

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH

KHOA CƠ KHÍ ĐỘNG LỰC

ĐỒ ÁN TỐT NGHIỆP

**BIÊN SOẠN TÀI LIỆU GIẢNG DẠY MÔN HỌC
VI ĐIỀU KHIỂN - ỨNG DỤNG DÀNH CHO CHUYÊN
NGÀNH CÔNG NGHỆ KỸ THUẬT Ô TÔ
PHẦN THỰC HÀNH**

GVHD: ThS. NGUYỄN TRUNG HIẾU

SVTH: LÊ CHÍ TÂM

MSSV: 20145158

SVTH: NGUYỄN ĐỨC HOÀI

MSSV: 20145261

Tp. Hồ Chí Minh, tháng 01 năm 2025

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH

KHOA CƠ KHÍ ĐỘNG LỰC

ĐỒ ÁN TỐT NGHIỆP

Ngành: Công nghệ kỹ thuật ô tô

Tên đề tài

**BIÊN SOẠN TÀI LIỆU GIẢNG DẠY MÔN HỌC
VI ĐIỀU KHIỂN - ỨNG DỤNG DÀNH CHO CHUYÊN
NGÀNH CÔNG NGHỆ KỸ THUẬT Ô TÔ
PHẦN THỰC HÀNH**

GVHD: ThS. NGUYỄN TRUNG HIẾU

SVTH: LÊ CHÍ TÂM

MSSV: 20145158

SVTH: NGUYỄN ĐỨC HOÀI

MSSV: 20145261

Tp. Hồ Chí Minh, tháng 01 năm 2025

TP. Hồ Chí Minh, ngày 13 tháng 09 năm 2024

NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

Họ tên sinh viên:	1. Lê Chí Tâm (E-mail: 20145158@student.hcmute.edu.vn)	MSSV: 20145158 Điện thoại: 0387375070)
	2. Nguyễn Đức Hoài (E-mail: 20145261@student.hcmute.edu.vn)	MSSV: 20145261 Điện thoại: 0358524295)
Ngành:	Công nghệ Kỹ thuật Ô tô	
Khóa:	2020	Lớp: 201451A

1. Tên đề tài

Biên soạn tài liệu giảng dạy môn học Vi Điều Khiển Ứng dụng dành cho chuyên ngành Công nghệ Kỹ thuật Ô tô – Phần thực hành.

2. Nhiệm vụ đề tài

Thiết kế các bài giảng thực hành phù hợp cho sinh viên ngành Công nghệ Kỹ thuật Ô tô. Biên soạn tài liệu hướng dẫn từng bước thực hiện các bài thực hành, kết nối phần cứng, lập trình vi điều khiển.

3. Sản phẩm của đề tài

01 mô hình và 01 bản thuyết minh.

4. Ngày giao nhiệm vụ đề tài: 13/09/2024.

5. Ngày hoàn thành nhiệm vụ: 04/01/2025.

TRƯỞNG BỘ MÔN

CÁN BỘ HƯỚNG DẪN

Bộ môn: Điện tử ô tô

PHIẾU NHẬN XÉT ĐỒ ÁN TỐT NGHIỆP

(Dành cho giảng viên hướng dẫn)

Họ và tên sinh viên: Lê Chí Tâm

MSSV: 20145158

Họ và tên sinh viên: Nguyễn Đức Hoài

MSSV: 20145261

Tên đề tài: Biên soạn tài liệu giảng dạy môn học Vi Điều Khiển Ứng dụng dành cho chuyên ngành Công nghệ Kỹ thuật Ô tô – Phần thực hành.

Ngành đào tạo: Công nghệ Kỹ thuật Ô tô.

Họ và tên GV hướng dẫn: ThS. Nguyễn Trung Hiếu.

Ý KIẾN NHẬN XÉT

1. Nhận xét về tinh thần, thái độ làm việc của sinh viên

.....
.....
.....

2. Nhận xét về kết quả thực hiện của ĐATN

2.1. Kết cấu, cách thức trình bày ĐATN:

.....
.....
.....
.....
.....

2.2 Nội dung đồ án:

(Cơ sở lý luận, tính thực tiễn và khả năng ứng dụng của đồ án, các hướng nghiên cứu có thể tiếp tục phát triển)

.....
.....
.....

2.3. Kết quả đạt được:

.....
.....
.....

2.4. Những tồn tại (nếu có):

.....
.....

3. Đánh giá:

TT	Mục đánh giá	Điểm tối đa	Điểm đạt được
1.	Hình thức và kết cấu ĐATN	30	
	Đúng format với đầy đủ cả hình thức và nội dung của các mục	10	
	Mục tiêu, nhiệm vụ, tổng quan của đề tài	10	
	Tính cấp thiết của đề tài	10	
2.	Nội dung ĐATN	50	
	Khả năng ứng dụng kiến thức toán học, khoa học và kỹ thuật, khoa học xã hội...	5	
	Khả năng thực hiện/phân tích/tổng hợp/danh giá	10	
	Khả năng thiết kế chế tạo một hệ thống, thành phần, hoặc quy trình đáp ứng yêu cầu đưa ra với những ràng buộc thực tế.	15	
	Khả năng cải tiến và phát triển	15	
	Khả năng sử dụng công cụ kỹ thuật, phần mềm chuyên ngành...	5	
3.	Đánh giá về khả năng ứng dụng của đề tài	10	
4.	Sản phẩm cụ thể của ĐATN	10	
	Tổng điểm	100	

4. Kết luận:

- Được phép bảo vệ
 Không được phép bảo vệ

TP.HCM, ngày tháng 01 năm 2025

Giảng viên hướng dẫn

(Ký, ghi rõ họ tên)

Bộ môn: Điện tử ô tô

PHIẾU NHẬN XÉT ĐỒ ÁN TỐT NGHIỆP

(Dành cho giảng viên phản biện)

Họ và tên sinh viên: Lê Chí Tâm

MSSV: 20145158

Họ và tên sinh viên: Nguyễn Đức Hoài

MSSV: 20145261

Tên đề tài: Biên soạn tài liệu giảng dạy môn học Vi Điều Khiển Ứng dụng dành cho chuyên ngành Công nghệ Kỹ thuật Ô tô – Phản thực hành.

Ngành đào tạo: Công nghệ Kỹ thuật Ô tô.

Họ và tên GV phản biện: ThS. Nguyễn Trọng Thức.

Ý KIẾN NHẬN XÉT

1. Kết cấu, cách thức trình bày ĐATN:

.....
.....
.....

2. Nội dung đồ án:

(Cơ sở lý luận, tính thực tiễn và khả năng ứng dụng của đồ án, các hướng nghiên cứu có thể tiếp tục phát triển)

.....
.....
.....
.....

3. Kết quả đạt được:

.....
.....
.....
.....

4. Những thiếu sót và tồn tại của ĐATN:

.....

.....
.....
.....
.....
.....

5. Câu hỏi:

.....
.....
.....
.....

6. Đánh giá:

TT	Mục đánh giá	Điểm tối đa	Điểm đạt được
1.	Hình thức và kết cấu ĐATN	30	
	Dung format với đầy đủ cả hình thức và nội dung của các mục	10	
	Mục tiêu, nhiệm vụ, tổng quan của đề tài	10	
	Tính cấp thiết của đề tài	10	
2.	Nội dung ĐATN	50	
	Khả năng ứng dụng kiến thức toán học, khoa học và kỹ thuật, khoa học xã hội...	5	
	Khả năng thực hiện/phân tích/tổng hợp/đánh giá	10	
	Khả năng thiết kế, chế tạo một hệ thống, thành phần, hoặc quy trình đáp ứng yêu cầu đưa ra với những ràng buộc thực tế.	15	
	Khả năng cải tiến và phát triển	15	
	Khả năng sử dụng công cụ kỹ thuật, phần mềm chuyên ngành...	5	
3.	Đánh giá về khả năng ứng dụng của đề tài	10	
4.	Sản phẩm cụ thể của ĐATN	10	
	Tổng điểm	100	

7. Kết luận:

- Được phép bảo vệ
 Không được phép bảo vệ

TP.HCM, ngày tháng 01 năm 2025
Giảng viên phản biện
(Ký, ghi rõ họ tên)

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT
THÀNH PHỐ HỒ CHÍ MINH
KHOA CƠ KHÍ ĐỘNG LỰC

XÁC NHẬN HOÀN THÀNH ĐÒ ÁN

Tên đề tài: Biên soạn tài liệu giảng dạy môn học Vi Điều Khiển Ứng dụng dành cho chuyên ngành Công nghệ Kỹ thuật Ô tô – Phân thực hành.

Họ và tên Sinh viên: Lê Chí Tâm
Nguyễn Đức Hoài

MSSV: 20145158
MSSV: 20145261

Ngành: Công nghệ Kỹ thuật ô tô

Sau khi tiếp thu và điều chỉnh theo góp ý của Giảng viên hướng dẫn, Giảng viên phản biện và các thành viên trong Hội đồng bảo vệ. Đò án đã được hoàn chỉnh đúng theo yêu cầu về nội dung và hình thức.

Chủ tịch Hội đồng: _____

Giảng viên hướng dẫn: _____

Giảng viên phản biện: _____

Tp. Hồ Chí Minh, ngày tháng năm 2025

LỜI CẢM ƠN

Đồ án tốt nghiệp với đề tài “Biên soạn tài liệu giảng dạy môn học Vật liệu kẽm - Ứng dụng cho chuyên ngành Công nghệ Kỹ thuật Ô tô – Phản ứng hành” thuộc bộ môn Điện ô tô, ngành Công nghệ Kỹ thuật Ô tô là kết quả cố gắng không ngừng trong hơn 4 tháng qua không chỉ của riêng cá nhân nhóm em mà còn là những sự giúp đỡ nhau của bạn bè, sự chỉ dạy tận tình của tất cả các giảng viên trường Đại học Sư phạm Kỹ thuật TP.HCM và đặc biệt nhất là Th.S Nguyễn Trung Hiếu đã hướng dẫn và chỉ bảo chúng em ở chặng đường cuối cùng của thời sinh viên này.

Đầu tiên, chúng em xin gửi lời cảm ơn đến tất cả các thầy cô của trường Đại học Sư phạm Kỹ thuật TP.HCM đã giành thời gian, kiến thức để giảng dạy cho chúng em trong suốt 4 năm đại học vừa qua. Và chúng em xin gửi lời cảm ơn sâu sắc đến thầy Nguyễn Trung Hiếu, người đã tận tình hướng dẫn chúng em trong suốt quá trình thực hiện đồ án. Thầy đã giúp nhóm giải quyết nhiều vấn đề nảy sinh và đưa ra những định hướng quan trọng để hoàn thành đồ án đúng mục tiêu ban đầu và đặc biệt là giúp chúng em rèn luyện, mở mang tư duy nhằm phục vụ cho công việc sau khi ra trường. Nhờ có thầy Nguyễn Trung Hiếu mà chúng em đã có cơ hội phát triển thêm nhiều kỹ năng và thành tựu của mình.

Tiếp theo, chúng em cảm ơn đến tất cả bạn bè, những người bạn đã đồng hành, chia sẻ vui buồn cùng nhau trong suốt chặng đường đại học này. Những người bạn giúp đỡ nhau trong lúc khó khăn, không chỉ trong học tập mà kể cả trong cuộc sống hàng ngày. Đặc biệt hơn nữa, chúng em xin gửi lời cảm ơn đến tất cả các bạn trong phòng thí nghiệm của thầy Nguyễn Trung Hiếu đã nhiệt tình chia sẻ, giúp đỡ chúng em hoàn thành đồ án tốt nghiệp này.

Cuối cùng, chúng em xin chân thành cảm ơn và kính chúc tất cả bạn bè và các thầy cô trong trường đại học Sư phạm Kỹ thuật TP.HCM sức khỏe, thành công và hạnh phúc.

TÓM TẮT

Trong bối cảnh công nghệ điện tử và tự động hóa ngày càng phát triển, vi điều khiển đóng vai trò quan trọng trong các hệ thống trên ô tô. Với mục tiêu nghiên cứu và thiết kế mô hình thực hành tiện lợi, đa năng cùng với các bài giảng phù hợp cho sinh viên ngành Công nghệ Kỹ thuật Ô tô tại trường Đại học Sư phạm Kỹ thuật TPHCM. Chính vì vậy, việc xây dựng một tài liệu chuyên sâu, trực quan và dễ tiếp cận, phục vụ cho việc học tập và giảng dạy về vi điều khiển ứng dụng là rất cần thiết. Vì lý do này, nhóm chúng em quyết định chọn đề tài “ Biên soạn tài liệu giảng dạy môn học vi điều khiển ứng dụng cho chuyên ngành Công nghệ Kỹ thuật Ô tô – Phân thực hành”. Nội dung tài liệu được biên soạn gồm: kiến thức cơ bản về vi điều khiển, kỹ thuật lập trình, giao tiếp ngoại vi và các giao thức truyền thông. Đặc biệt, đồ án tập trung vào dòng vi điều khiển AVR (ATMEGA328P, ATMEGA32A) với các bài thực hành thực tế. Tài liệu được thiết kế nhằm hỗ trợ sinh viên nắm vững lý thuyết, thực hành hiệu quả, góp phần nâng cao chất lượng đào tạo kỹ thuật. Cách tiếp cận này giúp sinh viên rèn luyện tư duy logic, kỹ năng lập trình và khả năng làm chủ công nghệ hiện đại, từ đó đáp ứng các yêu cầu khắt khe của ngành công nghiệp ô tô và nâng cao chất lượng đào tạo của nhà trường.

MỤC LỤC

	Trang
LỜI CẢM ƠN	i
TÓM TẮT	ii
MỤC LỤC	iii
DANH MỤC CÁC CHỮ VIẾT TẮT VÀ KÝ HIỆU	viii
DANH MỤC CÁC HÌNH	xiv
DANH MỤC CÁC BẢNG	xviii
CHƯƠNG 1. TỔNG QUAN	1
1.1. Lý do chọn đề tài	1
1.2. Tính cấp thiết của đề tài.....	1
1.3. Mục tiêu của đề tài	2
1.4. Phương pháp và phạm vi nghiên cứu	2
1.4.1. Phương pháp nghiên cứu.....	2
1.4.2. Phạm vi nghiên cứu.....	3
CHƯƠNG 2. TỔNG QUAN VỀ VI XỬ LÝ VÀ VI ĐIỀU KHIỂN	4
2.1. Đặc điểm của ATMEGA328P.....	4
2.2. Sơ đồ chân của ATMEGA328P	5
2.3. Sơ đồ khối của ATMEGA328P.....	8
2.4. Lõi CPU AVR	10
2.4.1. Tổng quan.....	10
2.4.2. Thanh ghi trạng thái – SREG (Status Register)	12
2.4.3. Stack Pointer	13
2.5. Cấu trúc bộ nhớ của ATMEGA328P	15
2.5.1. Bộ nhớ chương trình (Program Memory)	16
2.5.2. Bộ nhớ dữ liệu (Data Memory).....	20
CHƯƠNG 3. NGÔN NGỮ LẬP TRÌNH	23
3.1. Cấu trúc chương trình CAVR.....	23
3.2. Lập trình xuất nhập PORT	28
3.3. Định nghĩa số hàng, khai báo biến, toán tử C, cú pháp C	30

3.3.1. Định nghĩa hằng	30
3.3.2. Khai báo biến	31
3.3.3. Định nghĩa và cú pháp khai báo hàm	32
3.4.2. Các toán tử trong ngôn ngữ lập trình C.....	33
3.4.3. Các loại hàm.....	36
CHƯƠNG 4. LẬP TRÌNH HIỂN THỊ	37
4.1. Lập trình hiển thị LED 7 đoạn.....	37
4.1.1. Giới thiệu.....	37
4.1.2. Cấu tạo LED 7 đoạn.....	37
4.1.3. Kết nối LED 7 đoạn với vi điều khiển	38
4.1.4. Bài thực hành	40
4.2. Lập trình hiển thị LCD	43
4.2.1. Giới thiệu.....	43
4.2.2. Sơ đồ chân LCD 16x2.....	44
4.2.3. Cấu trúc thanh ghi và tổ chức bộ nhớ	46
4.2.4. Điều khiển hiển thị LCD	49
4.2.5. Bài thực hành	54
CHƯƠNG 5. CẤU TRÚC NGẮT.....	58
5.1. Khái niệm về ngắt.....	58
5.2. Trình phục vụ ngắt và bảng vector ngắt	59
5.3. Ngắt ngoài	61
5.3.1. Thanh ghi A điều khiển ngắt ngoài – EICRA.....	62
5.3.2. Thanh ghi mặt nạ ngắt – EIMSK	63
5.3.3. Thanh ghi cờ ngắt ngoài – EIFR	64
5.3.4. Thanh ghi Pin Change Interrupt Control – PCICR	64
5.3.5. Thanh ghi Pin Change Interrupt Flag – PCIFR	65
5.3.6. Các thanh ghi Pin Change Mask – PCMSK	65
5.4. Bài thực hành.....	66
CHƯƠNG 6. BỘ CHUYỂN ĐỔI ADC.....	69
6.1. Cấu trúc bộ ADC	69

6.1.1. Giới thiệu.....	69
6.1.2. Độ phân giải	71
6.1.3. Điện áp tham chiếu.....	72
6.2. Các thanh ghi của bộ ADC.....	73
6.2.1. Thanh ghi lựa chọn ghép kênh ADC – ADMUX	73
6.2.2. Điều khiển ADC và thanh ghi trạng thái A – ADCSRA	74
6.2.3. Thanh ghi dữ liệu ADC – ADCL và ADCH.....	76
6.2.4. Điều khiển ADC và Thanh ghi trạng thái B – ADCSRB	77
6.2.5. Thanh ghi Digital Input Disable 0 – DIDR0.....	78
6.3. Bài thực hành.....	78
CHƯƠNG 7. LẬP TRÌNH ĐIỀU KHIỂN.....	82
7.1. Lập trình cổng xuất nhập.....	82
7.1.1. Các thanh ghi lập trình cổng xuất nhập.....	82
7.1.2. Bài thực hành	84
7.2. Lập trình các ứng dụng điều khiển	85
7.2.1. Điều khiển động cơ DC.....	85
7.2.2. Cảm biến nhiệt độ nước làm mát	99
CHƯƠNG 8. BỘ SO SÁNH TƯƠNG TỰ	106
8.1. Cấu trúc bộ so sánh tương tự	106
8.1.1. Khái quát	106
8.1.2. Đầu vào ghép kênh của bộ so sánh tương tự	106
8.1.3. Sơ đồ khối bộ so sánh tương tự.....	107
8.2. Các thanh ghi của bộ so sánh tương tự.....	110
8.2.1. ADCSRB – Thanh ghi điều khiển và trạng thái của ADC B	110
8.2.2. ACSR – Thanh ghi điều khiển và trạng thái của bộ so sánh tương tự	110
8.2.3. DIDR1 – Thanh ghi vô hiệu hóa đầu vào số.....	112
8.3. Bài thực hành.....	113
CHƯƠNG 9. CẤU TRÚC BỘ TIMER/COUNTER/PWM	116
9.1. Cấu trúc Timer/Counter.....	116
9.1.1. Tổng quan.....	116

9.1.2. Định nghĩa khi sử dụng Timer/Counter	119
9.2. Các chế độ làm việc Timer/Counter.....	120
9.2.1. Timer/Counter0	120
9.2.2. Timer/Counter1	123
9.2.3. Timer/Counter2	125
9.3. Các thanh ghi chức năng Timer/Counter.....	126
9.3.1. Timer/Counter0	126
9.3.2. Timer/Counter1	132
9.3.3. Timer/Counter2	137
9.4. Bài thực hành.....	139
CHƯƠNG 10. CẤU TRÚC BỘ TIMER/COUNTER/PWM (tiếp theo).....	143
10.1. Cấu trúc PWM	143
10.2. Các chế độ làm việc PWM	144
10.2.1. Timer/Counter0	144
10.2.2. Timer/Counter1	146
10.2.3. Timer/Counter2	150
10.3. Các thanh ghi chức năng PWM.....	152
10.3.1. Timer/Counter0	152
10.3.2. Timer/Counter1	155
10.3.3. Timer/Counter2	158
10.4. Bài thực hành.....	160
CHƯƠNG 11. THIẾT KẾ GIAO TIẾP.....	164
11.1. Giao thức truyền thông nối tiếp.....	164
11.1.1. Tổng quan.....	164
11.1.2. Các thành phần quan trọng của UART	165
11.1.3. Các thanh ghi của USART	167
11.1.4. Bài thực hành	172
11.2. Giao tiếp I2C	175
11.2.1. Tổng quan về giao tiếp I2C.....	175
11.2.2. Cơ chế hoạt động.....	178

11.2.3. Các thanh ghi của I2C	180
11.2.4. Bài thực hành	185
11.3. Giao tiếp SPI	190
11.3.1. Tổng quan về giao tiếp SPI	190
11.3.2. Cơ chế hoạt động	192
11.3.3. Các thanh ghi của SPI.....	194
11.3.4. Bài thực hành.....	197
CHƯƠNG 12: LẬP TRÌNH VỚI VI ĐIỀU KHIỂN ATMEGA32A	202
12.1. Lập trình hiển thị	202
12.1.1. Lập trình hiển thị LED 7 đoạn	202
12.1.2. Lập trình hiển thị LCD	206
12.2. Cấu trúc ngắn.....	209
12.3. Bộ chuyển đổi ADC	211
12.4. Lập trình điều khiển kim phun nhiên liệu	215
12.4.1. Giới thiệu.....	215
12.4.2. Thực hành.....	216
12.5. Bộ so sánh tương tự	223
12.6. Cấu trúc bộ Timer/Counter.....	225
12.7. Cấu trúc PWM	229
12.8. Thiết kế giao tiếp	231
12.8.1. Giao tiếp UART	231
12.8.2. Giao tiếp I2C	234
12.8.3. Giao tiếp SPI	239
CHƯƠNG 13: KẾT LUẬN VÀ KIẾN NGHỊ	245
13.1. Kết luận	245
13.1.1. Kết quả đạt được	245
13.1.2. Hạn chế của đề tài	245
13.2. Kiến nghị và hướng phát triển	245
DANH MỤC TÀI LIỆU THAM KHẢO.....	247

DANH MỤC CÁC CHỮ VIẾT TẮT VÀ KÝ HIỆU

RISC: Reduced Instruction Set Computer

CISC: Complex Instructions Set Computer

MIPS: Million Instructions Per Second

ALU: Arithmetic Logic Unit

SPM: Store Program Memory

SP: Stack Pointer

BLD: Bit Load

BST: Bit Store

BCD: Binary-Coded Decimal

SPL: Stack Pointer Low

SPH: Stack Pointer High

ISP: In-System Programming

GPR: General Purpose Register

LPM: Load Program Memory

RAM: Random Access Memory

ROM: Read Only Memory

SRAM: Static Random Access Memory

EEFROM: Electrically Erasable Programmable ROM

IR: Instruction Register

DR: Data Register

DDRAM: Display Data RAM

CGROM: Character Generator ROM

CGRAM: Character Generator RAM

RS: Register Select

BF: Busy Flag

EN: Enable

MSB: Most Significant Bit

LSB: Least Significant Bit

DL: Data Length

ISR: Interrupt Service Routine

INT: Interrupt

PCINT: Pin Change Interrupt

PCMSK: Pin Change Mask

EICRA: External Interrupt Control Register A

MCU: Microcontroller Unit

MCUCR: MCU Control Register

EIMSK: External Interrupt Mask

SREG: Status Register

EIFR: External Interrupt Flag Register

PCICR: Pin Change Interrupt Control

PCIFR: Pin Change Interrupt Flag

ADC: Analog-to-Digital Converter

ADEN: ADC Enable

ADMUX: ADC Multiplexer Selection Register

ADCSR: ADC Control and Status Register

ADCH: ADC Data Register High

ADCL: ADC Data Register Low

ADLAR: ADC Left Adjust Result

ADSC: ADC Start Conversion

ADATE: ADC Auto Trigger Enable

ADIF: ADC Interrupt Flag

ADIE: ADC Interrupt Enable

ADPS: ADC Prescaler Select

ADTS: ADC Auto Trigger Source

DIDR: Digital Input Disable

AIN: Analog Input

ACO: Analog Comparator Output

ACIS: Analog Comparator Interrupt Select

ACME: Analog Comparator Multiplexer Enable

ADCSR: ADC Control and Status Register

ACIC: Analog Comparator Input Capture

ACIE: Analog Comparator Interrupt Enable

ACI: Analog Comparator Interrupt Flag

ACSR : Analog Comparator Control and Status Register

ACD: Analog Comparator Disable

ACBG: Analog Comparator Bandgap

TCNT: Timer/Counter Register

CS: Clock Select

TCCR : Timer/Counter Control Register

CTC: Clear Timer on Compare

OCF: Output Compare Flag

OCR: Output Compare Register

ICR: Input Capture Register

COM: Compare Output Mode

WGM: Waveform Generation Mode

OC: Output Compare

TIMSK: Timer/Counter Interrupt Mask

OCIE: Output Compare Match Interrupt Enable

TOIE: Timer Overflow Interrupt Enable

TIFR: Timer/Counter Interrupt Flag Register

TOV: Timer Overflow
FOC: Force Output Compare
ICF: Input Capture Flag
ICR: Input Capture Register
PWM: Pulse Width Modulation
DAC: Digital-to-Analog Converter
ICNC: Input Capture Noise Canceler
ICES: Input Capture Edge Select
UART: Universal Asynchronous Receiver/Transmitter
USART: Universal Synchronous/Asynchronous Receiver/Transmitter
UDR: USART Data Register
RXB: Receive Buffer
UCSR: USART Control and Status Register
RXC: Receive Complete
TXC: Transmit Complete
UDRE: USART Data Register Empty
DOR: Data OverRun
UPE: USART Parity Error
U2X: USART Double Transmission Speed
MPCM: Multi Processor Communication
RXCIE: Receive Complete Interrupt Enable
TXCIE: Transmit Complete Interrupt Enable
UDRIE: USART Data Register Empty Interrupt Enable
RXEN: Receiver Enable
TXEN: Transmitter Enable
UCSZ: USART Character Size
UMSEL: USART Mode Select

UPM: USART Parity Mode
USBS: USART Stop Bit Select
UCSZ: USART Character Size
UCPOL: USART Clock Polarity
UBRR: USART Baud Rate Register
I2C: Inter- Integrated Circuit
TWI: Two-Wire Serial Intereafce
SDA: Serial Data
SCL: Serial Clock
TWINT: TWI Interrupt
TWIE: TWI Interrupt Enable
TWCR: TWI Control Register
TWEA: TWI Enable Acknowledge
TWBR: TWI Bit Rate Register
TWPS: Two-Wire Prescaler
TWSR: TWI Status Register
TWSTA: TWI Start
TWSTO: TWI Stop
TWWC: TWI Write Collision
TWEN: TWI Enable
TWIE: TWI Interrupt Enable
TWDR: TWI Data Register
TWAR: TWI Address Register
TWAMR: TWI Address Mask Register
SPI: Serial Peripheral Bus
SCK: Serial Clock
MISO: Master Input / Slave Output

MOSI: Master Output / Slave Input

SS: Slave Select

CPOL: Clock Polarity

CPHA: Clock Phase

SPCR: SPI Control Register

SPIE: SPI Interrupt Enable

SPE: SPI Enable

DORD: Data Order

MSTR: Master/Slave Select

SPR: SPI Clock Rate

SPSR: SPI Status Register

SPIF: SPI Interrupt Flag

WCOL: Write Collision Flag

SPDR: SPI Data Register

SPI2X: SPI Double Speed

PID: Proportional Integral Derivative

DANH MỤC CÁC HÌNH

	Trang
Hình 2.1. Sơ đồ chân của ATMEGA328P [1]	5
Hình 2.2. Sơ đồ khói ATMEGA328P [2]	8
Hình 2.3. Sơ đồ khói của kiến trúc AVR [2]	10
Hình 2.4. Vùng nhớ Stack.....	14
Hình 2.5. Tổ chức bộ nhớ của ATMEGA328P [4]	16
Hình 2.6. Bộ nhớ chương trình của ATMEGA328P [2]	17
Hình 2.7. Bộ nhớ dữ liệu của ATmega328P [2]	20
Hình 2.8. Cấu trúc của một thanh ghi 8 bit.....	20
Hình 2.9. Register File [4]	21
Hình 3.1. General Digital I/O [2].....	28
Hình 3.2. I/O Pin Equivalent Schematic [2]	29
Hình 3.3. Nguyên lý hoạt động của hàm trong ngôn ngữ C	32
Hình 3.4. Thành phần mặc định của hàm	33
Hình 4.1. Giới thiệu LED 7 đoạn.....	37
Hình 4.2. Cấu tạo bên trong LED 7 đoạn [6].....	38
Hình 4.3. Sơ đồ nối dây bài thực hành LED 7 đoạn	40
Hình 4.4. Mô hình thực tế bài thực hành LED 7 đoạn	43
Hình 4.5. Cấu trúc của bộ nhớ DDRAM	47
Hình 4.6. Vùng nhớ CGROM [7]	48
Hình 4.7 Hoạt động của chân RS [8]	49
Hình 4.8. Sơ đồ nối dây bài thực hành với LCD	55
Hình 4.9. Mô hình thực tế bài thực hành hiển thị LCD	57
Hình 5.1. Quá trình thực hiện ngắn [9].....	59
Hình 5.2. Kết nối ngắn ngoài cho ATMEGA328P [9].....	62
Hình 5.3. Sơ đồ nối dây bài thực hành ngắn ngoài.....	66
Hình 5.4. Mô hình thực tế bài thực hành ngắn ngoài	68

Hình 6.1. Sơ đồ khói của bộ chuyển đổi ADC [2].....	71
Hình 6.2. Sơ đồ nối dây bài thực hành với LM35	79
Hình 6.3. Mô hình thực tế bài thực hành với LM35	81
Hình 7.1. Sơ đồ kết nối bài thực hành LED đơn	84
Hình 7.2. Sơ đồ khói bộ điều khiển PID [10]	86
Hình 7.3. Sơ đồ kết nối bài thực hành điều khiển động cơ DC	90
Hình 7.4. Mô hình thực tế bài thực hành điều khiển động cơ DC.....	97
Hình 7.5. Đồ thị vận tốc mong muốn và vận tốc thực tế	98
Hình 7.6. Đồ thị giá trị sai số	98
Hình 7.7. Đồ thị điều chế độ rộng xung PWM	99
Hình 7.8. Biểu đồ giá trị điện trở của cảm biến ECT theo nhiệt độ [11].....	100
Hình 7.9. Sơ đồ kết nối bài thực hành cảm biến nhiệt độ nước làm mát.....	101
Hình 7.10. Mô hình thực tế bài thực hành cảm biến nhiệt độ nước làm mát	105
Hình 8.1. Sơ đồ khói của bộ so sánh tương tự [2]	107
Hình 8.2. Sơ đồ kết nối bài thực hành bộ so sánh tương tự.....	113
Hình 8.3. Mô hình thực tế bài thực hành bộ so sánh tương tự	115
Hình 9.1. Sơ đồ khói Timer/Counter0 – 8 bit [2]	117
Hình 9.2. Sơ đồ khói Timer/Counter1 – 16 bit [2]	118
Hình 9.3. Sơ đồ khói Timer/Counter2 – 8 bit [2]	119
Hình 9.4. So sánh giữa hai chế độ làm việc [12]	120
Hình 9.5. Quá trình thực hiện hàm delay.....	122
Hình 9.6. Biểu đồ thời gian của chế độ CTC – Timer/Counter0 [2]	123
Hình 9.7. Biểu đồ thời gian của chế độ CTC – Timer/Counter1 [2]	125
Hình 9.8. Biểu đồ thời gian của chế độ CTC – Timer/Counter2 [2]	126
Hình 9.9. Sơ đồ kết nối bài thực hành bộ định thời và bộ đếm thời gian.....	139
Hình 9.10. Mô hình thực tế bài thực hành bộ định thời và bộ đếm thời gian.....	142
Hình 10.1. Chu kỳ điều chế độ rộng xung	143
Hình 10.2. Biểu đồ thời gian của chế độ Fast PWM – Timer/Counter0 [2]	144
Hình 10.3. Biểu đồ thời gian của chế độ Phase Correct PWM – Timer/Counter0 [2]	145

Hình 10.4. Biểu đồ thời gian ở chế độ Fast PWM [2]	146
Hình 10.5. Mode 14 trong chế độ Fast PWM [12]	147
Hình 10.6. Biểu đồ thời gian ở chế độ Phase Correct PWM [2]	148
Hình 10.7. Mode 10 trong chế độ Phase Correct PWM [12]	149
Hình 10.8. Biểu đồ thời gian ở chế độ Phase and Frequency Correct PWM [2].....	150
Hình 10.9. Biểu đồ thời gian của chế độ Fast PWM – Timer/Counter2 [2]	151
Hình 10.10. Biểu đồ thời gian của chế độ Phase Correct PWM – Timer/Counter2 [2]	152
Hình 10.11. Độ rộng xung điều khiển servo	161
Hình 10.12. Sơ đồ kết nối bài thực hành xung PWM.....	161
Hình 10.13. Mô hình thực tế bài thực hành xung PWM	163
Hình 11.1. Truyền thông nối tiếp đồng bộ - USART	164
Hình 11.2. Truyền thông nối tiếp bất đồng bộ - UART	165
Hình 11.3. Cấu tạo khung dữ liệu của giao thức UART [2]	166
Hình 11.4. Sơ đồ kết nối bài thực hành giao thức UART	172
Hình 11.5. Mô hình thực tế bài thực hành UART.....	174
Hình 11.6 Mạng TWI (I2C) với nhiều thiết bị.....	175
Hình 11.7. Cấu tạo một khung truyền của giao thức I2C	176
Hình 11.8. Start Condition	176
Hình 11.9. Stop Condition	177
Hình 11.10. Sơ đồ logic so sánh địa chỉ TWI [2]	185
Hình 11.11. Sơ đồ kết nối bài thực hành giao thức I2C.....	186
Hình 11.12. Mô hình thực tế bài thực hành I2C	190
Hình 11.13. Giao diện SPI [13].....	192
Hình 11.14. Truyền dữ liệu SPI [13].....	193
Hình 11.15. Sơ đồ kết nối bài thực hành giao thức SPI.....	197
Hình 11.16. Mô hình thực tế bài thực hành SPI	201
Hình 12.1. Sơ đồ nối dây bài thực hành LED 7 đoạn.....	202
Hình 12.2. Mô hình thực tế bài thực hành LED 7 đoạn	205
Hình 12.3. Sơ đồ nối dây bài thực hành với LCD	206

Hình 12.4. Mô hình thực tế bài thực hành hiển thị LCD	208
Hình 12.5. Sơ đồ nối dây bài thực hành ngắt ngoài.....	209
Hình 12.6. Mô hình thực tế bài thực hành ngắt ngoài	211
Hình 12.7. Sơ đồ nối dây bài thực hành với LM35	212
Hình 12.8. Mô hình thực tế bài thực hành với LM35	215
Hình 12.9. Sơ đồ kết nối bài thực hành điều khiển kim phun	216
Hình 12.10. Sơ đồ kết nối bài thực hành bộ so sánh tương tự.....	224
Hình 12.11. Mô hình thực tế bài thực hành bộ so sánh tương tự.....	225
Hình 12.12. Sơ đồ kết nối bài thực hành bộ định thời và bộ đếm thời gian.....	226
Hình 12.13. Mô hình thực tế bài thực hành bộ định thời và bộ đếm thời gian.....	228
Hình 12.15. Sơ đồ kết nối bài thực hành xung PWM.....	229
Hình 12.16. Mô hình thực tế bài thực hành xung PWM	231
Hình 12.17. Sơ đồ kết nối bài thực hành giao thức UART.....	232
Hình 12.18. Mô hình thực tế bài thực hành UART	234
Hình 12.19. Sơ đồ kết nối bài thực hành giao thức I2C	235
Hình 12.20. Mô hình thực tế bài thực hành I2C	239
Hình 12.21. Sơ đồ kết nối bài thực hành giao thức SPI.....	240
Hình 12.22. Mô hình thực tế bài thực hành SPI	244

DANH MỤC CÁC BẢNG

	Trang
Bảng 2.1. Chức năng của từng chân ATMEGA328P	6
Bảng 2.2. Các lệnh liên quan đến Stack Pointer [3]	15
Bảng 2.3. Boot Reset Fuse [2]	18
Bảng 2.4. Boot Size Configuration, ATMEGA328P [2].....	19
Bảng 3.1. Các từ khóa trong ngôn ngữ C	32
Bảng 3.2. Các toán tử đại số [5].....	33
Bảng 3.3. Toán tử truy cập và kích thước [5]	34
Bảng 3.4. Toán tử logic và quan hệ [5]	34
Bảng 3.5. Toán tử thao tác Bit [5].....	35
Bảng 4.1. Hiển thị LED 7 đoạn loại Anode chung	39
Bảng 4.2. Hiển thị LED 7 đoạn loại Cathode chung	39
Bảng 4.3 Sơ đồ chân LCD 16x2	44
Bảng 4.4. Tóm tắt các lệnh ghi vào LCD [8].....	50
Bảng 5.1 Các vector ngắt và Reset trên Atmega328P [9]	60
Bảng 5.2. Bảng thiết lập các chế độ ngắt ngoài của INT[0:1].....	63
Bảng 6.1. Lựa chọn điện áp tham chiếu [2]	73
Bảng 6.2. Lựa chọn kênh đầu vào của bộ ADC [2]	74
Bảng 6.3. Lựa chọn hệ số chia ADC [2]	75
Bảng 6.4. Nguồn kích ADC trong chế độ Auto Trigger [2].....	77
Bảng 8.1. Đầu vào ghép kênh của bộ so sánh tương tự [2]	106
Bảng 8.2. Thiết lập hai bit ACIS1 và ACISO [2]	112
Bảng 9.1. Định nghĩa khi sử dụng Timer/Counter.....	119
Bảng 9.2. Mô tả giá trị bộ chia cho Timer/Counter0 [2]	121
Bảng 9.3. Mô tả giá trị bộ chia cho Timer/Counter1 [2]	124
Bảng 9.4. Cấu hình ngõ ra của chân OC0A [2]	127
Bảng 9.5. Cấu hình ngõ ra của chân OC0B [2]	127

Bảng 9.6. Mô tả cấu hình chế độ làm việc của Timer/Counter0 [2]	128
Bảng 9.7. Mô tả các bit chọn xung nhịp – Timer/Counter0 [2]	128
Bảng 9.8. Cấu hình ngõ ra của chân OC1A/OC1B [2]	132
Bảng 9.9. Mô tả cấu hình chế độ làm việc của Timer/Counter1 [2]	133
Bảng 9.10. Mô tả các bit chọn xung nhịp – Timer/Counter1 [2]	133
Bảng 9.11. Chế độ đầu ra so sánh, không PWM – Timer/Counter0 [2]	137
Bảng 9.12. Mô tả cấu hình chế độ làm việc của Timer/Counter2 [2]	138
Bảng 9.13. Mô tả các bit chọn xung nhịp – Timer/Counter1 [2]	138
Bảng 10.1. Ngõ ra chân OC0A với chế độ Fast PWM – Timer/Counter0 [2]	153
Bảng 10.2. Ngõ ra chân OC0A với chế độ Phase Correct PWM – Timer/Counter0 [2]	153
Bảng 10.3. Ngõ ra chân OC0B với chế độ Fast PWM – Timer/Counter0 [2]	153
Bảng 10.4. Ngõ ra chân OC0B với chế độ Phase Correct PWM – Timer/Counter0 [2]	154
Bảng 10.5. Mô tả cấu hình các chế độ làm việc PWM – Timer/Counter0 [2]	154
Bảng 10.6. Ngõ ra chân OC1A/OC1B với chế độ Fast PWM – Timer/Counter1 [2]	155
Bảng 10.7. Ngõ ra chân OC1A/OC1B với chế độ Phase Correct và Phase and Frequency Correct PWM – Timer/Counter1 [2]	156
Bảng 10.8. Mô tả cấu hình chế độ làm việc PWM của Timer/Counter1 [2]	156
Bảng 10.9. Ngõ ra chân OC2A với chế độ Fast PWM – Timer/Counter2 [2]	158
Bảng 10.10. Ngõ ra chân OC2A với chế độ Phase Correct PWM – Timer/Counter2 [2]	158
Bảng 10.11. Ngõ ra chân OC2B với chế độ Fast PWM – Timer/Counter2 [2]	159
Bảng 10.12. Ngõ ra chân OC2B với chế độ Phase Correct PWM – Timer/Counter0 [2]	159
Bảng 10.13. Mô tả cấu hình các chế độ làm việc PWM – Timer/Counter2 [2]	159
Bảng 11.1. Công thức tính tốc độ Baud [2]	165
Bảng 11.2. Cấu hình các chế độ hoạt động của UART [2]	169
Bảng 11.3. Cấu hình sử dụng Parity Bit [2]	169
Bảng 11.4. Lựa chọn độ dài Stop Bit [2]	170
Bảng 11.5. Cấu hình lựa chọn độ dài dữ liệu của bộ truyền [2]	170
Bảng 11.6. Cấu hình bit UCPOLn [2]	170
Bảng 11.7. Cấu hình thanh ghi UBRR và tốc độ Baud mẫu.....	171

Bảng 11.8. Bộ chia tần số tốc độ bit TWI [2]	183
Bảng 11.9. Chế độ SPI	193
Bảng 11. 10. Chức năng của CPOL [2]	195
Bảng 11.11. Chức năng của CPHA [2]	195
Bảng 11. 12. Mối quan hệ giữa tốc độ xung nhịp SCK và tần số dao động (fOSC) [2]	195

CHƯƠNG 1. TỔNG QUAN

1.1. Lý do chọn đề tài

Trong thời đại hiện nay, sự phát triển của công nghệ điện tử và tự động hóa đã mang lại sự thay đổi mạnh mẽ cho các hệ thống ô tô, đặc biệt là các hệ thống điều khiển và quản lý. Vì điều khiển đóng vai trò cốt lõi trong các hệ thống điều khiển động cơ, hệ thống phanh, túi khí, hệ thống điều hòa không khí và nhiều hệ thống phụ trợ khác trên xe, đòi hỏi sinh viên phải nắm vững kiến thức và kỹ năng lập trình vi điều khiển. Đề tài được xây dựng không chỉ tập trung vào cung cấp kiến thức nền tảng, mà còn chú trọng vào thực hành ứng dụng, từ đó giúp sinh viên rèn luyện tư duy logic và kỹ năng lập trình điều khiển cho các hệ thống phức tạp. Điều này nhằm mục tiêu giúp sinh viên có thể nắm bắt và làm chủ các công nghệ tiên tiến, từ đó dễ dàng hòa nhập vào môi trường làm việc sau khi ra trường. Với cách tiếp cận linh hoạt, thực tế và cập nhật theo sự phát triển của ngành công nghiệp ô tô, đề tài sẽ là công cụ quan trọng giúp các kỹ sư tương lai của ngành ô tô đáp ứng được những yêu cầu khắt khe từ các doanh nghiệp hiện nay, đồng thời góp phần nâng cao chất lượng đào tạo và khẳng định vị thế của nhà trường trong lĩnh vực giáo dục.

