ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN – ĐIỆN TỬ

BỘ MÔN ĐIỆN TỬ

----------o0o----------

BÁO CÁO BÀI TẬP LỚN

MÔN LẬP TRÌNH HỆ THỐNG NHÚNG

TỔNG QUAN VỀ NGOẠI VI I/O PORT VÀ BỘ CHUYỂN ĐỔI TƯƠNG TỰ – SỐ (ADC)

GVHD: NGUYỄN PHAN HẢI PHÚ

NHÓM 2 - L01

TP. HỒ CHÍ MINH, THÁNG 12 NĂM 2025

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành đến thầy Nguyễn Phan Hải Phú vì đã tận tình hướng dẫn nhóm chúng em trong quá trình thực hiện đề tài "Tìm hiểu về I/O Port và ADC" của môn Lập trình hệ thống nhúng. Nhờ sự chỉ dẫn tận tâm, những kiến thức sâu rộng và các góp ý cụ thể của thầy, chúng em đã hiểu rõ hơn về cách hoạt động và lập trình các cổng vào/ra, cũng như nguyên lý chuyển đổi ADC trên vi điều khiển.

Sự hỗ trợ của thầy đã giúp nhóm hoàn thành bài tập lớn một cách chính xác, hiệu quả và tích lũy thêm nhiều kinh nghiệm hữu ích cho các môn học và công việc sau này. Chúng em xin chân thành cảm ơn và kính chúc thầy luôn mạnh khỏe, thành công trong sự nghiệp giảng dạy.

Tp. Hồ Chí Minh, tháng 12 năm 2025

Nhóm 2 – L01

Thành viên	**MSSV**	**Nhiệm vụ**	**Mức độ hoàn thành**
Trần Thái Bảo	2210279	Giới thiệu tổng quan lý thuyết và ví dụ minh họa các chương trình	40%
Vũ Thanh Danh	2110904	Tìm hiểu và mô tả các thanh ghi của I/O Port	20%
Trần Thanh Hiếu	2211014	Tìm hiểu và mô tả các thanh ghi của ADC	20%
Văn Đức Khánh	2211540	Mô tả các chế độ hoạt động của cả 2 ngoại vi	20%
Trần Nguyễn Duy Khoa	2211652		0%

PHẦN 1 :TỔNG QUAN VỀ NGOẠI VI I/O PORT VÀ BỘ CHUYỂN ĐỔI TƯƠNG TỰ – SỐ (ADC)

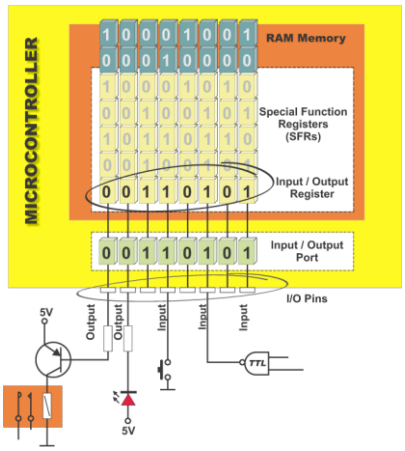
Trong các hệ thống vi điều khiển hiện đại, ngoại vi đóng vai trò đặc biệt quan trọng trong việc hỗ trợ vi điều khiển giao tiếp với thế giới bên ngoài. Mỗi loại ngoại vi đảm nhiệm một chức năng riêng biệt, từ việc nhận và truyền tín hiệu số đến chuyển đổi và xử lý các tín hiệu tương tự. Chương này trình bày tổng quan về hai nhóm ngoại vi phổ biến và quan trọng nhất: I/O Port và ADC, cùng với vai trò của chúng trong vi điều khiển RA6M5 của Renesas.

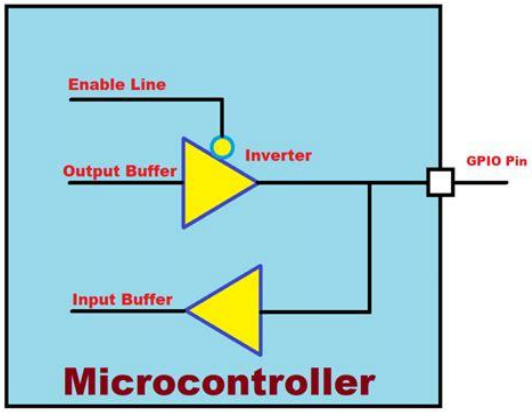
1. Tổng quan về I/O Port và ADC

1.1 I/O Port

I/O Port (Input/Output Port) là tập hợp các chân vào/ra được tích hợp sẵn trong vi điều khiển, cho phép chip có thể giao tiếp trực tiếp với các linh kiện ngoại vi như cảm biến, công tắc, LED, động cơ hoặc các module mở rộng. Mỗi chân I/O có thể được cấu hình làm ngõ vào hoặc ngõ ra tùy theo yêu cầu ứng dụng. Về bản chất, các chân I/O là các cổng số, chỉ làm việc với hai mức logic cơ bản là mức cao (1) và mức thấp (0), tương ứng với các trạng thái bật/tắt của phần cứng.

Trong vi điều khiển RA6M5, hệ thống I/O Port được thiết kế rất linh hoạt và mạnh mẽ, hỗ trợ nhiều chế độ hoạt động nhờ có khả năng gán chức năng cho từng chân thông qua bộ định tuyến tín hiệu (Pin Function Controller). Nhờ đó, một chân I/O không chỉ đóng vai trò là chân vào/ra thông thường, mà còn có thể được sử dụng cho các chức năng chuyên dụng như giao tiếp UART, SPI, I2C hoặc xuất xung PWM. Sự linh hoạt này giúp tối ưu hóa số lượng chân sử dụng và tăng tính đa dạng trong thiết kế hệ thống nhúng. Bên cạnh đó, RA6M5 còn hỗ trợ các đặc tính quan trọng như chống nhiễu, cấu hình điện trở kéo lên/kéo xuống, điều khiển tốc độ chuyển mức và bảo vệ quá dòng, giúp hệ thống hoạt động ổn định trong các môi trường có nhiều tác động nhiễu.

