
    else
        resSr = false(1,0);
    end
```

Không giá trị nào vượt ngưỡng

```
else
    resAck = false(0,1);
    resSr = false (srOnly,1);
end
ucival = {resAck resSr}
```

ucival = 1×2 cell

	1	2
1	[0;0]	1

```
uci
```

uci = 1×2 cell

	1	2
1	[0;0]	1

```
% thêm để so sánh với hàm của MATLAB
fun = decodePUCCH0dx(carrier,pucch,[lenACK lenSR],seq)
```

fun = 1×2 cell

	1	2
1	[0;0]	1

Tài liệu tham khảo

[1] ShareTechnote. *5G/NR — Waveform*. https://www.sharetechnote.com/html/5G/5G_Phy_Numerology.html. Truy cập ngày 25/7/2025.

[2] ShareTechnote. *5G/NR — Waveform*. https://www.sharetechnote.com/html/5G/5G_Waveform.html. Truy cập ngày 25/7/2025.

[3] Juho Kivijakola. "Implementation of 5G NR PUCCH Formats 0 and 1 on Arm Cortex-M4". Degree Programme in Electronics and Communications Engineering, 66 pages. Diploma thesis. Faculty of Information Technology **and** Electrical Engineering, University of Oulu, **may** 2024.

[4] 3GPP. *5G; NR; Physical channels and modulation (3GPP TS 38.211 version 17.4.0 Release 17)*. techreport. 3GPP, 2023.

[5] 3GPP. *NR; Physical channels and modulation*. techreport TS 38.211 V17.5.0. Section 5.2.1: Pseudo-random sequence generation. 3rd Generation Partnership Project (3GPP), **march** 2024. URL: https://www.etsi.org/deliver/etsi_ts/138200_138299/138211/17.05.00_60/ts_138211v170500p.pdf.

[6] AMD. *Versal Adaptive SoC AI Engine Architecture Manual*. AM009 v1.3, truy cập ngày 10/07/2025. 2023. URL: https://0x04.net/~mwk/xidocs/am/am009-versal-ai-engine.pdf.

[7] AMD. *AI Engine Kernel and Graph Programming Guide (UG1079)*. 2023. URL: https://docs.amd.com/r/en-US/ug1079-ai-engine-kernel-coding/Matrix-Multiplication?tocId=WRfMSn4WUW5h2B8YMVqO8Q.

[8] AMD Xilinx. *AI Engine API: Matrix Multiply ('aie::mmul') Documentation*. Accessed: July 2025. 2023. URL: https://download.amd.com/docnav/aiengine/xilinx2023_1/aiengine_api/aie_api/doc/group__group__mmul.html.

[9] Xilinx University Program. *Matrix Multiplication Explained – AI Engine Training*. Accessed: July 2025. 2023. URL: https://xilinx.github.io/xup_aie_training/matmult_explained.html.

[10] Xilinx University Program. *Matrix Multiplication Lab – AI Engine Training*. Accessed: July 2025. 2023. URL: https://xilinx.github.io/xup_aie_training/matmult_lab.html.