Tuy nhiên, thực tế trong quá trình học tập môn “Vi Điều Khiển Ứng Dụng”, sinh viên chủ yếu được tiếp cận với lý thuyết, mô phỏng, thiếu mạch thực tế để thực hành dẫn đến hạn chế trong việc tiếp thu và ứng dụng kiến thức. Chính vì vậy nhóm quyết định chọn đề tài **“Biên soạn tài liệu giảng dạy môn học vi điều khiển ứng dụng dành cho chuyên ngành Công nghệ Kỹ thuật Ô tô - Phần thực hành”** làm đề tài đồ án tốt nghiệp. Đề tài hướng đến việc tạo ra một board mạch tiện lợi, giá thành rẻ nhưng vẫn đầy đủ các linh kiện điện tử và bám sát chương trình giảng dạy môn học “Vi Điều Khiển Ứng Dụng” kết hợp với lý thuyết nhằm nâng cao chất lượng đào tạo.

1.2. Tính cấp thiết của đề tài

Vi điều khiển ứng dụng là môn học được giảng dạy cho sinh viên năm 3 ngành Công nghệ Kỹ thuật Ô tô của trường Đại học Sư phạm Kỹ thuật TPHCM. Môn học trang bị cho sinh

viên các kiến thức về kỹ thuật số và kỹ thuật xung, cấu tạo phần cứng của Vi điều khiển (các bộ nhớ bên trong, bộ định thời, các chức năng đặc biệt hỗ trợ khi sử dụng như tạo ngắt, timer...), cách lập trình cho vi điều khiển để nhận các tín hiệu, tính toán xử lý và điều khiển ra bộ chấp hành. Tuy nhiên, môn học vẫn còn thiếu mô hình thực tế để giảng dạy và thực hành.

Từ những vấn đề trên chúng em nhận thấy rất cần thiết để nghiên cứu và thiết kế một mô hình tiện lợi, sử dụng vi điều khiển nói chung và ATMEGA328P và ATMEGA32A nói riêng với giá thành hợp lý, giúp sinh viên dễ dàng thực hiện các bài học theo giáo trình giảng dạy.

1.3. Mục tiêu của đề tài

Năm vững cấu trúc phần cứng, chức năng và phương thức hoạt động của vi điều khiển.

Nghiên cứu và thiết kế một Board mạch nhỏ gọn, linh hoạt, giá rẻ giúp sinh viên dễ dàng thao tác, có góc nhìn trực quan về môn học, kích thích sự sáng tạo và tinh thần học hỏi.

Thiết kế các bài giảng “Vi điều khiển ứng dụng” phù hợp cho sinh viên ngành Công nghệ Kỹ thuật Ô tô, hướng dẫn từng bước thực hiện các bài thực hành, kết nối phần cứng và lập trình cho vi điều khiển.

1.4. Phương pháp và phạm vi nghiên cứu

1.4.1. Phương pháp nghiên cứu

Nghiên cứu các tài liệu về vi điều khiển. Đề biên soạn bài giảng lý thuyết giảng dạy.

Ứng dụng kỹ thuật hàn các board mạch.

Ứng dụng các kiến thức đã học để thiết kế mạch điện cho mô hình.

Nghiên cứu các nguồn tài liệu trên Internet về cấu trúc vi điều khiển ATMEGA328P.

Thu thập và phân tích tài liệu liên quan đến giảng dạy vi điều khiển và các ứng dụng trên ô tô.

Tham khảo các chương trình giảng dạy, giáo trình và các bài thực hành thông qua Internet, từ đó tổng hợp và thiết kế tài liệu phù hợp với sinh viên ngành Công nghệ Kỹ thuật Ô tô.

1.4.2. Phạm vi nghiên cứu

Kiến thức cơ bản về vi điều khiển bao gồm các khái niệm cơ bản như cấu trúc vi điều khiển, cách lập trình, sử dụng các cổng vào/ra và giao tiếp ngoại vi như UART, I2C, SPI, v.v. để sinh viên nắm được nền tảng.

Đào tạo sinh viên về tư duy lập trình, áp dụng và vận dụng ngôn ngữ lập trình C vào thực tế. Đồng thời, sinh viên sẽ được tiếp cận phần mềm mô phỏng Proteus, với đa dạng các linh kiện điện tử, phù hợp cho việc nghiên cứu và dễ dàng phát hiện ra sai sót trong quá trình học tập, từ đó hạn chế hư hỏng khi thực hiện trên mô hình thực tế.

CHƯƠNG 2. TỔNG QUAN VỀ VI XỬ LÝ VÀ VI ĐIỀU KHIỂN

2.1. Đặc điểm của ATMEGA328P

ATMEGA328P là một vi điều khiển 8-bit thuộc họ AVR của Microchip Technology (trước đây là Atmel), thường được sử dụng trong các ứng dụng nhúng nhỏ gọn, nổi bật nhất là trong các board mạch như Arduino Uno, Arduino Nano. ATMEGA328P có giá thành hợp lý, hiệu suất tốt, cùng mức tiêu thụ năng lượng thấp, phù hợp cho tất cả các dự án DIY. Dưới đây là một số đặc điểm chính của ATMEGA328P:

- Kiến trúc RISC (Reduced Instruction Set Computer): ATMEGA328P sử dụng kiến trúc RISC 8-bit với 131 lệnh, giúp thực hiện các lệnh nhanh chóng, thường chỉ trong một chu kỳ xung nhịp.
- Tốc độ xử lý lên đến 20 MIPS (Million Instructions Per Second) ở tốc độ xung nhịp 20 MHz cho phép thực hiện các tác vụ nhanh chóng.

- Bộ nhớ:

- Flash: 32 KB bộ nhớ Flash cho chương trình (trong đó 2KB dành cho Bootloader), cho phép lưu trữ và chạy mã nguồn chương trình dài. Với độ bền khoảng 10.000 chu kỳ ghi/xoá.
- SRAM: 2 KB bộ nhớ SRAM dùng để lưu trữ biến tạm thời và dữ liệu trong quá trình hoạt động.
- EEPROM: 1 KB bộ nhớ EEPROM dùng để lưu trữ dữ liệu không bị mất khi mất nguồn, thích hợp cho các ứng dụng lưu trữ thông tin cần thiết giữa các lần khởi động. Với độ bền khoảng 100.000 chu kỳ ghi/xoá.

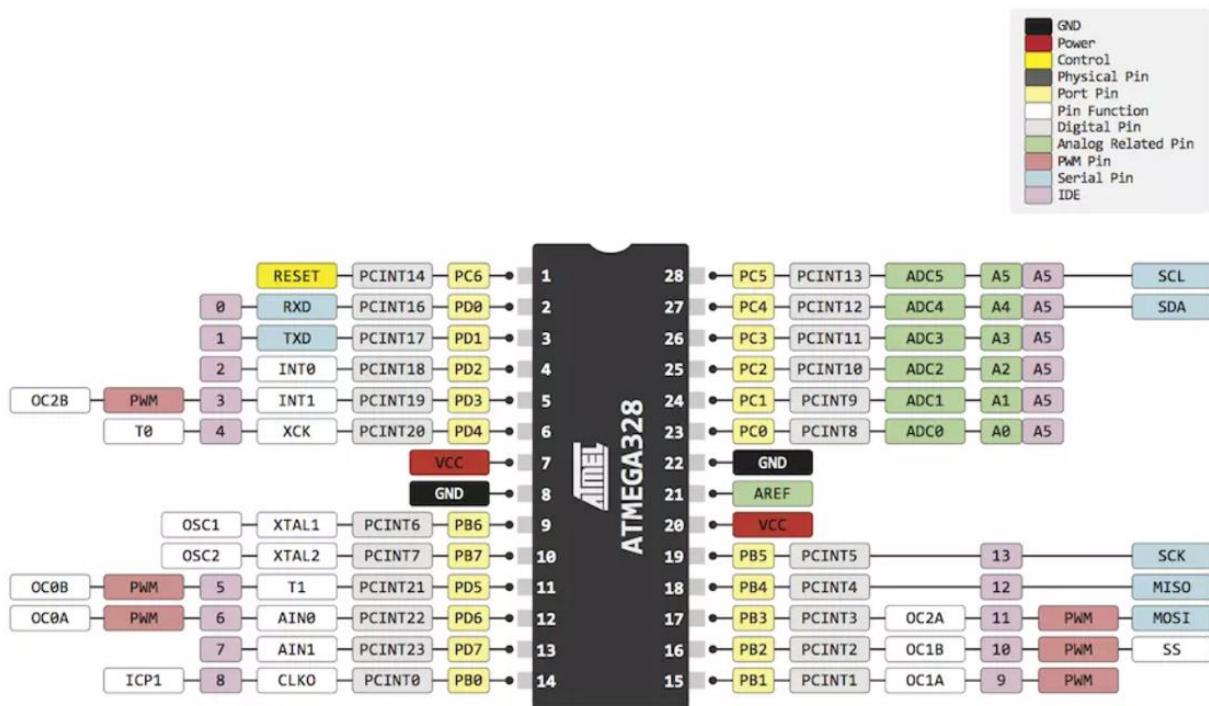
- Có 2 Timer/Counter 8 bit với khả năng chia tần số riêng và so sánh, 1 Timer/Counter 16 bit với bộ chia tần số riêng, chế độ so sánh và chế độ bắt tín hiệu. Bộ đếm thời gian thực với dao động riêng biệt cho phép thực hiện các tác vụ tính toán thời gian chính xác.
- Bộ PWM có 6 kênh và bộ ADC có 8 kênh 10 bit cho phép đo lường tín hiệu Analog từ các cảm biến.

- Tích hợp nhiều tính năng quản lý năng lượng như khả năng reset khi bật nguồn, phát hiện ngưỡng bơm thấp và sáu chế độ ngủ khác nhau như Idle, giảm nhiễu ADC, tiết kiệm năng lượng, tắt nguồn, chờ và chờ mở rộng để tối ưu hóa mức tiêu thụ điện. Bộ dao động RC nội được hiệu chỉnh và tùy chọn dao động ngoài giúp thiết bị có thể thích ứng với nhiều môi trường hoạt động.

- Có 23 chân I/O lập trình được, cho phép giao tiếp với nhiều thiết bị ngoại vi.

- Điện áp hoạt động của ATMEGA328P có thể hoạt động trong khoảng 1,8 – 5,5V với xung nhịp lên đến 20 MHz. Mức tiêu thụ điện năng của ATMEGA328P rất thấp, ở chế độ hoạt động là 0,2 mA, chế độ nghỉ là 0,75 mA và chế độ tắt nguồn là dưới 0,1 µA.

2.2. Sơ đồ chân của ATMEGA328P



Hình 2.1. Sơ đồ chân của ATMEGA328P [1]

Trong vi điều khiển, PORT là khái niệm dùng để chỉ các ngõ vào/ra (I/O), cho phép giao tiếp hai chiều (bi-directional) để thực hiện chức năng xuất hoặc nhận dữ liệu. Mỗi PORT bao gồm 8 chân và trên vi điều khiển ATMEGA328P có 3 PORT được gọi là PORTB,

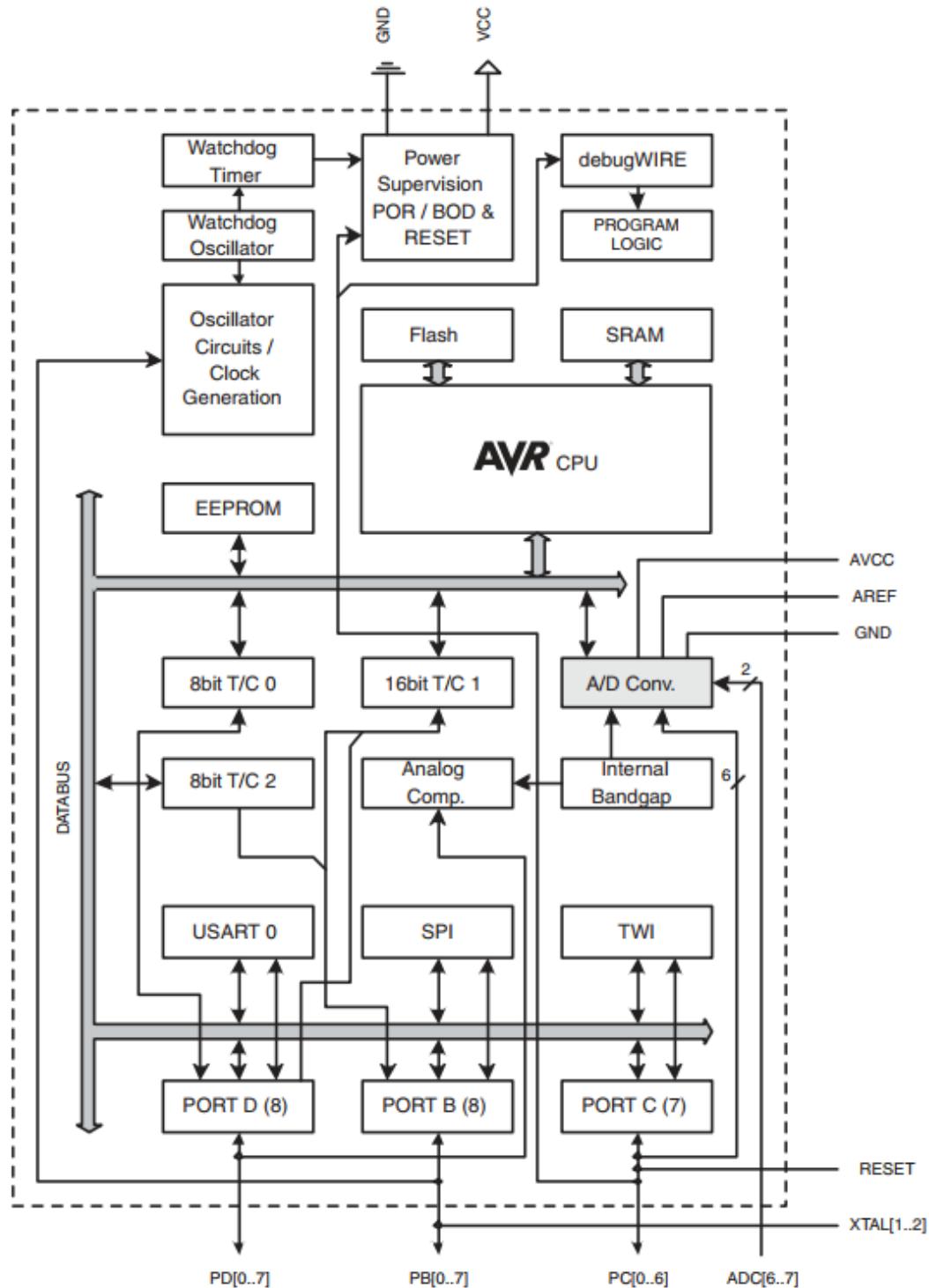
PORTC, và PORTD (một số dòng chip AVR khác có thể có 4 hoặc 6 PORT). Các PORT đóng vai trò như “cửa ngõ” quan trọng của vi điều khiển. Với 28 chân, ATMEGA328P được thiết kế để đảm nhiệm nhiều chức năng khác nhau.

Bảng 2.1. Chức năng của từng chân ATMEGA328P

Số thứ tự chân	Kí hiệu chân	Mô tả chức năng
1	PC6	Chân này được dùng để khởi động lại vi điều khiển. Khi chân reset này ở mức thấp, bộ vi điều khiển và chương trình của nó sẽ được khởi động lại
2	PD0	Chân RXD trong giao tiếp UART, dùng để nhận dữ liệu từ thiết bị ngoại vi.
3	PD1	Chân TXD trong giao tiếp UART, dùng để truyền dữ liệu đến thiết bị ngoại vi.
4	PD2	Chân I/O kỹ thuật số hoặc ngắt ngoài (INT0).
5	PD3	Chân I/O kỹ thuật số, ngắt ngoài (INT1) hoặc chức năng xuất tín hiệu PWM (OC2B).
6	PD4	Chân I/O kỹ thuật số hoặc làm chân ngõ vào xung nhịp T0 cho bộ đếm (Timer/Counter 0).
7	VCC	Chân cấp điện áp cho vi điều khiển (1.8V – 5.5V)
8	GND	Chân nối đất, dùng để nối với cực âm của nguồn điện.
9	PB6	Thường được dùng làm chân XTAL1 kết nối với thạch anh ngoại vi cho dao động (Oscillator) nhưng cũng có thể sử dụng làm I/O.
10	PB7	Thường được dùng làm chân XTAL2 kết nối với thạch anh ngoại vi cho dao động (Oscillator) nhưng cũng có thể sử dụng làm I/O.
11	PD5	Chân I/O kỹ thuật số hoặc chức năng xuất tín hiệu PWM (OC0B).
12	PD6	Chân I/O kỹ thuật số, chân đầu vào dương của bộ so sánh tương tự hoặc chức năng xuất tín hiệu PWM (OC0A).

13	PD7	Chân I/O kỹ thuật số hoặc chân đầu vào âm của bộ so sánh tương tự
14	PB0	Sử dụng cho I/O, chân đầu vào của chức năng Input Capture trong bộ Timer/Counter1 hoặc chức năng PWM (OC0A).
15	PB1	Sử dụng cho I/O hoặc chức năng PWM (OC1A).
16	PB2	Sử dụng cho I/O, làm chức năng xuất tín hiệu PWM (OC1B) hoặc chân SS (Slave Select) trong giao tiếp SPI.
17	PB3	Sử dụng cho I/O, làm chức năng xuất tín hiệu PWM (OC2A) hoặc làm chân MOSI (Master Out Slave In) trong giao tiếp SPI.
18	PB4	Sử dụng cho I/O hoặc làm chân MISO (Master In Slave Out) trong giao tiếp SPI.
19	PB5	Sử dụng cho I/O hoặc làm chân SCK (Serial Clock) trong giao tiếp SPI.
20	AVCC	Chân cấp điện áp dương cho bộ chuyển đổi ADC và các cổng I/O liên quan đến ADC (AVCC thường nối chung với VCC qua 1 bộ lọc để ổn định điện áp cho ADC)
21	AREF	Chân tham chiếu cho ADC. Có thể được kết nối với một nguồn điện áp tham chiếu ngoài hoặc sử dụng điện áp nội (AVCC).
22	GND	Chân nối đất, dùng để nối với cực âm của nguồn điện.
23	PC0	Sử dụng cho I/O hoặc chân đầu vào Analog (ADC0)
24	PC1	Sử dụng cho I/O hoặc chân đầu vào Analog (ADC1)
25	PC2	Sử dụng cho I/O hoặc chân đầu vào Analog (ADC2)
26	PC3	Sử dụng cho I/O hoặc chân đầu vào Analog (ADC3)
27	PC4	Sử dụng cho I/O, chân đầu vào Analog (ADC4) hoặc SDA (Serial Data) trong giao tiếp I2C.
28	PC5	Sử dụng cho I/O, chân đầu vào Analog (ADC5) hoặc SCL (Serial Clock) trong giao tiếp I2C.

2.3. Sơ đồ khói của ATMEGA328P



Hình 2.2. Sơ đồ khói ATMEGA328P [2]

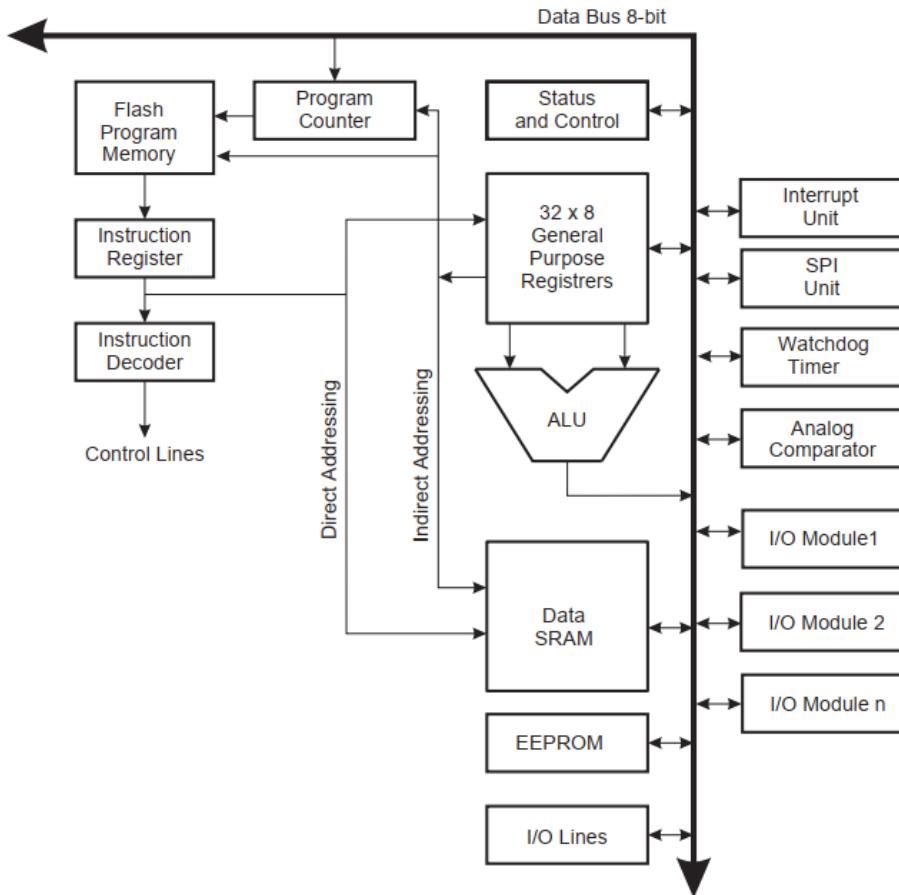
Lõi AVR được thiết kế với 32 thanh ghi làm việc đa dụng, tất cả đều được kết nối trực tiếp với đơn vị logic số học (ALU). Điều này cho phép truy cập đồng thời hai thanh ghi trong một lệnh duy nhất và thực thi trong một chu kỳ xung nhịp, giúp tăng hiệu suất mã hóa và tốc độ xử lý lên đến mươi lần so với các vi điều khiển CISC truyền thống.

Vi điều khiển ATMEGA328P tích hợp nhiều tính năng nổi bật như: bộ nhớ Flash có dung lượng 32K byte, 1K byte EEPROM, 2K byte SRAM, 23 chân I/O đa dụng, 32 thanh ghi làm việc, 3 bộ Timer/Counter với các chế độ so sánh linh hoạt, cùng khả năng ngắt bên trong và bên ngoài. Ngoài ra, thiết bị còn được trang bị USART, TWI (I2C), SPI, bộ chuyển đổi ADC 10-bit hỗ trợ 6 kênh (hoặc 8 kênh), bộ định thời Watchdog có thể lập trình kèm dao động nội. Trong chế độ Idle, CPU ngừng hoạt động nhưng SRAM, Timer/Counter, USART, TWI (I2C), SPI và ngắt ngoài vẫn hoạt động. Ở chế độ Power-down, nội dung các thanh ghi được giữ lại trong khi dao động và các chức năng khác của chip bị vô hiệu hóa cho đến khi có tín hiệu ngắt hoặc reset phần cứng. Chế độ Power-save cho phép bộ đếm thời gian bắt đồng bộ tiếp tục hoạt động để duy trì cơ sở thời gian, trong khi các phần còn lại của thiết bị ở trạng thái nghỉ. Chế độ giám nhiễu ADC tạm ngừng CPU và các mô-đun I/O ngoại trừ ADC và bộ đếm thời gian bắt đồng bộ, giúp giảm thiểu nhiễu trong quá trình chuyển đổi ADC. Chế độ Standby cho phép dao động tinh thể hoặc bộ cộng hưởng tiếp tục hoạt động, trong khi phần còn lại của thiết bị ở chế độ ngủ, hỗ trợ khởi động nhanh và tiết kiệm năng lượng.

ATMEGA328P được sản xuất bằng công nghệ bộ nhớ điện tĩnh mật độ cao, tích hợp bộ nhớ Flash ISP trên chip, hỗ trợ lập trình bộ nhớ chương trình qua giao diện SPI, trình lập trình truyền thông hoặc chương trình khởi động. Khu vực Flash khởi động (Boost Flash Section) cho phép chạy chương trình trong khi phần Flash ứng dụng được cập nhật, đảm bảo khả năng đọc-ghi đồng thời. Với CPU RISC 8-bit và bộ nhớ flash tự lập trình tích hợp trên một chip. Bên cạnh đó, vi điều khiển này còn được hỗ trợ bởi các công cụ phát triển phong phú như: trình biên dịch C (C Compilers), Macro Assembler, trình giả lập/chạy thử chương trình (Program Debugger/Simulators), trình giả lập trong mạch (In-Circuit Emulators – ICE), cùng các bộ kit đánh giá (Evaluation Kits).

2.4. Lõi CPU AVR

2.4.1. Tổng quan



Hình 2.3. Sơ đồ khái niệm về kiến trúc AVR [2]

Để tối ưu hóa hiệu suất và khả năng xử lý song song, lõi AVR được thiết kế theo kiến trúc Harvard, sử dụng bộ nhớ và bus riêng biệt cho chương trình và dữ liệu. Các lệnh trong bộ nhớ chương trình được thực hiện theo cơ chế đường ống đơn cấp, khi một lệnh đang được thực thi, lệnh kế tiếp được lấy từ bộ nhớ chương trình. Điều này cho phép mỗi lệnh được thực thi trong một chu kỳ xung nhịp. Bộ nhớ chương trình của AVR là Flash có thể tái lập trình trực tiếp trong hệ thống (In-System Reprogrammable Flash memory). Tệp thanh ghi với 32 thanh ghi đa dụng 8-bit được tối ưu hóa với thời gian truy cập chỉ một chu kỳ xung nhịp, giúp đơn

vị logic số học (ALU) hoạt động hiệu quả. Một phép toán ALU điển hình thực hiện việc xuất hai toán hạng từ tệp thanh ghi, xử lý phép toán và lưu kết quả trở lại thanh ghi – tất cả chỉ trong một chu kỳ. Sáu thanh ghi trong số này có thể kết hợp thành ba cặp con trỏ địa chỉ gián tiếp 16-bit, cho phép truy cập không gian dữ liệu một cách linh hoạt và hiệu quả. Một trong số các con trỏ này còn có thể sử dụng để định địa chỉ bảng tra cứu trong bộ nhớ chương trình Flash. Các thanh ghi X, Y và Z 16-bit này mở rộng khả năng định địa chỉ và hỗ trợ hiệu quả.

ALU cung cấp các phép toán số học và logic giữa các thanh ghi hoặc giữa một thanh ghi và hằng số. Ngoài ra, các phép toán trên một thanh ghi đơn cũng có thể được thực thi trong ALU. Sau mỗi phép toán số học, thanh ghi trạng thái (Status Register) sẽ được cập nhật để phản ánh thông tin về kết quả của phép toán. Chương trình được thực hiện bằng các lệnh nhảy (Jump) hoặc gọi (Call) theo điều kiện hoặc không điều kiện, với khả năng định địa chỉ trực tiếp toàn bộ không gian bộ nhớ. Hầu hết các lệnh AVR có định dạng 16-bit, trong khi một số lệnh đặc biệt có định dạng 32-bit. Bộ nhớ Flash chương trình được chia thành hai phần: khu vực chương trình khởi động (Boot Flash Section) và khu vực chương trình ứng dụng (Application Flash Section). Cả hai đều được bảo vệ bởi các bit khóa chuyên dụng để ngăn ghi hoặc đọc-ghi trái phép. Lệnh SPM được sử dụng để ghi vào bộ nhớ Flash chương trình ứng dụng, nhưng chỉ có thể được thực thi từ khu vực chương trình khởi động. Khi có ngắt hoặc gọi chương trình con, địa chỉ trả về của bộ đếm chương trình (Program Counter – PC) sẽ được lưu trên Stack. Stack được phân bổ trong SRAM dữ liệu chung, với kích thước phụ thuộc vào dung lượng SRAM và mức sử dụng hiện tại. Trước khi thực thi chương trình con hoặc xử lý ngắt, con trỏ Stack (Stack Pointer – SP) cần được khởi tạo trong quy trình Reset. Con trỏ SP có thể được đọc/ghi từ không gian I/O. SRAM dữ liệu hỗ trợ năm chế độ định địa chỉ khác nhau, mang lại sự linh hoạt trong việc truy cập bộ nhớ. Các không gian bộ nhớ trong kiến trúc AVR được tổ chức dưới dạng bản đồ tuyến tính và nhất quán.

Mô-đun ngắt linh hoạt có các thanh ghi điều khiển trong không gian I/O, với một bit kích hoạt ngắt toàn cục (Global Interrupt Enable) trong thanh ghi trạng thái. Mỗi ngắt được liên kết với một vector ngắt riêng trong bảng vector ngắt, ưu tiên thực hiện dựa trên vị trí của vector. Vector ngắt có địa chỉ thấp hơn được ưu tiên cao hơn. Không gian bộ nhớ I/O chứa 64

địa chỉ dành cho các chức năng ngoại vi của CPU như các thanh ghi điều khiển, SPI và các chức năng I/O khác. Bộ nhớ I/O có thể được truy cập trực tiếp hoặc như các địa chỉ trong không gian dữ liệu tiếp theo tệp thanh ghi từ 0x20 đến 0x5F. Ngoài ra, ATMEGA328P còn có không gian I/O mở rộng từ 0x60 đến 0xFF trong SRAM.

2.4.2. Thanh ghi trạng thái – SREG (Status Register)

Bit	7	6	5	4	3	2	1	0	
0x3F(0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – Kích hoạt ngắt toàn cục (Global Interrupt Enable): Bit kích hoạt ngắt toàn cục phải được thiết lập để cho phép các ngắt hoạt động. Sau khi kích hoạt ngắt toàn cục, việc kiểm soát từng ngắt riêng lẻ sẽ được thực hiện thông qua các thanh ghi điều khiển tương ứng. Nếu bit này bị xóa (cleared), toàn bộ các ngắt sẽ bị vô hiệu hóa, bất kể trạng thái của các ngắt riêng lẻ. Khi xảy ra ngắt, phần cứng sẽ tự động xóa bit I và nó sẽ được khôi phục thông qua lệnh RETI, nhằm đảm bảo khả năng xử lý các ngắt tiếp theo.

Bit 6 – Lưu trữ sao chép: Bit T được sử dụng bởi các lệnh thao tác bit như BLD (Bit Load) và BST (Bit Store), đóng vai trò là nguồn hoặc đích của các bit được thao tác. Lệnh BST cho phép sao chép một bit từ thanh ghi trong Register File vào bit T, trong khi lệnh BLD cho phép sao chép bit T vào một vị trí bit trong thanh ghi thuộc Register File.

Bit 5 – Half Carry Flag: Cờ này được sử dụng để chỉ ra rằng đã xảy ra một “mang nửa” (half carry) trong quá trình thực hiện phép toán số học BCD (Binary-Coded Decimal) giữa các số 4-bit thấp nhất của một byte. Điều này thường xảy ra khi kết quả của phép toán vượt quá giá trị 15 (0xF) trong nhóm 4 bit thấp.

Bit 4 – Bit dấu (Sign Bit), $S = N \oplus V$: Bit S luôn là kết quả của phép XOR (phép hoặc loại trừ) giữa cờ âm (Negative Flag) và cờ tràn bù 2 (Two's Complement Overflow Flag).

Bit 3 – Cờ tràn của bù 2 (Two's Complement Overflow Flag): Cờ tràn của bù 2 (V) hỗ trợ các phép toán số học theo hệ bù 2.

Bit 2 – Cờ âm (Negative Flag): Cờ âm (N) biểu thị kết quả âm trong một phép toán đại số hoặc logic.

Bit 1 – Cờ không (Zero Flag): Cờ không (Z) biểu thị kết quả bằng không trong một phép toán đại số hoặc logic.

Bit 0 – Cờ nhớ (Carry Flag): Cờ nhớ I là bit nhớ trong một phép toán số học hoặc logic.

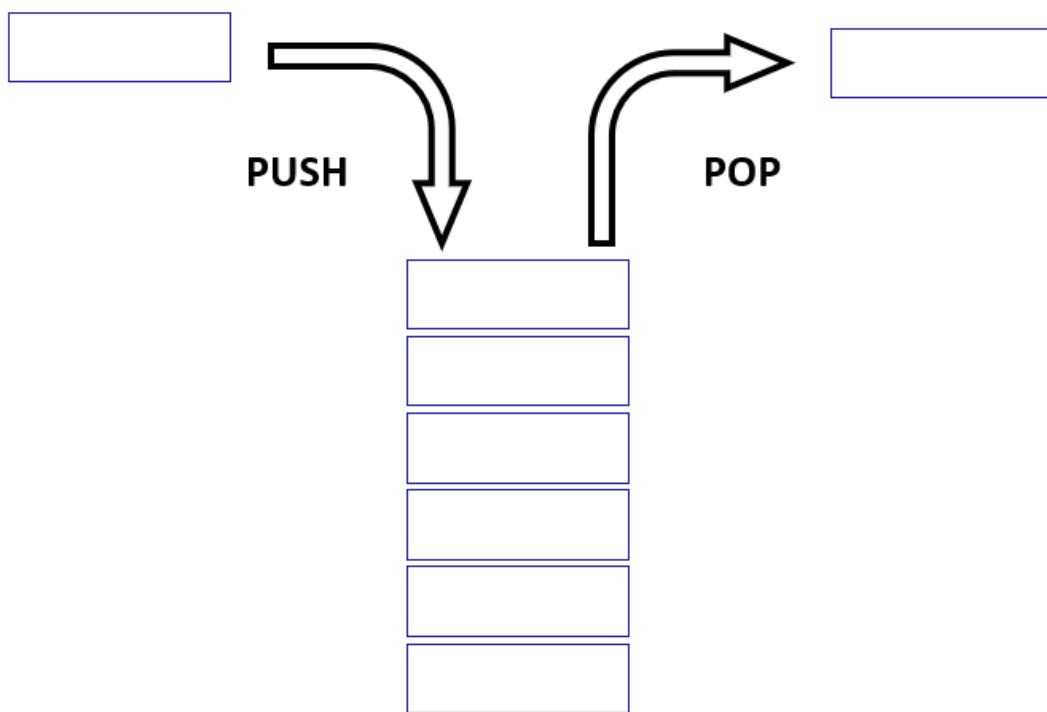
2.4.3. Stack Pointer

Stack là một khu vực trong bộ nhớ được sử dụng để lưu trữ dữ liệu tạm thời, các biến cục bộ, cũng như địa chỉ trả về khi xảy ra ngắt hoặc khi chương trình gọi các chương trình con (subroutine calls). Trong vi điều khiển AVR, Stack hoạt động theo cơ chế từ các địa chỉ bộ nhớ cao xuống thấp. Con trỏ Stack (Stack Pointer Register) luôn trỏ đến vị trí đỉnh của Stack tại mọi thời điểm.

Để sử dụng Stack trong AVR, cần khai báo một khu vực trong SRAM làm Stack. Điều này được thực hiện bằng cách thiết lập địa chỉ khởi đầu của Stack thông qua việc cấu hình con trỏ Stack (SP – Stack Pointer). Con trỏ SP sẽ trỏ đến vùng nhớ Stack trong SRAM, nơi lưu trữ thông tin tạm thời cho các chương trình con và ngắt. Trong AVR, con trỏ Stack được biểu diễn bởi hai thanh ghi 8-bit, gồm SPL (Stack Pointer Low) và SPH (Stack Pointer High). SPL chứa giá trị byte thấp, trong khi SPH chứa giá trị byte cao của địa chỉ SP. Hai thanh ghi này nằm trong vùng nhớ I/O. Mặc định, giá trị ban đầu của Stack Pointer thường được thiết lập bằng địa chỉ cuối cùng của bộ nhớ SRAM bên trong vi điều khiển. Trước khi sử dụng, cần cấu hình Stack Pointer sao cho nó trỏ đến vị trí bắt đầu của khu vực Stack trong SRAM. Điều này đảm bảo Stack hoạt động hiệu quả, hỗ trợ tốt cho chương trình con và xử lý ngắt.

Thanh ghi con trỏ Stack cao và thanh ghi con trỏ Stack thấp – SPH and SPL [2]:

Bit	15	14	13	12	11	10	9	8	
(0x3E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
(0x3D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	
	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	



Hình 2.4. Vùng nhớ Stack

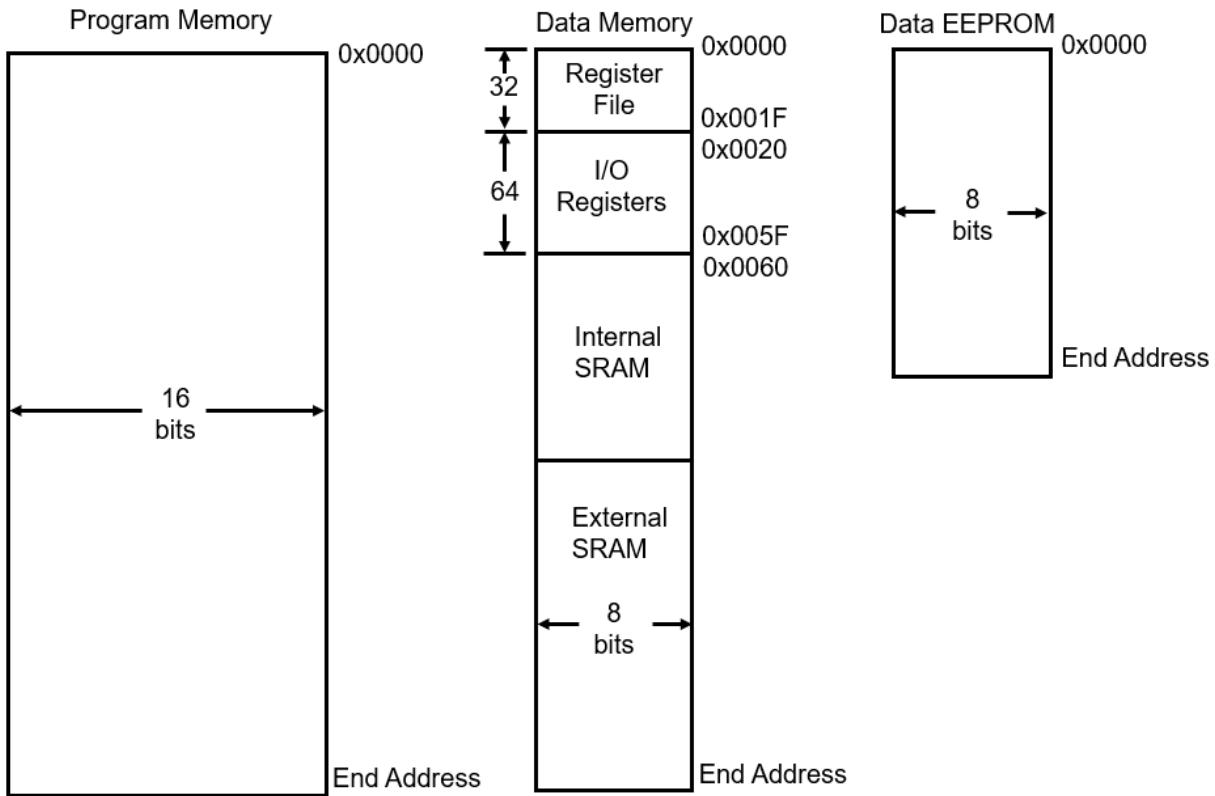
Hai lệnh chính để thao tác với Stack là PUSH và POP. Lệnh PUSH được sử dụng để đưa dữ liệu vào Stack, trong khi POP dùng để lấy dữ liệu từ Stack. Việc truy cập dữ liệu được thực hiện tại vị trí mà con trỏ Stack (SP) đang trỏ tới, đảm bảo dữ liệu được lưu trữ và truy xuất theo đúng thứ tự.

Bảng 2.2. Các lệnh liên quan đến Stack Pointer [3]

Lệnh	Stack Pointer	Mô tả
PUSH	Giảm 1	Dữ liệu được đẩy (push) vào Stack.
CALL	Giảm 2	Địa chỉ trả về được đẩy vào Stack khi thực hiện lệnh gọi chương trình con (subroutine call) hoặc ngắt (interrupt).
ICALL	Giảm 2	Địa chỉ trả về được đẩy vào Stack khi thực hiện lệnh gọi gián tiếp (indirect call).
RCALL	Giảm 2	Địa chỉ trả về được đẩy vào Stack khi thực hiện lệnh gọi tương đối (relative call).
POP	Tăng 1	Dữ liệu được lấy (pop) ra từ Stack.
RET	Tăng 2	Địa chỉ trả về được lấy ra từ Stack khi trở về từ chương trình con (subroutine).
RETI	Tăng 2	Địa chỉ trả về được lấy ra từ Stack khi trở về từ ngắt (interrupt).