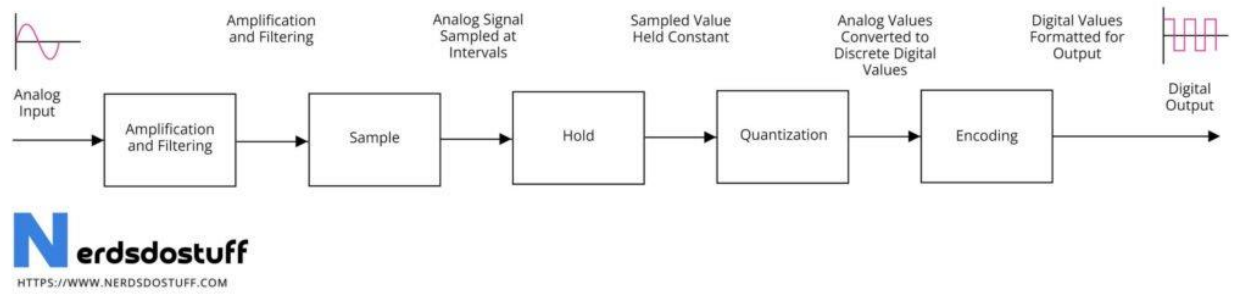




1.2. Ngoại vi ADC (Analog-to-Digital Converter)

ADC là khối ngoại vi có chức năng chuyển đổi tín hiệu tương tự (analog) thành tín hiệu số (digital) để vi điều khiển có thể xử lý bằng phần mềm. Trong nhiều ứng dụng nhúng, tín hiệu từ môi trường thực – chẳng hạn như nhiệt độ, ánh sáng, âm thanh, áp suất – thường tồn tại dưới dạng tương tự. Do đó, để có thể sử dụng được các tín hiệu này trong hệ thống tính toán số, ADC đóng vai trò như chiếc cầu nối quan trọng giữa thế giới vật lý và môi trường số hóa của vi điều khiển.

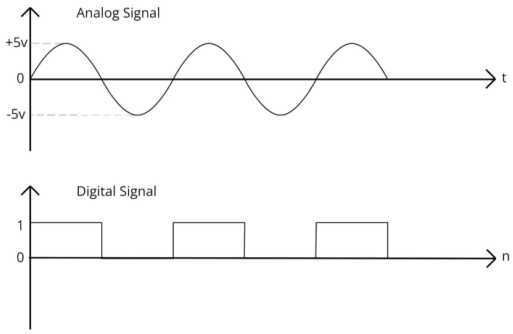
Nguyên lý hoạt động cơ bản của ADC dựa trên việc lấy mẫu tín hiệu analog và lượng tử hóa nó thành một giá trị nhị phân tương ứng. Độ phân giải của bộ ADC (thường là 12-bit hoặc 14-bit trong các vi điều khiển tiên tiến như RA6M5) quyết định độ chính xác của việc chuyển đổi: số bit càng lớn thì khả năng biểu diễn tín hiệu càng chi tiết. Vi điều khiển RA6M5 được trang bị ADC độ phân giải cao, hỗ trợ nhiều kênh chuyển đổi độc lập, cho phép đọc nhiều tín hiệu tương tự cùng lúc. Ngoài ra, RA6M5 còn tích hợp các chức năng nâng cao như kích hoạt ADC bằng sự kiện ngoại vi, bộ lọc, độ lợi đầu vào và khả năng chuyển đổi siêu nhanh, rất phù hợp cho các hệ thống yêu cầu đo lường chính xác và thời gian thực.



1.3. So sánh giữa I/O Port và ADC

Mặc dù đều thuộc nhóm ngoại vi của vi điều khiển, I/O Port và ADC có bản chất và mục đích sử dụng hoàn toàn khác nhau. I/O Port hoạt động thuần số, chỉ làm việc với hai mức logic và thường được dùng để giao tiếp với các thiết bị số hoặc thực hiện điều khiển trạng thái đơn giản. Trong khi đó, ADC lại xử lý tín hiệu tương tự – loại tín hiệu mang thông tin liên tục theo thời gian và biên độ. Vì vậy, ADC được dùng trong các bài toán đo lường, thu thập dữ liệu từ cảm biến analog hoặc theo dõi sự biến đổi của các đại lượng vật lý.

Ngoài sự khác biệt về bản chất tín hiệu, mức độ phức tạp giữa hai ngoại vi cũng khác nhau đáng kể. Một I/O Port thường chỉ cần cấu hình hướng vào/ra, mức kéo lên/kéo xuống và trạng thái logic. Ngược lại, ADC yêu cầu quy trình cấu hình phức tạp hơn như đặt chế độ lấy mẫu, chọn điện áp tham chiếu, lựa chọn kênh chuyển đổi, độ phân giải và phương thức kích hoạt. Tuy nhiên, hai ngoại vi này cũng bổ trợ cho nhau trong nhiều ứng dụng: một chân I/O có thể dùng để bật nguồn cảm biến, trong khi ADC đọc dữ liệu từ cảm biến đó. Sự kết hợp giữa I/O Port và ADC tạo nên khả năng tương tác linh hoạt giữa vi điều khiển và môi trường xung quanh, nâng cao hiệu quả của hệ thống nhúng.