2.5. Cấu trúc bộ nhớ của ATMEGA328P

ATMEGA328P có hai không gian bộ nhớ chính, bao gồm không gian bộ nhớ dữ liệu (Data Memory) và không gian bộ nhớ chương trình (Program Memory). Ngoài ra, ATMEGA328P còn có bộ nhớ EEPROM để lưu trữ dữ liệu. Cả ba không gian bộ nhớ này đều có cấu trúc tuyến tính và đồng nhất.

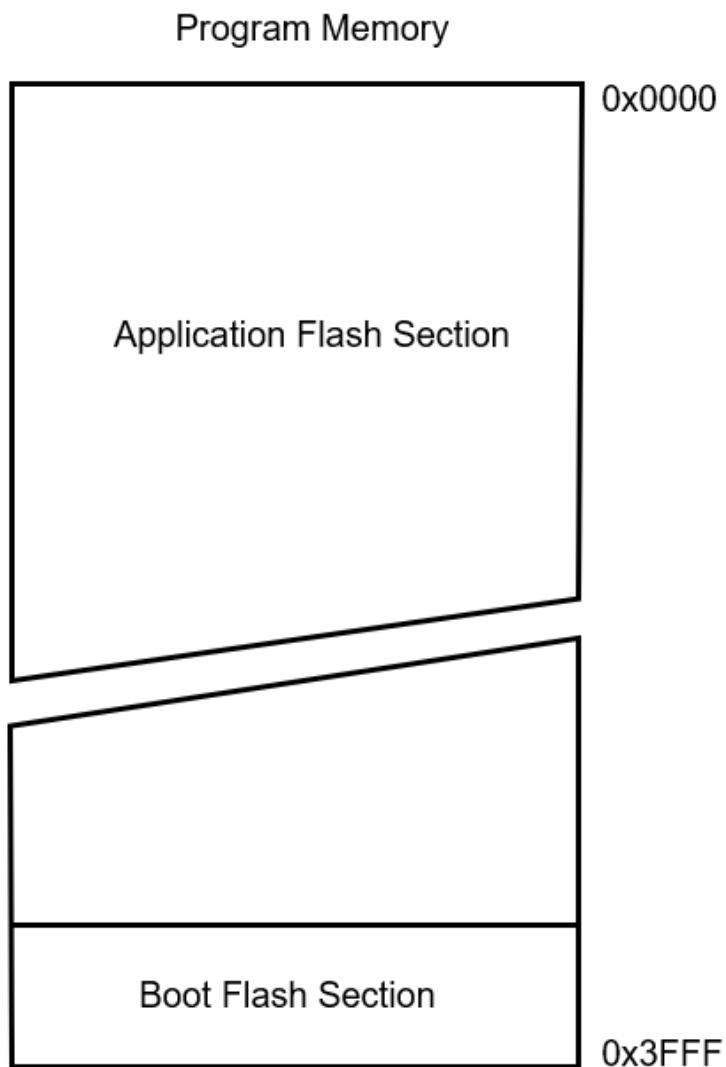


Hình 2.5. Tổ chức bộ nhớ của ATMEGA328P [4]

2.5.1. Bộ nhớ chương trình (Program Memory)

Bộ nhớ chương trình, hay bộ nhớ Flash là nơi lưu trữ mã chương trình mà vi điều khiển sẽ thực thi. Bộ nhớ Flash của ATMEGA328P có dung lượng 32KB, có thể lập trình lại nhiều lần và duy trì dữ liệu ngay cả khi mất nguồn. Trong các vi điều khiển AVR cũ, bộ nhớ chương trình chỉ có một phần duy nhất là Application Flash Section. Tuy nhiên, để nâng cao tính bảo mật phần mềm và tổ chức không gian bộ nhớ, các dòng vi điều khiển AVR mới như ATMEGA328P đã bổ sung thêm Boot Flash Section. Trong đó, Application Flash Section là phần chính của bộ nhớ chương trình, nơi chứa mã chương trình chính mà vi điều khiển sẽ thực thi bao gồm 2 phần: phần chứa các instruction (mã lệnh cho hoạt động của chip) và phần chứa các vector ngắn (interrupt vectors). Như trường hợp dưới đây thì các vector ngắn nằm ở phần đầu của Application Flash Section (từ địa chỉ 0x0000) và dài đến bao nhiêu tùy thuộc vào loại

chip. Ngay sau khu vực vector ngắn là vùng dành cho mã lệnh (instruction), nơi chứa chương trình được tải vào để vi điều khiển thực thi.



Hình 2.6. Bộ nhớ chương trình của ATMEGA328P [2]

- Application Flash Section là khu vực trong bộ nhớ Flash được sử dụng để lưu trữ mã chương trình ứng dụng. Đây là khu vực bộ nhớ chính mà CPU sẽ đọc và thực thi khi chip được khởi động. Đặc điểm của Application Flash Section:

- + Địa chỉ bộ nhớ: Bắt đầu từ 0x0000 và kéo dài tới vị trí gần Boot Flash Section.

+ Dung lượng: Phụ thuộc vào dung lượng Flash tổng (32 KB) và kích thước Boot Flash Section (có thể tùy chỉnh).

Ví dụ: Nếu Boot Flash Section chiếm 2 KB, thì Application Flash Section có 30 KB.

+ Thực thi chương trình: Khi chip được khởi động, CPU sẽ bắt đầu thực thi mã từ địa chỉ 0x0000 trong Application Flash Section.

+ Khả năng ghi/đọc: Có thể đọc/ghi thông qua ISP hoặc các giao thức khác như UART.

+ Bảo vệ bộ nhớ: Có thể thiết lập các fuse bits để bảo vệ Application Flash Section khỏi việc ghi đè hoặc đọc trái phép.

+ Khả năng sửa đổi động: Khi sử dụng Bootloader, chương trình trong Application Flash Section có thể được cập nhật mà không cần bộ lập trình bên ngoài.

- Boot Flash Section (bộ nhớ Flash khởi động): Trong vi điều khiển ATMEGA328P nó cung cấp một khu vực riêng biệt trong bộ nhớ Flash để lưu trữ và thực thi chương trình Bootloader, Bootloader là một đoạn mã nhỏ được thực thi đầu tiên khi vi điều khiển khởi động. Nhiệm vụ của nó thường là: Kiểm tra và nạp chương trình mới vào bộ nhớ Application Program Section thông qua giao thức như UART, SPI, I2C hoặc các giao thức khác mà không cần bộ lập trình ISP. Kích thước vùng Boot Flash Section có thể cấu hình được từ 256 bytes đến 2 KB bằng các fuse bits (BOOTSIZE). Vùng Boot Flash Section được bảo vệ bằng phần cứng và phần mềm để tránh bị ghi đè bởi chương trình trong Application Flash Section.

Khi ATMEGA328P khởi động, vị trí mà CPU bắt đầu thực thi chương trình phụ thuộc vào cấu hình fuse bits, cụ thể là bit BOOTRST (Bit BOOTRST được sử dụng để xác định địa chỉ khởi động của vector Reset).

Bảng 2.3. Boot Reset Fuse [2]

BOOTRST	Reset Address
1	Reset vector = Application reset (address 0x0000)

0	Reset vector = Boot loader reset
---	----------------------------------

- Khi BOOTRST không được lập trình (unprogrammed) (giá trị bit là 1), vector Reset sẽ trả đến địa chỉ 0x0000, CPU sẽ bắt đầu thực thi chương trình từ địa chỉ 0x0000 trong vùng Application Flash Section của bộ nhớ Flash. Hình 2.6 thuộc trường hợp này.

- Khi BOOTRST được lập trình (programmed) (giá trị bit là 0), vector Reset sẽ trả đến địa chỉ bắt đầu của Boot Flash Section, CPU sẽ bắt đầu thực thi chương trình từ vùng Boot Flash Section, được xác định bởi các tham số kích thước vùng Boot (BOOTSZ).

Điều này cho phép linh hoạt trong thiết kế, đặc biệt khi cần sử dụng Bootloader để cập nhật firmware hoặc kiểm soát quá trình khởi động.

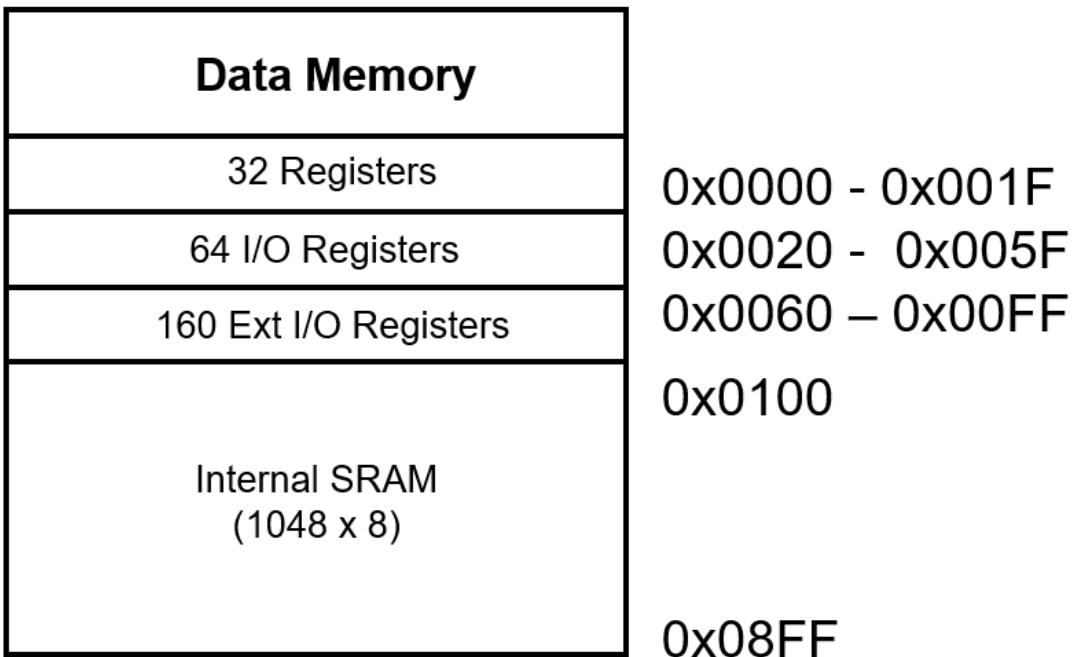
Bảng 2.4. Boot Size Configuration, ATMEGA328P [2]

BOOTSZ1	BOOTSZ0	Boot Size	Pages	Application Flash Section	Boot Loader Flash Section	End Application Section	Boot Reset Address
1	1	256 words	4	0x0000 – 0x3EFF	0x3F00 – 0x3FFF	0x3EFF	0x3F00
1	0	512 words	8	0x0000 – 0x3DFF	0x3E00 – 0x3FFF	0x3DFF	0x3E00
0	1	1024 words	16	0x0000 – 0x3BFF	0x3C00 – 0x3FFF	0x3BFF	0x3C00
0	0	2048 words	32	0x0000 – 0x37FF	0x3800 – 0x3FFF	0x37FF	0x3800

BOOTSZ1 và BOOTSZ0 là hai fuse bits dùng để chọn kích thước của Boot Flash Section trong bộ nhớ Flash của vi điều khiển. Boot Size biểu thị kích thước Boot Flash Section bằng số từ (mỗi từ là 16 bits). Kích thước này xác định số trang bộ nhớ Flash (Pages) tương ứng với Boot Flash Section. Application Flash Section là phạm vi địa chỉ bộ nhớ Flash dành cho mã ứng dụng, trong khi Boot Flash Section là phạm vi dành riêng cho Boot Loader. Địa

chỉ kết thúc của Application Flash Section được gọi là End Application Section. Nếu fuse BOOTRST được lập trình, vi điều khiển sẽ khởi động từ địa chỉ Boot Reset Address của Boot Flash Section thay vì từ địa chỉ mặc định của mã ứng dụng.

2.5.2. Bộ nhớ dữ liệu (Data Memory)



Hình 2.7. Bộ nhớ dữ liệu của ATmega328P [2]

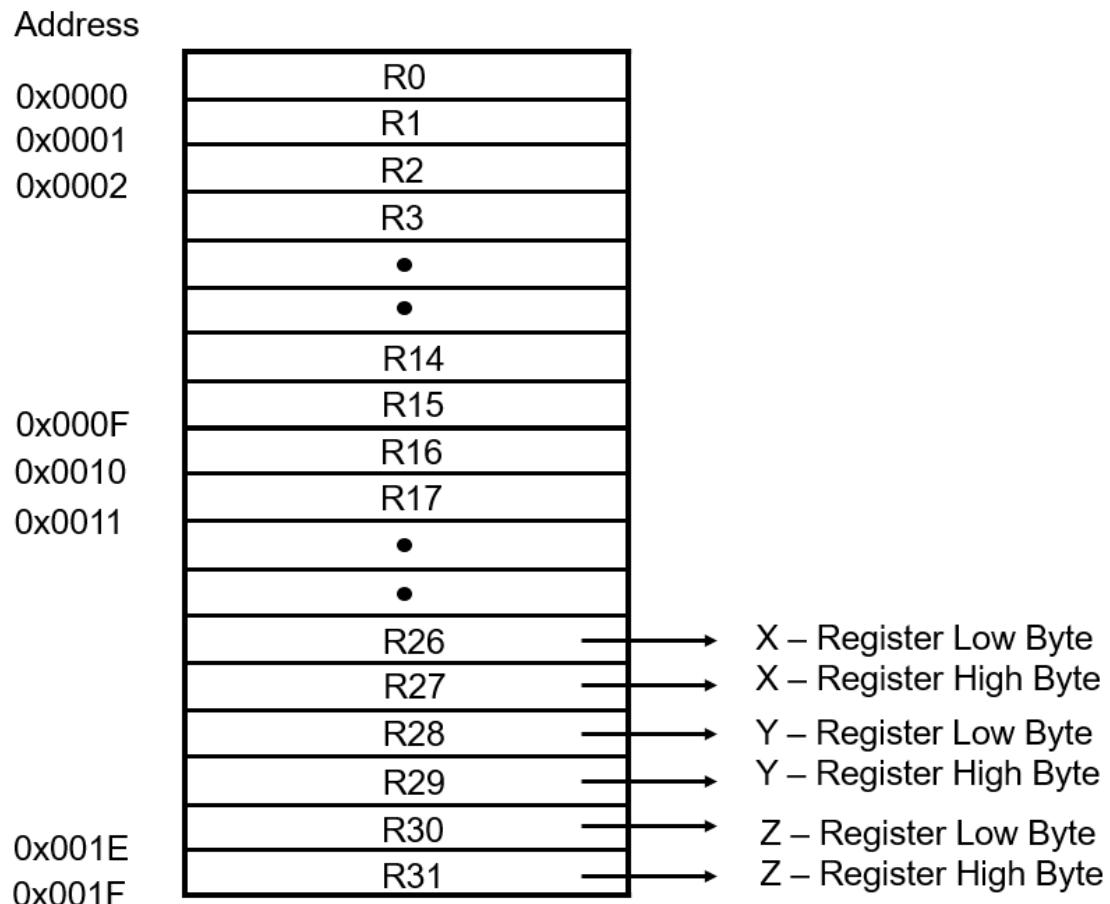
- Register File: là phần đầu tiên trong bộ nhớ dữ liệu, phần này bao gồm 32 thanh ghi có tên gọi là Register File (RF) hay General Purpose Register – GPR hoặc đơn giản là các thanh ghi. Tất cả các thanh ghi này đều là các thanh ghi 8 bit như hình bên dưới.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-------	-------	-------	-------	-------	-------	-------	-------

Hình 2.8. Cấu trúc của một thanh ghi 8 bit

Hộ chip AVR có 32 thanh ghi trong RF, với địa chỉ từ 0x0000 đến 0x001F. Mỗi thanh ghi, ký hiệu từ R0 đến R31, lưu trữ các giá trị không dấu từ 0 đến 255, giá trị có dấu từ -128 đến 127 hoặc mã ASCII của một ký tự bất kỳ. Các thanh ghi này được chia làm hai nhóm: nhóm đầu tiên gồm các thanh ghi từ R0 đến R15 và nhóm thứ hai gồm các thanh ghi từ R16

đến R31. Những đặc điểm chính của các thanh ghi này bao gồm: chúng có thể được truy cập trực tiếp trong các lệnh (instruction); các phép toán thực hiện trên các thanh ghi chỉ cần một chu kỳ xung clock và chúng được kết nối trực tiếp với CPU của chip, đảm nhiệm vai trò lưu trữ các số hạng đầu vào và kết quả đầu ra của các phép toán.



Hình 2.9. Register File [4]

Thanh ghi R0 là thanh ghi duy nhất được sử dụng trong lệnh LPM (Load Program Memory). Trong khi đó, các thanh ghi từ R26 đến R31 ngoài chức năng thông thường còn có thể hoạt động như các thanh ghi con trỏ (Pointer Register) để hỗ trợ truy xuất gián tiếp trong một số lệnh. Tóm lại, 32 thanh ghi Register File trong AVR được xem là một phần của CPU, cho phép CPU sử dụng chúng một cách trực tiếp và nhanh chóng. Khi gọi các thanh ghi này,

thay vì sử dụng địa chỉ, chúng ta chỉ cần gọi trực tiếp theo tên. Trong lập trình, Register File thường được sử dụng làm toán hạng cho các phép toán.

- I/O register: là phần tiếp nối ngay sau Register File là một khu vực gồm 64 thanh ghi nhập/xuất (I/O Register) hay còn gọi là vùng nhớ I/O (I/O Memory). Đây chính là điểm kết nối giữa CPU và các thiết bị ngoại vi. Tất cả các thanh ghi điều khiển và trạng thái của các thiết bị ngoại vi đều được lưu trữ trong khu vực này. Đối với việc điều khiển các PORT trên AVR, người dùng sẽ làm việc với ba thanh ghi chính: DDRx, PORTx, và PINx, tất cả đều thuộc vùng nhớ I/O. Ngoài ra, các thiết bị ngoại vi khác như Timer, bộ chuyển đổi Analog/Digital hoặc giao tiếp USART cũng được điều chỉnh thông qua các thanh ghi trong khu vực này. Vùng nhớ I/O có thể được truy cập theo hai cách: dưới dạng SRAM hoặc dưới dạng các thanh ghi I/O. Khi sử dụng các lệnh truy xuất SRAM, địa chỉ của vùng này được quy định từ 0x0020 đến 0x005F. Ngược lại, khi truy xuất như các thanh ghi I/O, địa chỉ sẽ nằm trong từ 0x0000 đến 0x003F.

- Internal SRAM: RAM tĩnh nội là khu vực bộ nhớ được sử dụng để lưu trữ các biến tạm thời hoặc toàn cục trong quá trình chương trình chạy. Tương tự như RAM trên máy tính, vùng nhớ này có dung lượng nhỏ (chỉ khoảng vài KB) tùy thuộc vào từng loại vi điều khiển.

- External SRAM: RAM ngoại là vùng bộ nhớ ngoài có thể được bổ sung nếu người dùng cần thêm dung lượng để lưu trữ biến. Tính năng này chỉ khả dụng khi bộ nhớ ngoài được gắn vào vi điều khiển.

- EEPROM (Electrically Erasable Programmable ROM): là một thành phần quan trọng trong các dòng vi điều khiển AVR mới. Đây là loại bộ nhớ ROM, có khả năng giữ nguyên dữ liệu ngay cả khi không có nguồn cấp, rất phù hợp cho các ứng dụng yêu cầu lưu trữ dữ liệu lâu dài. Như trong Hình 2.5, phần bộ nhớ EEPROM được tách riêng và có địa chỉ từ 0x0000.

CHƯƠNG 3. NGÔN NGỮ LẬP TRÌNH

3.1. Cấu trúc chương trình CAVR

Cấu trúc một chương trình CAVR gồm có :

- Khai báo tiền xử lý.
- Khai báo các biến toàn cục.
- Khai báo các hàm.
- Chương trình chính.

Trong một chương trình CAVR thường sẽ có thêm các thành phần như: chú thích (comments), biểu thức (expressions), câu lệnh (statements), khối (blocks), cấu trúc điều khiển (flow controls),...

- + Chú thích (Comments): có 2 cách để tạo phần chú thích trong C là chú thích từng dòng bằng 2 dấu “//”
- + Hoặc chú thích block bằng cách kẹp block cần chú thích vào giữa /* */

Ví dụ :

1	// day la chu thich
2	
3	/*
4	Ghi chu thich o day
5	*/

- + Tiền xử lí (Preprocessor): là một tiện ích của ngôn ngữ C, các preprocessor được trình biên dịch xử lý trước khi xử lý các phần khác của chương trình. Các tiền xử lí luôn được bắt đầu bằng dấu “#”, trong số các preprocessors trong ngôn ngữ C có 2 preprocessors được sử dụng phổ biến nhất là #include và #define. Preprocessor #include chỉ định 1 file được đính kèm trong quá trình biên dịch và #define để định nghĩa 1 chuỗi thay thế hoặc 1 macro. Các

chỉ thị tiền xử lý giúp ta viết chương trình ngắn gọn hơn và tổ chức biên dịch, gỡ rối chương trình linh hoạt, hiệu quả hơn.

Ví dụ :

1	#include <mega328.h>
2	#define Pi 3.14

+ Biểu thức (Expressions): là một thành phần của câu lệnh, bao gồm các biến, toán tử, hoặc gọi hàm,... và luôn trả về một giá trị duy nhất. Tuy nhiên, biểu thức không phải là một câu lệnh đầy đủ.

+ Câu lệnh (Statement): thường là 1 dòng lệnh hoàn chỉnh, có thể chứa các từ khoá, biểu thức, các câu lệnh khác và được kết thúc bằng dấu “ ; ”.

Ví dụ :

1	unsigned char val=1; val*=2;
---	------------------------------

+ Khối (Blocks): là sự kết hợp của nhiều câu lệnh để thực hiện chung 1 nhiệm vụ nào đó, khối được bao bởi 2 dấu mở khối “{“ và đóng khối “}”

+ Cấu trúc điều khiển (Flow controls) :

❖ **Cấu trúc if**

1	if (Điều kiện)
2	{
3	Khối lệnh ;
4	}

Nếu điều kiện là đúng thì thực hiện khối lệnh sau, khối lệnh có thể được trình bày cùng dòng hoặc dòng sau if. Điều kiện có thể là một biểu thức bất kỳ, có thể là sự kết hợp của nhiều điều kiện bằng các toán tử quan hệ AND (&&), OR (||)... Điều kiện được cho là đúng khi nó khác 0.

❖ Câu trúc if Else

```
1   if (Điều kiện){  
2       Khối lệnh 1;  
3   }  
4   else{  
5       Khối lệnh 2;  
6   }
```

Nếu điều kiện đúng thì thực hiện khối lệnh 1, ngược lại thực hiện khối lệnh 2. Việc đặt các khối lệnh if và else trên cùng 1 dòng hay trên những dòng khác nhau đều không ảnh hưởng đến kết quả. Ngoài ra, cũng có thể đặt nhiều câu trúc if.....else..... lồng vào nhau.

❖ Câu trúc switch

```
1   switch (Biểu thức) {  
2       case hằng số_1 :  
3           các khối lệnh 1;  
4           break;  
5       case hằng số_2:  
6           các khối lệnh 2;  
7           break;  
8       ...  
9       default:  
10          các khối lệnh khác;  
11      }
```

Trong trường hợp có nhiều khả năng có thể xảy ra cho 1 biểu thức (hay 1 biến), ứng với mỗi khả năng bạn cần chương trình thực hiện một việc nào đó, khi này bạn nên sử dụng câu trúc switch. Break được dùng để thoát khỏi câu trúc điều khiển hiện tại ngay lập tức, switch sẽ kết thúc mà không cần xét đến các trường hợp khác.

❖ Câu trúc while

```
1   while (Điều kiện )  
2   {  
3       Khối lệnh ;  
4   }
```

Là một cấu trúc lặp (Loop) không biết trước số lần lặp, ý nghĩa của cấu trúc while là khi điều kiện còn đúng thì sẽ thực hiện khối lệnh. Cần phải cẩn trọng khi sử dụng vì rất dễ rơi vào một vòng lặp “không lối thoát” với while nếu điều kiện luôn luôn đúng.

❖ Cấu trúc for

1	<code>for (biểu_thức_1; biểu_thức_2; biểu_thức_3)</code>
2	{
3	Khối lệnh;
4	}

Hàm for có chức năng làm một vòng lặp, hiểu một cách đơn giản nó làm đi làm lại một công việc có một tính chất chung nào đó. Trong cấu trúc for, biểu_thức_1 thường được hiểu là khởi tạo, biểu_thức_2 là điều kiện và biểu_thức_3 là biểu thức được thực hiện sau. Các biểu thức trong cấu trúc for có thể bị lược bỏ khi sử dụng nhưng các dấu “;” thì vẫn phải giữ lại. Trong trường hợp viết theo cú pháp for(; ;) sẽ tương đương với vòng lặp vô tận while (1).

❖ Cấu trúc dowhile....

1	<code>do{</code>
2	Khối lệnh ;
3	}
4	<code>While (Điều kiện);</code>

Chức năng của vòng lặp do-while hoàn toàn giống vòng lặp while chỉ trừ có một điều là điều kiện điều khiển vòng lặp được tính toán sau khi khối lệnh được thực hiện. Vì vậy, khối lệnh sẽ được thực hiện ít nhất một lần ngay cả khi điều kiện không bao giờ được thỏa mãn.

❖ Lệnh break

1	<code>break;</code>
---	---------------------

Khi gặp câu lệnh break trong một vòng lặp, vòng lặp bị kết thúc ngay lập tức và câu lệnh kế tiếp sau vòng lặp được thực thi. Lệnh break có thể được sử dụng để kết thúc một case trong câu lệnh switch. Nếu sử dụng vòng lặp lồng nhau, câu lệnh break sẽ dừng việc thực hiện vòng lặp trong cùng và bắt đầu thực hiện câu lệnh kế tiếp sau vòng lặp trong cùng.

❖ Lệnh continue

1	continue;
---	------------------

Hoạt động giống như câu lệnh break. Thay vì buộc kết thúc vòng lặp, nó buộc trở về kiểm tra điều kiện để thực hiện vòng lặp tiếp theo và bỏ qua các lệnh bên trong vòng lặp hiện tại sau lệnh continue. Đối với vòng lặp for, câu lệnh continue làm cho chương trình tăng hoặc giảm biến đếm của vòng lặp. Đối với vòng lặp while và do-while thì câu lệnh continue làm cho chương trình quay về đầu vòng lặp và kiểm tra điều kiện của vòng lặp.

❖ Lệnh goto

1	identifier :
2	Khối lệnh;
3	goto identifier ;

Lệnh goto làm cho chương trình nhảy đến một nhãn identifier. Nhãn là một tên hợp lệ trong C, theo sau bởi dấu “ : ”. Lệnh goto làm cho cấu trúc chương trình trở nên khó theo dõi, tuy nhiên trong nhiều trường hợp nó làm cho chương trình thực hiện nhanh hơn và giảm kích thước chương trình.

❖ Lệnh return

1	// có hai dạng trả về
2	return; // dạng 1
3	return value; // dạng 2

Return có nhiệm vụ trả về một giá trị (cùng kiểu dữ liệu với hàm) mà nó được gọi. Đôi khi câu lệnh này cũng được sử dụng như một hành động kết thúc cho một đoạn chương trình đang được thực thi.

Dạng đầu tiên của câu lệnh return được sử dụng để kết thúc hàm và chuyển điều khiển cho hàm đang gọi. Không có giá trị nào từ hàm được gọi trả về khi sử dụng dạng return này.

Dạng thứ hai của câu lệnh return value được sử dụng để trả về các giá trị từ một hàm.

+ Biểu thức trả về sau có thể là bất kỳ lệnh gọi hàm, biến, hằng số, mảng, con trỏ...

Main: một chương trình C cho AVR phải bao gồm 1 chương trình chính main, tất cả các nội dung chính sẽ được đặt bên trong chương trình chính. Cấu trúc chương trình chính có thể như sau:

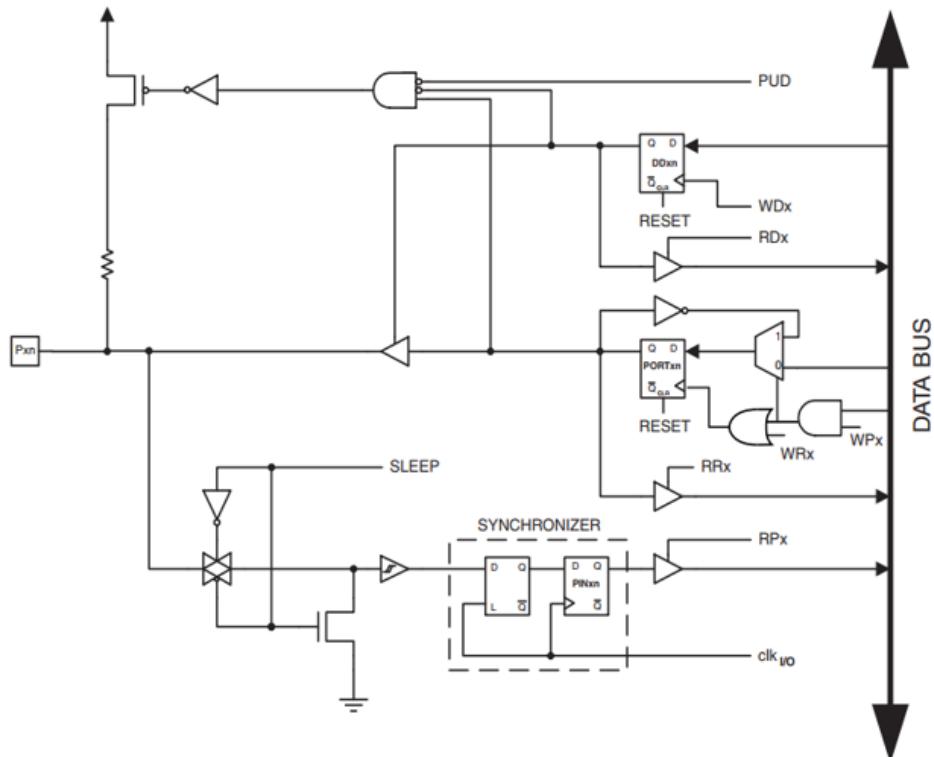
```

1 int main(void)
2 {
3     // noi dung chuong trinh chinh
4     return 0 // gia tri tra ve cho chuong trinh chinh
5 }
```

Trong đó, int là kiểu giá trị trả về của main, từ khoá void để chương trình chính không cần bất kỳ tham số nào kèm theo.

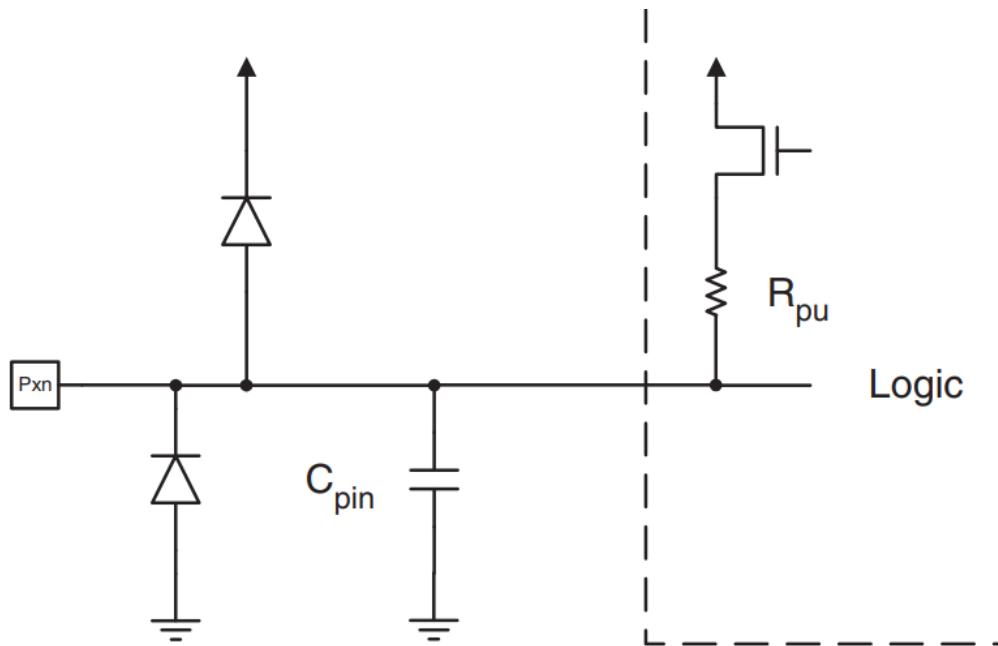
3.2. Lập trình xuất nhập PORT

Mô hình mô tả chức năng của một chân cổng I/O, được gọi chung là Px_n.



Hình 3.1. General Digital I/O [2]

Tất cả các cổng AVR có chức năng đọc – ghi thực sự khi được sử dụng làm cổng I/O kỹ thuật số chung. Điều này có nghĩa là hướng của một chân cổng có thể thay đổi mà không vô tình thay đổi hướng của bất kỳ chân nào khác. Tất cả các chân cổng đều có điện trở kéo lên có thể chọn riêng lẻ với điện trở không phụ thuộc vào điện áp cung cấp. Trong mạch điện Hình 3.2, các diode và tụ điện chỉ có chức năng bảo vệ chân PORT, nhưng điện trở R_{pu} (R Pull up) đóng vai trò quan trọng như là điện trở kéo lên khi chân của PORT làm nhiệm vụ nhận tín hiệu (ngõ nhập). Tuy nhiên trong AVR, điện trở kéo lên này không phải luôn kích hoạt, chúng ta biết rằng mỗi PORT của AVR có 3 thanh ghi : DDRx, PORTx, PINx, nếu DDRx=0 thì PORTx là ngõ nhập, lúc này thanh ghi PINx là thanh ghi chưa dữ liệu nhận về, đặc biệt thanh ghi PORTx vẫn được sử dụng trong mode này, đó là thanh ghi xác lập điện trở kéo lên, như thế nếu DDRx=0 và PORTx=0xFF thì các chân PORTx là ngõ nhập và được kéo lên bởi 1 điện trở trong chip, nghĩa là các chân của PORTx luôn ở mức cao, muốn kích để thay đổi trạng thái chân này chúng ta cần nối chân đó trực tiếp với GND, đây là lý do tại sao các button trong mạch điện của chúng ta có 1 đầu nối với chân của chip còn đầu kia được nối với GND. Đây cũng là ý nghĩa của khái niệm điện trở kéo lên (Pull up resistor) trong kỹ thuật điện tử.



Hình 3.2. I/O Pin Equivalent Schematic [2]

Tất cả các thanh ghi và tham chiếu bit trong phần này được viết dưới dạng tổng quát. Chữ “x” viết thường đại diện cho ký hiệu của cổng, và chữ “n” viết thường đại diện cho số thứ tự của bit. Tuy nhiên, khi sử dụng các định nghĩa thanh ghi hoặc bit trong chương trình, phải dùng đúng dạng cụ thể. Ví dụ, PORTB3 để chỉ bit số 3 của cổng B, trong khi ở đây được ghi tổng quát là PORTxn.

Ba vị trí địa chỉ bộ nhớ I/O được cấp phát cho mỗi cổng, lần lượt cho thanh ghi dữ liệu – PORTx, thanh ghi hướng dữ liệu – DDRx, và các chân đầu vào của cổng – PINx. Vị trí I/O của các chân đầu vào cổng (PINx) chỉ có thể đọc, trong khi thanh ghi dữ liệu (PORTx) và thanh ghi hướng dữ liệu (DDRx) có thể đọc/ghi. Tuy nhiên, khi ghi một giá trị logic 1 vào một bit trong thanh ghi PINx sẽ làm đảo ngược bit tương ứng trong thanh ghi dữ liệu. Ngoài ra, bit tắt kéo lên – PUD trong thanh ghi MCUCR sẽ vô hiệu hóa chức năng kéo lên cho tất cả các chân trong mọi cổng khi được thiết lập.

- Thanh ghi MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35(0x55)	-	BODS	BODSE	PUD	-	-	IVSEL	IVCE	MCUCR
Read/Write	R	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Bit 4 – PUD: Tắt Pull-up (Pull-up Disable) Khi bit này được ghi là 1, các điện trở pull-up trong các cổng I/O sẽ bị vô hiệu hóa ngay cả khi các thanh ghi DDxn và PORTxn được cấu hình để kích hoạt pull-up.

3.3. Định nghĩa số hằng, khai báo biến, toán tử C, cú pháp C

3.3.1. Định nghĩa hằng

Số hằng là các giá trị không thay đổi trong suốt thời gian chạy chương trình. Chúng có thể là số nguyên, số thực, hoặc chuỗi ký tự. Để định nghĩa hằng số trong C, có hai cách chính:

- Sử dụng từ khóa *const* :

+ Khai báo hằng bằng cách sử dụng từ khóa const. Khi một biến được khai báo với const, giá trị của nó không thể thay đổi.

Ví dụ : const int max = 100; // max là một hằng số có giá trị bằng 100

- Sử dụng #define :

+ Sử dụng tiền xử lý #define để định nghĩa hằng . Lưu ý rằng #define chỉ thay thế văn bản trước khi biên dịch, không tạo ra biến thực sự.

Ví dụ:

1	#define Pi 3.14159 // Định nghĩa hằng số Pi
2	#define max 100 // Định nghĩa hằng số max là 100

3.3.2. Khai báo biến

Biến là một vùng nhớ được đặt tên mà có thể lưu trữ và thay đổi giá trị trong khi chương trình chạy. Biến cần được khai báo trước khi sử dụng. Mỗi biến có một kiểu dữ liệu xác định, như số nguyên, số thực, ký tự,...

Cú pháp khai báo biến :

data_type variable_name = value

Trong đó : data_type là kiểu dữ liệu của biến

variable_name là tên của biến do người dùng đặt

value là giá trị khởi tạo (có thể có hoặc không)

Ví dụ :

1	int x = 10; //Biến số nguyên x và gán giá trị 10
2	float y = 5.5; //Biến số thực y và gán giá trị 5.5
3	char c = 'A'; //Biến ký tự c với giá trị 'A'

- Quy tắc đặt tên biến :

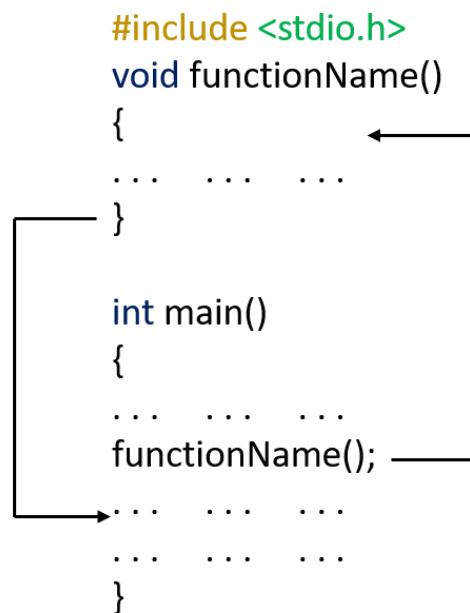
+ Chỉ nên chứa: chữ cái, số, dấu gạch chân (_).

- + Bắt đầu với: chữ cái hoặc chỉ dấu gạch chân (_).
- + Không thể bắt đầu với con số.
- + Không dấu cách.
- + Không đặt tên biến trùng với keyword.

Bảng 3.1. Các từ khóa trong ngôn ngữ C

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

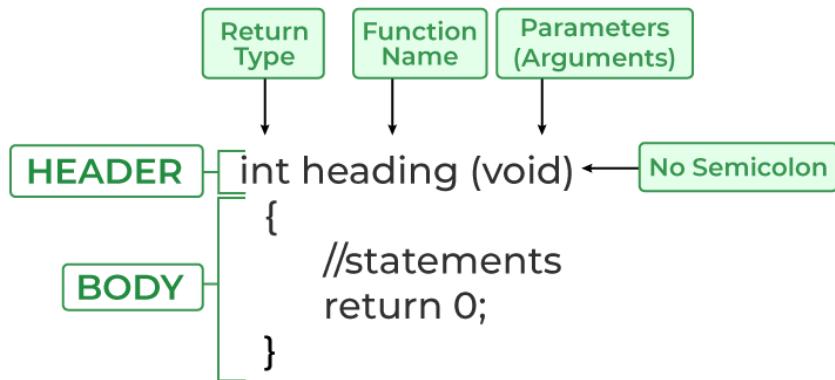
3.3.3. Định nghĩa và cú pháp khai báo hàm



Hình 3.3. Nguyên lý hoạt động của hàm trong ngôn ngữ C

Hàm trong lập trình C dành cho AVR là một nhóm các câu lệnh khác nhau kết hợp thành một và đặt tên cho nó. Sau khi tạo và đặt tên cho một hàm, chúng ta chỉ cần gọi tên hàm ra mỗi khi cần sử dụng đến nó trong chương trình.

Cú pháp khai báo hàm:



Hình 3.4. Thành phần mặc định của hàm

Trong đó:

Return Type: là kiểu dữ liệu trả về

Function Name: là tên hàm

Parameter: là tham số truyền vào, nếu không có tham số sẽ để là void

Body: phần chương trình của hàm

3.4.2. Các toán tử trong ngôn ngữ lập trình C

Bảng 3.2. Các toán tử đại số [5]

Toán tử	Tên	Ví dụ	Định nghĩa
*	Nhân	x*y	x nhân với y
/	Chia lấy phần nguyên	x/y	x chia y
%	Chia lấy phần dư	x%y	Provide the remainder of x divided by y
+	Cộng	x+y	x cộng y

-	Trừ	x-y	x trừ y
++	Tăng	x++	x tăng thêm 1 sau khi sử dụng nó
--	Giảm	--x	x giảm đi 1 trước khi sử dụng nó

Bảng 3.3. Toán tử truy cập và kích thước [5]

Toán tử	Tên	Ví dụ	Định nghĩa
[]	Phần tử mảng	X[6]	Phần tử thứ 7 của mảng X
.	Chọn thành viên	PORTB.x	Bit x của Port B
->	Chọn thành viên	pStruct -> x	Thành viên x của cấu trúc được con trỏ pStruct trỏ tới
*	Gián tiếp(Indirection)	*p	Nội dung của địa chỉ được lưu trong con trỏ p
&	Địa chỉ	&x	Địa chỉ của biến x

Bảng 3.4. Toán tử logic và quan hệ [5]

Toán tử	Tên	Ví dụ	Định nghĩa
>	Lớn hơn	x>y	1 nếu x lớn hơn y, ngược lại 0
>=	Lớn hơn bằng	x>=y	1 nếu x lớn hơn bằng y, ngược lại 0
<	Nhỏ hơn	x<y	1 nếu x nhỏ hơn y , ngược lại 0
<=	Nhỏ hơn bằng	x<=y	1 nếu x nhỏ hơn bằng y , ngược lại 0
==	Bằng	x==y	1 nếu x bằng y , ngược lại 0

$!=$	Không bằng	$x!=y$	1 nếu x không bằng y , ngược lại 0
!	NOT	$!x$	1 nếu x là 0, ngược lại 0
$&&$	AND	$x\&\&y$	0 nếu x hoặc y là 0, ngược lại 1
$ $	OR	$x y$	0 nếu cả x và y là 0, ngược lại 1

Bảng 3.5. Toán tử thao tác Bit [5]

Toán tử	Tên	Ví dụ	Định nghĩa
\sim	Đảo Bit	$\sim x$	Đảo bit 1 thành bit 0 và ngược lại
$\&$	AND	$x\&y$	Bit của số kết quả sẽ bằng 1 nếu cả hai bit tương ứng của x và y đều là 1
$ $	OR	$x y$	Bit của số kết quả sẽ bằng 1 nếu một trong hai bit tương ứng của x và y là 1.
$^$	XOR	x^y	Bit của số kết quả sẽ bằng 1 nếu hai bit tương ứng của x và y khác nhau, và ngược lại.
$<<$	Dịch trái	$x<<2$	Dịch các bit của một số sang trái. Các bit ở phía bên trái sẽ bị loại bỏ và các bit 0 sẽ được điền vào các vị trí trống.
$>>$	Dịch phải	$x>>3$	Dịch các bit của một số sang phải. Các bit ở phía bên phải sẽ bị loại bỏ và các bit 0 sẽ được điền vào các vị trí trống.

3.4.3. Các loại hàm

Hàm cơ bản: Hàm cơ bản là một đoạn mã thực hiện một tác vụ cụ thể. Nó thường được sử dụng để tách riêng các đoạn mã lặp lại nhằm tăng tính tổ chức và dễ bảo trì cho chương trình. Hàm cơ bản không nhận tham số, không có giá trị trả về và chỉ thực hiện các thao tác cố định.

Hàm với tham số: Hàm nhận một hoặc nhiều tham số để thực hiện tác vụ dựa trên giá trị đầu vào. Tham số giúp hàm trở nên linh hoạt và tái sử dụng cho nhiều trường hợp khác nhau.

Hàm có giá trị trả về: Hàm thực hiện tính toán hoặc xử lý, sau đó trả về một giá trị cụ thể bằng từ khóa return, loại dữ liệu trả về được chỉ định khi khai báo hàm. Giá trị này có thể được sử dụng trong chương trình chính hoặc các hàm khác.

Hàm không có tham số: Hàm không nhận bất kỳ tham số nào, thường được sử dụng để thực hiện một nhiệm vụ cố định hoặc độc lập với dữ liệu đầu vào, phù hợp cho các tác vụ cố định.

Hàm đệ quy: Hàm gọi lại chính nó để giải quyết bài toán bằng cách chia nhỏ thành các bước đơn giản hơn. Tuy nhiên, cần có điều kiện dừng để tránh vòng lặp vô hạn. Trong AVR, cần hạn chế vì có thể gây tràn stack.

Hàm với con trỏ: Hàm sử dụng con trỏ để thay đổi giá trị của biến được truyền vào. Điều này cho phép hàm làm việc trực tiếp trên dữ liệu gốc.

Hàm cục bộ: Hàm cục bộ chỉ được sử dụng trong phạm vi file nơi nó được định nghĩa, thường đi kèm với từ khóa static.

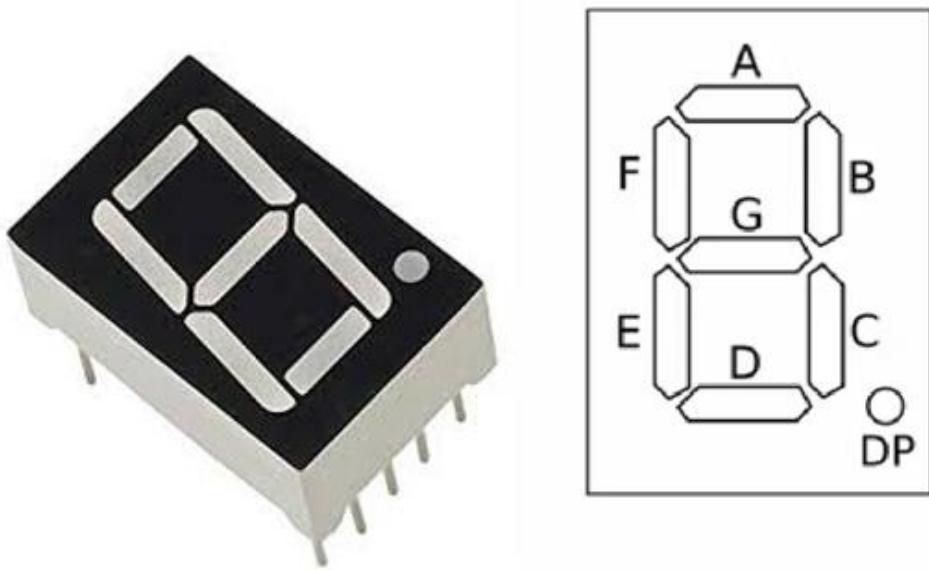
Hàm ngắt: Hàm ngắt được sử dụng làm trình xử lý ngắt (Interrupt Service Routine – ISR), hàm sẽ được gọi tự động khi một ngắt xảy ra.

CHƯƠNG 4. LẬP TRÌNH HIỂN THỊ

4.1. Lập trình hiển thị LED 7 đoạn

4.1.1. Giới thiệu

Trong các thiết bị, khi cần thông báo trạng thái hoạt động cho người dùng thông qua các giá trị số đơn giản, LED 7 đoạn thường được sử dụng. Ví dụ, nó có thể hiển thị nhiệt độ phòng hoặc đếm số lượng sản phẩm đã được kiểm tra sau một giai đoạn nào đó. LED 7 đoạn rất phổ biến vì nó có thể hiển thị các con số một cách rõ ràng và dễ hiểu.



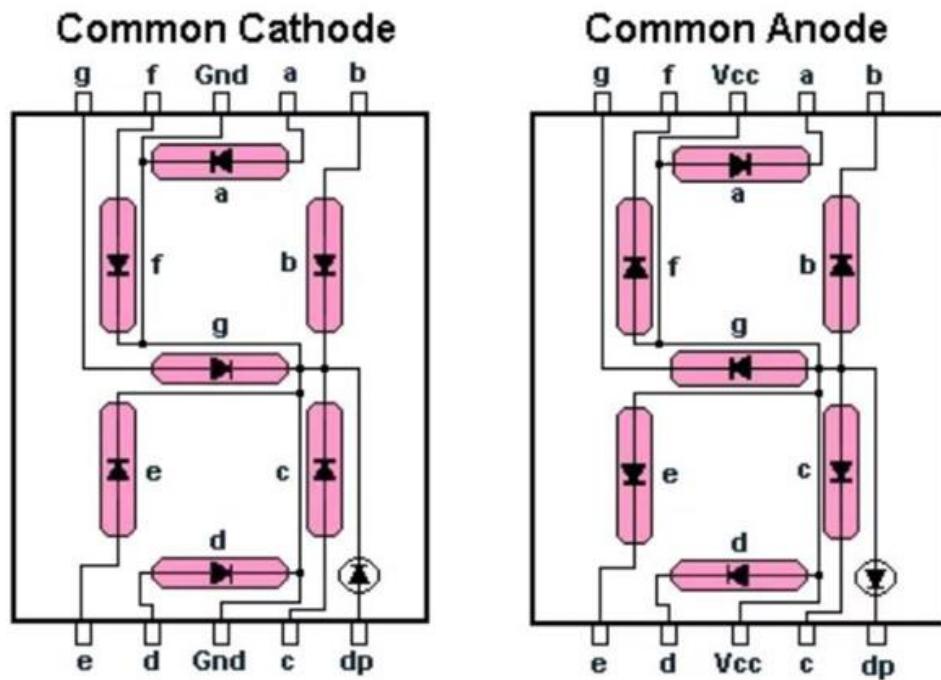
Hình 4.1. Giới thiệu LED 7 đoạn

4.1.2. Cấu tạo LED 7 đoạn

LED 7 đoạn gồm 7 đèn LED nhỏ, được sắp xếp theo hình dạng giống số 8, cho phép hiển thị các chữ số từ 0 đến 9. Ngoài ra, còn có một đèn LED nhỏ hình tròn dùng để hiển thị dấu chấm thập phân ở góc dưới.

Do mỗi đoạn là một đèn LED độc lập, nên có hai cách để kết nối chúng. Các đèn LED trong LED 7 đoạn có thể có cực Anode (cực dương) hoặc Cathode (cực âm) chung. Trong loại có Anode chung, các đèn được nối với nguồn dương (VCC) và chỉ sáng khi tín hiệu điều khiển

ở các chân tương ứng là mức thấp (0). Trong loại Cathode chung, điểm chung được nối với đất (Ground) và đèn sẽ sáng khi tín hiệu điều khiển ở mức cao 1.



Hình 4.2. Cấu tạo bên trong LED 7 đoạn [6]

Do LED 7 đoạn được cấu tạo từ các LED đơn bên trong, nên cần đảm bảo dòng điện đi qua mỗi LED đơn nằm trong khoảng 10mA đến 20 mA. Nếu nguồn điện cung cấp là 5V thì phải kết nối với các điện trở 330 Ohm.

4.1.3. Kết nối LED 7 đoạn với vi điều khiển

LED 7 đoạn có 8 chân nhận tín hiệu, vì vậy có thể sử dụng một cổng (Port) của vi điều khiển để điều khiển các LED này. Mỗi LED 7 đoạn nhận một dữ liệu 8 bit từ vi điều khiển và dữ liệu này sẽ điều khiển trạng thái của từng LED đơn trong module. Dữ liệu xuất ra để điều khiển LED 7 đoạn được gọi là “mã hiển thị”.

Có hai loại mã hiển thị: một dành cho LED 7 đoạn Anode chung và một dành cho Cathode chung. Ví dụ, để hiển thị số 1 các LED ở vị trí b và c cần sáng. Đối với LED 7 đoạn

Anode chung, cần đặt điện áp 0V (mức 0) vào các chân b và c, trong khi các chân còn lại đặt mức 5V (mức 1). Đối với Cathode chung cách điều khiển sẽ ngược lại.

Nếu kết nối mỗi Port của vi điều khiển với một LED 7 đoạn, tối đa có thể kết nối được 4 LED 7 đoạn. Tuy nhiên, cách này sẽ hạn chế khả năng thực hiện các tác vụ khác của vi điều khiển. Do đó, cần tối ưu hóa số chân điều khiển khi muốn kết nối nhiều LED 7 đoạn. Có hai giải pháp chính: sử dụng IC chuyên dụng để hiển thị trên LED 7 đoạn hoặc kết nối nhiều LED 7 đoạn vào cùng một đường tín hiệu hiển thị.

Bảng 4.1. Hiển thị LED 7 đoạn loại Anode chung

Số cần hiển thị	Mã nhị phân								Mã thập lục phân
	dp	G	F	E	D	C	B	A	
0	1	1	0	0	0	0	0	0	C0
1	1	1	1	1	1	0	0	1	F9
2	1	0	1	0	0	1	0	0	A4
3	1	0	1	1	0	0	0	0	B0
4	1	0	0	1	1	0	0	1	99
5	1	0	0	1	0	0	1	0	92
6	1	1	0	0	0	0	1	0	82
7	1	1	1	1	1	0	0	0	F8
8	1	0	0	0	0	0	0	0	80
9	1	0	0	0	0	0	0	0	90

Bảng 4.2. Hiển thị LED 7 đoạn loại Cathode chung

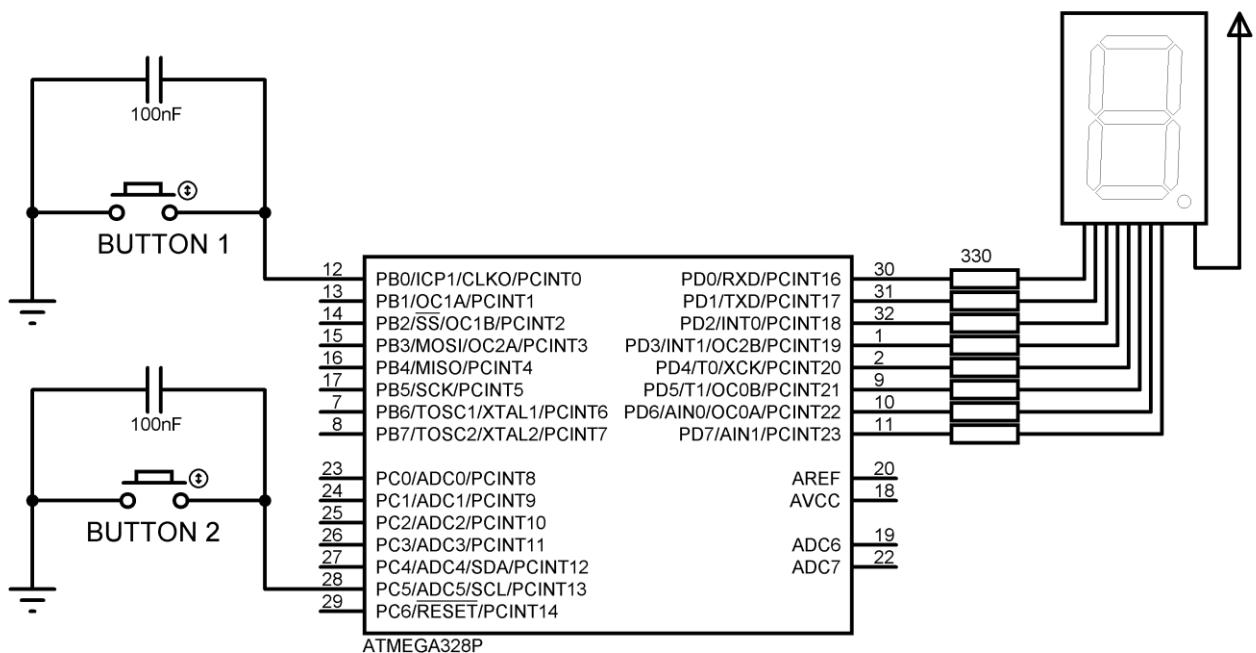
Số cần hiển thị	Mã nhị phân								Mã thập lục phân
	dp	G	F	E	D	C	B	A	
0	0	0	1	1	1	1	1	1	3F
1	0	0	0	0	0	1	1	0	06
2	0	1	0	1	1	0	1	1	5B

3	0	1	0	0	1	1	1	1	4F
4	0	1	1	0	0	1	1	0	66
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	1	7D
7	0	0	0	0	0	1	1	1	07
8	0	1	1	1	1	1	1	1	7F
9	0	1	1	0	1	1	1	1	6F

4.1.4. Bài thực hành

Yêu cầu bài thực hành: Hiển thị giá trị biến đếm lên LED 7 đoạn, khi nhấn BUTTON 1 giá trị biến đếm sẽ tăng một đơn vị và khi nhấn BUTTON 2 sẽ làm giảm một đơn vị.

❖ Phản mô phỏng



Hình 4.3. Sơ đồ nối dây bài thực hành LED 7 đoạn

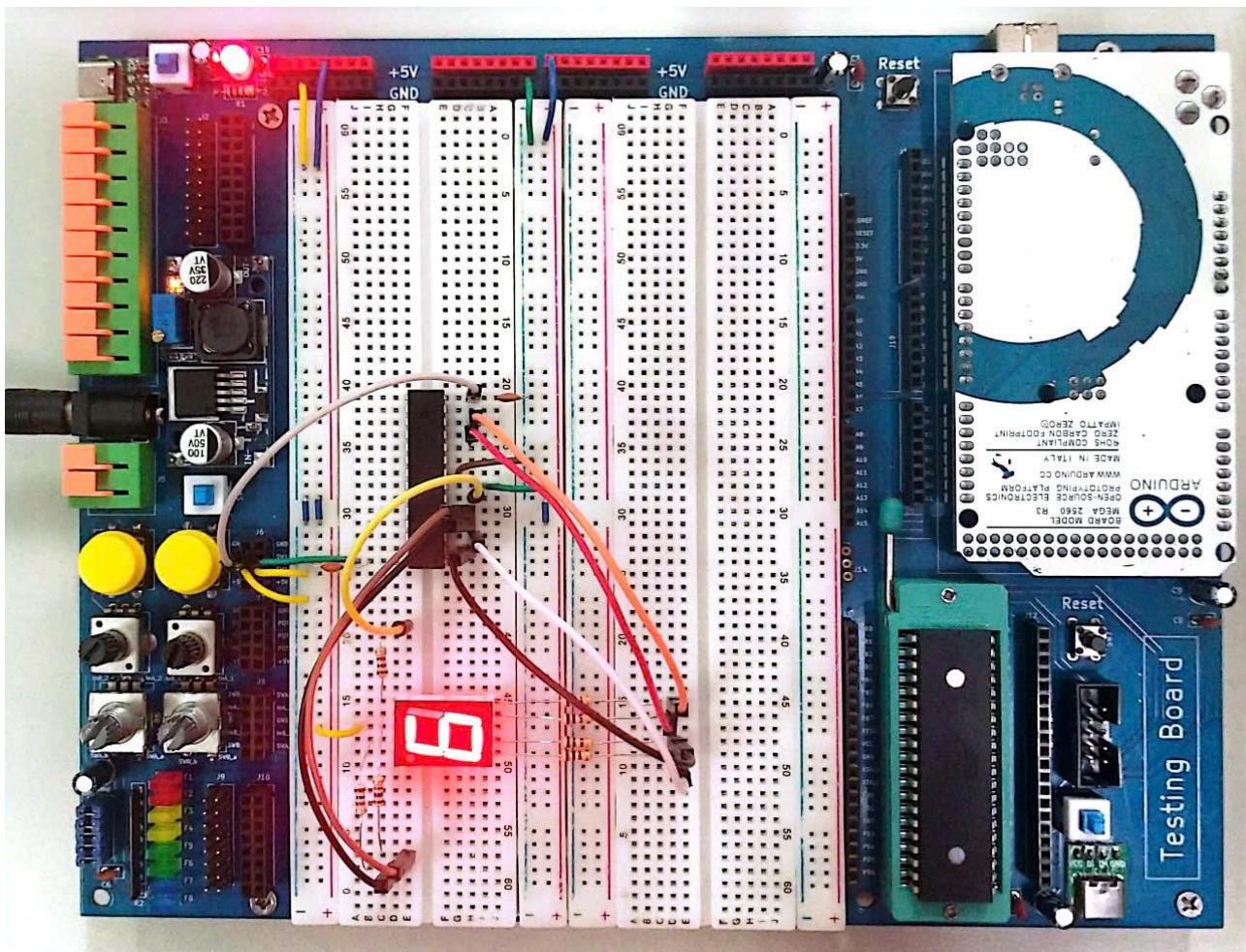
❖ Phần lập trình

	SourceCode _ATMEGA328P\P4_1_Seg_Led_Button
	<pre>1 #include <mega328p.h> 2 #include <delay.h> 3 char Led_7Seg[] = { 4 0b01000000, 5 0b01111001, 6 0b00100100, 7 0b00110000, 8 0b00011001, 9 0b00010010, 10 0b00000010, 11 0b01111000, 12 0b00000000, 13 0b00010000 14 }; 15 char current_button_1, previous_button_1; 16 char current_button_2, previous_button_2; 17 char counter = 0; 18 void main(void) 19 { 20 #pragma optsize- 21 CLKPR=(1<<CLKPCE); 22 CLKPR=0x00; 23 #ifdef _OPTIMIZE_SIZE_ 24 #pragma optsize+ 25 #endif 26 // dat muc logic cac chan PORTD voi muc thap 27 PORTD = 0b11111111; 28 // cau hinh cac chan PORTD la chan xuat 29 DDRD = 0b11111111; 30 31 // cau hinh dien tro keo len ben trong MCU 32 PORTB.0 = 1; 33 PORTC.5 = 1;</pre>

```

34 // cau hinh cac chan PB0 va PC6 la chan nhan
35 DDRB.0 = 0;
36 DDRC.5 = 0;
37 while (1)
38 {
39     previous_button_1 = current_button_1;
40     current_button_1 = PINB.0;
41     previous_button_2 = current_button_2;
42     current_button_2 = PINC.5;
43     if(PINB.0 == 0 && previous_button_1 != 0)
44     {
45         if(counter < 9) counter++;
46         else counter = 0;
47     }
48     else if(PINC.5 == 0 && previous_button_2 != 0)
49     {
50         if(counter > 0) counter--;
51         else counter = 9;
52     }
53     PORTD = Led_7Seg[counter];
54     delay_ms(50);
55 }
56 }
```

❖ Phản mô hình



Hình 4.4. Mô hình thực tế bài thực hành LED 7 đoạn

4.2. Lập trình hiển thị LCD

4.2.1. Giới thiệu

Text LCD là loại màn hình tinh thể lỏng nhỏ gọn, chuyên dùng để hiển thị các ký tự hoặc số theo bảng mã ASCII. Không giống như các loại LCD lớn có khả năng hiển thị hình ảnh, Text LCD được chia thành các ô vuông cố định, mỗi ô chỉ có thể hiển thị một ký tự ASCII. Vì đặc điểm chỉ hiển thị ký tự, loại màn hình này được gọi là Text LCD, để phân biệt với Graphic LCD có khả năng hiển thị cả hình ảnh.

Mỗi ô trong Text LCD bao gồm nhiều điểm ảnh tinh thể lỏng nhỏ. Khi các điểm ảnh này được bật hoặc tắt chúng kết hợp lại để tạo thành ký tự cần hiển thị. Các mẫu ký tự này đã được định sẵn bên trong màn hình.

Kích thước của Text LCD được xác định bởi số ký tự hiển thị trên mỗi dòng và số dòng của màn hình. Chẳng hạn, màn hình LCD 16x2 có thể hiển thị 16 ký tự trên mỗi dòng và có tổng cộng 2 dòng. Các kích thước thông dụng khác của Text LCD gồm có 16x1, 16x4, 20x2, 20x4 và nhiều biến thể khác.

Text LCD có hai phương thức giao tiếp chính: nối tiếp (như giao thức I2C) và song song. Chúng ta sẽ tập trung vào giao tiếp song song, cụ thể là sử dụng LCD 16x2 được điều khiển bởi chip HD44780U của hãng Hitachi. Đối với các loại LCD khác, cần tham khảo datasheet riêng, nhưng HD44780U được xem là tiêu chuẩn chung cho nhiều loại Text LCD, do đó có thể dùng chung các đoạn chương trình cho các loại LCD khác mà không cần chỉnh sửa quá nhiều.

HD44780U là một chip điều khiển dành cho các Text LCD kiểu ma trận điểm (dot-matrix), hỗ trợ các màn hình hiển thị 1 hoặc 2 dòng. Chip này có hai chế độ giao tiếp là 4 bit và 8 bit. Bên trong HD44780U có sẵn 208 ký tự mẫu với kích thước font 5x8 và 32 ký tự mẫu font 5x10, tạo thành tổng cộng 240 ký tự khác nhau.

4.2.2. Sơ đồ chân LCD 16x2

Các loại Text LCD dùng chip HD44780U thường có 16 chân, bao gồm 14 chân kết nối với bộ điều khiển và 2 chân nguồn cho LED nền.

Bảng 4.3 Sơ đồ chân LCD 16x2

Chức năng	Thứ tự chân	Tên	Trạng thái Logic	Mô tả
Ground	1	V _{ss} (GND)	-	0V
Nguồn	2	V _{dd} (V _{cc})	-	+5V
Độ tương phản	3	V _{ee}	-	0 đến V _{dd}

Điều khiển LCD	4	RS	0 1	D0-D7: lệnh D0-D7: dữ liệu
	5	R/W	0 1	Ghi (từ AVR vào LCD)
	6	E	0 1 Từ 1 xuống 0	Vô hiệu hóa LCD LCD hoạt động Bắt đầu ghi/đọc LCD
Dữ liệu/Lệnh	7	D0	0/1	Bit 0 LSB
	8	D1	0/1	Bit 1
	9	D2	0/1	Bit 2
	10	D3	0/1	Bit 3
	11	D4	0/1	Bit 4
	12	D5	0/1	Bit 5
	13	D6	0/1	Bit 6
	14	D7	0/1	Bit 7 MSB

Ở một vài LCD các chân LED nền ở vị trí 15 và 16 được ký hiệu là A (Anode) và K (Cathode). Hình bên dưới mô tả cách kết nối giữa LCD và vi điều khiển với các linh kiện điện tử.

Chân 1 và chân 2 của LCD là chân nguồn, được nối lần lượt với GND và nguồn 5V. Chân 3 dùng để điều chỉnh độ tương phản của màn hình và được kết nối với một biến trở phân áp như hình minh họa. Trong quá trình hoạt động, có thể điều chỉnh giá trị biến trở để đạt được độ tương phản mong muốn.

Các chân điều khiển RS, R/W, EN cùng với các chân dữ liệu sẽ được kết nối trực tiếp với vi điều khiển. Tùy thuộc vào chế độ giao tiếp (4 bit hoặc 8 bit), các chân dữ liệu D0 đến D3 có thể được bỏ qua hoặc kết nối với vi điều khiển.

4.2.3. Cấu trúc thanh ghi và tổ chức bộ nhớ

HD44780U có hai thanh ghi 8 bit là INSTRUCTION REGISTER (IR) và DATA REGISTER (DR). Thanh ghi IR lưu trữ các lệnh điều khiển LCD và chỉ cho phép ghi vào, không thể đọc ra từ thanh ghi này. Trong khi đó, thanh ghi DR lưu trữ dữ liệu, chẳng hạn như các ký tự cần hiển thị hoặc dữ liệu đọc ra từ bộ nhớ của LCD. Cả hai thanh ghi này đều kết nối với các đường dữ liệu D0:7 của Text LCD và được điều khiển dựa trên các chân RS và RW.

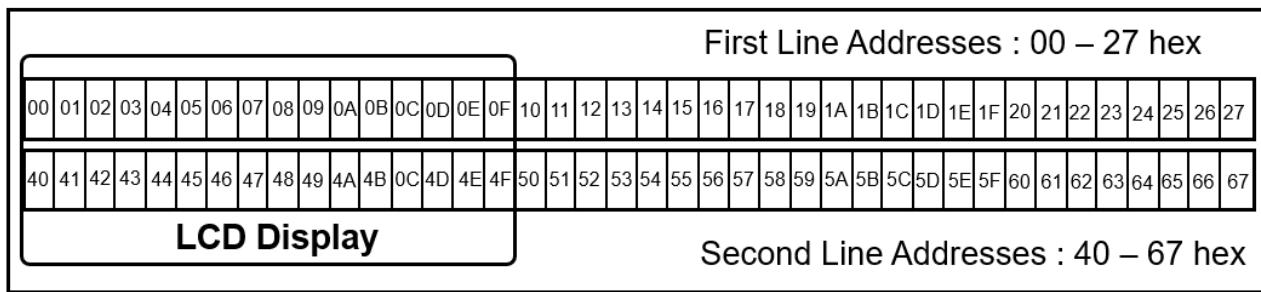
Trên thực tế, khi điều khiển Text LCD chúng ta không cần phải quá quan tâm đến cách các thanh ghi này hoạt động, nên không cần phải tìm hiểu chi tiết về chúng.

HD44780U sử dụng ba loại bộ nhớ chính: DDRAM (Display Data RAM) là bộ nhớ lưu trữ dữ liệu hiển thị, CGROM (Character Generator ROM) là bộ nhớ chứa các mẫu ký tự cố định, và CGRAM (Character Generator RAM) là bộ nhớ cho phép tạo ra các ký tự tùy chỉnh. Để lập trình hiển thị trên LCD cần hiểu cấu trúc cũng như nguyên lý hoạt động của các bộ nhớ này.

❖ Bộ nhớ DDRAM – Display Data RAM

DDRAM là bộ nhớ tạm thời dùng để lưu trữ các ký tự cần hiển thị trên màn hình LCD. Bộ nhớ này có tổng cộng 80 ô, được chia làm 2 hàng, mỗi ô rộng 8 bit. Các ô nhớ ở dòng 1 được đánh số từ 0 đến 0x27, và ở dòng 2 từ 0x40 đến 0x67. Mỗi ô nhớ trong DDRAM tương ứng với một ô hiển thị trên LCD. Với màn hình LCD loại 16x2, có thể hiển thị tối đa 32 ký tự, tương đương với 32 ô hiển thị. Điều này đồng nghĩa với việc có một số ô nhớ trong DDRAM sẽ không được sử dụng cho việc hiển thị.

DDRAM Memory



Hình 4.5. Cấu trúc của bộ nhớ DDRAM

Chỉ có 16 ô nhớ từ địa chỉ 0 đến 0x0F và 16 ô từ địa chỉ 0x40 đến 0x4F trong DDRAM được hiển thị trên LCD. Vì vậy, nếu muốn hiển thị một ký tự nào đó trên LCD, chúng ta cần ghi ký tự đó vào một trong 32 địa chỉ này trong DDRAM. Các ký tự được lưu ngoài 32 địa chỉ này sẽ không hiển thị, nhưng chúng vẫn được giữ lại trong bộ nhớ và có thể được sử dụng cho các mục đích khác khi cần.

❖ Bộ nhớ CGROM

CGROM là vùng nhớ cố định chứa định nghĩa các font ký tự và chúng ta không thể truy cập trực tiếp vào. Thay vào đó, chip HD44780U tự động thực hiện việc truy xuất khi cần lấy font để hiển thị. Điều đặc biệt là địa chỉ font của mỗi ký tự trong CGROM tương ứng với mã ASCII của ký tự đó. Ví dụ, ký tự ‘a’ có mã ASCII là 97. Khi nhìn vào cách tổ chức của CGROM sẽ thấy địa chỉ font của ‘a’ gồm 4 bit thấp là 0001 và 4 bit cao là 0110, tổng hợp lại thành 01100001 tương đương với giá trị 97.

Trong quá trình hiển thị, CGROM và DDRAM phối hợp một cách tự động. Giả sử chúng ta muốn hiển thị ký tự ‘a’ tại vị trí đầu tiên của dòng thứ hai trên LCD, các bước sẽ diễn ra như sau: Vị trí đầu tiên của dòng 2 có địa chỉ 0x40 trong DDRAM (theo hình minh họa). Bạn sẽ ghi giá trị 97 (mã ASCII của ‘a’) vào ô nhớ có địa chỉ 64 trong DDRAM. Sau đó, chip HD44780U sẽ đọc giá trị 97 này và sử dụng nó làm địa chỉ để truy xuất vùng nhớ CGROM. Tại đây, nó sẽ tìm bảng font của ký tự ‘a’, sau đó hiển thị ký tự này bằng cách kích hoạt các

điểm ảnh phù hợp trên vị trí đầu tiên của dòng 2 trên màn hình LCD. Đây là cách mà DDRAM và CGROM phối hợp để hiển thị ký tự trên LCD.

Upper 4 bits Lower 4 bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
CG RAM (1)	ØøP^P												—øEøP			
CG RAM (2)	!1AQaq												øAøCäq			
CG RAM (3)	"2BRbr												「イツXøø			
CG RAM (4)	#3CScs												」ウテモ ^{oo}			
CG RAM (5)	\$4DTdt												、エトトム ^o			
CG RAM (6)	%5EUeu												・オナコ ^o			
CG RAM (7)	&6FUVv												ヲカニヨ ^{PΣ}			
CG RAM (8)	?7GUgw												アキヌラ ^{gπ}			
CG RAM (1)	(BH)hx												イクネリ ^{gx}			
CG RAM (2))9IYiy												ウケルレ ^{iy}			
CG RAM (3)	*:JZjz												エコハレ ^{jz}			
CG RAM (4)	+;KCKk{												オサヒロ ^{^x}			
CG RAM (5)	;L¥1												ヤシフワ [¥]			
CG RAM (6)	--=M)m)												ミズヘンモ ⁼			
CG RAM (7)	:#N^n*												ミセホ ^{^n}			
CG RAM (8)	/?O_o*												ミソラ ^o	ø		

Hình 4.6. Vùng nhớ CGROM [7]

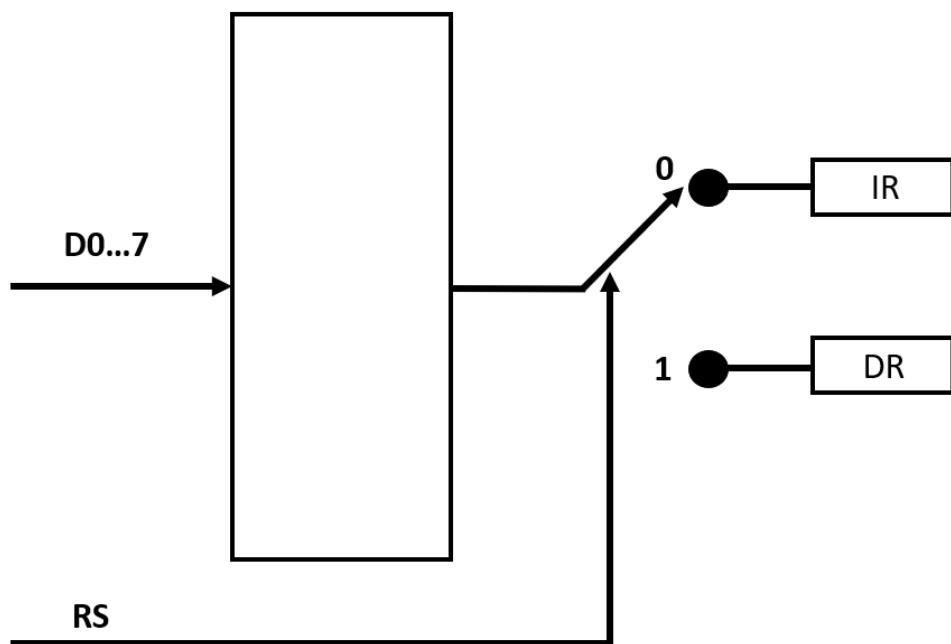
❖ Bộ nhớ CGRAM

CGRAM là vùng nhớ cho phép người dùng tự tạo các ký tự hoặc biểu tượng tùy chỉnh. Mỗi biểu tượng có kích thước 5×8 và chiếm 8 ô nhớ 8 bit. Thông thường, các biểu tượng này được định nghĩa trước và có thể được gọi để hiển thị khi cần. Với tổng cộng 64 ô nhớ, CGRAM cho phép định nghĩa tối đa 8 biểu tượng khác nhau.

4.2.4. Điều khiển hiển thị LCD

Để thực hiện việc đọc và ghi dữ liệu, hiển thị các ký tự lên màn hình LCD, chúng ta cần quan tâm đến các chân điều khiển bao gồm RS, R/W và EN.

RS (chân số 4): Đây là chân chọn thanh ghi (Register Select), giúp lựa chọn giữa hai thanh ghi IR và DR để làm việc. Vì cả hai thanh ghi này đều được kết nối với các chân dữ liệu của LCD, cần một bit để xác định thanh ghi nào sẽ được sử dụng. Khi $RS=0$, thanh ghi IR sẽ được chọn, còn khi $RS=1$ thanh ghi DR sẽ được chọn. Thanh ghi IR chứa mã lệnh điều khiển LCD, vì vậy nếu bạn muốn gửi một lệnh đến LCD, cần đặt $RS=0$. Ngược lại, để ghi mã ASCII của ký tự cần hiển thị, ta sẽ đặt $RS=1$ để chọn thanh ghi DR.



Hình 4.7 Hoạt động của chân RS [8]

R/W (chân số 5): Đây là chân quyết định việc đọc hoặc ghi dữ liệu. Nếu R/W=0, dữ liệu sẽ được ghi từ vi điều khiển (ví dụ như AVR) vào LCD. Ngược lại, nếu R/W=1, dữ liệu sẽ được đọc từ LCD ra ngoài. Tuy nhiên, LCD chỉ cho phép đọc dữ liệu trong một tình huống duy nhất, đó là khi kiểm tra trạng thái bận của LCD thông qua cờ Busy Flag (BF). Do LCD hoạt động chậm hơn so với vi điều khiển, cờ BF được sử dụng để báo hiệu rằng LCD đang bận. Nếu BF=1, LCD chưa sẵn sàng và cần chờ cho đến khi BF=0 trước khi có thể tiếp tục thao tác. Vì vậy, khi điều khiển LCD cần có một hàm chương trình để chờ LCD hoàn tất nhiệm vụ. Có hai phương pháp để viết hàm này:

- Phương pháp 1: Đọc cờ BF để kiểm tra xem khi nào BF=0, tức là LCD đã sẵn sàng. Phương pháp này yêu cầu lệnh đọc từ LCD, vì vậy chân R/W phải được kết nối với vi điều khiển.
- Phương pháp 2: Sử dụng một hàm delay với khoảng thời gian cố định (tốt nhất là trên 1ms). Phương pháp này đơn giản hơn vì không cần phải đọc từ LCD, do đó có thể nối chân R/W của LCD xuống GND. Tuy nhiên, nếu thời gian delay quá dài, quá trình điều khiển LCD sẽ chậm, còn nếu quá ngắn sẽ dẫn đến lỗi hiển thị.

EN (chân số 6): Đây là chân cho phép hoạt động của LCD (Enable), cần kết nối với vi điều khiển để thao tác với LCD. Để thực hiện đọc hoặc ghi dữ liệu, phải tạo ra một “xung cạnh xuống” trên chân EN. Nói cách khác, trước khi ghi dữ liệu vào LCD, chân EN phải được đảm bảo ở mức 0. Sau khi xuất dữ liệu ra các chân D0:7, đặt EN lên mức 1 và sau đó hạ EN trở lại mức 0 để tạo ra xung cạnh xuống, hoàn thành quá trình ghi dữ liệu.

❖ Nhóm tập lệnh dành cho LCD

Bảng 4.4. Tóm tắt các lệnh ghi vào LCD [8]

Command	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	Execution Time
Clear display	0	0	0	0	0	0	0	0	0	1	1.64 ms
Cursor home	0	0	0	0	0	0	0	0	1	x	1.64 ms

Entry mode set	0	0	0	0	0	0	0	1	I/D	S	40 us
Display on/off control	0	0	0	0	0	0	1	D	C	B	40 us
Cursor/Display Shift	0	0	0	0	0	1	S/C	R/L	x	x	40 us
Function set	0	0	0	0	1	DL	N	F	x	x	40 us
Set CGRAM address	0	0	0	1	CGRAM address						40 us
Set DDRAM address	0	0	1	DDRAM address						40 us	
Read “BUSY” flag (BF)	0	1	BF	DDRAM address						-	
Write to DDRAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	40 us
Read from CGRAM or DDRAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	40 us

Bảng tập lệnh bao gồm các hàm được sử dụng thường xuyên để hiển thị LCD và một số hàm chỉ sử dụng một lần để khởi tạo.

❖ Nhóm lệnh dùng để hiển thị LCD:

Clear display (xóa hiển thị): xóa toàn bộ nội dung hiện có trên bộ nhớ DDRAM, do đó xóa hết các nội dung hiển thị trên màn hình LCD. Vì đây là lệnh ghi Instruction, trước khi thực hiện, chân RS phải được đặt về mức 0. Mã lệnh dùng để xóa LCD là 0x01, được ghi vào các chân D0 của màn hình LCD.

Cursor home (đưa con trỏ về vị trí đầu tiên): đưa con trỏ về vị trí đầu tiên trong bộ nhớ DDRAM, cụ thể là vị trí đầu của dòng 1 trên LCD. Sau khi thực hiện lệnh, nếu có dữ liệu mới được ghi vào DDRAM, nó sẽ nằm ở vị trí đầu tiên. Trước khi ghi lệnh này, chân RS cũng phải được đặt về mức 0. Mã lệnh để đưa con trỏ về đầu là 0x02 hoặc 0x03, người dùng có thể chọn một trong hai mã lệnh tùy ý.

Set DDRAM Address (Đặt địa chỉ cho con trỏ trong DDRAM): Lệnh này cho phép di chuyển con trỏ đến một vị trí tùy ý trong DDRAM, điều này đồng nghĩa với việc người dùng

có thể tùy chọn vị trí muốn hiển thị trên LCD. Khi thực hiện lệnh, chân RS cần được đặt về mức 0. Bit MSB của mã lệnh D7 phải là 1 và 7 bit còn lại là địa chỉ DDRAM cần di chuyển đến. Ví dụ, để di chuyển con trỏ đến vị trí thứ 3 của dòng 2 (có địa chỉ là 42 trong DDRAM), mã lệnh sẽ là 0xAA, với $0x\text{AA} = 10101010$ trong hệ nhị phân, trong đó bit MSB là 1, còn 7 bit tiếp theo là 0101010 tương ứng với địa chỉ 42.

Write to DDRAM (ghi dữ liệu vào DDRAM): Đây là lệnh ghi dữ liệu, không phải là lệnh instruction, do đó chân RS phải được đặt lên mức 1 trước khi ghi vào LCD. Lệnh này cho phép ghi mã ASCII của ký tự cần hiển thị vào DDRAM.