2. Mô tả các thanh ghi của hai ngoại vi

2.1 I/O Port

- PCNTR1/PODR/PDR:

Bit	Tên	Chức năng
15:0	PDR15 to PDR00	Chiều dữ liệu chân Pmn (0: Đầu vào, 1: Đầu ra)
31:16	PODR15 to PODR00	Dữ liệu đầu ra chân Pmn (0: Đầu ra thấp, 1: Đầu ra cao)

Chức năng: điều khiển hướng cổng và dữ liệu đầu ra cổng.

Các bit PDRn: Các bit PDRn chọn hướng đầu vào hoặc đầu ra cho từng chân riêng lẻ trên cổng liên quan khi các chân được cấu hình là chân I/O chung.

Các bit PODRn: Các bit PODRn lưu giữ dữ liệu được xuất ra từ các chân I/O chung.

- PCNTR2/EIDR/PIDR:

Bit	Tên	Chức năng
15:0	PIDR15 đến PIDR00	Trạng thái chân Pmn (0: Mức thấp, 1: Mức cao)
31:16	EIDR15 đến EIDR00	Dữ liệu đầu vào sự kiện cổng khi một tín hiệu ELC_PORTx diễn ra (0: Mức thấp, 1: Mức cao)

Chức năng: cho phép truy cập đọc trạng thái chân Pmn và dữ liệu đầu vào sự kiện cổng.

Các bit PIDRn: ác bit PIDRn phản ánh trạng thái chân riêng lẻ của cổng, bất kể các giá trị được đặt trong PmnPFS.PMR và PORTm.PCNTR1.PDRn.

Các bit EIDRn: Các bit EIDRn chốt trạng thái chân khi một tín hiệu ELC_PORTx xảy ra.

- PCNTR3/PORR/POSR:

Bit	Tên	Chức năng
15:0	POSR15 to POSR00	Thiết lập đầu ra chân Pmn (0: Không có đầu ra, 1: Đầu ra ở mức cao)
31:16	PORR15 to PORR00	Đặt lại đầu ra chân Pmn (0: Không có đầu ra, 1: Đầu ra ở mức thấp)

Chức năng: điều khiển việc thiết lập hoặc đặt lại dữ liệu đầu ra cổng.

Các bit POSRn: Các bit POSRn thay đổi PODR khi được thiết lập bằng thao tác ghi của phần mềm.

Các bit PORRn: Các bit PORRn thay đổi PODR khi được đặt lại bằng thao tác ghi của phần mềm.

- PCNTR4/EORR/EOSR:

Bit	Tên	Chức năng
15:0	EOSR15 to EOSR00	Thiết lập sự kiện đầu ra chân Pmn (0: Không có đầu ra, 1: Đầu ra ở mức cao)
31:16	EORR15 to EORR0	Đặt lại sự kiện đầu ra chân Pmn (0: Không có đầu ra, 1: Đầu ra ở mức thấp)

Chức năng: điều khiển việc thiết lập hoặc đặt lại dữ liệu đầu ra cổng bằng một đầu vào sự kiện từ ELC

Các bit EOSRn: Các bit EOSRn thay đổi PODR thành 1 khi được thiết lập do một tín hiệu ELC_PORTx xảy ra.

Các bit EORRn: Các bit EORRn thay đổi PODR thành 0 khi được đặt lại do một tín hiệu ELC_PORTx xảy ra.

- PmnPFS/PmnPFS_HA/PmnPFS_BY:

Bit	Tên	Chức năng
0	PODR	Dữ liệu đầu ra cổng (0: Đầu ra ở mức thấp, 1: Đầu ra ở mức cao)
1	PIDR	Trạng thái chân Pmn (0: Mức thấp, 1: Mức cao)
2	PDR	Chiều dữ liệu của cổng (0: Đầu vào, 1: Đầu ra)
4	PCR	Điều khiển kéo lên (0: Không thiết lập kéo lên đầu vào, 1: Thiết lập kéo lên đầu vào)
6	NCODR	Điều khiển Kênh N máng mở (0: Đầu ra CMOS, 1: Đầu ra máng mở NMOS)
11:10	DSCR[1:0]	Khả năng điều khiển cổng (0 0: Điều khiển thấp 0 1: Điều khiển trung 1 0: Điều khiển tăng cường tốc độ cao 1 1: Điều khiển tăng cường)
13:12	EOFR[1:0]	Sự kiện trên cạnh lên/ cạnh xuống (0 0: Không quan tâm 0 1: Phát hiện cạnh lên 1 0: Phát hiện cạnh xuống 1 1: Phát hiện cả hai cạnh)
14	ISEL	Thiết lập đầu vào IRQ

		(0: Không thiết lập làm đầu vào IRQ, 1: Thiết lập làm đầu vào IRQ)
15	ASEL	Thiết lập đầu vào analog (0: Không thiết lập làm chân analog, 1: Thiết lập làm chân analog)
16	PMR	Điều khiển chế độ cổng (0: Thiết lập làm chân I/O chung, 1: Thiết lập làm cổng I/O cho các chức năng ngoại vi)
28:24	PSEL[4:0]	Chọn ngoại vi

Chức năng: điều khiển đọc/ghi dùng để chọn chức năng của chân cổng mn

Các bit PODR, PIDR, PDR: có chức năng tương tự như các bit trong thanh ghi PCNTR

Bit PCR: bật hoặc tắt điện trở kéo lên đầu vào trên từng chân cổng riêng lẻ.

Bit NCODR: chỉ định loại đầu ra cho các chân cổng.

Các bit DSCR[1:0]: chuyển đổi khả năng điều khiển của cổng. Nếu khả năng điều khiển của một chân là cố định, bit liên quan là bit đọc/ghi, nhưng khả năng điều khiển không thể thay đổi.