❖ Nhóm lệnh khởi tạo cho LCD

Entry mode set (xác định các hiển thị liên tiếp trên LCD): quy định cách hiển thị các ký tự tiếp theo sau ký tự hiện tại trên màn hình LCD. Ví dụ muốn hiển thị hai ký tự “A” và “B” liên tiếp, bạn có thể viết “A” ở vị trí thứ 5 của dòng 1. Khi viết ký tự “B”, có 4 cách mà LCD có thể hiển thị:

- Hiển thị “B” bên phải “A” tại vị trí thứ 6.
- Hiển thị “B” bên trái “A” tại vị trí thứ 4.
- Dịch chuyển “A” sang trái đến vị trí số 4, rồi hiển thị “B” bên phải “A” tại vị trí 5.
- Dịch chuyển “A” sang phải đến vị trí số 6, rồi hiển thị “B” bên trái “A” tại vị trí 5.

Người dùng có thể chọn một trong bốn cách hiển thị này thông qua lệnh Entry Mode Set. Đây là lệnh instruction, vì vậy chân RS phải được đặt về mức 0. Trong mã lệnh, 5 bit cao D7:3 có giá trị 00000, bit D2 luôn bằng 1 và hai bit còn lại D1:0 xác định chế độ hiển thị. Bit D1 (I/D): Xác định hướng hiển thị tiếp theo. Nếu I/D = 1, ký tự sau sẽ hiển thị bên phải ký tự trước (tăng), nếu I/D = 0, ký tự sau sẽ hiển thị bên trái ký tự trước (giảm). Bit D0 (S): Xác định việc dịch chuyển các ký tự trước. Nếu S = 1, các ký tự đã hiển thị trước đó sẽ được dịch chuyển và ký tự mới sẽ thay thế vị trí của ký tự cũ. Ngược lại, nếu S = 0, các ký tự trước đó sẽ giữ nguyên vị trí. Có thể tóm tắt 4 mode như sau:

- D7:0 = 0b00000110 (0x06): hiển thị tăng và không shift.
- D7:0 = 0b00000100 (0x04): hiển thị giảm và không shift.

- D7:0 = 0b00000111 (0x07): hiển thị tăng và shift.

- D7:0 = 0b00000101 (0x05): hiển thị giảm và shift.

Display on/off control (xác lập hiển thị LCD): cho phép cấu hình các tùy chọn hiển thị của LCD như bật/tắt màn hình, hiển thị con trỏ (cursor) và bật/tắt chế độ nháy nháy (blinking). Đây là lệnh ghi Instruction, nên chân RS phải được đặt về mức 0. Mã lệnh có dạng 00001DCB, trong đó:

- D (Display): Cho phép hiển thị LCD khi giá trị bằng 1.

- C (Cursor): Hiển thị con trỏ khi giá trị bằng 1.

- B (Blinking): Nếu bằng 1, con trỏ sẽ nháy nháy tại vị trí hiện tại, tạo hiệu ứng “blinking” (một ô vuông đen nháy nháy tại vị trí ký tự đang hiển thị).

- Mã lệnh phổ biến cho lệnh này là 0x0E (00001110), bật hiển thị con trỏ nhưng không bật chế độ nháy nháy.

Function set – thiết lập chức năng cho LCD: Lệnh này thiết lập các phương thức giao tiếp với LCD, số dòng hiển thị và kích thước font chữ. Trước khi thực hiện lệnh, chân RS phải được đặt về mức 0. Mã lệnh có dạng 001DLNFxx, trong đó:

- DL (Data Length): Xác định chế độ giao tiếp, nếu DL = 1 LCD sẽ sử dụng chế độ 8 bit, yêu cầu tất cả các chân từ D0 đến D7 phải được kết nối với bộ điều khiển. Nếu DL = 0 thì chế độ 4 bit sẽ được sử dụng, chỉ cần kết nối các chân D[4:7] với vi điều khiển còn các chân PD[0:3] để trống.

- N (Number of Lines): Quy định số dòng hiển thị, nếu N = 1 LCD sẽ hiển thị 2 dòng (với điều kiện LCD đó thể hiển thị tối đa 2 dòng), nếu N = 1 thì LCD sẽ hiển thị 1 dòng.

- F (Font Size): Quy định kích thước phông chữ, nếu F = 1 sẽ sử dụng cỡ chữ 5 x 10, nếu F = 0 sẽ dùng cỡ chữ 5 x 8.

- Hai bit thấp (xx) có thể được gán giá trị tùy ý.

- Mã lệnh phổ biến cho lệnh này là 0x38 (00111000 – giao tiếp 8 bit, 2 dòng, font 5x8) hoặc 0x28 (00101000 – giao tiếp 4 bit, 2 dòng, font 5x8).

❖ Giao tiếp 8 bit và 4 bit

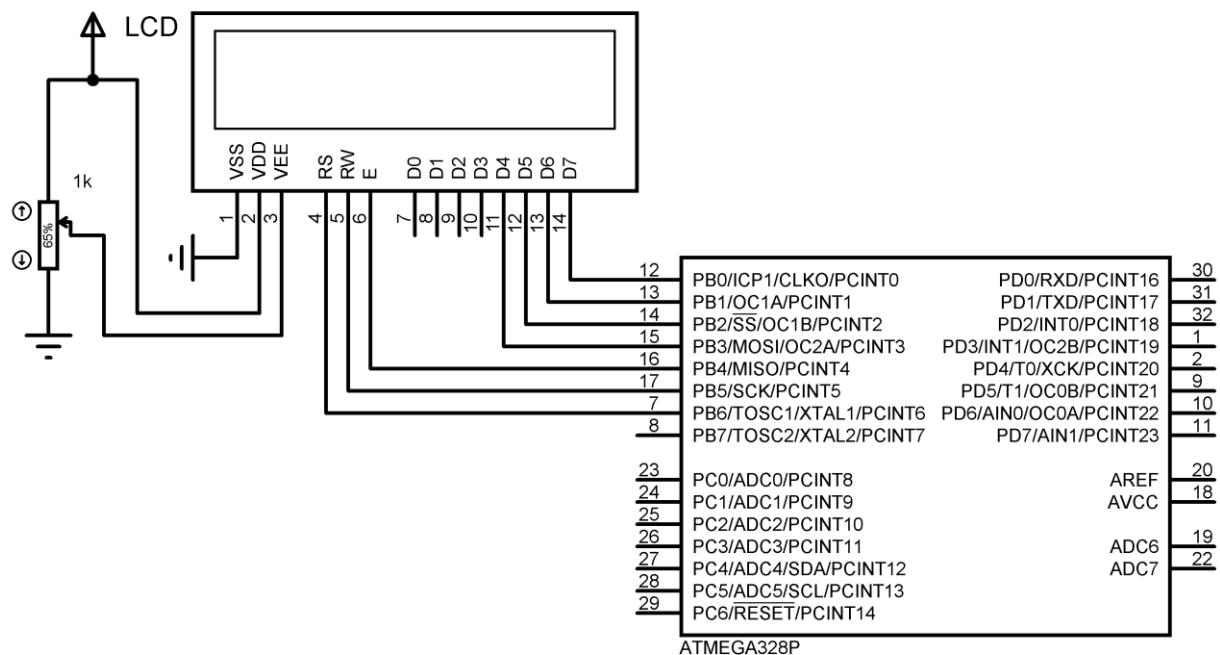
- Mode 4 bit: LCD cũng hỗ trợ chế độ giao tiếp 4 bit với vi điều khiển. Ở chế độ này, các chân D0, D1, D2 và D3 của LCD không được kết nối mà để trống, chỉ sử dụng 4 chân từ D4 đến D7 để truyền và nhận dữ liệu. Mỗi instruction và dữ liệu 8 bit sẽ được chia thành hai phần gọi là Nibbles, mỗi nibble chứa 4 bit. Giao tiếp diễn ra bằng cách truyền nibble cao trước, sau đó là nibble thấp, thông qua các chân D[7:4]. Ưu điểm lớn nhất của chế độ 4 bit là giảm thiểu số lượng chân cần thiết để giao tiếp với LCD. Tuy nhiên, quá trình đọc và ghi dữ liệu bằng cách xử lý từng nibble phức tạp hơn so với việc xử lý toàn bộ 8 bit cùng lúc.

- Mode 8 bit: Nếu bit DL trong lệnh Function Set được đặt bằng 1, LCD sẽ sử dụng chế độ giao tiếp 8 bit. Để kích hoạt chế độ này, tất cả các chân dữ liệu của LCD từ D0 đến D7 (từ chân 7 đến chân 14) phải được kết nối với một PORT của vi điều khiển. Ưu điểm của chế độ 8 bit là quá trình đọc và ghi dữ liệu diễn ra nhanh chóng và dễ dàng, vì vi điều khiển chỉ cần sử dụng một PORT để xuất hoặc nhận toàn bộ 8 bit dữ liệu. Tuy nhiên, nhược điểm là chế độ này sử dụng quá nhiều chân để giao tiếp với LCD, tổng cộng là 11 đường (gồm 8 chân dữ liệu và 3 chân điều khiển).

4.2.5. Bài thực hành

Yêu cầu bài thực hành: Sử dụng LCD 16x2 để hiển thị đoạn ký tự, tạo hiệu ứng chạy dòng chữ “HELLO WORLD” từ trái sang phải.

❖ Phản mô phỏng



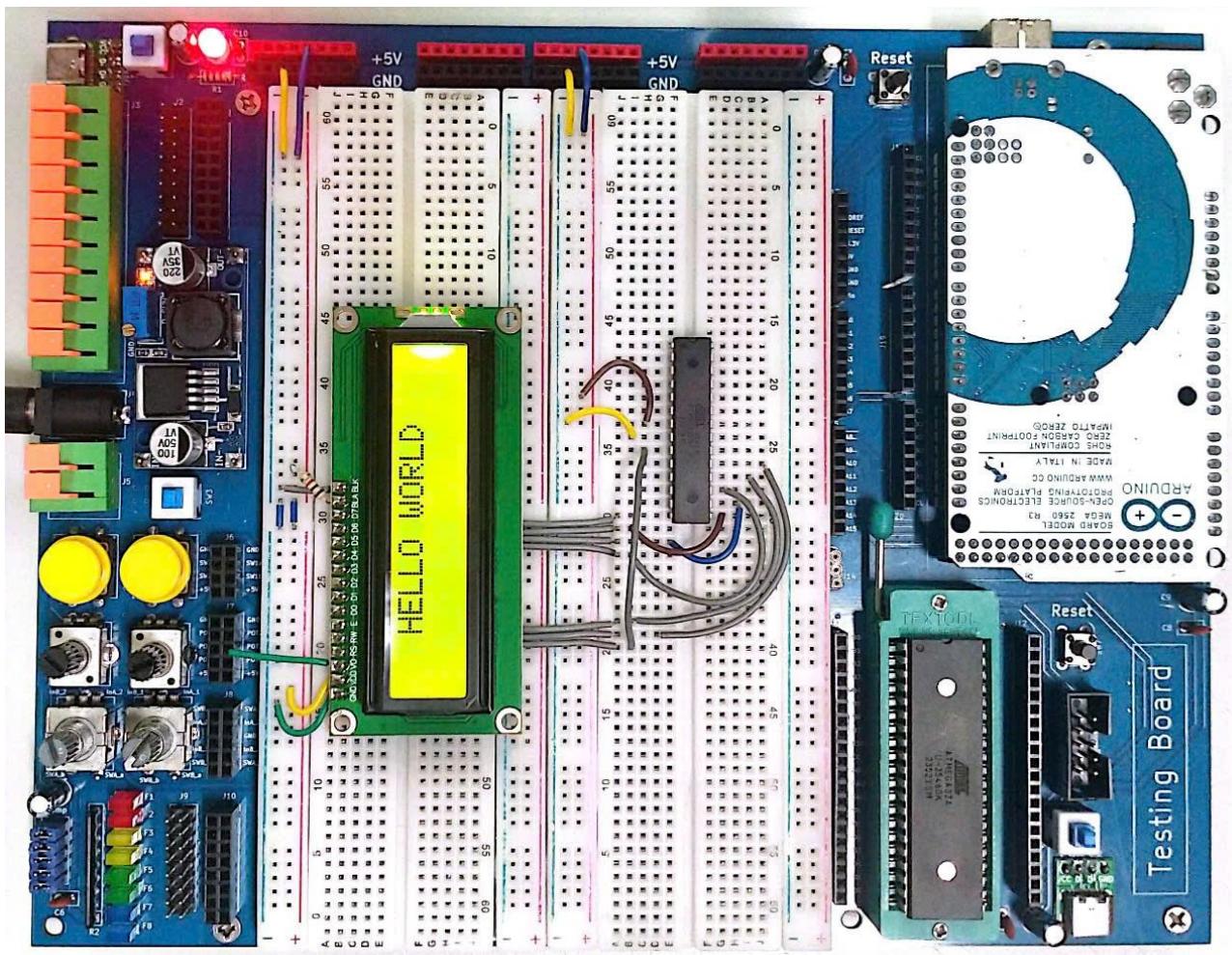
Hình 4.8. Sơ đồ nối dây bài thực hành với LCD

❖ Phản lập trình

	SourceCode _ATMEGA328P\P4_2_LCD16x2
1	#include <mega328p.h>
2	#include <alcd.h>
3	#include <delay.h>
4	//mang chua doan ky tu can hien thi
5	//myStr gom (16+11+16) ky tu
6	char myStr[] = " HELLO WORLD ";
7	int length = sizeof(myStr) - 1; //tru 1 ky tu ket thuc
8	int lcd_width = 16;
9	void main(void){
10	#pragma optsize-
11	CLKPR=(1<<CLKPCE);
12	CLKPR=0x00;
13	#ifdef _OPTIMIZE_SIZE_
14	#pragma optsize+
15	#endif

```
16  lcd_init(16);
17  lcd_gotoxy(2,0);
18  lcd_putsf("Demo of the");
19  lcd_gotoxy(0,1);
20  lcd_putsf("2x16 LCD Display");
21  delay_ms(2000);
22  lcd_clear();
23  while (1){
24      int i, j;
25      for (i = length - lcd_width; i >= 0; i--){
26          lcd_clear();
27          for (j = 0; j < lcd_width; j++) {
28              lcd_gotoxy(j,0);
29              lcd_putchar(myStr[i + j]);
30          }
31          delay_ms(500);
32      }
33  }
34 }
```

❖ Phản mô hình



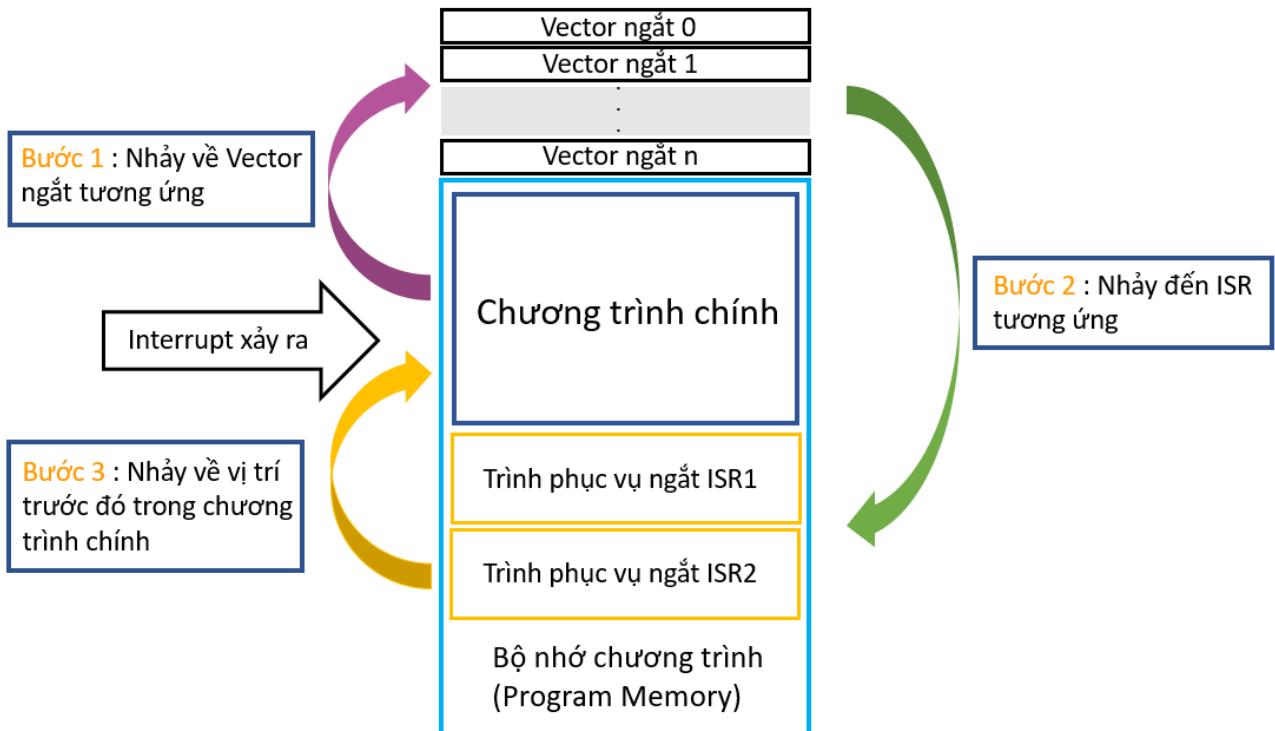
Hình 4.9. Mô hình thực tế bài thực hành hiển thị LCD

CHƯƠNG 5. CẤU TRÚC NGẮT

5.1. Khái niệm về ngắt

Ngắt (Interrupt) là một tín hiệu ưu tiên cao gửi đến bộ xử lý, yêu cầu tạm dừng công việc hiện tại để chuyển sang xử lý một nhiệm vụ khẩn cấp khác, gọi là trình phục vụ ngắt (ISR – Interrupt Service Routine). Sau khi hoàn tất việc xử lý trong ISR, bộ đếm chương trình sẽ được khôi phục về giá trị trước đó, cho phép bộ xử lý tiếp tục công việc đang thực hiện dang dở. Ngắt có độ ưu tiên rất cao và thường được dùng để xử lý các sự kiện bất ngờ, nhưng không đòi hỏi nhiều thời gian xử lý. Các tín hiệu ngắt có thể xuất phát từ các thiết bị nội bộ của chip (như ngắt do tràn bộ đếm Timer/Counter hoặc do kết thúc quá trình truyền dữ liệu qua RS232) hoặc từ các nguồn bên ngoài (chẳng hạn như ngắt do một nút nhấn được kích hoạt hoặc ngắt do nhận được một gói dữ liệu).

Số lượng ngắt trên mỗi dòng chip là khác nhau và mỗi ngắt đều có một vector ngắt tương ứng. Vector ngắt là các thanh ghi có địa chỉ cố định được định nghĩa trước, nằm ở phần đầu của bộ nhớ chương trình. Ví dụ, vector ngắt ngoài 0 (external interrupt 0) trên chip ATMEGA328P có địa chỉ là 0x001 (theo tài liệu kỹ thuật từ Atmel). Khi chương trình chính đang thực thi, nếu có sự thay đổi xảy ra dẫn đến ngắt tại chân INT0 (chân 4), bộ đếm chương trình (Program Counter) sẽ nhảy đến địa chỉ 0x009. Nếu tại địa chỉ 0x001, chúng ta đặt một lệnh RJMP để nhảy đến một trình phục vụ ngắt (ISR1 chặng hạn), thì bộ đếm chương trình sẽ tiếp tục nhảy đến ISR1 để xử lý ngắt. Sau khi hoàn thành ISR1, bộ đếm chương trình sẽ quay lại địa chỉ của chương trình chính trước khi ngắt xảy ra và kết thúc quá trình ngắt.



Hình 5.1. Quá trình thực hiện ngắt [9]

5.2. Trình phục vụ ngắt và bảng vector ngắt

Trình phục vụ ngắt là một đoạn mã chương trình được viết để xử lý một nhiệm vụ cụ thể khi ngắt xảy ra. Khi một ngắt được kích hoạt, vi điều khiển sẽ nhảy đến đoạn mã này (dựa vào vector ngắt) để thực hiện nhiệm vụ liên quan đến ngắt đó. ISR thực hiện các công việc nhanh chóng, vì sau khi hoàn thành, vi điều khiển sẽ quay trở lại chương trình chính và tiếp tục công việc trước khi ngắt xảy ra. ISR phải được viết gọn gàng và nhanh chóng để không làm gián đoạn quá lâu quá trình xử lý của hệ thống.

Vector ngắt là một địa chỉ cụ thể trong bộ nhớ chương trình của vi điều khiển, nơi chương trình sẽ chuyển tới khi một ngắt xảy ra. Mỗi ngắt trên vi điều khiển có một vector ngắt riêng biệt giúp vi điều khiển biết phải đi đến đâu trong bộ nhớ để thực hiện nhiệm vụ xử lý ngắt. Các vector ngắt được định nghĩa trước và thường nằm ở đầu bộ nhớ chương trình. Khi có một ngắt xảy ra, vi điều khiển dừng chương trình chính và sử dụng vector ngắt tương ứng để nhảy đến một địa chỉ cụ thể trong bộ nhớ, nơi bắt đầu mã của trình phục vụ ngắt.

Bảng 5.1 Các vector ngắt và Reset trên Atmega328P [9]

Vector	Address	Source	Interrupt Definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x004	INT1	External Interrupt Request 1
4	0x006	PCINT0	Pin change interrupt request 0
5	0x008	PCINT1	Pin change interrupt request 1
6	0x00A	PCINT2	Pin change interrupt request 2
7	0x00C	WDT	Watchdog time-out interrupt
8	0x00E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x012	TIMER2 OVF	Timer/Counter2 overflow
11	0x014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x01A	TIMER1 OVF	Timer/Counter1 overflow
15	0x01C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x01E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x020	TIMER0 OVF	Timer/Counter0 overflow
18	0x022	SPI, STC	SPI serial transfer complete
19	0x024	USART, RX	USART Rx complete
20	0x026	USART, UDRE	USART, data register empty
21	0x028	USART, TX	USART Tx complete
22	0x02A	ADC	ADC conversion complete
23	0x02C	EE READY	EEPROM ready
24	0x02E	ANALOG COMP	Analog comparator
25	0x030	TWI	2-wire serial interface
26	0x032	SPM READY	Store program memory ready

5.3. Ngắt ngoài

Các ngắt ngoài trên ATmega328P có thể được kích hoạt qua các chân INT0, INT1 hoặc qua dãy chân PCINT[23:0]. Khi các ngắt này được bật, chúng sẽ hoạt động ngay cả khi các chân INT0, INT1 hoặc PCINT[23:0] được thiết lập là ngõ ra.

Ngắt thay đổi trạng thái chân (Pin Change Interrupt) được chia thành ba nhóm chính:

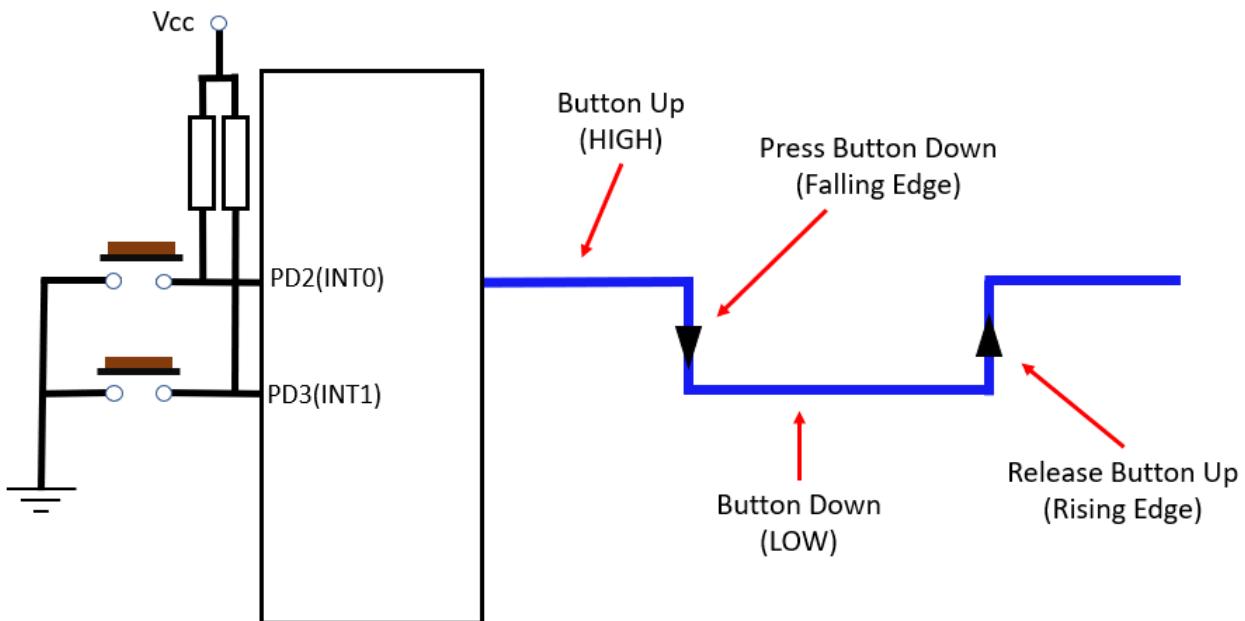
- PCI2: Kích hoạt khi bất kỳ chân nào trong nhóm PCINT[23:16] thay đổi trạng thái.
- PCI1: Kích hoạt khi có sự thay đổi trên các chân trong nhóm PCINT[14:8].
- PCI0: Kích hoạt khi bất kỳ chân nào trong nhóm PCINT[7:0] có sự thay đổi.

Việc kích hoạt cho từng chân trong các nhóm này được điều khiển thông qua các thanh ghi PCMSK2, PCMSK1 và PCMSK0. Ngắt thay đổi trạng thái chân được phát hiện không đồng bộ, vì vậy có thể dùng các ngắt này để đánh thức vi điều khiển từ các chế độ ngủ ngoài chế độ Idle.

Riêng với ngắt INT0 và INT1, chúng có thể kích hoạt khi tín hiệu đầu vào có sườn lên, sườn xuống hoặc giữ ở mức thấp, được cấu hình qua thanh ghi EICRA. Khi cấu hình ngắt INT0 hoặc INT1 với trường hợp mức thấp, ngắt sẽ duy trì miễn là tín hiệu đầu vào giữ ở mức thấp.

Nếu sử dụng ngắt theo kiểu level triggered để đánh thức từ chế độ Power-down, tín hiệu mức thấp cần được giữ đủ lâu để MCU có thời gian hoàn thành quá trình thức dậy và ghi nhận ngắt. Nếu tín hiệu biến mất trước khi kết thúc thời gian khởi động, MCU vẫn sẽ thức dậy, nhưng ngắt sẽ không được kích hoạt.

5.3.1. Thanh ghi A điều khiển ngắt ngoài – EICRA



Hình 5.2. Kết nối ngắt ngoài cho ATMEGA328P [9]

Khi kết nối các ngắt ngoài trên ATMEGA328P với các nút nhấn (button) để tạo ngắt, sẽ có 4 tình huống xảy ra trong quá trình nhấn và thả nút. Khi không nhấn nút, chân INT sẽ ở trạng thái mức HIGH nhờ điện trở kéo lên. Khi nút được nhấn, chân INT thay đổi trạng thái từ HIGH xuống LOW, được gọi là cạnh xuống (Falling Edge). Nếu giữ nút nhấn, chân INT sẽ duy trì ở mức LOW. Cuối cùng, khi thả nút, trạng thái chuyển từ LOW lên HIGH, được gọi là cạnh lên (Rising Edge).

Thanh ghi EICRA chứa các bit cho phép lựa chọn một trong bốn chế độ cho các ngắt ngoài, dưới đây là cấu trúc của thanh ghi EICRA.

Bit	7	6	5	4	3	2	1	0	
(0x69)	-	-	-	-	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Ngắt ngoài 0 hoặc 1 được kích hoạt bởi chân ngoài INT0, INT1 nếu cờ I trong thanh ghi trạng thái SREG và mặt nạ ngắt tương ứng được thiết lập. Nếu chọn chế độ ngắt theo cạnh lên hoặc cạnh xuống hoặc có bất kỳ sự chuyển đổi trạng thái, các xung có thời gian dài hơn một chu kỳ xung clock sẽ tạo ra ngắt. Các xung ngắn hơn thì không đảm bảo tạo ra ngắt. Nếu chọn chế độ ngắt theo mức thấp, mức thấp phải được giữ cho đến khi hoàn tất lệnh đang thực thi để tạo ra ngắt.

Bảng 5.2. Bảng thiết lập các chế độ ngắt ngoài của INT[0:1]

ISC01 ISC11	ISC00 ISC10	Mô tả
0	0	Mức thấp của chân INT[0:1] tạo ra yêu cầu ngắt – low level
0	1	Bất kì sự thay đổi nào từ chân INT[1:0] tạo ra yêu cầu ngắt – logical change
1	0	Cạnh xuống trên chân INT[0:1] tạo ra yêu cầu ngắt – falling edge
1	1	Cạnh lên trên chân INT[0:1] tạo ra yêu cầu ngắt – rising edge

5.3.2. Thanh ghi mặt nạ ngắt – EIMSK

Bit	7	6	5	4	3	2	1	0	EIMSK
0x1D (0x3D)	-	-	-	-	-	-	INT1	INT0	
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Khi thanh ghi INT0 hoặc INT1 hoặc cả hai được cấu hình ở mức 1 và bit thứ 7 – bit I trong thanh ghi SREG được cấu hình là mức 1 thì ngắt ngoài được xác lập. Tùy thuộc vào 2 bit ISC01, ISC00 đối với ngắt 0 và ISC11, ISC10 đối với ngắt 1 trong thanh ghi EICRA mà ngắt ngoài xảy ra trong trường hợp rising edge, low edge, low level hoặc logical change. Việc thiết lập này sẽ vẫn được phát hiện ngay cả khi chân INT0 và INT1 là ngõ ra.

5.3.3. Thanh ghi cờ ngắt ngoài – EIFR

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	-	-	-	-	-	-	INTF1	INTF0	EIFR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Hai bit INTF0 và INTF1 sẽ được ghi ở mức 1 nếu chân INT0 và INT1 bắt được một tín hiệu ngắt thỏa mãn yêu cầu của chúng. Nếu bit thứ 7 – bit I của thanh ghi SREG và bit INT0:1 của thanh ghi EIMSK ở mức 1 thì chương trình sẽ nhảy đến vector ngắt tương ứng, cờ ngắt được xóa khi ngắt đã được thực hiện, ngoài ra cờ ngắt sẽ bị xóa nếu ghi mức logic 1 vào nó.

5.3.4. Thanh ghi Pin Change Interrupt Control – PCICR

Bit	7	6	5	4	3	2	1	0	
(0x68)	-	-	-	-	-	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 0 – PCIE0 (Pin Change Interrupt Enable 0): khi ghi mức logic 1 vào bit PCIE0 và bit I của thanh ghi trạng thái SREG thì bất kỳ sự thay đổi mức logic nào xảy ra trên các chân PCINT[7:0] sẽ gây ra ngắt.

Tương tự với bit 0, hai bit PCIE1 và PCIE2 sẽ kiểm tra sự thay đổi mức logic xảy ra trên các chân tương ứng PCINT[14:8] và PCINT[23:16] và nhảy đến vector ngắt PCINT1 và PCINT2.

Các thanh ghi PCMSK0, PCMSK1 và PCMSK2 sẽ chịu trách nhiệm điều khiển các chân PCINT[7:0], PCINT[14:8] và PCINT[23:16].

5.3.5. Thanh ghi Pin Change Interrupt Flag – PCIFR

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	-	-	-	-	-	PCIF2	PCIF1	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Khi có bất kỳ sự thay đổi mức logic nào xảy ra ở các chân PCINT[7:0], PCINT[14:8], PCINT[23:16] thì lần lượt các bit PCIF0, PCIF1, PCIF2 được ghi lên mức 1, cũng giống như ngắt 0 và ngắt 1, MCU sẽ lập tức nhảy đến vector ngắt tương ứng và xóa cờ ngắt khi đã thực hiện xong (xóa khi ghi mức logic 1 vào nó).

5.3.6. Các thanh ghi Pin Change Mask – PCMSK

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0x6C)	-	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

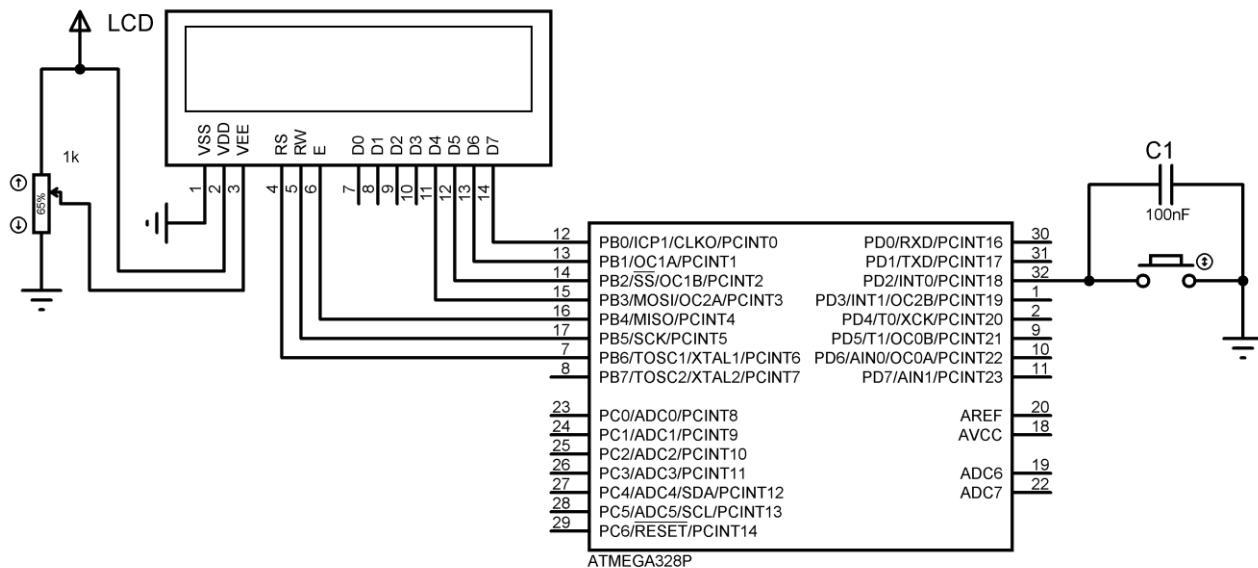
Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

Các thanh ghi PCMSK0, PCMSK1 và PCMSK2 điều khiển việc bật/tắt ngắt ở các chân PCINT[7:0], PCINT[14:8] và PCINT[23:16]. Nếu muốn sử dụng ngắt ở một trong các chân I/O trên, bắt buộc phải cấu hình bit tương ứng trong thanh ghi PCMSKx và thanh ghi PCICR lên mức logic 1.

5.4. Bài thực hành

Yêu cầu bài thực hành: Lập trình nhận tín hiệu nút nhấn thông qua INT0 và hiển thị giá trị biến đếm lên màn hình LCD. Sau mỗi lần nhấn nút thì biến đếm sẽ tăng một giá trị.

❖ Phản mô phỏng



Hình 5.3. Sơ đồ nối dây bài thực hành ngắt ngoài

❖ Phản lập trình

SourceCode_ATmega328P\P5_Interrupt_LCD	
1	#include <mega328p.h>
2	#include <alcd.h>
3	#include <delay.h>
4	#include <stdlib.h>
5	
6	unsigned char counter = 0;
7	unsigned char myStr[3];

```

8
9 interrupt [EXT_INT0] void ext_int0_isr(void)
10 {
11     counter++;      //dem len sau moi lan ngat
12     itoa(counter, myStr);
13     lcd_gotoxy(0,1);
14     lcd_puts("Counter:");
15     lcd_puts(myStr);
16 }
17
18 void main(void)
19 {
20     #pragma optsize-
21     CLKPR=(1<<CLKPCE);
22     CLKPR=0x00;
23     #ifdef _OPTIMIZE_SIZE_
24     #pragma optsize+
25     #endif
26
27     lcd_init(16);
28     lcd_gotoxy(3,0);
29     lcd_puts("INTERRUPT");
30     lcd_gotoxy(0,1);
31     lcd_puts("Counter:");
32     itoa(counter, myStr);
33     lcd_puts(myStr);
34
35     // su dung chan PD2 de doc nut nhan
36     // cau hinh dien tro keo len chan PD2
37     PORTD |= (1<<PORTD2);    //PORTD.2 = 1;
38     DDRD  &= ~(1<<DDD2);    //DDRD.2 = 0;
39
40     //su dung ngat 0, lay canh xuong
41     EICRA = (1<<ISC01) | (0<<ISC00);
42     EIMSK = (0<<INT1) | (1<<INT0);
43     EIFR  = (0<<INTF1) | (1<<INTF0);

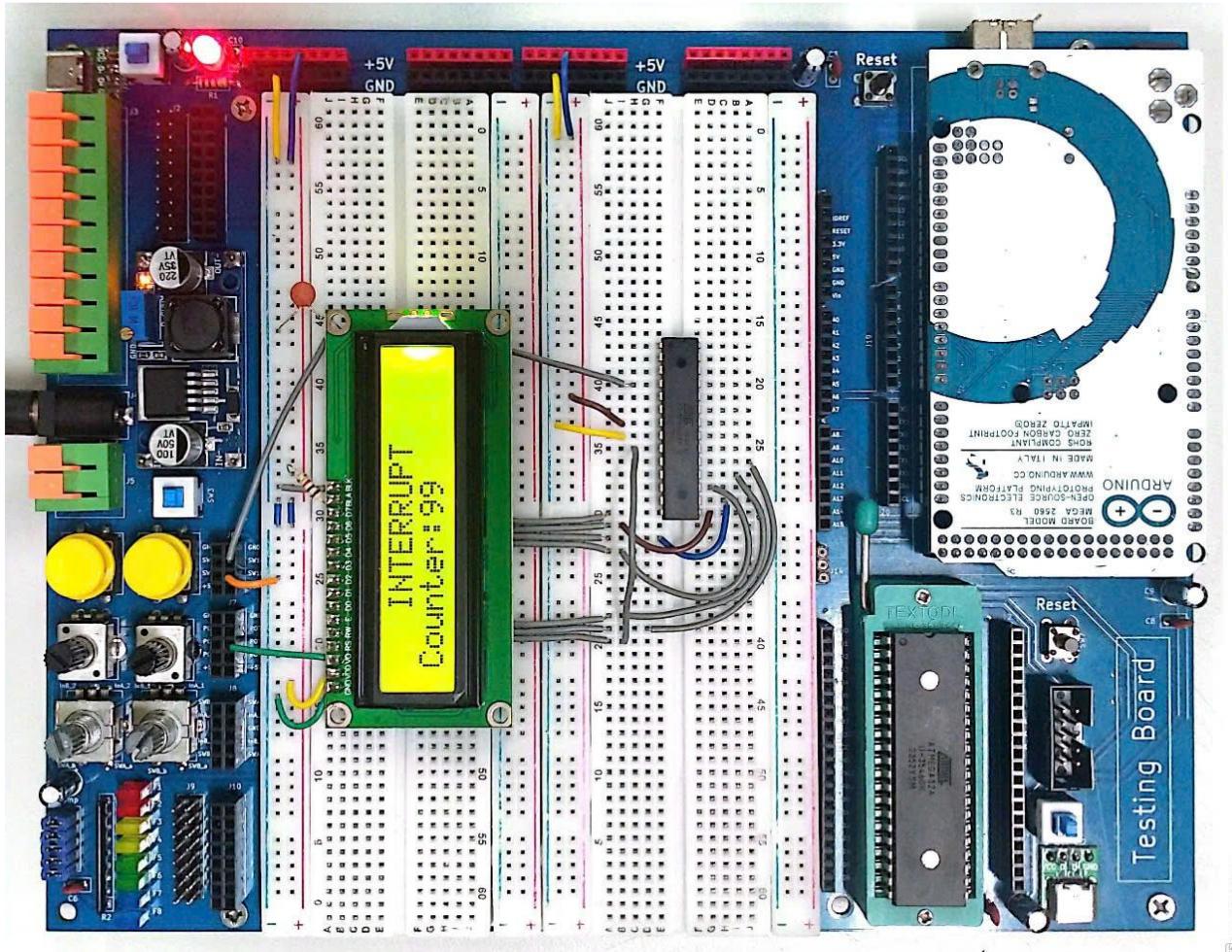
```

```

44 #asm("sei")
45 while (1){}
46 }

```

❖ Phân mô hình



Hình 5.4. Mô hình thực tế bài thực hành ngắt ngoài

CHƯƠNG 6. BỘ CHUYỂN ĐỔI ADC

6.1. Cấu trúc bộ ADC

6.1.1. Giới thiệu

Trong các hệ thống đo lường và điều khiển sử dụng vi điều khiển, bộ chuyển đổi tương tự - số (ADC) đóng vai trò rất quan trọng. Lý do là bởi dữ liệu trong thế giới thực thường ở dạng tương tự (analog). Chẳng hạn, nhiệt độ môi trường có thể thay đổi liên tục từ 25°C vào buổi sáng đến 32°C vào buổi trưa và trong quá trình đó nhiệt độ có thể đi qua vô số giá trị trung gian. Một đại lượng như nhiệt độ thay đổi liên tục và không rời rạc được gọi là dữ liệu analog. Ngược lại, vi điều khiển là một thiết bị số (digital) chỉ có khả năng xử lý các giá trị rời rạc dưới dạng tổ hợp các mức nhị phân 0 và 1.