Các bit EOFR[1:0]: chọn phương pháp phát hiện cạnh cho tín hiệu đầu vào nhóm cổng. Các bit này hỗ trợ phát hiện cạnh lên, cạnh xuống hoặc cả hai. Khi các bit EOFR[1:0] được đặt thành 01b, 10b, hoặc 11b, đầu vào I/O cell được khẳng định.

Bit ISEL: chỉ định các chân đầu vào IRQ. Cài đặt này có thể được sử dụng kết hợp với các chức năng ngoại vi, nhưng ngắt IRQn cùng số chỉ có thể được bật cho một chân duy nhất.

Bit ASEL: chỉ định các chân analog.

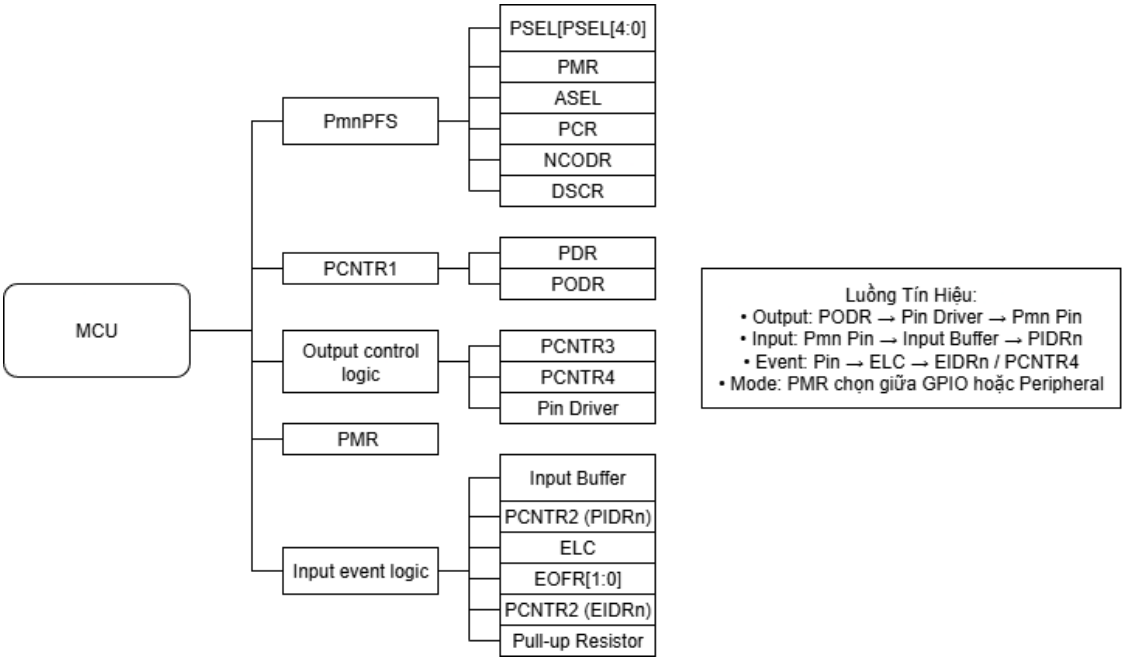
Bit PMR: chỉ định chức năng của chân cổng.

Các bit PSEL[4:0]: gán chức năng ngoại vi cụ thể.

- PWPR:

Bit	Tên	Chức năng
6	PFSWE	Cho phép ghi thanh ghi PmnPFS (0: Không cho phép ghi thanh ghi PmnPFS, 1: Cho phép ghi thanh ghi PmnPFS)
7	B0WI	Không cho phép ghi bit PFSWE (0: Cho phép ghi bit PmnPFS, 1: Không cho phép ghi bit PmnPFS)

- Sơ đồ cổng I/O



2.2 Ngoại vi ADC

** Các thanh ghi chính của ADC*

- ADCSR – A/D Control Register

Thanh ghi điều khiển chính của ADC.

Bit	Tên	Chức năng
15	ADST	Bit khởi động/dừng chuyển đổi (1: bắt đầu; 0: dừng).
14	TRGE	Cho phép kích hoạt bằng trigger ngoài (1: enable).
13	EXTRG	Chọn loại trigger ngoài (0: sync từ ELC, 1: async từ chân ADTRG).
8–9	ADCS	Chọn chế độ quét (00: Single, 01: Continuous, 10: Group scan).
6	GBADIE	Cho phép ngắt khi kết thúc quét nhóm B.
0	DBLE	Bật chế độ double-trigger.

Chức năng: điều khiển hoạt động tổng thể của ADC, chọn chế độ quét, bật trigger và bắt đầu chuyển đổi.

- ADANSAx / ADANSBx – A/D Channel Select Registers

Các thanh ghi chọn kênh cần chuyển đổi:

- ADANSA0, ADANSA1: chọn kênh cho group A (hoặc single/continuous).
- ADANSB0, ADANSB1: chọn kênh cho group B.

Mỗi bit tương ứng với một kênh analog (AN0, AN1,…). Ví dụ: ADANSA0 = 0x0005 → quét kênh AN0 và AN2.

Chức năng: xác định kênh đầu vào tham gia quá trình chuyển đổi.

- ADSTRGR – A/D Conversion Start Trigger Select Register

Chọn nguồn kích hoạt cho quá trình chuyển đổi.

Bit	Tên	Chức năng
5–0	TRSA[5:0]	Chọn trigger cho group A / chế độ single/continuous.
13–8	TRSB[5:0]	Chọn trigger cho group B.

Các giá trị phổ biến:

- 0x00: Software trigger (dùng ADCSR.ADST).
- 0x01: Kích hoạt bởi ELC (Event Link Controller).
- 0x15: Trigger từ chân ADTRG0 (ngoại vi).

Chức năng: quyết định nguồn bắt đầu chuyển đổi (phần mềm, ELC, hoặc chân ngoài).

- ADDRn – A/D Data Register n

Mỗi kênh analog có một thanh ghi ADDRn (n = 0–23).

- Lưu kết quả 12-bit sau khi chuyển đổi.
- Dữ liệu có thể right-justified hoặc left-justified (chọn trong ADCER.ADRFMT).
- Đọc bằng halfword (16-bit), không đọc byte.

Chức năng: chứa kết quả chuyển đổi tương ứng với kênh đã chọn.

- ADADC – A/D Addition/Average Count Select Register

Chọn số lần cộng hoặc trung bình kết quả:

Bit	Tên	Chức năng
2–0	ADC[2:0]	000: 1 lần, 001: 2 lần, 010: 3 lần, 011: 4 lần, 111: 16 lần.
7	AVEE	1: Lấy trung bình; 0: Cộng dồn.

Chức năng: dùng để giảm nhiễu bằng cách cộng hoặc trung bình nhiều mẫu.

- ADCER – A/D Control Extended Register

Thiết lập độ phân giải và định dạng dữ liệu:

Bit	Tên	Chức năng
1–0	ADPRC	Chọn độ phân giải: 00=12-bit, 01=10-bit, 10=8-bit.
15	ADRFMT	0: Right-justified, 1: Left-justified.

Chức năng: điều chỉnh định dạng và độ chính xác dữ liệu ADC.

**** Cách cấu hình và đọc dữ liệu ADC***

Bước 1: Bật nguồn và giải phóng module-stop

Đảm bảo module ADC được cấp clock PCLKA và thoát khỏi trạng thái dừng.

Bước 2: Dừng ADC trước khi cấu hình

ADCSR.ADST = 0 để đảm bảo an toàn khi thay đổi các thanh ghi.

Bước 3: Chọn kênh cần đo

Ghi bit tương ứng vào ADANSA0 hoặc ADANSB0.

Bước 4: Chọn độ phân giải và định dạng dữ liệu

Cấu hình trong ADCER:

ADCER = 0x0000; // 12-bit, right-justified

Bước 5: Chọn chế độ quét và trigger

Ví dụ dùng software trigger (single scan):

ADCSR = 0x8000; // ADST=1 để bắt đầu chuyển đổi

Bước 6: Chờ quá trình chuyển đổi hoàn tất

Có thể dùng polling hoặc ngắt:

while (ADCSR & 0x8000); // chờ ADST=0 (kết thúc)

Bước 7: Đọc dữ liệu kết quả

uint16_t result = ADDR0; // đọc giá trị từ kênh AN0

**** Quy trình chuyển đổi ADC (Start → Sample → Convert → Store)***

Giai đoạn	Mô tả chi tiết
1. Start	Quá trình bắt đầu khi ADCSR.ADST = 1 (hoặc nhận trigger ngoài từ ELC/ADTRG).
2. Sample	Mạch lấy mẫu của ADC đóng, điện áp đầu vào được nạp vào tụ lưu mẫu trong khoảng thời gian xác định bởi clock ADCLK.
3. Convert	Bộ SAR (Successive Approximation Register) thực hiện so sánh tuần tự để tạo ra giá trị số tương ứng với điện áp đầu vào (12/10/8 bit).
4. Store	Kết quả được lưu vào thanh ghi ADDRn tương ứng. Nếu chế độ cộng/trung bình được bật, dữ liệu được xử lý trước khi lưu.
5. End/Interrupt	Khi chuyển đổi hoàn tất, cờ trạng thái được đặt; nếu ngắt được bật, CPU nhận interrupt "scan end".

3. Các chế độ hoạt động của hai ngoại vi

3.1 I/O Port

+ Input:

MCU đọc giá trị đầu vào từ GPIO. GPIO lúc này có trở kháng vào rất cao.

Giá trị 0 là không có điện áp vào, 1 là có điện áp vào.

Có thể sử dụng thêm khối CMOS pull-up nếu cần dữ liệu chính xác

+ Output:

MCU xuất logic ra GPIO để điều khiển thiết bị ngoài.

MCU kéo GPIO xuống GND/0V để xuất mức 0. Mức 1 thường tương đương điện áp thiết kế (3.3V cho ARM)

+ Open-drain:

Biến thể từ output, MCU chỉ có thể kéo xuống chứ không thể kéo lên.

Lúc này, MCU sẽ sử dụng khối N-MOS để thực hiện. N-MOS sẽ kéo tín hiệu xuống khi MCU xuất mức 0, và sẽ ngắt khi MCU xuất mức 1

+ Pull-up:

1 khối CMOS đóng vao trò như điện trở.

Khối CMOS giữ input luôn ở mức 1 nếu không có tín hiệu.

Có 1 trường hợp ngược là Pull-down, luôn giữ input ở mức 0 nếu không có tín hiệu

3.2 Ngoại vi ADC

+ Single Scan (Quét 1 lần):

Kênh ADC lấy mẫu 1 lần, xử lý và lưu vào thanh ghi dữ liệu ADC.

+ Continuous Scan (Quét liên tục) :

ADC sẽ lấy mẫu liên tục, xử lý và nạp giá trị mới vào thanh ghi dữ liệu ADC.