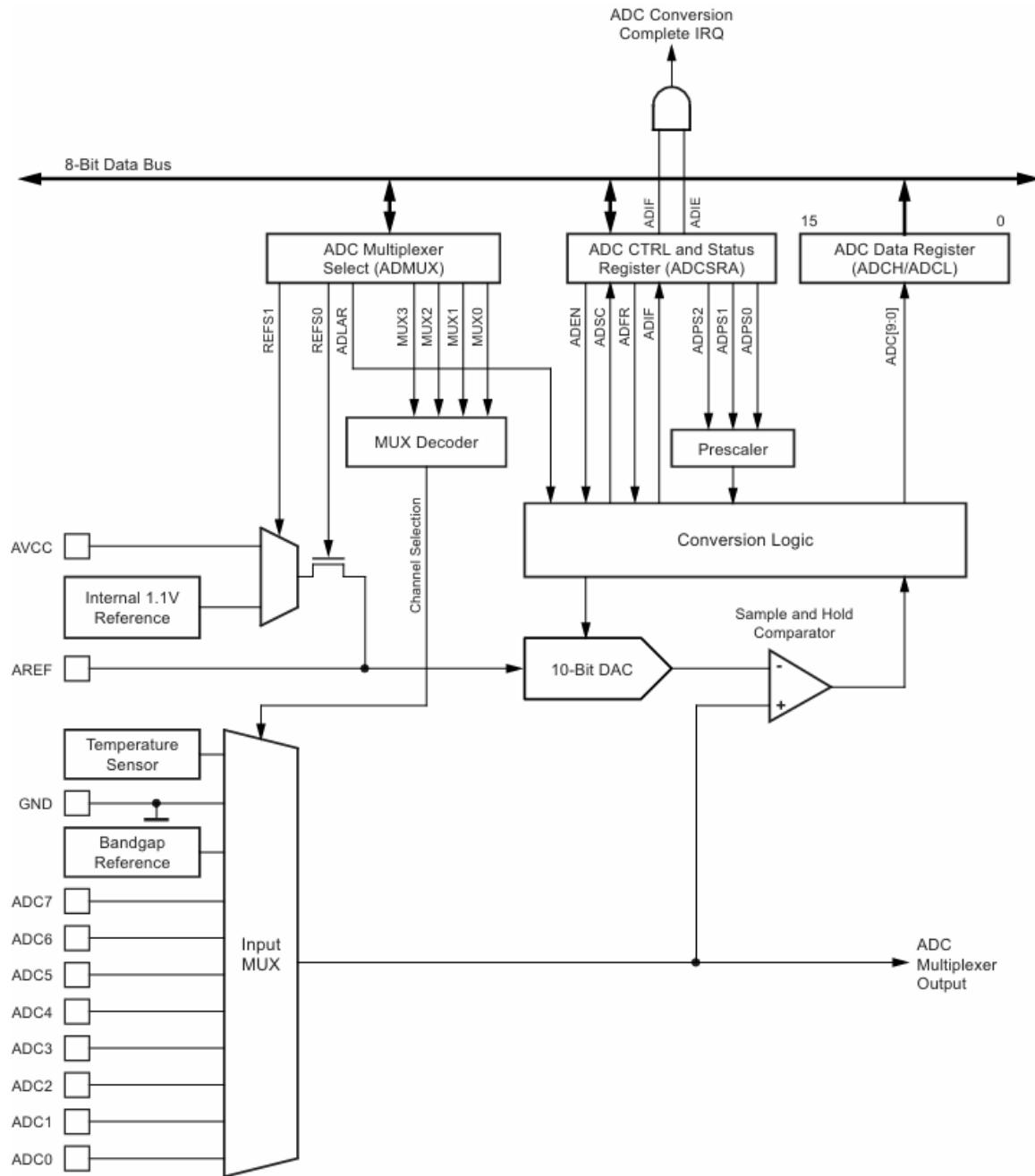
Giả sử chúng ta cần lưu giá trị nhiệt độ vào một thanh ghi 8 bit của vi điều khiển, với giá trị từ 0°C đến 255°C. Một thanh ghi 8 bit có khả năng biểu diễn 256 giá trị nguyên, từ 0 đến 255. Vì vậy, các giá trị nhiệt độ như 28.123°C sẽ không thể được lưu trữ chính xác. Điều này cho thấy rằng dữ liệu analog đã được chuyển đổi thành dữ liệu digital, hay còn gọi là quá trình “số hóa” (digitize). Việc chuyển đổi này được thực hiện thông qua một thiết bị chuyển đổi từ tín hiệu tương tự sang tín hiệu số, được gọi là ADC (Analog to Digital Converter).

Bộ chuyển đổi ADC của ATMEGA328P có độ phân giải 10 bit, với sai số tích phân phi tuyến tối đa là 0.5 LSB và độ chính xác tuyệt đối trong khoảng ± 2 LSB. Thời gian chuyển đổi dao động từ 65 đến 260 μ s, đạt tốc độ tối đa lên đến 15kSPS (15 nghìn mẫu mỗi giây). Bộ ADC hỗ trợ 6 kênh đầu vào cùng với 2 kênh đầu vào bổ sung và một kênh đầu vào dành cho cảm biến nhiệt độ. Dải điện áp đầu vào của ADC từ 0 đến V_{CC}, với tùy chọn sử dụng điện áp tham chiếu 1.1V. Bộ chuyển đổi có thể hoạt động ở chế độ tự do (free running) hoặc chế độ chuyển đổi đơn lẻ (single conversion) và hỗ trợ ngắt khi quá trình chuyển đổi ADC hoàn tất. Hơn nữa, bộ ADC còn tích hợp tính năng giảm nhiễu trong chế độ ngủ (sleep mode noise canceler).

Giá trị nhỏ nhất đại diện cho mức GND, trong khi giá trị lớn nhất tương ứng với điện áp trên chân AREF trừ đi 1 LSB. Người dùng có thể tùy chọn kết nối điện áp tham chiếu AVCC hoặc tham chiếu nội 1.1V đến chân AREF bằng cách thiết lập các bit REFSn trong thanh ghi ADMUX. Để tăng khả năng chống nhiễu, điện áp tham chiếu nội có thể được ghép nối với một tụ điện bên ngoài tại chân AREF.

Để chọn kênh đầu vào analog cần cấu hình các bit MUX trong thanh ghi ADMUX. ADC cho phép sử dụng các chân đầu vào, GND hoặc điện áp tham chiếu bandgap cố định làm đầu vào. Việc kích hoạt ADC được thực hiện bằng cách bật bit ADEN (ADC Enable) trong thanh ghi ADCSRA. Lưu ý rằng các thiết lập về kênh đầu vào và điện áp tham chiếu sẽ chỉ có hiệu lực sau khi bit ADEN được bật. Khi bit ADEN bị tắt, ADC sẽ không tiêu thụ năng lượng, do đó, nên tắt ADC trước khi đưa vi điều khiển vào chế độ ngủ để tiết kiệm năng lượng.

Bộ chuyển đổi ADC tạo ra dữ liệu 10 bit và lưu trữ trong hai thanh ghi ADCH và ADCL. Theo mặc định, kết quả được căn phải, tuy nhiên, người dùng có thể điều chỉnh để căn trái bằng cách thiết lập bit ADLAR trong thanh ghi ADMUX. Khi kết quả được căn trái và độ chính xác yêu cầu không vượt quá 8 bit, chỉ cần đọc giá trị từ thanh ghi ADCH. Ngược lại, nếu muốn đảm bảo dữ liệu thuộc cùng một lần chuyển đổi, thứ tự đọc phải là đọc ADCL trước, sau đó mới đến ADCH.



Hình 6.1. Sơ đồ khái của bộ chuyển đổi ADC [2]

6.1.2. Độ phân giải

Độ phân giải của ADC thể hiện số lượng bit cần thiết để biểu diễn tất cả các mức giá trị số ở đầu ra. Ví dụ, với 8 mức giá trị đầu ra, cần 3 bit nhị phân để mã hóa các mức này. Vì

vậy, một bộ chuyển đổi ADC sử dụng 7 bộ so sánh sẽ có độ phân giải là 3 bit. Nói chung, nếu ADC có độ phân giải là n bit, đầu ra sẽ có tổng cộng 2^n mức giá trị số có thể xuất hiện ở đầu ra.

Để thiết kế một mạch chuyển đổi flash ADC với độ phân giải n bit, cần đến 2^n-1 bộ so sánh. Số lượng này tăng nhanh khi độ phân giải cao, khiến flash ADC thường chỉ có độ phân giải dưới 8 bit để duy trì tính khả thi trong thiết kế.

Độ phân giải ảnh hưởng trực tiếp đến chất lượng của quá trình chuyển đổi ADC. Khi lựa chọn độ phân giải, cần xem xét kỹ lưỡng giữa yêu cầu về độ chính xác và khả năng xử lý của vi điều khiển.

6.1.3. Điện áp tham chiếu

Trong các ứng dụng sử dụng bộ chuyển đổi ADC, việc lựa chọn điện áp tham chiếu (V_{ref}) là yếu tố quan trọng để đảm bảo độ chính xác của quá trình chuyển đổi. Điện áp tham chiếu được định nghĩa là mức điện áp lớn nhất mà bộ ADC có thể chuyển đổi. Điều này đặc biệt quan trọng khi các ứng dụng yêu cầu dải đo khác nhau.

Giả sử nếu người dùng A chỉ cần đo điện áp trong khoảng 0-1V, trong khi người dùng B cần đo trong khoảng 0-5V, việc cả hai sử dụng một bộ ADC có khả năng chuyển đổi tối đa 5V sẽ dẫn đến việc người dùng A không tận dụng tối đa độ chính xác của thiết bị. Để giải quyết vấn đề này, điện áp tham chiếu thường được cấu hình bởi người sử dụng nhằm tối ưu hóa độ phân giải của ADC cho từng ứng dụng cụ thể.

Ví dụ với một bộ ADC 10-bit có $V_{ref} = 3V$, nếu điện áp đầu vào là 1V, giá trị số nhận được từ quá trình chuyển đổi sẽ được tính như sau:

$$\text{Giá trị số} = 1023 \times \frac{1}{3} = 34$$

Trong đó, 1023 là giá trị lớn nhất mà một bộ ADC 10-bit có thể biểu diễn. Như vậy, việc chọn điện áp tham chiếu phù hợp là cần thiết để tối ưu hóa độ chính xác của quá trình

chuyển đổi, đảm bảo rằng Vref không nhỏ hơn giá trị lớn nhất của điện áp đầu vào, đồng thời tránh việc chọn Vref quá lớn làm giảm độ phân giải của hệ thống.

6.2. Các thanh ghi của bộ ADC

6.2.1. Thanh ghi lựa chọn ghép kênh ADC – ADMUX

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Là thanh ghi điều khiển việc lựa chọn điện áp tham chiếu và ghép kênh ADC. Các chức năng cụ thể như sau:

Bit REFS[1:0] cho phép lựa chọn điện áp tham chiếu cho bộ ADC, tùy thuộc vào mức logic gán vào hai bit này mà ATMEGA328P sẽ lựa chọn giá trị điện áp để so sánh. Bảng bên dưới thể hiện cách cấu hình điện áp tham chiếu cho bộ chuyển đổi ADC.

Bảng 6.1. Lựa chọn điện áp tham chiếu [2]

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, internal V_{REF} turned off
0	1	Avcc with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V voltage reference with external capacitor at AREF pin

Bit ADLAR (ADC Left Adjust Result): là thanh ghi ảnh hưởng đến việc trình bày kết quả chuyển đổi ADC. Do bộ ADC có độ phân giải 10 bit nên cần phải có hai thanh ghi ADCL (8 bit thấp) và ADCH (8 bit cao) để chứa giá trị, vì thế có thể tùy chỉnh việc lưu kết quả sau khi chuyển đổi về phía trái hoặc phải. Nếu bit ADLAR ở mức logic 0 thì kết quả sẽ được điều chỉnh về phía phải và ngược lại khi ADLAR ở mức logic 1 thì kết quả được điều chỉnh về phía

trái. Do đó, việc chuyển đổi bit ADLAR sẽ làm ảnh hưởng trực tiếp đến quá trình lưu trữ dữ liệu của ADC.

Các thanh ghi MUX[3:0] (Analog Channel Selection Bits): là 4 bit cho phép chọn kênh đầu vào và chế độ của bộ ADC. Những thay đổi ảnh hưởng đến 4 bit này diễn ra trong quá trình đọc ADC sẽ không có hiệu lực cho đến khi quá trình đọc kết thúc. Vì thế trước khi thực hiện chuyển đổi ADC cần phải thiết lập các bit MUX để chọn kênh và chế độ làm việc.

Bảng 6.2. Lựa chọn kênh đầu vào của bộ ADC [2]

MUX3.0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8 (Temperature sensor)
1110	1.1V (V_{BG})
1111	0V (GND)

6.2.2. Điều khiển ADC và thanh ghi trạng thái A – ADCSRA

Bit	7	6	5	4	3	2	1	0	ADCSRA
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7: ADEN (ADC Enable): Cho phép bộ chuyển đổi ADC được sử dụng khi ghi một mức logic 1 vào bit này. Tuy nhiên, vẫn cần kích hoạt bit ADSC để quá trình chuyển đổi được bắt đầu.

Bit 6: ADSC (ADC Start Conversion): Ghi mức logic 1 vào bit này sẽ khởi động quá trình chuyển đổi ADC. Khi quá trình chuyển đổi hoàn tất, bit này sẽ tự động trả về giá trị 0, do đó không cần ghi mức logic 0 vào bit này.

Bit 5: ADATE (ADC Auto Trigger Enable): Ghi mức 1 vào bit này sẽ kích hoạt chế độ tự động khởi động của ADC. Quá trình chuyển đổi sẽ bắt đầu khi phát hiện sườn lên của tín hiệu kích hoạt được chọn. Nguồn tín hiệu kích hoạt được thiết lập thông qua các bit ADTS trong thanh ghi ADCSRB.

Bit 4: ADIF (ADC Interrupt Flag): Khi quá trình chuyển đổi kết thúc và các thanh ghi dữ liệu được cập nhật thì bit ADIF sẽ tự động xóa bằng cách ghi vào nó mức logic 1.

Bit 3: ADIE (ADC Interrupt Enable): Đây là bit cho phép ngắn. Khi bit này được kích hoạt, một ngắn sẽ xảy ra sau khi chuyển đổi hoàn tất và dữ liệu trong các thanh ghi ADCL và ADCH đã được cập nhật.

Bit [2:0] ADPS (ADC Prescaler Select Bits): Là các bit chọn hệ số chia xung nhịp cho bộ ADC.

Bảng 6.3. Lựa chọn hệ số chia ADC [2]

ADPS2	ADPS2	ADPS2	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32

1	1	0	64
1	1	1	128

6.2.3. Thanh ghi dữ liệu ADC – ADCL và ADCH

Là thanh ghi chứa dữ liệu của bộ chuyển đổi ADC, do ATMEGA328P có độ phân giải 10 bit nên việc chứa dữ liệu chuyển đổi cần đến hai thanh ghi. Tuy nhiên, hai thanh ghi có đến 16 bit (8 bit trên mỗi thanh ghi) sẽ tạo ra các bit trống và có thể lựa chọn cách lưu trữ về phía trái hay phải thông qua bit ADLAR trong thanh ghi ADMUX.

❖ ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	-	-	-	-	-	-	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

❖ ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	-	-	-	-	-	-	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Thông thường hai thanh ghi sẽ được sắp xếp theo định dạng ADLAR = 0. Ví dụ, kết quả ADC là 0b10 1011 1101 và ADLAR = 0 thì giá trị của hai thanh ghi:

- ADCH = 0b0000 0010

- ADCL = 0b1011 1101

6.2.4. Điều khiển ADC và Thanh ghi trạng thái B – ADCSRB

Bit	7	6	5	4	3	2	1	0	ADCSRB
(0x7B)	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0	
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 2:0 – ADTS[2:0] (ADC Auto Trigger Source): Khi bit ADATE trong thanh ghi ADCSRA được thiết lập ở mức 1, giá trị của các bit ADTS2:0 sẽ quyết định nguồn kích nào sẽ được dùng để khởi động quá trình chuyển đổi ADC. Quá trình chuyển đổi được kích hoạt bởi cạnh lên của cờ ngắt đã được chọn. Lưu ý, nếu chuyển từ một nguồn kích hoạt đã bị xóa sang một nguồn đang được thiết lập sẽ tạo ra một kích hoạt. Tuy nhiên, khi chuyển sang chế độ Free running mode (ADC tự động chuyển đổi liên tục) sẽ không có sự kích hoạt nào xảy ra, ngay cả khi cờ ngắt ADC được thiết lập.

Bảng 6.4. Nguồn kích ADC trong chế độ Auto Trigger [2]

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free running mode
0	0	1	Analog comparator
0	1	0	External interrupt request 0
0	1	1	Timer/Counter0 compare match A
1	0	0	Timer/Counter0 overflow
1	0	1	Timer/Counter1 compare match A
1	1	0	Timer/Counter1 overflow
1	1	1	Timer/Counter1 capture event

6.2.5. Thanh ghi Digital Input Disable 0 – DIDR0

Bit	7	6	5	4	3	2	1	0	
(0x7B)	-	-	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	DIDR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Khi bit được đặt thành 1, bộ đệm đầu vào kỹ thuật số trên chân ADC tương ứng sẽ bị vô hiệu hóa. Lúc này, giá trị đọc từ bit tương ứng trong thanh ghi PIN luôn là 0. Nếu một tín hiệu analog được đưa vào chân ADC[5:0] và đầu vào kỹ thuật số từ chân này không được sử dụng, nên kích hoạt bit này bằng 1 để giảm mức tiêu thụ năng lượng của bộ đệm đầu vào kỹ thuật số. Lưu ý rằng các chân ADC7 và ADC6 không được trang bị bộ đệm đầu vào kỹ thuật số, do đó không cần thiết lập bit vô hiệu hóa đầu vào kỹ thuật số cho các chân này.

6.3. Bài thực hành

Yêu cầu bài thực hành: Đọc giá trị cảm biến nhiệt độ LM35 thông qua ADC và hiển thị lên màn hình LCD. LM35 được kết nối với chân PA0, đây là chân ADC0 của vi điều khiển và bộ ADC được sử dụng với độ phân giải 10 bit, ngoài ra cần nối chân AVCC của vi điều khiển với điện áp tham chiếu 5V.

- Giá trị ADC được chuyển đổi thành điện áp đầu ra của LM35 theo công thức:

$$V_{out} = \frac{ADC_{value} \times V_{AVCC}}{1024} [mV] \quad (1)$$

Với: V_{AVCC} là điện áp tham chiếu ở chân AVCC ($5V = 5000mV$)

Do sử dụng độ phân giải 10 bit nên phải chia cho 1024 (từ 0 đến 1023)

ADC_value là giá trị mà ADC đọc được sau mỗi frame

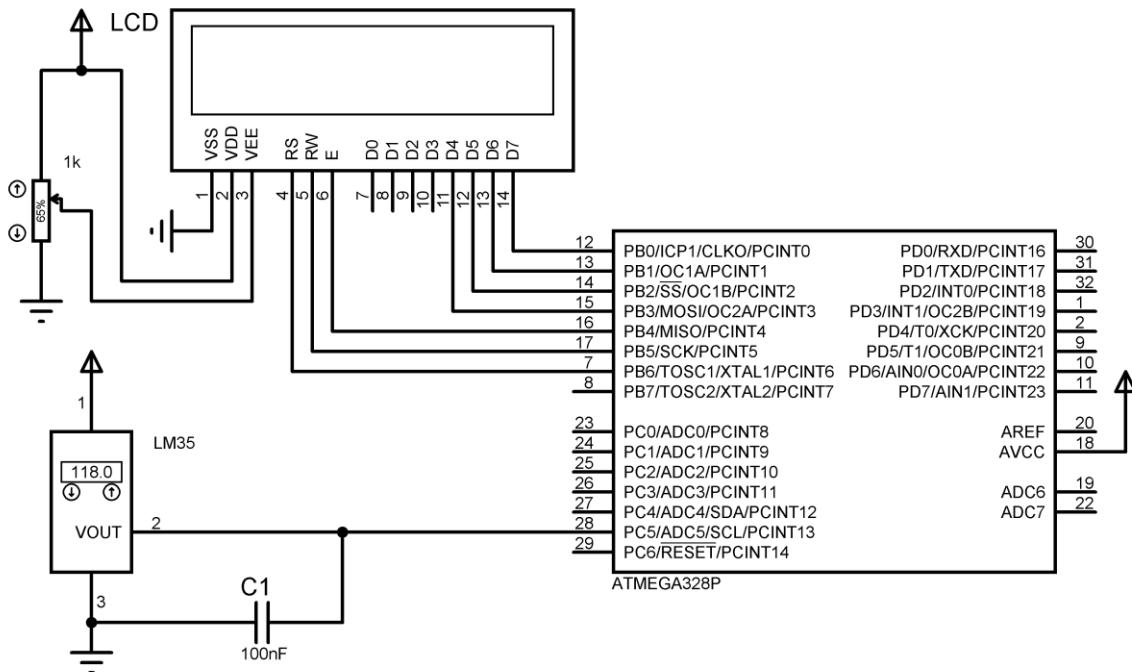
- Nhiệt độ được tính toán dựa trên công thức:

$$T = \frac{V_{out}}{10} = \frac{ADC_{value} \times V_{AVCC}}{1024} \times \frac{1}{10} [^{\circ}C] \quad (2)$$

do 10 mV tương đương với 1°C nên từ đó kết luận:

$$T = \frac{ADC_{value} \times 5000}{1024} \times \frac{1}{10} = \frac{ADC_{value} \times 125}{256} \quad (3)$$

❖ Phần mô phỏng



Hình 6.2. Sơ đồ nối dây bài thực hành với LM35

❖ Phần lập trình

	SourceCode _ATMEGA328P\P6_ADC_LM35
1	#include <mega328p.h>
2	#include <delay.h>
3	#include <alcd.h>
4	
5	#define ADC_VREF_TYPE ((0<<REFS1) (1<<REFS0) (0<<ADLAR))
6	
7	void Display_Value(unsigned int number);
8	unsigned int resultADC5;
9	unsigned int Temp;
10	

```

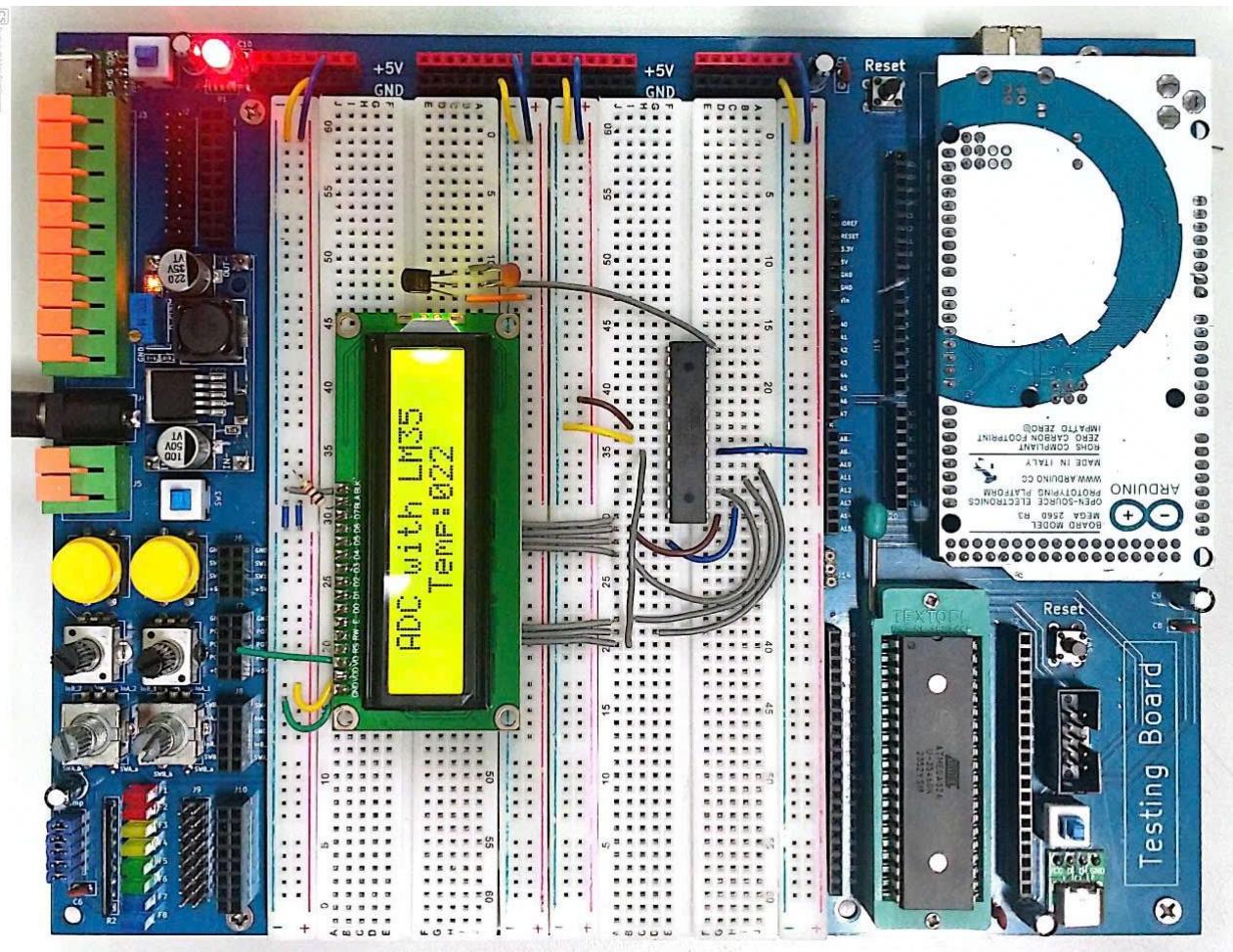
11 unsigned int read_adc(unsigned char adc_input)
12 {
13     ADMUX=adc_input | ADC_VREF_TYPE;
14     delay_us(10);
15     // bat dau qua trinh chuyen doi
16     ADCSRA|=(1<<ADSC);
17     //cho den khi qua trinh chuyen doi ket thuc
18     while ((ADCSRA & (1<<ADIF))==0);
19     ADCSRA|=(1<<ADIF);
20     return ADCW;
21 }
22 void main(void) {
23     #pragma optsize-
24     CLKPR = (1<<CLKPCE);
25     CLKPR = 0x00;
26     #ifdef _OPTIMIZE_SIZE_
27     #pragma optsize+
28     #endif
29     ADMUX=ADC_VREF_TYPE;
30     // kich hoat ADC, chon bo chia 8 ⇔ 125.000kHz
31     ADCSRA =(1<<ADEN) | (1<<ADATE) | (1<<ADPS2) | (1<<ADPS1);
32     ADCSRB=(0<<ADTS2) | (0<<ADTS1) | (0<<ADTS0);
33     lcd_init(16);
34     lcd_gotoxy(1,0);
35     lcd_puts("ADC with LM35 ");
36     delay_ms(1000);
37     while (1) {
38         //du lieu duoc doc tu ADC
39         resultADC5 = read_adc(5);
40         //Temp = resultADC5*500/1024
41         Temp = (resultADC5*125/256); //da rut gon
42         lcd_gotoxy(4,1);
43         lcd_putsf("Temp:");
44         Display_Value(Temp);
45     }
46 }
```

```

47 void Display_Value(unsigned int number) {
48     unsigned int tram,chuc,donvi;
49     tram = number/100%10;
50     chuc = (number/10)%10;
51     donvi = number%10;
52     lcd_putchar(tram+48);
53     lcd_putchar(chuc+48);
54     lcd_putchar(donvi+48);
55 }

```

❖ Phân mô hình



Hình 6.3. Mô hình thực tế bài thực hành với LM35

CHƯƠNG 7. LẬP TRÌNH ĐIỀU KHIỂN

7.1. Lập trình cổng xuất nhập

Lập trình xuất nhập (I/O) cổng trên vi điều khiển AVR sử dụng ngôn ngữ C (CAVR) liên quan đến việc điều khiển trạng thái của các chân trên vi điều khiển. Các chân được tổ chức thành các nhóm gọi là Port. Mỗi port có thể có nhiều chân (pin) và mỗi chân có thể được cấu hình là ngõ vào (input) hoặc ngõ ra (output).

- DDRx (Data Direction Register) : Thanh ghi xác định hướng của từng chân (input hoặc output). Với x là B, C, D; gán bit là 1 để cấu hình chân làm ngõ ra (Output); gán bit là 0 để cấu hình chân làm ngõ vào (Input).

7.1.1. Các thanh ghi lập trình cổng xuất nhập

❖ Thanh ghi DDRB – The port B data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04(0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

❖ Thanh ghi DDRC – The port C data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x07(0x27)	-	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R	R/W							
Initial Value	0	0	0	0	0	0	0	0	

❖ Thanh ghi DDRD – The port D data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A(0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

- PORTx (Data Register) : Thanh ghi xuất dữ liệu, kiểm soát trạng thái của chân nếu nó được cấu hình làm ngõ ra. Với x là B, C, D

+ Gán bit là 1 để xuất mức logic cao (HIGH).

+ Gán bit là 0 để xuất mức logic thấp (LOW).

- Còn nếu nó được cấu hình làm ngõ vào

+ Gán bit là 1 để kích hoạt điện trở kéo lên.

+ Gán bit là 0 để tắt điện trở kéo lên.

❖ Thanh ghi PORTB – The port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05(0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W								
InitialValue	0	0	0	0	0	0	0	0	

❖ Thanh ghi PORTC – The port C Data Register

Bit	7	6	5	4	3	2	1	0	
0x08(0x28)	-	PORTC6	PORTC5	PORTC4	PORTC3	PORTC	PORTC1	PORTC0	PORTC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
InitialValue	0	0	0	0	0	0	0	0	

❖ Thanh ghi PORTD – The port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B(0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W								
InitialValue	0	0	0	0	0	0	0	0	

- PINx (Input Pins Register) : Thanh ghi đọc dữ liệu từ các chân khi chúng được cấu hình làm ngõ vào (với x là B, C, D). Các bit của thanh ghi này cho biết trạng thái của chân (0 hoặc 1).

❖ Thành ghi PINB – The port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	PINB
0x03(0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W								
Initial Value	N/A								

❖ Thành ghi PINC – The port C Input Pins Address

Bit	7	6	5	4	3	2	1	0	PINC
0x06(0x26)	-	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R/W							
Initial Value	0	N/A							

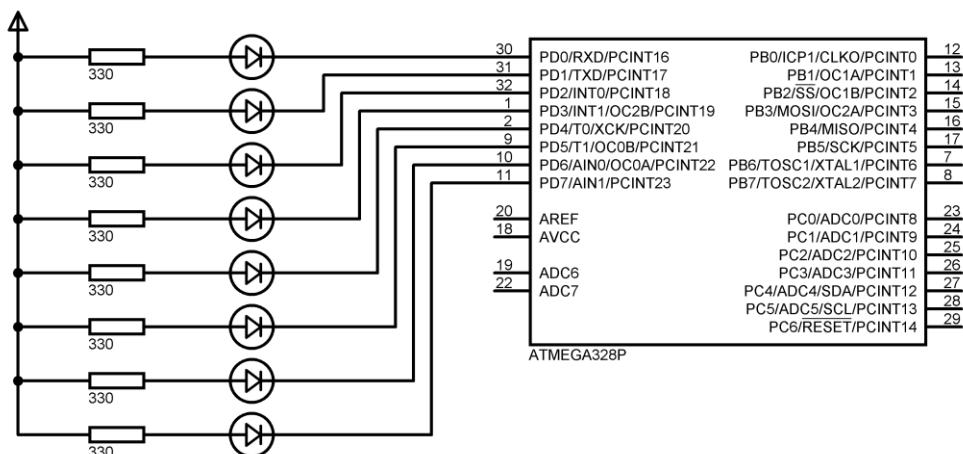
❖ Thành ghi PIND – The port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	PIND
0x09(0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R/W								
Initial Value	N/A								

7.1.2. Bài thực hành

Yêu cầu bài thực hành: Kết nối tám LED đơn với vi điều khiển, lập trình tạo hiệu ứng nháy LED.

❖ Phần mô phỏng



Hình 7.1. Sơ đồ kết nối bài thực hành LED đơn

❖ Phân lập trình

	SourceCode_ATMEGA328P\P7_1_BlinkLED
1	#include <mega328p.h>
2	#include <delay.h>
3	void main(void){
4	// Khai bao PORTD la ngo xuat tin hieu
5	DDRD = 0b11111111; //0xFF
6	//Keo PORTD len muc logic 1, tat ca den deu tat
7	PORTD = 0b11111111; //0xFF
8	// vong lap vo tan
9	while(1)
10	{
11	int I;
12	for(i=0; i<8; i++){
13	PORTB = ~(1<<i);
14	delay_ms(100);
15	}
16	}
17	}

7.2. Lập trình các ứng dụng điều khiển

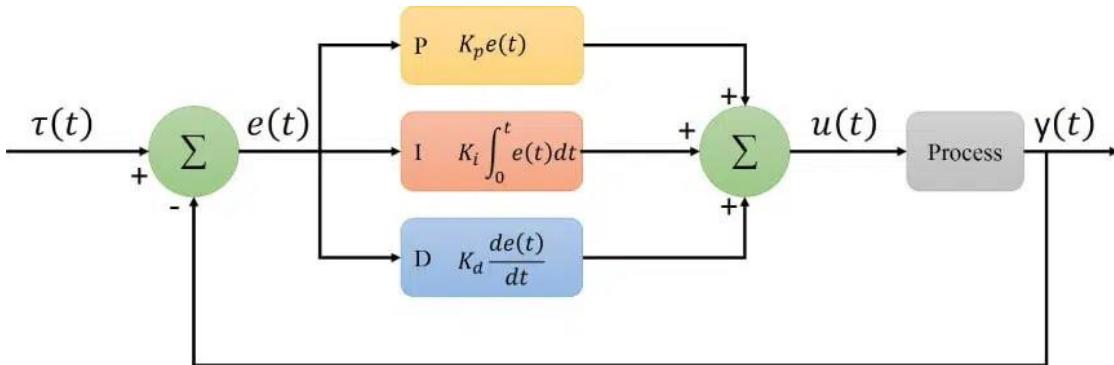
7.2.1. Điều khiển động cơ DC

7.2.1.1. Giới thiệu bộ điều khiển PID

PID (Proportional Integral Derivative) là một phương pháp điều khiển vòng phản hồi được ứng dụng rộng rãi trong các hệ thống điều khiển công nghiệp. Đây là bộ điều khiển phổ biến trong các hệ thống vòng kín và các hệ thống có tín hiệu phản hồi, nhờ khả năng điều chỉnh hiệu quả để đạt được sự ổn định và chính xác trong hoạt động. Chức năng chính của bộ điều khiển PID là tính toán và xử lý sai số, được định nghĩa là chênh lệch giữa giá trị thực đo lường và giá trị mong muốn do người dùng hoặc nhà thiết kế đặt ra. Dựa trên các thuật toán điều chỉnh tỉ lệ, tích phân và vi phân, PID tạo ra tín hiệu điều khiển phù hợp để giảm thiểu sai

số một cách tối ưu. Kết quả là hệ thống không chỉ vận hành chính xác mà còn nâng cao hiệu suất tổng thể, đáp ứng tốt các yêu cầu kỹ thuật và ứng dụng thực tế.

Bộ điều khiển PID thường được gọi là điều khiển ba khâu, hoạt động dựa trên ba tham số chính: tỉ lệ (P), tích phân (I), và đạo hàm (D). Mỗi khâu đóng một vai trò khác nhau trong việc hiệu chỉnh sai số của hệ thống. Thành phần tỉ lệ (P) phản ánh trực tiếp mức độ sai số hiện tại, thành phần tích phân (I) xem xét ảnh hưởng của các sai số tích lũy từ quá khứ, còn thành phần đạo hàm (D) dự đoán xu hướng sai số trong tương lai dựa trên tốc độ thay đổi hiện tại. Sự kết hợp của cả ba khâu này tạo ra tín hiệu điều khiển, giúp điều chỉnh hệ thống qua các thiết bị như van, bộ gia nhiệt, hoặc các phần tử điều khiển khác. Thông qua việc điều chỉnh ba hằng số của bộ điều khiển PID, ta có thể tinh chỉnh đáp ứng của hệ thống để phù hợp với những yêu cầu đặc thù. Đáp ứng của bộ điều khiển có thể được mô tả dưới dạng độ nhạy sai số của bộ điều khiển, giá trị mà bộ điều khiển vọt lô điểm đặt và giá trị dao động của hệ thống. Lưu ý là công dụng của giải thuật PID trong điều khiển không đảm bảo tính tối ưu hoặc ổn định cho hệ thống.



Hình 7.2. Sơ đồ khối bộ điều khiển PID [10]

Tín hiệu điều khiển đầu ra của bộ điều khiển PID, thường được gọi là biến điều khiển (MV), là tổng hợp của ba thành phần: tỉ lệ, tích phân, và đạo hàm. Biểu thức tổng quát được viết như sau:

$$MV(t) = P_{OUT} + I_{OUT} + D_{OUT} \quad (4)$$

Trong đó :

P_{OUT} : Thành phần đầu ra từ khâu tỉ lệ, thể hiện tác động dựa trên sai số hiện tại.

I_{OUT} : Thành phần đầu ra từ khâu tích phân, phản ánh ảnh hưởng của tổng sai số tích lũy qua thời gian.

D_{OUT} : Thành phần đầu ra từ khâu đạo hàm, dự đoán xu hướng của sai số dựa trên tốc độ thay đổi hiện tại.

Ba khâu này phối hợp chặt chẽ để tạo ra tín hiệu điều khiển, giúp hệ thống phản ứng một cách hiệu quả với sai số và duy trì trạng thái ổn định theo mong muốn.

❖ Khâu tỉ lệ

Khâu tỉ lệ (P), còn được gọi là khâu độ lợi, điều chỉnh đầu ra của hệ thống dựa trên sai số hiện tại. Tác động này được tính toán bằng cách nhân sai số với một hằng số tỉ lệ K_p được gọi là hệ số tỉ lệ. Công thức khâu tỉ lệ được biểu diễn như sau:

$$P_{OUT} = K_p \cdot e(t) \quad (5)$$

Trong đó :

P_{OUT} : Đầu ra khâu tỉ lệ.

K_p : Hệ số tỉ lệ, quyết định độ mạnh của đáp ứng.

$e(t)$: Sai số, được tính bằng hiệu giữa giá trị đặt SP (Set Point) và giá trị đo thực tế PV (Process Variable)

$$e(t) = SP - PV$$

t : Thời gian hiện tại

Hệ số tỉ lệ K_p đóng vai trò quan trọng trong việc điều chỉnh đáp ứng của hệ thống. Nếu K_p quá lớn, sự thay đổi nhỏ trong sai số sẽ dẫn đến biến động mạnh ở đầu ra, làm hệ

thông dễ trở nên không ổn định. Ngược lại, nếu K_p quá nhỏ, hệ thống sẽ phản ứng chậm hoặc không đủ nhạy cảm với sai số, dẫn đến hiệu suất kém trong việc điều chỉnh sai lệch. Việc lựa chọn giá trị K_p phù hợp là yếu tố then chốt để đảm bảo hệ thống đạt được sự cân bằng giữa độ nhạy và tính ổn định.

❖ Khâu tích phân

Khâu tích phân (I), đôi khi được gọi là khâu "reset," tạo ra tín hiệu điều chỉnh tỉ lệ thuận với cả biên độ sai số và thời gian mà sai số tồn tại. Bằng cách tính tích phân của sai số theo thời gian, khâu này phản ánh tác động tích lũy của các sai số trước đó, giúp loại bỏ sai số ổn định. Kết quả của phép tính này được nhân với hệ số độ lợi tích phân K_i và thêm vào đầu ra của bộ điều khiển. Biên độ của khâu tích phân trong tín hiệu điều chỉnh phụ thuộc trực tiếp vào giá trị K_i .

Công thức tính khâu tích phân được mô tả như sau:

$$I_{OUT} = K_i \int_0^t e(\tau) d\tau \quad (6)$$

Trong đó:

I_{OUT} : Đầu ra của khâu tích phân.

K_i : Độ lợi tích phân, một tham số điều chỉnh.

e : Sai số, được tính bằng $e = SP - PV$ (hiệu giữa giá trị đặt và giá trị đo thực tế).

t : Thời điểm hiện tại.

τ : Biến tích phân, dùng để biểu diễn tích lũy sai số qua thời gian.

Khi được kết hợp với khâu tỉ lệ, khâu tích phân giúp đẩy nhanh quá trình đưa hệ thống về giá trị đặt (Set Point) và loại bỏ sai số ổn định. Tuy nhiên, do khâu tích phân dựa vào tích lũy sai số trong quá khứ, nó có thể dẫn đến hiện tượng vọt lố, khi giá trị điều khiển vượt quá điểm đặt và gây ra dao động xung quanh giá trị mục tiêu. Việc điều chỉnh độ lợi tích phân K_i đóng vai trò quan trọng trong việc cân bằng giữa tốc độ đáp ứng và độ ổn định của hệ thống.

❖ Khâu đạo hàm

Khâu vi phân (D) tính toán tốc độ thay đổi của sai số theo thời gian, hay chính là đạo hàm của sai số bậc nhất, sau đó nhân giá trị này với một hệ số K_d , gọi là độ lợi vi phân. Khâu vi phân giúp phản ứng nhanh với những thay đổi đột ngột trong sai số, nhờ vậy đóng vai trò quan trọng trong việc kiểm soát tốc độ biến thiên của hệ thống.

Công thức của khâu vi phân được viết như sau:

$$D_{OUT} = K_d \cdot \frac{d}{dt} e(t) \quad (7)$$

Trong đó :

D_{OUT} : Đầu ra của khâu vi phân.

K_d : Độ lợi vi phân, một tham số điều chỉnh tác động của khâu này.