+ Group Scan (Quét theo nhóm) :

ADC sẽ xử lý 1 chuỗi kênh đã được cấu hình sẵn theo chế độ single hoặc continuous

+ Trigger (kích thích) :

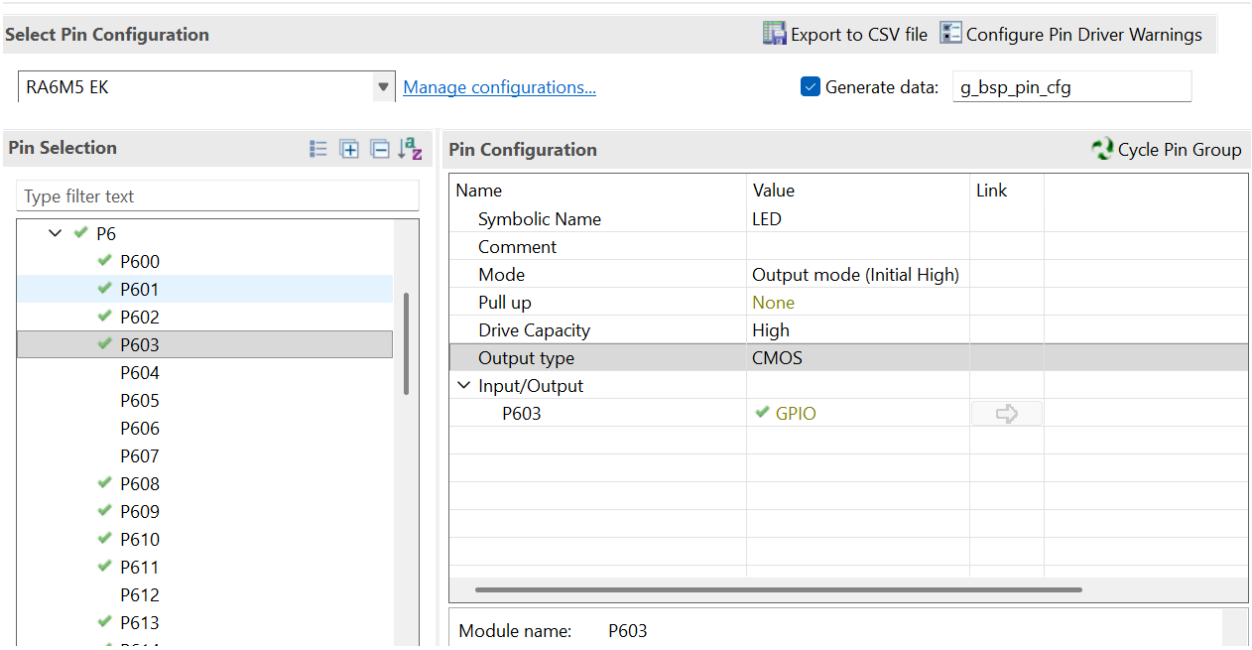
ADC chỉ chạy khi có tín hiệu phần cứng (nút nhấn, cạnh lên/xuống,..).

4 . Các chương trình ví dụ về hai ngoại vi

4.1 Chương trình sử dụng FSP

- GPIO

Chương trình nhấp nháy LED với ngõ ra là chân P603 được cấu hình như sau :

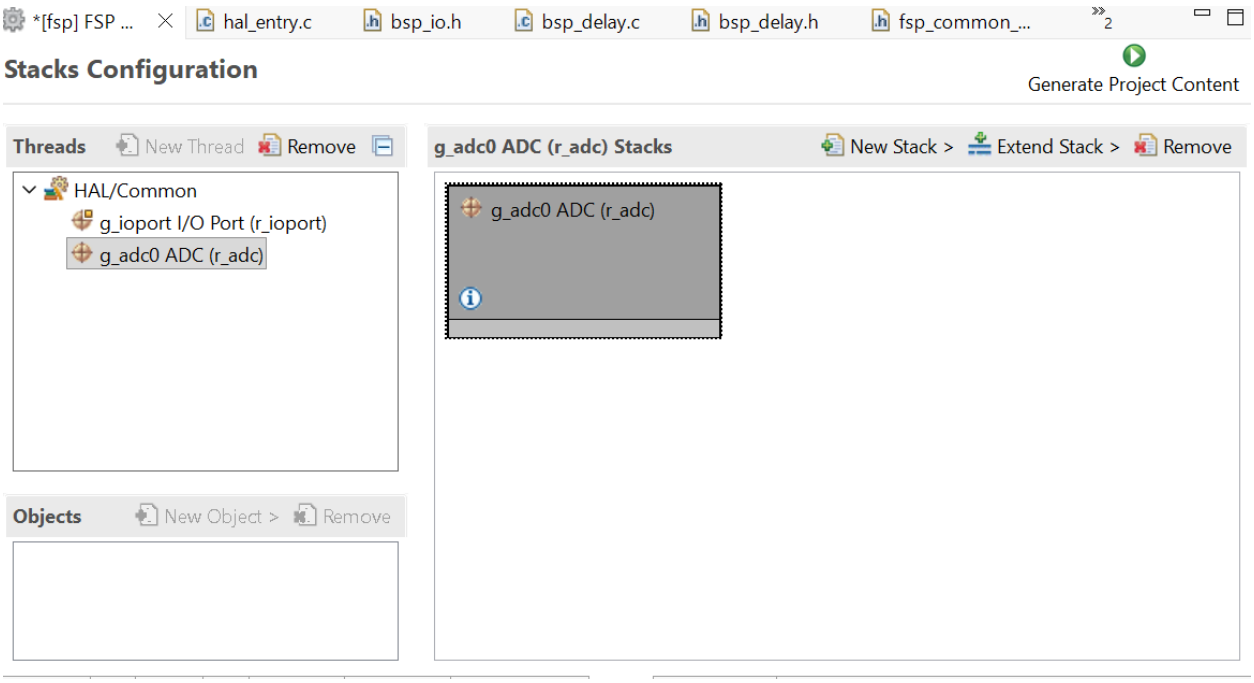


Chương trình trong hal_entry.c :

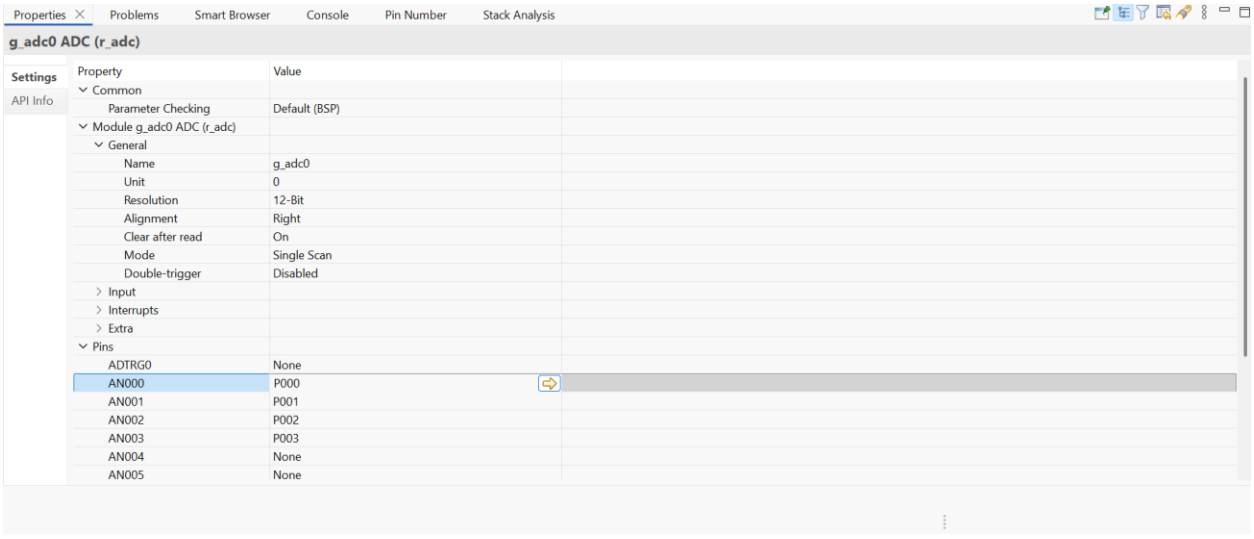
```
#include "hal_data.h"

FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);
FSP_CPP_FOOTER
/*******************************************************************************************************************
 * is called by main() when no RTOS is used.
 ******************************************************************************************************************/
void hal_entry(void)
{

  while(1)
   {
        R_IOPORT_PinWrite(&g_ioport_ctrl, LED, BSP_IO_LEVEL_HIGH); // bật LED
        R_BSP_SoftwareDelay(500, BSP_DELAY_UNITS_MILLISECONDS); //delay 500ms
        R_IOPORT_PinWrite(&g_ioport_ctrl, LED, BSP_IO_LEVEL_LOW);  // tắt LED
        R_BSP_SoftwareDelay(500, BSP_DELAY_UNITS_MILLISECONDS);//delay 500ms
   }


#if BSP_TZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}
```

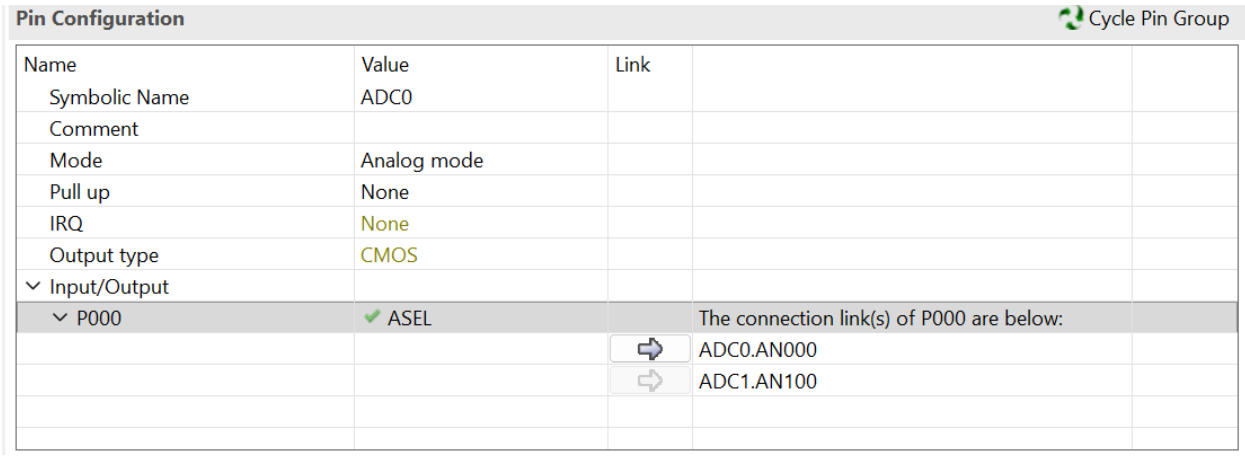
- ADC

Trong FSP Configuration chọn Stack -> New Stack -> Analog -> ADC



Sau đó nháy chuột vào g_adc0_ADC(r_adc) và chọn properties để cấu hình ADC





Chương trình đọc giá trị điện áp của ADC0 , Channel 0 :

```
#include "hal_data.h"

FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);
FSP_CPP_FOOTER
/*******************************************************************************
 * is called by main() when no RTOS is used.
 ******************************************************************************/
void hal_entry(void)
{
    fsp_err_t err;
    uint16_t  adc_val = 0;
        /* Mở ADC */
        R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
        R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
        R_ADC_ScanStart(&g_adc0_ctrl);

        while (1)
        {
            /* Đọc ADC */
            err = R_ADC_Read(&g_adc0_ctrl, ADC_CHANNEL_0, &adc_val);
            if (FSP_SUCCESS == err)
            {
                unused variable 'voltage' [-Wunused-variable]
                float voltage = (3.3f * (float)adc_val) / 4095.0f;
            }
            R_BSP_SoftwareDelay(400, BSP_DELAY_UNITS_MILLISECONDS);
        }

#if BSP_TZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}
```