$e(t)$: Sai số hiện tại, được tính bằng hiệu giữa giá trị đặt SP và giá trị đo thực tế PV

$$e(t) = SP - PV$$

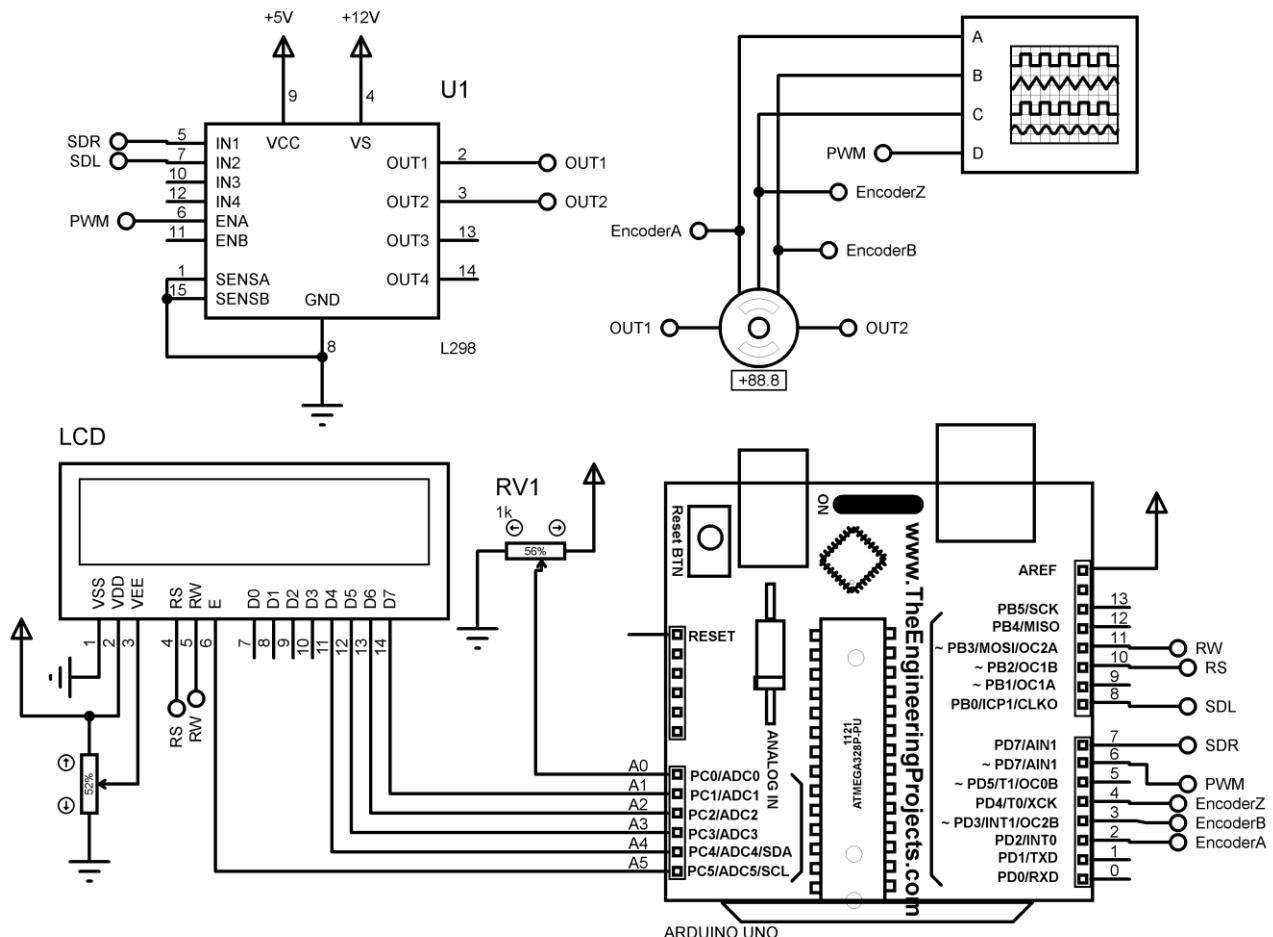
t : Thời gian hiện tại.

Khâu vi phân hoạt động như một cơ chế dự đoán, giúp làm chậm tốc độ thay đổi của đầu ra và giảm hiện tượng vọt lố do khâu tích phân gây ra, từ đó cải thiện sự ổn định của hệ thống. Tuy nhiên, đặc tính vi phân cũng làm tăng độ nhạy đối với nhiễu trong tín hiệu sai số, bởi nhiễu thường xuất hiện dưới dạng biến đổi nhanh theo thời gian. Nếu nhiễu quá lớn hoặc độ lợi K_d được thiết lập không phù hợp, khâu vi phân có thể làm hệ thống dao động và mất ổn định. Do nhược điểm này, trong thực tế, người ta thường sử dụng các bộ lọc vi phân xấp xỉ với băng thông giới hạn để giảm thiểu ảnh hưởng của nhiễu, đồng thời vẫn giữ được ưu điểm của khâu vi phân trong việc cải thiện đáp ứng động của hệ thống.

7.2.1.2. Bài thực hành

Yêu cầu bài thực hành: Điều khiển tốc độ của động cơ DC sử dụng phương pháp điều khiển PID, dùng một biến trớ để giả lập tín hiệu tốc độ mong muốn và hiển thị thông số lên màn hình LCD, bao gồm D là giá trị tốc độ mong muốn (vòng/phút), C là giá trị tốc độ hiện tại và E là giá trị sai số.

❖ Phản mô phỏng



Hình 7.3. Sơ đồ kết nối bài thực hành điều khiển động cơ DC

❖ Phần lập trình

	SourceCode_ATMEGA328P\P7_2_DC_Motor
1	#include <mega328p.h>
2	#include <delay.h>
3	#include <alcd.h>
4	#include <stdio.h>
5	#include <stdint.h>
6	
7	#define TCNT1 0xC1
8	#define PWM_R 6 //PD6 (OC0A)
9	#define PWM_L 5 //PD5 (OC0B)
10	
11	#define pulsesPerRevolution 2000 //encoder co 2000 rang
12	#define sampleTime 0.01 //thoi gian lay mau
13	
14	// Voltage Reference: AREF pin
15	#define ADC_VREF_TYPE ((0<<REFS1) (0<<REFS0) (0<<ADLAR))
16	
17	void PID_Calculate();
18	void pwmPulse(int pwm);
19	//void Display_Value(unsigned int number);
20	void UART_putchar(char data);
21	void UART_Init();
22	void data_write(char* data, unsigned int size);
23	
24	//char error[3], deSp[5], currSp[5];
25	volatile long pulseCount = 0; //bien dem xung
26	
27	float Kp = 0.2;
28	float Ki = 0.35;
29	float Kd = 0.0001;
30	float input, error, output, setpoint;
31	
32	float maxOutput = 255;
33	float minOutput = 0;

```

34
35 int input_LCD, err_LCD;
36
37 // Read the AD conversion result
38 unsigned int read_adc(unsigned char adc_input)
39 {
40 ADMUX=adc_input | ADC_VREF_TYPE;
41 delay_us(10);
42 // Start the AD conversion
43 ADCSRA|=(1<<ADSC);
44 // Wait for the AD conversion to complete
45 while ((ADCSRA & (1<<ADIF))==0);
46 ADCSRA|=(1<<ADIF);
47 return ADCW;
48 }
49 void main(void)
50 {
51 #pragma optsize-
52 CLKPR=(1<<CLKPCE);
53 CLKPR=0x00;
54 #ifdef _OPTIMIZE_SIZE_
55 #pragma optsize+
56 #endif
57
58 // INT0 Mode: Rising Edge
59 EICRA=(0<<ISC11)|(0<<ISC10)|(1<<ISC01)|(1<<ISC00);
60 EIMSK=(0<<INT1) | (1<<INT0);
61 EIFR=(0<<INTF1) | (1<<INTF0);
62 PCICR=(0<<PCIE2) | (0<<PCIE1) | (0<<PCIE0);
63
64 //cau hinh chan doc Phase A, B la chan nhan;
65 //co dien tro keo len
66 //Phase A:INT0, Phase B:INT1
67 DDRD = (0<<DDD2) | (0<<DDD3);
68 PORTD = (1<<PORTD2) | (1<<PORTD3);
69

```

```

70 //chan xuat PWM, chan PD6, PD5
71 DDRD.6 = 1;
72 DDRD.5 = 1;
73 //chan xac dinh chieu quay
74 DDRD.7 = 1;
75 DDRB.0 = 1;
76 PORTD.7 = 1;
77 PORTB.0 = 1;
78
79 // Timer/Counter 0 initialization
80 // Clock value: 31.250 kHz
81 // Fast PWM, Prescaler: 256
82 TCCR0A=(1<<COM0A1)|(1<<WGM01)|(1<<WGM00);
83 TCCR0B=(0<<WGM02)|(0<<CS02)|(1<<CS01)|(0<<CS00);
84 TCNT0=0xC2;
85 OCR0A=0x00;
86 OCR0B=0x00;
87
88 // Timer/Counter 1 initialization
89 // Normal mode, Prescaler: 8
90 TCCR1A=(0<<WGM11)|(0<<WGM10);
91 TCCR1B=(0<<WGM13)|(0<<WGM12)|(0<<CS12)|(0<<CS11)|(1<<CS10);
92 //gia tri ban dau cua TCNT1 = 45535
93 TCNT1H = 0xC1;
94 TCNT1L = 0x80;
95
96 // Timer/Counter 2 initialization
97 // Normal mode, Prescaler:1024, tao delay 25ms
98 ASSR=(0<<EXCLK) | (0<<AS2);
99 TCCR2A=0x00;
100 TCCR2B=(0<<WGM22)|(1<<CS22)|(0<<CS21)|(0<<CS20);
101 TCNT2=0x06;
102
103 // Timer/Counter 0 Interrupt(s) initialization
104 TIMSK0=(0<<OCIE0B) | (0<<OCIE0A) | (0<<TOIE0);
105 // Timer/Counter 1 Interrupt(s) initialization

```

```

106 TIMSK1=(0<<ICIE1)|(0<<OCIE1B)|(0<<OCIE1A)|(1<<TOIE1);
107 // Timer/Counter 2 Interrupt(s) initialization
108 TIMSK2=(0<<OCIE2B) | (0<<OCIE2A) | (1<<TOIE2);
109
110 // ADC initialization
111 // ADC Voltage Reference: AREF pin
112 ADMUX=ADC_VREF_TYPE;
113 ADCSRA=(1<<ADEN)|(1<<ADATE)
114 |(0<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
115 ADCSRB=(0<<ADTS2)|(0<<ADTS1)|(0<<ADTS0);
116
117 lcd_init(16);
118 DDRB.4 = 1;
119 PORTB.4 = 0;
120
121 DDRD.1 = 1;
122 UART_Init();
123 #asm("sei")
124     while (1){
125         setpoint = (float)read_adc(0);
126     }
127 }
128 void UART_Init() {
129
130     UBRR0H = 0;
131     UBRR0L = 8;
132     UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
133     UCSR0B = (1 << RXEN0) | (1 << TXEN0);
134 }
135 void UART_putchar(char data){
136     while(!(UCSR0A & (1<<UDRE0)));
137     UDR0 = data;
138 }
139 void data_write(char* data, unsigned int size ){
140     int i =0;
141     for (i = 0; i < size; i++)

```

```

142    {
143        UART_putchar(data[i]);
144    }
145}
146void PID_Calculate(){
147    static float lastError = 0;
148    static float pPart = 0;
149    static float iPart = 0;
150    static float dPart = 0;
151    static long lastPulseCount = 0;
152    long currentPulseCount = pulseCount;
153    float pulses=currentPulseCount-lastPulseCount;
154    lastPulseCount = currentPulseCount;
155
156    input = (float)(pulses / pulsesPerRevolution)
157        * (60.0 / sampleTime); // RPM
158    error = setpoint - input;
159    pPart = Kp * error;
160    iPart += Ki * error * sampleTime;
161    dPart = Kd * (error - lastError) / sampleTime;
162    output = pPart + iPart + dPart;
163    lastError = error;
164
165    if (iPart > maxOutput) {
166        iPart = maxOutput;
167    }
168    else if (iPart < minOutput) {
169        iPart = minOutput;
170    }
171    //output saturation
172    if (output > maxOutput) {
173        output = maxOutput;
174    }
175    else if (output < minOutput) {
176        output = minOutput;
177    }

```

```

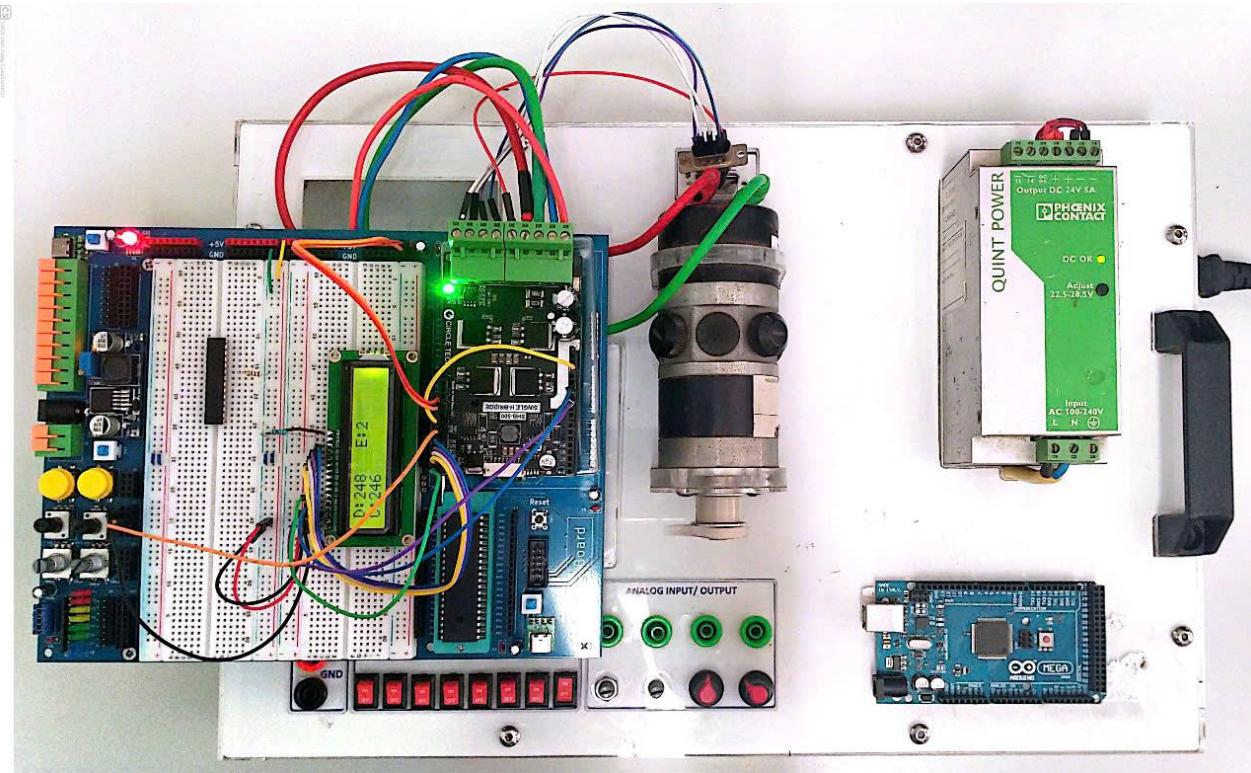
178     pwmPulse((int)output);
179     //pwmPulse(20);
180     input_LCD = (int)(input);
181     err_LCD = (int)(error);
182 }
183 void pwmPulse(int pwm){
184     OCR0A = pwm;
185 }
186 interrupt [EXT_INT0] void ext_int0_isr(void)
187 {
188     pulseCount++;
189 }
190 // Timer1 overflow interrupt service routine
191 interrupt [TIM1_OVF] void timer1_ovf_isr(void)
192 {
193     static long counter = 0;
194     if (counter >= 10)
195     {
196         PID_Calculate();
197         counter = 0;
198     }
199     TCNT1H = 0xC1;
200     TCNT1L = 0x80;
201     counter++;
202 }
203 interrupt [TIM2_OVF] void timer2_ovf_isr(void)
204 {
205     static long counter = 0;
206     if(counter >= 20){
207         float data[4];
208         data[0] = setpoint;
209         data[1] = input;
210         data[2] = error;
211         data[3] = output;
212         UART_putchar(0xBB);
213         data_write((char*)&data, 16);

```

```

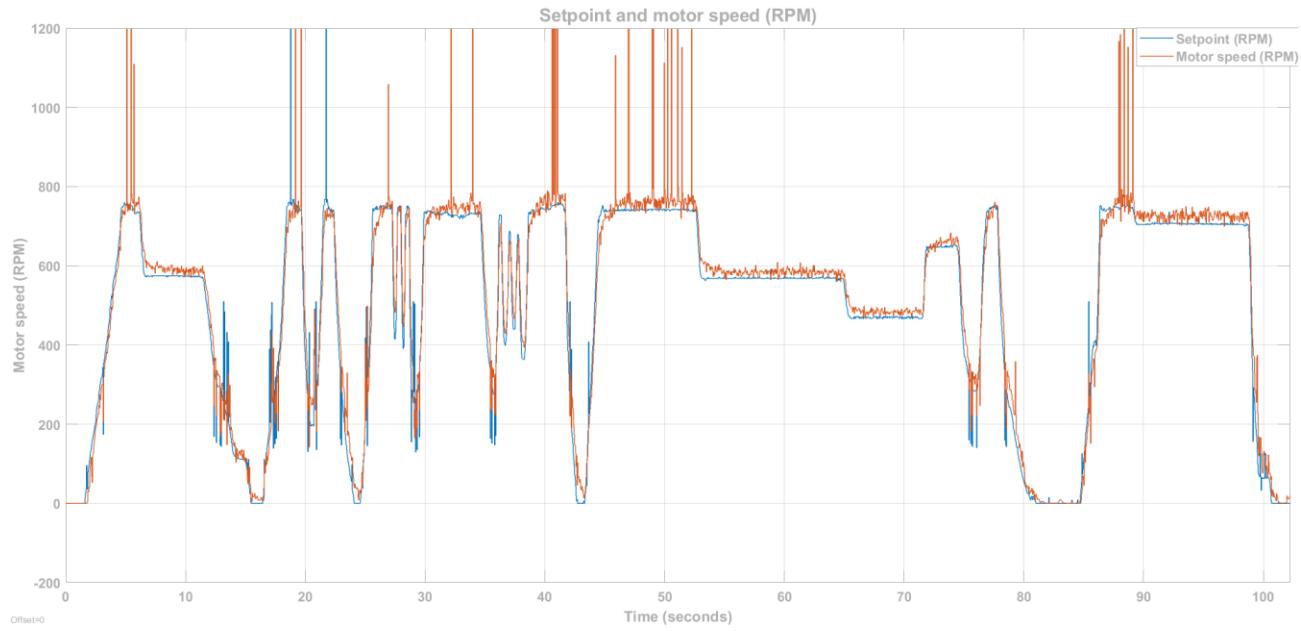
214     UART_putchar(0xCC);
215     counter = 0;
216 }
217 TCNT2 = 0x06;
218 counter++;
219 }
```

❖ Phần mô hình

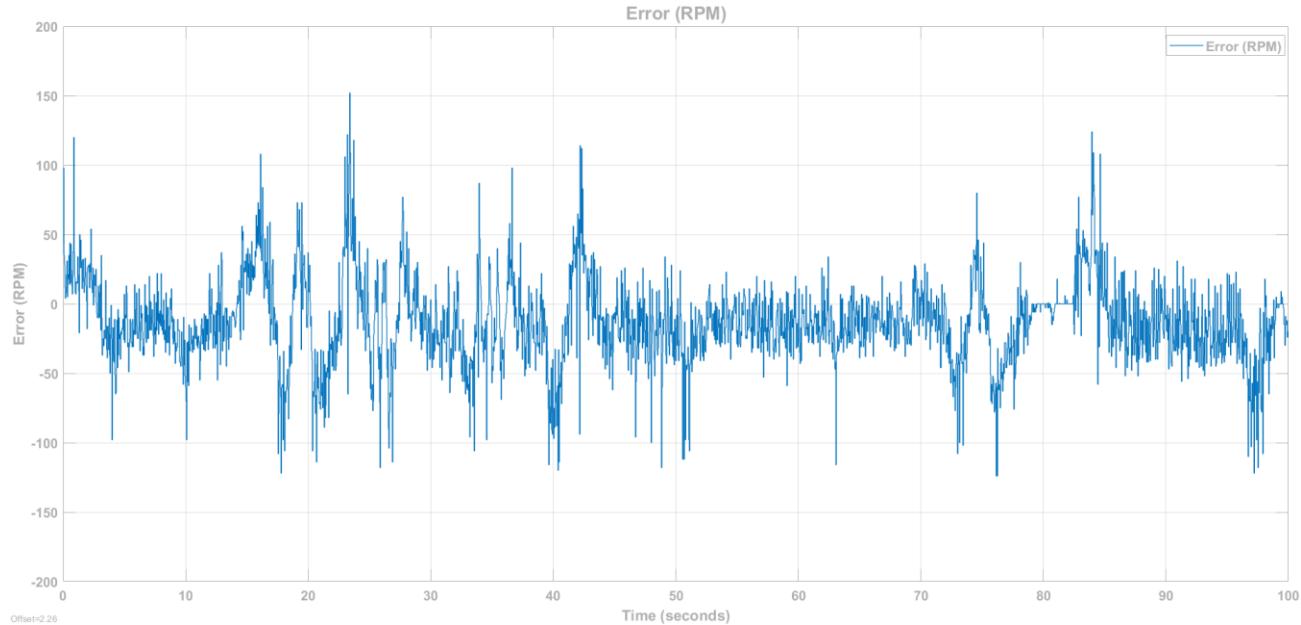


Hình 7.4. Mô hình thực tế bài thực hành điều khiển động cơ DC

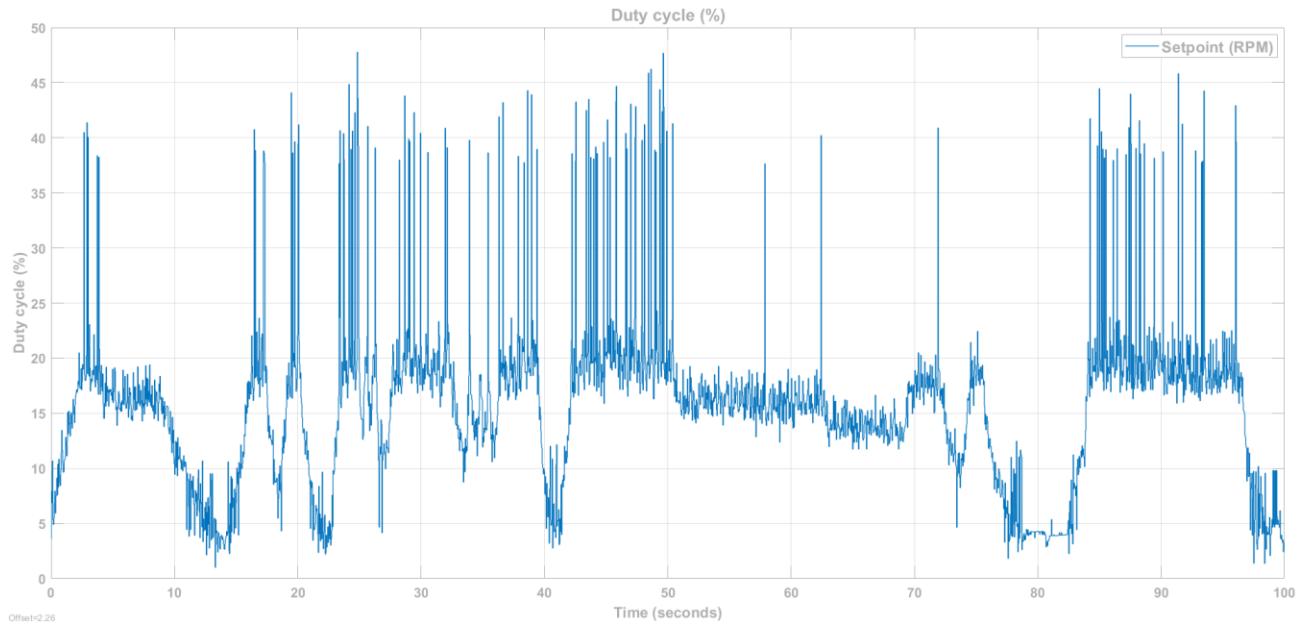
❖ Kết quả đồ thị



Hình 7.5. Đồ thị vận tốc mong muốn và vận tốc thực tế



Hình 7.6. Đồ thị giá trị sai số

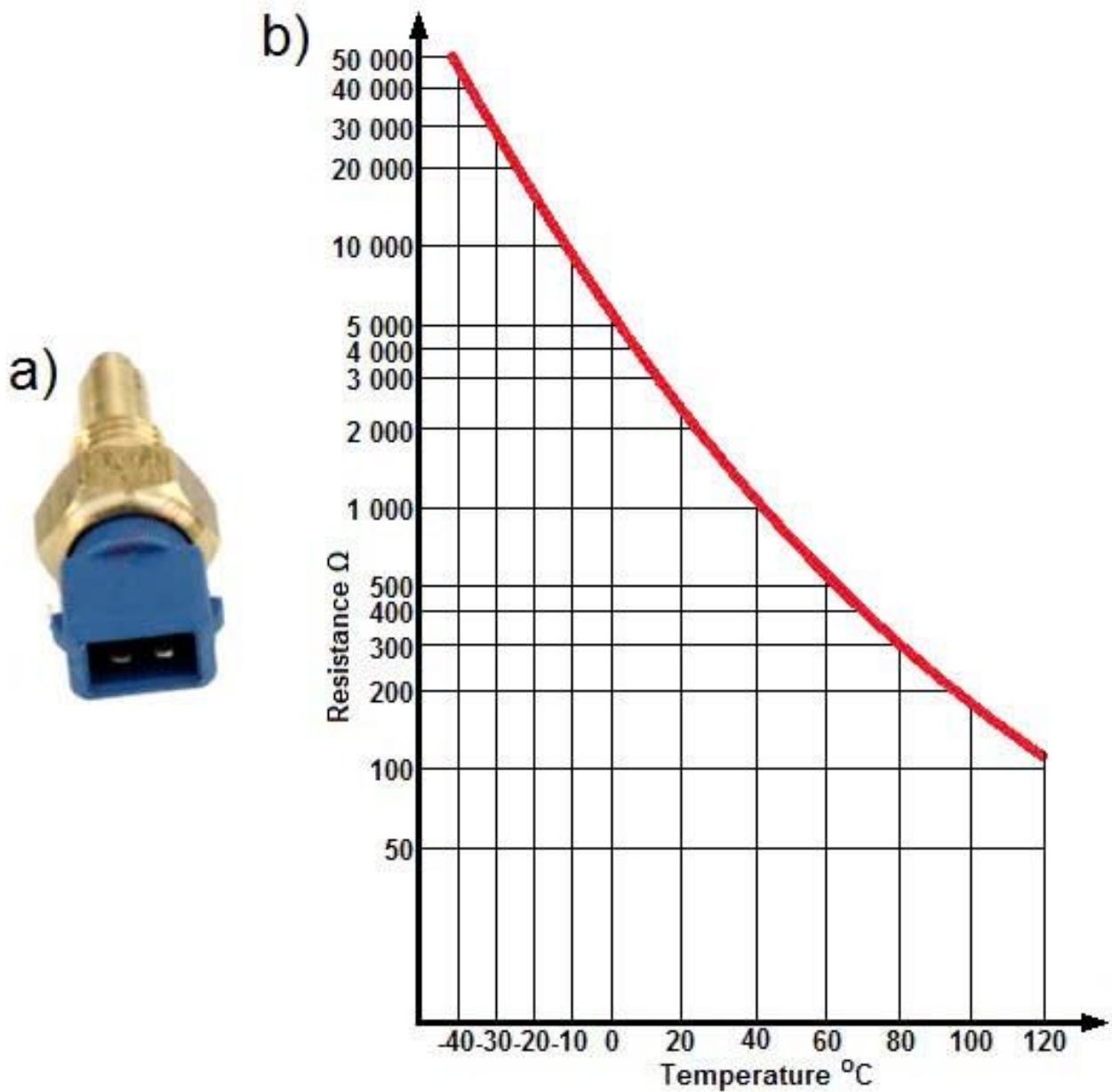


Hình 7.7. Đồ thị điều chế độ rộng xung PWM

7.2.2. Cảm biến nhiệt độ nước làm mát

7.2.2.1. Giới thiệu

Cảm biến nhiệt độ nước làm mát (Coolant Temperature Sensor - CTS) là một trong những cảm biến quan trọng trong hệ thống điều khiển động cơ ô tô. Cảm biến này có nhiệm vụ đo nhiệt độ của dung dịch làm mát trong hệ thống làm mát động cơ và gửi tín hiệu đến bộ điều khiển trung tâm (ECU). Dữ liệu từ cảm biến giúp ECU điều chỉnh các thông số vận hành như lượng nhiên liệu phun, thời điểm đánh lửa và hoạt động của quạt làm mát để đảm bảo động cơ hoạt động ở nhiệt độ tối ưu. Cảm biến nhiệt độ nước làm mát thường sử dụng nguyên lý điện trở nhiệt (theristor), trong đó điện trở thay đổi theo nhiệt độ.

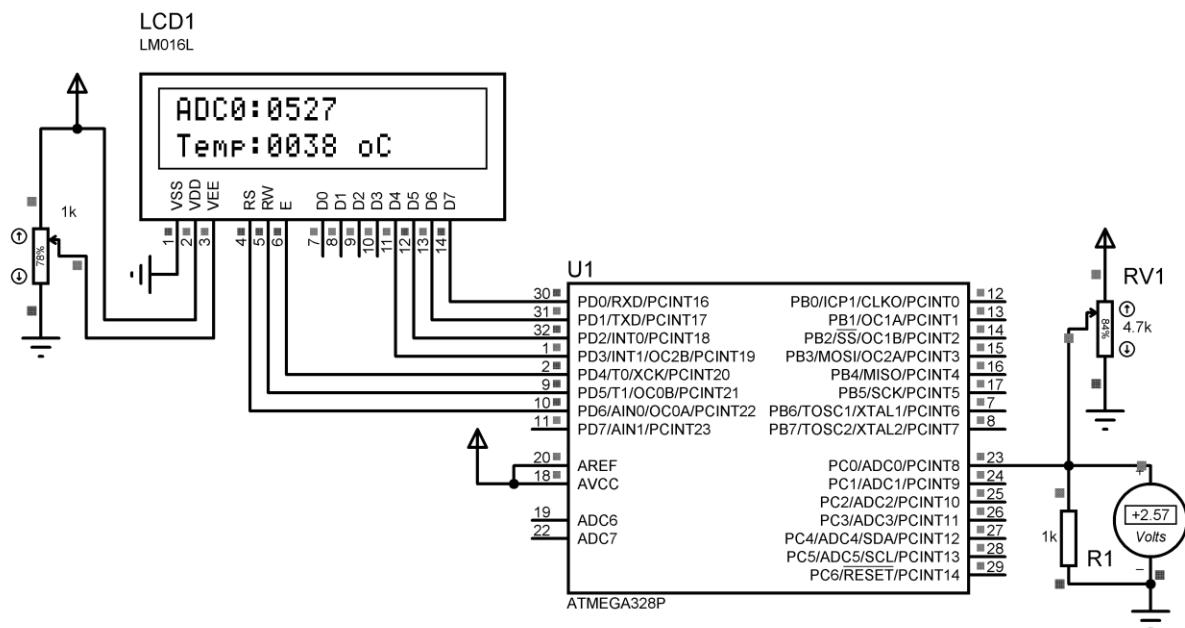


Hình 7.8. Biểu đồ giá trị điện trở của cảm biến ECT theo nhiệt độ [11]

7.2.2.2. Bài thực hành

Yêu cầu bài thực hành: Đọc giá trị cảm biến nhiệt độ nước làm mát và hiển thị lên màn hình LCD. Sử dụng một biến trớ để giả lập cảm biến nhiệt độ nước làm mát.

❖ Phân mô phỏng



Hình 7.9. Sơ đồ kết nối bài thực hành cảm biến nhiệt độ nước mát

❖ Phản lập trình

	SourceCode_ATmega328P\P7_1_TempSensor
1	#include <mega328p.h>
2	#include <delay.h>
3	#include <alcd.h>
4	#include <stdint.h>
5	// Voltage Reference: AVCC pin
6	#define ADC_VREF_TYPE ((0<<REFS1) (1<<REFS0) (0<<ADLAR))
7	void Display_Value(unsigned int number);
8	float Analog_to_Temp(unsigned int val);
9	float map(uint16_t input, uint16_t in_min, uint16_t in_max, uint16_t out_min, uint16_t out_max);
10	unsigned int ADC0;
11	float Temp = 0;
12	//unsigned int adc_value; //float voltage, temperature;

```

14 unsigned int read_adc(unsigned char adc_input){
15     ADMUX=adc_input | ADC_VREF_TYPE;
16     delay_us(10);
17     // Start the AD conversion
18     ADCSRA|=(1<<ADSC);
19     // Wait for the AD conversion to complete
20     while ((ADCSRA & (1<<ADIF))==0);
21     ADCSRA|=(1<<ADIF);
22     return ADCW;
23 }
24 void main(void)
25 {
26     #pragma optsize-
27     CLKPR=(1<<CLKPCE);
28     CLKPR=0x00;
29     #ifdef _OPTIMIZE_SIZE_
30     #pragma optsize+
31     #endif
32     // ADC initialization
33     ADMUX=ADC_VREF_TYPE;
34     ADCSRA=(1<<ADEN)|(1<<ADATE)
35         |(1<<ADPS1)|(1<<ADPS0);
36     ADCSRB=(0<<ADTS2)|(0<<ADTS1)|(0<<ADTS0);
37     lcd_init(16);
38     lcd_gotoxy(3,0);
39     lcd_puts("ECT Sensor");
40     lcd_gotoxy(4,1);
41     lcd_puts("Temp:");
42     while (1)
43     {
44         ADC0 = read_adc(0);
45         Temp = Analog_to_Temp(ADC0);
46
47         lcd_gotoxy(9,1);
48         Display_Value((unsigned int)(Temp));
49         lcd_gotoxy(11,1);

```

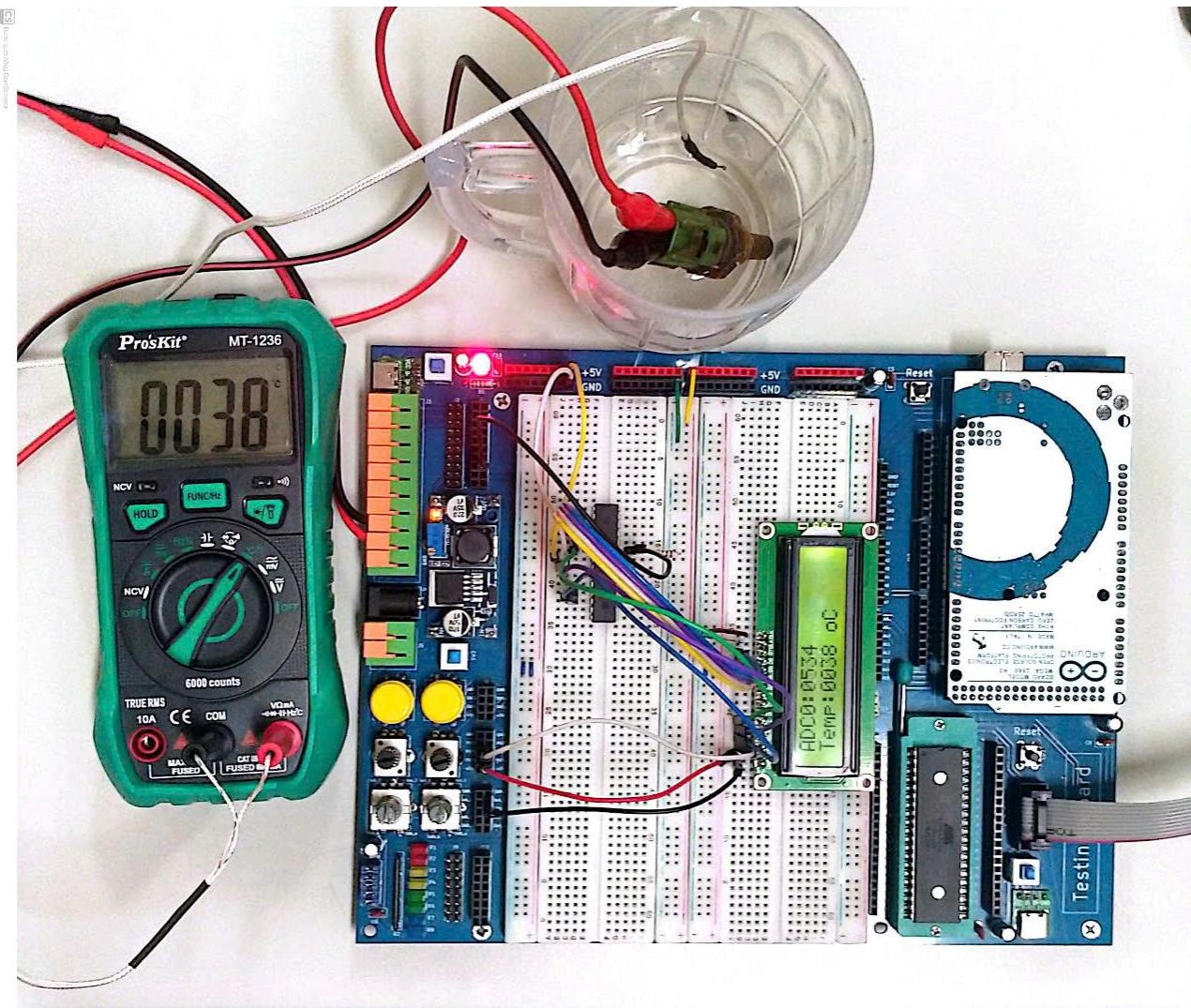
```

50         lcd_puts("oC");
51     }
52 }
53 float map(uint16_t input, uint16_t in_min,
54           uint16_t in_max,uint16_t out_min,uint16_t out_max){
55     return ((input - in_min) * (out_max - out_min))
56           / (in_max - in_min) + out_min;
57 }
58 float Analog_to_Temp(unsigned int val){
59     if(val >= 330 && val <= 415){
60         return map(val, 330, 415, 18, 25);
61     }
62     else if(val > 415 && val <= 450){
63         return map(val, 415, 450, 25, 29);
64     }
65     else if(val > 450 && val <= 490){
66         return map(val, 450, 490, 29, 35);
67     }
68     else if(val > 490 && val <= 550){
69         return map(val, 490, 550, 35, 40);
70     }
71     else if(val > 550 && val <= 620){
72         return map(val, 550, 620, 40, 46);
73     }
74     else if(val > 620 && val <= 665){
75         return map(val, 620, 665, 46, 51);
76     }
77     else if(val > 670 && val <= 760){
78         return map(val, 665, 707, 51, 60);
79     }
80     else if(val > 760 && val <= 805){
81         return map(val, 760, 805, 60, 65);
82     }
83     else if(val > 805 && val <= 840){
84         return map(val, 805, 840, 65, 70);
85     }

```

```
84     else if(val > 840 && val <= 862){
85         return map(val, 840, 855, 70, 74);
86     }
87     else if(val > 862){
88         return 75;
89     }
90     else if(val < 330){
91         return 17;
92     }
93 }
94 void Display_Value(unsigned int number)
95 {
96     unsigned int chuc,donvi;
97     chuc = (number/10)%10;
98     donvi = number%10;
99     lcd_putchar(chuc+48);
100    lcd_putchar(donvi+48);
101 }
```

❖ Phân mô hình



Hình 7.10. Mô hình thực tế bài thực hành cảm biến nhiệt độ nước mát

CHƯƠNG 8. BỘ SO SÁNH TƯƠNG TỰ

8.1. Cấu trúc bộ so sánh tương tự

8.1.1. Khái quát

Bộ so sánh tương tự (Analog Comparator) so sánh các giá trị điện áp đầu vào tại chân dương AIN0 và chân âm AIN1. Khi điện áp trên chân dương AIN0 cao hơn điện áp trên chân âm AIN1, đầu ra của bộ so sánh tương tự, ACO, sẽ được thiết lập. Đầu ra của bộ so sánh này có thể được cấu hình để kích hoạt chức năng bắt đầu vào (Input Capture) của Timer/Counter1.

Ngoài ra, bộ so sánh cũng có thể kích hoạt một ngắt riêng biệt, chỉ dành cho bộ so sánh tương tự. Người dùng có thể chọn kích hoạt ngắt dựa trên sự thay đổi trạng thái của đầu ra bộ so sánh, bao gồm: cạnh lên (rising edge), cạnh xuống (falling edge) hoặc chuyển đổi trạng thái (toggle).

8.1.2. Đầu vào ghép kênh của bộ so sánh tương tự

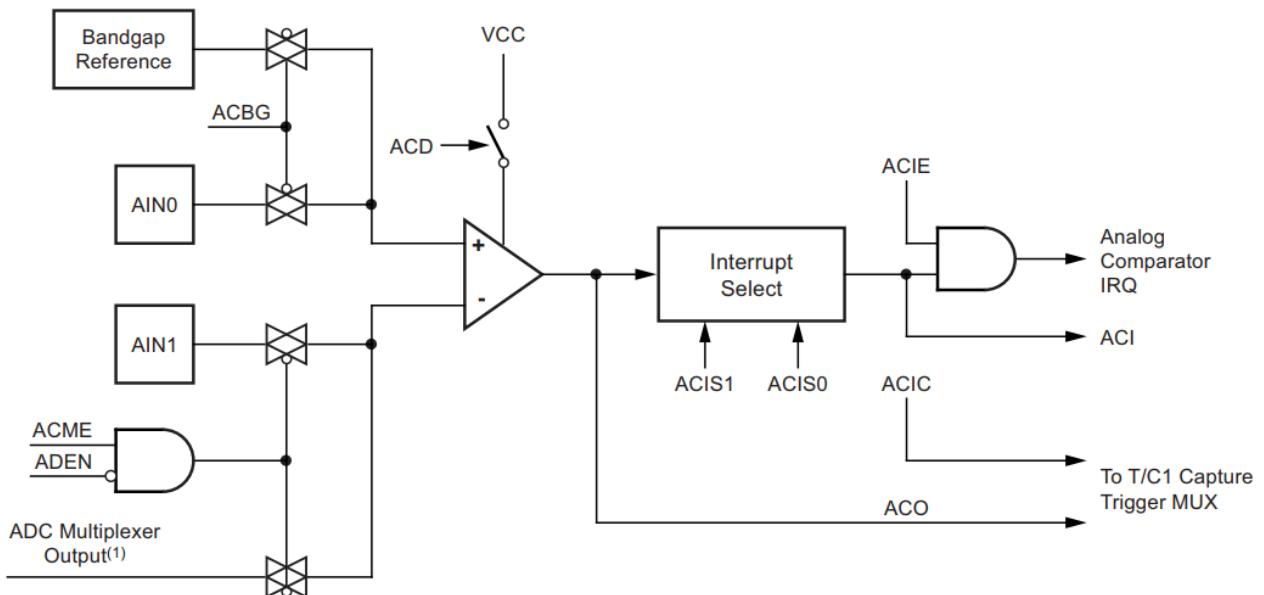
Có thể chọn bất kỳ chân ADC7...0 nào để thay thế đầu vào âm của bộ so sánh tương tự. Bộ ghép kênh ADC (ADC multiplexer) được sử dụng để chọn đầu vào này, và do đó, ADC phải được tắt để sử dụng tính năng này. Nếu bit cho phép ghép kênh của bộ so sánh tương tự (ACME trong thanh ghi ADCSRB) được thiết lập và ADC bị tắt (ADEN trong thanh ghi ADCSRA bằng 0), các bit MUX2...0 trong thanh ghi ADMUX sẽ chọn chân đầu vào để thay thế đầu vào âm của bộ so sánh tương tự. Nếu bit ACME bị xóa hoặc ADC được bật (ADEN được thiết lập), chân AIN1 sẽ được sử dụng làm đầu vào âm của bộ so sánh tương tự.