4.2 Chương trình lập trình bằng thanh ghi

- GPIO

```
#include "hal_data.h"

#define LED_PORT        6
#define LED_PIN         3  // led :p603
#define BUTTON_PORT     6  //button : p602
#define BUTTON_PIN      2
void delay_ms(volatile uint32_t ms)
{
    for (volatile uint32_t i = 0; i < (ms * 5000); i++)
    {
        __asm("NOP");
    }
}
void GPIO_Init(void)
{
    R_PMISC->PWPR_b.B0WI  = 0U;
    R_PMISC->PWPR_b.PFSWE = 1U;

    R_PFS->PORT[LED_PORT].PIN[LED_PIN].PmnPFS_b.PMR  = 0U;  // GPIO
    R_PFS->PORT[LED_PORT].PIN[LED_PIN].PmnPFS_b.DSCR = 1U;  // High drive
    R_PFS->PORT[LED_PORT].PIN[LED_PIN].PmnPFS_b.PODR = 0U;  // Mức ban đầu
    R_PFS->PORT[LED_PORT].PIN[LED_PIN].PmnPFS_b.PDR  = 1U;  // Output

    R_PFS->PORT[BUTTON_PORT].PIN[BUTTON_PIN].PmnPFS_b.PMR = 0U; // GPIO
    R_PFS->PORT[BUTTON_PORT].PIN[BUTTON_PIN].PmnPFS_b.PCR = 1U; // Pull-up enable
    R_PFS->PORT[BUTTON_PORT].PIN[BUTTON_PIN].PmnPFS_b.PDR = 0U; // Input

    R_PMISC->PWPR_b.PFSWE = 0U;
    R_PMISC->PWPR_b.B0WI  = 1U;
}
```

```
void hal_entry(void)
{
    GPIO_Init();
    while (1)
    {
        // Bật LED
        if(R_PORT6->PIDR_b.PIDR2 == 0)
        {
            delay_ms(20);
            R_PORT6->PODR_b.PODR3 = 1U;
                   delay_ms(500);
        }

        // Tắt LED
        else
        {
            delay_ms(20);
            R_PORT6->PODR_b.PODR3 = 0U;
                   delay_ms(500);
        }
    }
}
```

- ADC

```c
#include "hal_data.h"

void delay_ms(volatile uint32_t ms)
{
    for (volatile uint32_t i = 0; i < (ms * 5000); i++)
    {
        __asm("NOP");
    }
}
void ADC_PinConfig(void)
{
    R_PMISC->PWPR_b.B0WI  = 0U;
    R_PMISC->PWPR_b.PFSWE = 1U;
    R_PFS->PORT[0].PIN[0].PmnPFS_b.PMR  = 0;  // GPIO
    R_PFS->PORT[0].PIN[0].PmnPFS_b.ASEL = 1;  // chon analog
    R_PMISC->PWPR_b.PFSWE = 0U;
    R_PMISC->PWPR_b.B0WI  = 1U;
}

void ADC0_Init(void)
{
    R_MSTP->MSTPCRD_b.MSTPD16 = 0;        // bật clock ADC0
    R_ADC0->ADCSR = 0x0000;
    R_ADC0->ADCSR_b.ADCS = 0b00;          // Single Scan
    R_ADC0->ADSTRGR = 0x0000;             // software trigger
    R_ADC0->ADANSA[0] = 0x0001;             // enable AN000
}
```

```c
uint16_t ADC0_Read_AN000(void)
{
    R_ADC0->ADCSR_b.ADST = 1;
    while (R_ADC0->ADCSR_b.ADST == 1);
    return (R_ADC0->ADDR[0] & 0x0FFF);
}

void hal_entry(void)
{
    uint16_t adc_val;
    variable 'voltage' set but not used [-Wunused-but-set-variable]
    float voltage;

    ADC_PinConfig();
    ADC0_Init();

    while (1)
    {
        adc_val = ADC0_Read_AN000();
        voltage = (3.3f * adc_val) / 4095.0f;    // Quy đổi sang điện áp
        delay_ms(500);
    }
}
```

5 . Kết luận

Qua quá trình tìm hiểu và thực hiện đề tài “I/O Port và ADC trên vi điều khiển RA6M5”, nhóm chúng em đã nắm vững hơn cách thức hoạt động của hai ngoại vi quan trọng nhất trong các hệ thống nhúng. Việc nghiên cứu song song hai phương pháp lập trình — thông qua FSP và lập trình trực tiếp bằng thanh ghi — giúp chúng em nhận thấy rõ sự khác biệt về mức độ tiện lợi, khả năng kiểm soát cũng như phạm vi ứng dụng của từng phương pháp.

FSP mang lại sự trực quan, nhanh chóng và giảm thiểu sai sót, phù hợp cho các dự án cần triển khai nhanh. Ngược lại, lập trình bằng thanh ghi cho phép truy cập sâu vào phần cứng, giúp hiểu rõ bản chất hoạt động của vi điều khiển và tối ưu hệ thống khi cần thiết. Sự kết hợp kiến thức của cả hai hướng tiếp cận sẽ là nền tảng quan trọng để chúng em phát triển các ứng dụng nhúng phức tạp hơn trong tương lai.

Thông qua bài tập lớn này, nhóm không chỉ củng cố kiến thức lý thuyết mà còn rèn luyện kỹ năng phân tích, cấu hình ngoại vi và xây dựng chương trình thực tế. Đây là những kinh nghiệm quý báu, góp phần giúp chúng em tự tin hơn trong các môn học tiếp theo và trong công việc sau này.