Bảng 8.1. Đầu vào ghép kênh của bộ so sánh tương tự [2]

ACME	ADEN	MUX2...0	Analog Comparator Negative Input
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0

1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

8.1.3. Sơ đồ khói bộ so sánh tương tự



Hình 8.1. Sơ đồ khói của bộ so sánh tương tự [2]

Bộ so sánh bao gồm các khối:

- Bandgap Reference: Cung cấp một điện áp tham chiếu ổn định, không bị ảnh hưởng bởi nhiệt độ hoặc dao động của nguồn điện (VCC). Điện áp này thường được sử dụng để làm mức tham chiếu cho đầu vào âm (-) hoặc các phép đo chính xác. Thường được sử dụng để so sánh tín hiệu đầu vào với một giá trị cố định
- Hai đầu vào (AIN0 và AIN1 hoặc ADC[7:0])

+ Chân AIN0: Là đầu vào dương của bộ so sánh tương tự.

+ Chân AIN1: Là đầu vào âm mặc định của bộ so sánh.

+ Ngoài ra, đầu vào âm (Negative Input) có thể được thay thế bởi bất kỳ chân ADC nào (Từ ADC0 đến ADC7) thông qua bộ ghép kênh ADC (ADC Multiplexer).

- ADC Multiplexer Output: Đầu ra của bộ chọn tín hiệu (multiplexer) trong ADC (Analog-to-Digital Converter) có thể được định tuyến đến bộ so sánh. Điều này cho phép sử dụng kết quả của ADC như một tín hiệu đầu vào cho bộ so sánh.

- Khối Logic ACME và ADEN

+ ACME (Analog Comparator Multiplexer Enable): Kích hoạt đầu ra của ADC multiplexer làm nguồn tín hiệu đầu vào cho bộ so sánh.

+ ADEN (ADC Enable): Bật hoặc tắt chức năng của ADC, quyết định xem tín hiệu ADC có được cung cấp cho bộ so sánh hay không.

+ Để chọn đầu vào âm thay thế cho AIN1 thì bit ACME trong thanh ghi ADCSRB phải được thiết lập. ADC phải được tắt bằng cách đặt ADEN trong thanh ghi ADCSRA bằng 0. Các bit MUX2:0 trong thanh ghi ADMUX sẽ xác định chân ADC nào được sử dụng làm đầu vào âm.

- Bộ So Sánh Tương Tự (Analog Comparator):

+ Khi điện áp tại đầu vào dương (AIN0) lớn hơn điện áp tại đầu vào âm (AIN1 hoặc một chân ADC được chọn), ACO sẽ được thiết lập (giá trị logic 1).

+ Ngược lại, nếu điện áp tại AIN0 nhỏ hơn hoặc bằng điện áp tại AIN1, ACO sẽ bị xóa (giá trị logic 0).

- Interrupt Select (ACIS1, ACIS0): Bộ so sánh có thể kích hoạt một ngắt độc lập khi giá trị đầu ra ACO thay đổi. Hai bit ACIS1 và ACIS0 quyết định kích hoạt ngắt. Có bốn chế độ kích hoạt ngắt :

- + Cạnh lên (Rising edge) : Khi đầu ra ACO chuyển từ 0 lên 1.
- + Cạnh xuống (Falling edge) : Khi đầu ra ACO chuyển từ 1 xuống 0.
- + Chuyển đổi trạng thái (Toggle) : Khi bất kỳ sự thay đổi nào xảy ra tại ACO.
- + Không kích hoạt ngắn.
- Output Logic: Điều khiển đầu ra và các tín hiệu liên quan
 - + ACIE (Analog Comparator Interrupt Enable): Kích hoạt ngắn cho bộ so sánh. Khi bật, ngắn sẽ được tạo khi điều kiện của khối Interrupt Select được thỏa mãn.
 - + ACI (Analog Comparator Interrupt Flag): Là cờ báo hiệu ngắn của bộ so sánh. Cờ này được phần mềm kiểm tra để xử lý ngắn.
 - + ACIC (Analog Comparator Input Capture Enable): Kết nối đầu ra của bộ so sánh (ACO) đến Timer/Counter1 Input Capture (T/C1), cho phép bộ so sánh kích hoạt các sự kiện trong bộ đếm thời gian.
 - + ACO (Analog Comparator Output): Là tín hiệu đầu ra của bộ so sánh, cho biết kết quả so sánh điện áp ở đầu vào dương và âm.

❖ Cấu hình và điều khiển

Các thanh ghi chính điều khiển hoạt động của bộ so sánh:

- ACSR (Analog Comparator Control and Status Register): Điều khiển và giám sát trạng thái của bộ so sánh.
- Bit quan trọng:
 - + ACO: Giá trị đầu ra của bộ so sánh.
 - + ACIE: Bật hoặc tắt ngắn bộ so sánh.
 - + ACIS1, ACIS0: Chọn chế độ kích hoạt ngắn (cạnh lên, xuống, hoặc chuyển đổi).

- ADCSRB (ADC Control and Status Register B): Bit ACME: Kích hoạt tính năng ghép kênh.

- ADMUX (ADC Multiplexer Selection Register): Bit MUX2...0: Chọn chân ADC thay thế đầu vào âm.

8.2. Các thanh ghi của bộ so sánh tương tự

8.2.1. ADCSRB – Thanh ghi điều khiển và trạng thái của ADC B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0	ADCSRS
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 6 – ACME: Cho phép ghép kênh bộ so sánh tương tự (Analog Comparator Multiplexer Enable). Khi bit này được ghi giá trị logic 1 và ADC bị tắt (bit ADEN trong thanh ghi ADCSRA bằng 0), bộ ghép kênh ADC (ADC Multiplexer) sẽ chọn đầu vào âm cho bộ so sánh tương tự. Khi bit này được ghi giá trị logic 0, chân AIN1 sẽ được sử dụng làm đầu vào âm của bộ so sánh tương tự.

8.2.2. ACSR – Thanh ghi điều khiển và trạng thái của bộ so sánh tương tự

Bit	7	6	5	4	3	2	1	0	
0x30 (0x50)	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACISO	ACSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

Bit 7 – ACD: Vô hiệu hóa bộ so sánh tương tự (Analog Comparator Disable). Khi bit này được ghi giá trị logic 1, nguồn cung cấp cho bộ so sánh tương tự sẽ bị tắt. Bit này có thể được thiết lập bất kỳ lúc nào để tắt bộ so sánh tương tự, giúp giảm tiêu thụ năng lượng trong chế độ Active và Idle. Khi thay đổi bit ACD, ngắt của bộ so sánh tương tự (Analog Comparator Interrupt) phải được tắt bằng cách xóa bit ACIE trong thanh ghi ACSR. Nếu không, ngắt có thể xảy ra khi thay đổi bit này.

Bit 6 – ACBG: Chọn điện áp tham chiếu Bandgap cho bộ so sánh tương tự. Khi bit này được đặt, một điện áp tham chiếu Bandgap cố định sẽ thay thế đầu vào dương của bộ so sánh tương tự (Analog Comparator). Khi bit này được xóa, chân AIN0 sẽ được áp dụng làm đầu vào dương cho bộ so sánh tương tự. Khi sử dụng điện áp tham chiếu Bandgap làm đầu vào cho bộ so sánh tương tự, cần có một khoảng thời gian để điện áp ổn định. Nếu điện áp chưa ổn định, kết quả chuyển đổi đầu tiên có thể không chính xác.

Bit 5 – ACO: Đầu ra của bộ so sánh tương tự. Đầu ra của bộ so sánh tương tự được đồng bộ hóa và sau đó được kết nối trực tiếp tới ACO. Quá trình đồng bộ hóa này gây ra một độ trễ từ 1 đến 2 chu kỳ xung nhịp.

Bit 4 – ACI: Cờ ngắt bộ so sánh tương tự. Bit này được phần cứng tự động đặt, khi một sự kiện đầu ra của bộ so sánh kích hoạt chế độ ngắt được xác định bởi các bit ACIS1 và ACIS0. Chương trình xử lý ngắt của bộ so sánh tương tự sẽ được thực thi nếu bit ACIE được đặt và bit I trong thanh ghi trạng thái SREG cũng được đặt. Bit ACI sẽ được phần cứng tự động xóa, khi thực hiện vector xử lý ngắt tương ứng. Ngoài ra, ACI cũng có thể được xóa bằng cách ghi giá trị logic 1 vào cờ này.

Bit 3 – ACIE: Cho phép ngắt bộ so sánh tương tự. Khi bit ACIE được ghi giá trị logic 1 và bit I trong thanh ghi Trạng thái (Status Register) được đặt, ngắt của bộ so sánh tương tự sẽ được kích hoạt. Khi ghi giá trị logic 0, ngắt sẽ bị vô hiệu hóa.

Bit 2 – ACIC: Cho phép chức năng bắt tín hiệu đầu vào của bộ so sánh tương tự. Khi ghi giá trị logic 1, bit này cho phép chức năng bắt tín hiệu đầu vào (input capture) trong bộ Timer/Counter1 được kích hoạt bởi bộ so sánh tương tự. Trong trường hợp này, đầu ra của bộ so sánh sẽ được kết nối trực tiếp tới logic đầu vào của chức năng bắt tín hiệu, cho phép bộ so sánh sử dụng bộ loại nhiễu (noise canceler) và các tính năng chọn cạnh (edge select) của ngắt bắt tín hiệu đầu vào Timer/Counter1. Khi ghi giá trị logic 0, không có kết nối nào giữa bộ so sánh tương tự và chức năng bắt tín hiệu đầu vào. Để bộ so sánh kích hoạt ngắt bắt tín hiệu đầu vào của Timer/Counter1, bit ICIE1 trong thanh ghi Timer Interrupt Mask Register – TIMSK1 cần được đặt.

Bit 1:0 – ACIS[1:0]: Chọn chế độ ngắt bộ so sánh tương tự. Các bit này quyết định sự kiện nào của bộ so sánh sẽ kích hoạt ngắt của bộ so sánh tương tự. Các cấu hình khác nhau được hiển thị trong bảng dưới đây :

Bảng 8.2. Thiết lập hai bit ACIS1 và ACIS0 [2]

ACIS1	ACIS0	Chế độ ngắt
0	0	Ngắt bộ so sánh khi đầu ra thay đổi trạng thái
0	1	Dành riêng (Reserved)
1	0	Ngắt bộ so sánh khi có cạnh xuống của đầu ra
1	1	Ngắt bộ so sánh khi có cạnh lên của đầu ra

Khi thay đổi các bit ACIS1/ACIS0, ngắt của bộ so sánh tương tự phải được vô hiệu hóa bằng cách xóa bit cho phép ngắt (Interrupt Enable) trong thanh ghi ACSR. Nếu không, một ngắt có thể xảy ra khi các bit này được thay đổi.

8.2.3. DIDR1 – Thanh ghi vô hiệu hóa đầu vào số

Bit	7	6	5	4	3	2	1	0	
(0x7F)	-	-	-	-	-	-	AIN1D	AIN0D	DIDR1
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7:2 – Dự trữ (Reserved): Các bit này không được sử dụng trong vi điều khiển ATMEGA328P và luôn đọc giá trị bằng 0.

Bit 1:0 – AIN1D, AIN0D: Vô hiệu hóa đầu vào số của AIN1 và AIN0 (AIN1, AIN0 Digital Input Disable):

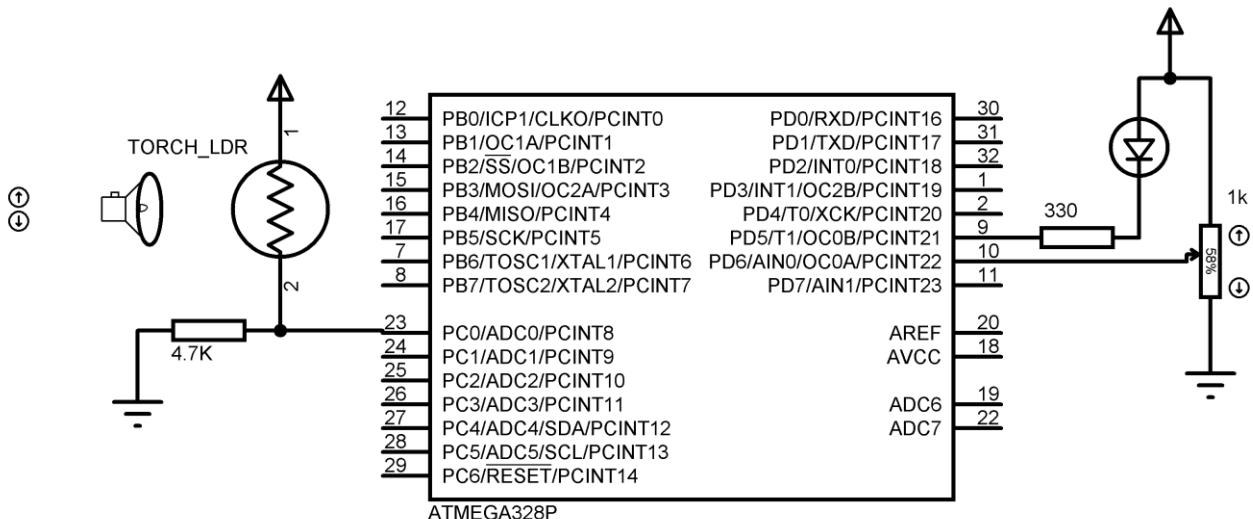
- + Khi bit này được ghi giá trị logic 1, bộ đệm đầu vào số (digital input buffer) trên chân AIN1 hoặc AIN0 sẽ bị vô hiệu hóa.
- + Khi bit này được đặt, giá trị đọc từ bit tương ứng trong thanh ghi PIN sẽ luôn là 0.

+ Nếu một tín hiệu analog được đưa vào chân AIN1 hoặc AIN0 mà đầu vào kỹ thuật số từ các chân này không cần thiết, nên thiết lập bit này thành logic 1 để giảm tiêu hao năng lượng của bộ đệm đầu vào kỹ thuật số.

8.3. Bài thực hành

Yêu cầu bài thực hành: So sánh điện áp giữa chân PA0 (kết nối với quang điện trở) với chân PB2 (kết nối với biến trở). Khi không có ánh sáng chiếu vào quang điện trở, lập tức đèn LED ở chân PD0 sẽ sáng, biến trở dùng để điều chỉnh độ nhạy khi có ánh sáng thay đổi.

❖ Phản mô phỏng



Hình 8.2. Sơ đồ kết nối bài thực hành bộ so sánh tương tự

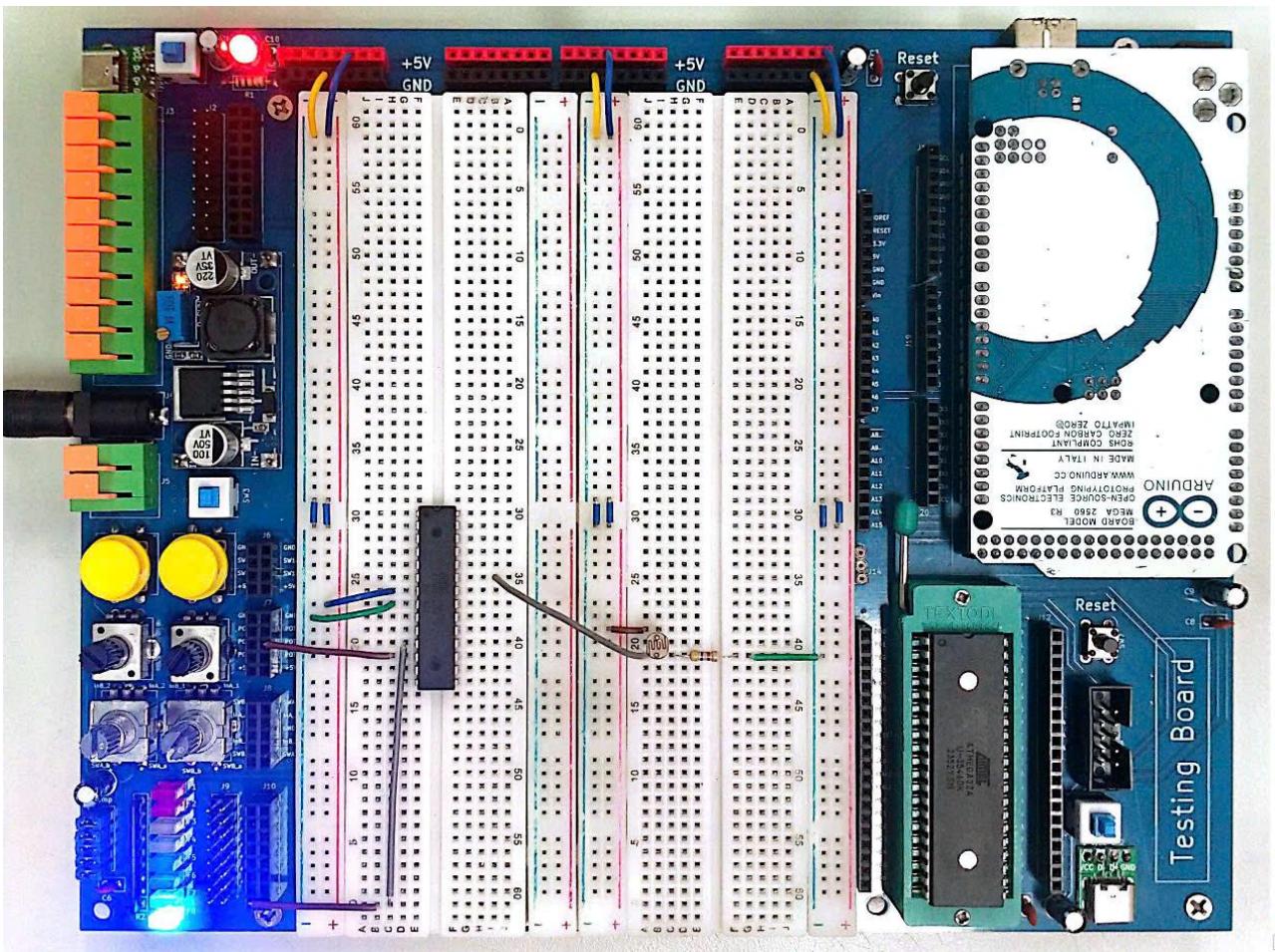
❖ Phản lập trình

	SourceCode_ATMEGA328P\P8_Analog_Comparator
1	#include <mega328p.h>
2	void main(void)
3	{
4	#pragma optsize-
5	CLKPR = (1<<CLKPCE);
6	CLKPR = 0x00;
7	#endif _OPTIMIZE_SIZE_
8	#pragma optsize+

```

9    #endif
10
11 //su dung chan PD5 dieu khien den LED
12 DDRD |= (1<<DDD5);           //DDRD.5 = 1;
13 PORTD &= ~ (1<<PORTD5);     //PORTD.5 = 0;
14
15 //che do Analog Comparator Multiplexed Input
16 ADCSRB=(1<<ACME);
17 while (1)
18 {
19     if(ACSR & (1<<AC0))    //neu V_AIN0 > V_AINT1
20         PORTD &= ~ (1<<PORTD5);
21     else
22         PORTD |= (1<<PORTD5);
23 }
24 }
```

❖ Phân mô hình



Hình 8.3. Mô hình thực tế bài thực hành bộ so sánh tương tự

CHƯƠNG 9. CẤU TRÚC BỘ TIMER/COUNTER/PWM

9.1. Cấu trúc Timer/Counter

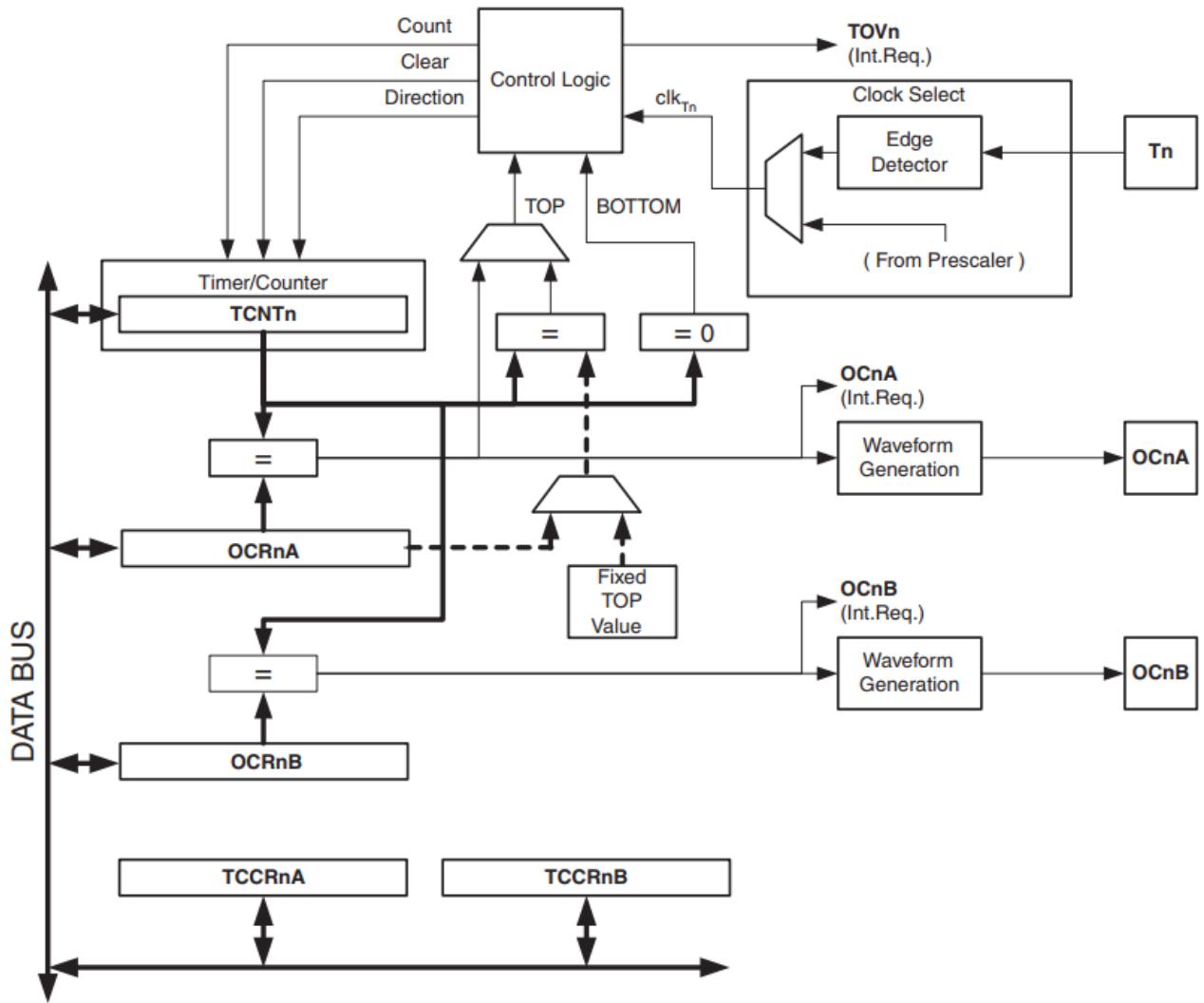
9.1.1. Tổng quan

Timer/Counter là các module hoạt động độc lập với CPU, được thiết kế để thực hiện các nhiệm vụ liên quan đến định thời (tạo ra khoảng thời gian hoặc đo thời gian) và đếm sự kiện. Ngoài ra các Timer/Counter còn có thể tạo xung điều rộng (PWM – Pulse Width Modulation). Ở một số dòng chip AVR, Timer/Counter còn được sử dụng làm bộ canh chỉnh thời gian (Calibration) cho các ứng dụng thời gian thực. Các Timer/Counter được phân loại dựa trên độ rộng của thanh ghi lưu giá trị định thời hoặc giá trị đếm. Trên vi điều khiển ATMEGA328P, có 2 Timer 8 bit (Timer/Counter0 và Timer/Counter2) và 1 Timer 16 bit (Timer/Counter1). Mỗi Timer/Counter có cách thức hoạt động và điều khiển riêng biệt.

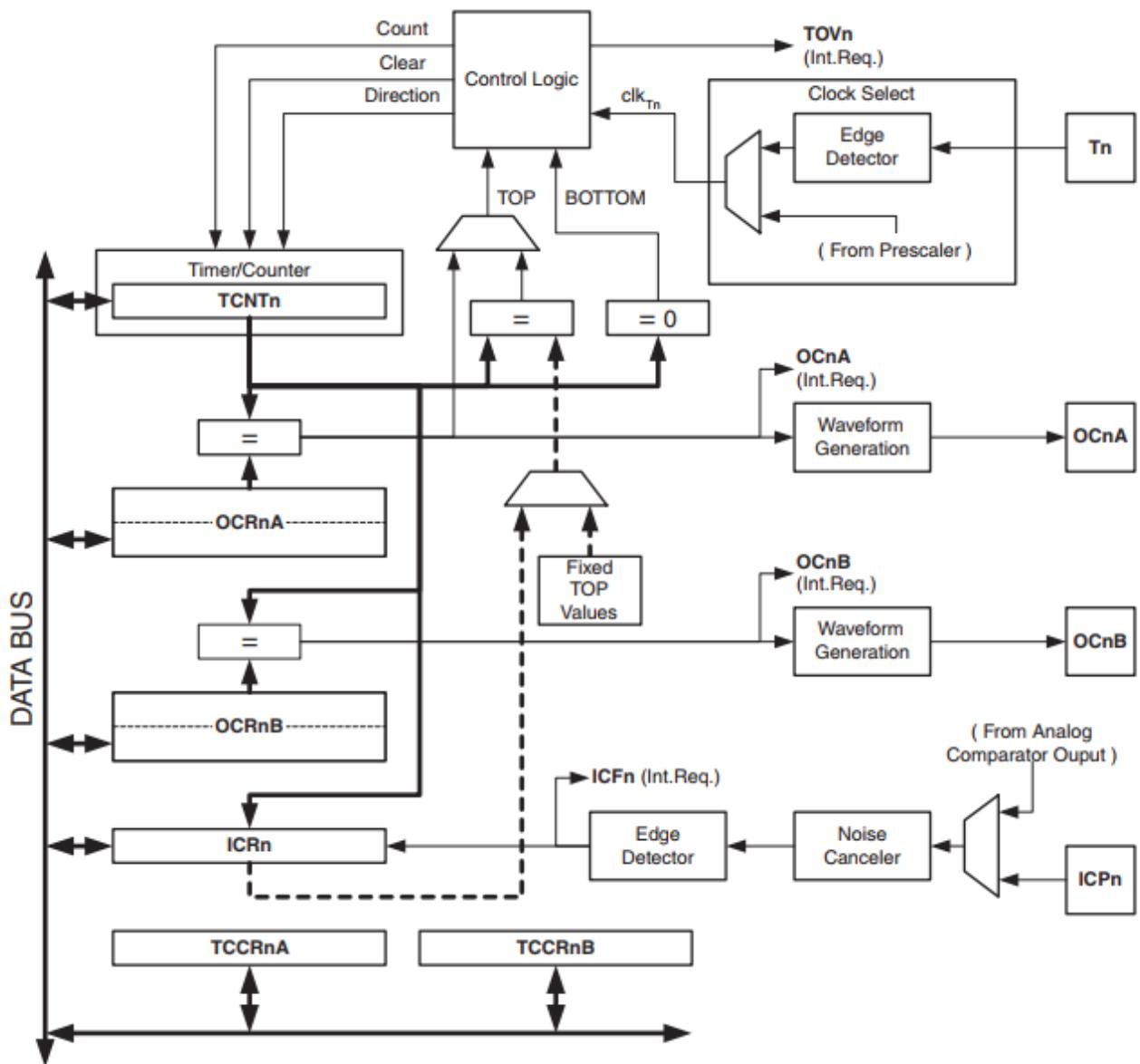
Timer/Counter0: Đây là một bộ định thời và đếm đơn giản với thanh ghi 8 bit. Với hai bộ so sánh đầu ra độc lập và cung cấp chế độ điều chế xung PWM. Ngoài ra, module này còn hỗ trợ ba nguồn ngắt độc lập: TOV0, OCF0A, OCF0B.

Timer/Counter1: Với thanh ghi 16 bit, Timer/Counter1 là một bộ định thời và đếm đa năng. Bên cạnh các chức năng cơ bản, nó có thể tạo xung PWM để hỗ trợ các ứng dụng điều khiển. Bộ này có khả năng tạo ra 2 tín hiệu PWM độc lập trên các chân OC1A (chân 15) và OC1B (chân 16).

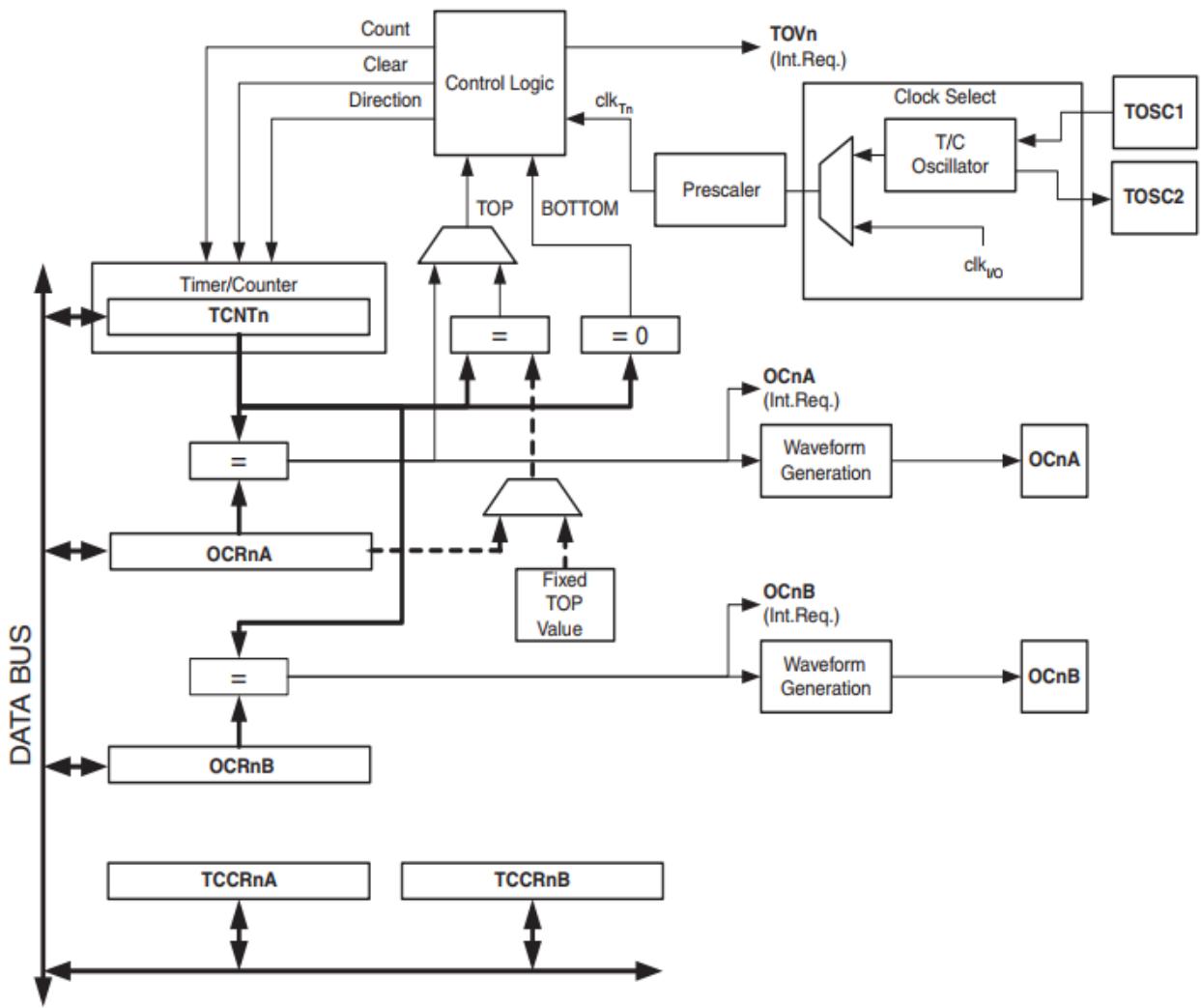
Timer/Counter2: Mặc dù cũng là một module 8 bit, có cấu trúc tương tự như Timer/Counter0 nhưng Timer/Counter2 hỗ trợ đến 4 chế độ bất đồng bộ (asynchronous mode), cho phép sử dụng như một bộ canh chỉnh thời gian cho các ứng dụng thời gian thực.



Hình 9.1. Sơ đồ khái niệm Timer/Counter0 – 8 bit [2]



Hình 9.2. Sơ đồ khói Timer/Counter1 – 16 bit [2]



Hình 9.3. Sơ đồ khối Timer/Counter2 – 8 bit [2]

9.1.2. Định nghĩa khi sử dụng Timer/Counter

Bảng 9.1. Định nghĩa khi sử dụng Timer/Counter

BOTTOM	Bộ đếm (Counter) đạt giá trị BOTTOM khi nó bằng 0x00.
MAX	Là giá trị tối đa mà một Timer/Counter có thể đạt được, giá trị này được quy định bởi giá trị lớn nhất mà thanh ghi đếm của Timer/Counter có thể chứa được. Đối với Timer/Counter 8 bit có giá trị MAX luôn là 0xFF (tức là 255 trong hệ thập phân), còn với bộ Timer/Counter 16 bit thì MAX bằng 0xFFFF (65535). Như vậy MAX là giá trị không thay đổi trong mỗi Timer/Counter.

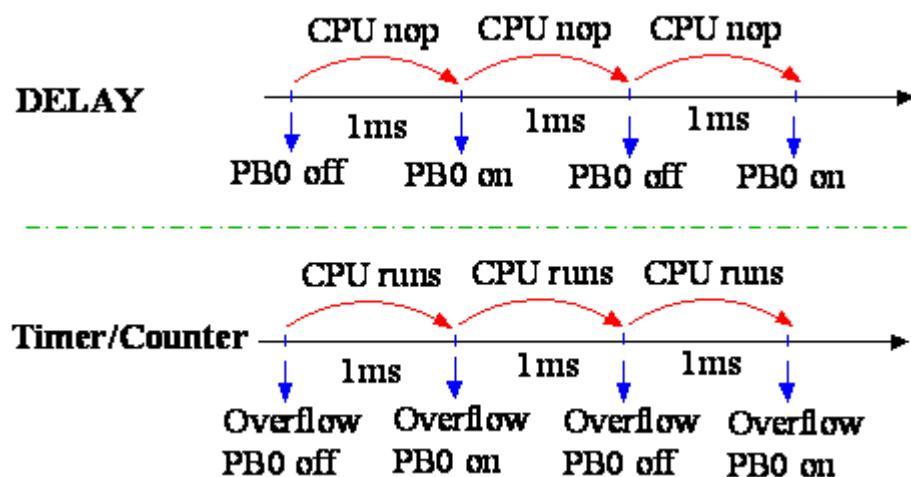
TOP	Bộ đếm đạt giá trị TOP khi nó bằng với giá trị cao nhất trong chuỗi đếm. Giá trị TOP có thể được gán là giá trị cố định 0xFF (MAX) hoặc giá trị được lưu trong thanh ghi OCRnx. Việc gán giá trị TOP phụ thuộc vào chế độ hoạt động của bộ đếm.
-----	---

9.2. Các chế độ làm việc Timer/Counter

9.2.1. Timer/Counter0

❖ Chế độ thường – Normal Mode

Đây là chế độ hoạt động đơn giản, có thể sử dụng Timer/Counter0 để tạo một bộ đếm thời với mục đích thực hiện một sự kiện nào đó sau một khoảng thời gian cố định, chẳng hạn như thay đổi trạng thái của chân PB0 sau mỗi 100ms (tạo hiệu ứng nhấp nháy). Thay vì sử dụng lệnh delay sẽ làm CPU phải chờ và không thể xử lý các tác vụ khác, phương pháp này giúp CPU rảnh rỗi để thực hiện công việc khác trong khi Timer hoạt động độc lập.



Hình 9.4. So sánh giữa hai chế độ làm việc [12]

Timer/Counter0 chỉ có thể đếm từ 0 đến 255 giá trị rồi xảy ra một ngắt tràn (giá trị sẽ quay về 0 khi tràn). Tuy nhiên, có thể cài đặt giá trị mong muốn ban đầu cho bộ Timer/Counter thông qua thanh ghi TCNT0, khi đó Timer/Counter 0 sẽ đếm lên từ giá trị đã được thiết lập và kết thúc ở giá trị 255. Cần chú ý đến việc gán lại giá trị cho thanh ghi TCNT0 sau mỗi lần ngắt

tràn, cách tốt nhất là đặt lại trong chương trình phục vụ ngắt tràn. Việc tiếp theo là chọn giá trị chia (prescaler) của bộ Timer/Counter0 thông qua 3 bit CS02, CS01, CS00 trong thanh ghi TCCR0B, từ đó xác định giá trị ban đầu của thanh ghi TCNT0.

Prescaler có vai trò quan trọng trong việc giảm tần số xung clock ($\text{clk}_{\text{I/O}}$) của bộ đếm Timer/Counter, cho phép điều chỉnh tốc độ đếm bằng cách chia tần số xung nhịp gốc của vi điều khiển. Nếu không có prescaler, Timer/Counter sẽ đếm rất nhanh, với tần số 8MHz tương đương với 0,125 μs thì Timer/Counter đã hoàn thành xong chu kỳ đếm và dẫn đến timer bị tràn quá nhanh, không phù hợp cho các ứng dụng yêu cầu đếm trong thời gian dài. Bảng 9.2 mô tả cách chọn bộ chia trong thanh ghi TCCR0B.

Bảng 9.2. Mô tả giá trị bộ chia cho Timer/Counter0 [2]

CS02	CS01	CS00	Mô tả
0	0	1	$\text{clk}_{\text{I/O}}/1$ (no prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (from prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (from prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (from prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (from prescaler)

Giả sử để tạo một chương trình delay 100 ms, nguồn xung clock của chip $\text{clk}_{\text{I/O}} = 8\text{MHz}$ và giá trị bộ chia được chọn là 1024, vậy tần số cho bộ Timer sẽ được tính bằng công thức:

$$f_{\text{clk}_{\text{timer}}} = \frac{\text{clk}_{\text{I}}}{\text{division factor}} \quad [\text{MHz}] \quad (8)$$

$$= \frac{8}{1024} \quad [\text{MHz}]$$

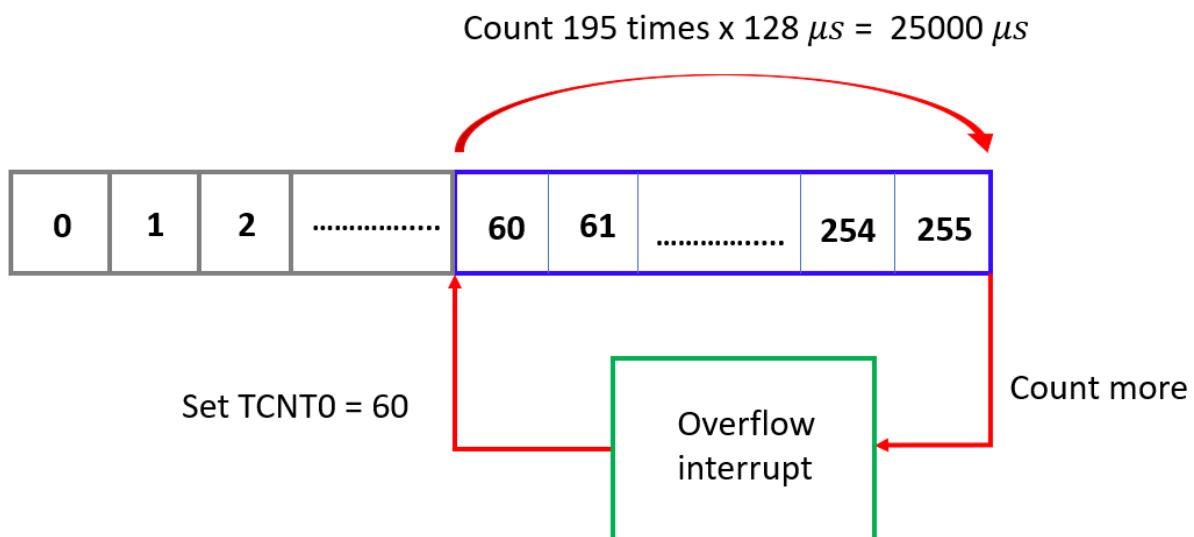
Suy ra chu kỳ của bộ Timer:

$$T_{\text{clk}_{\text{timer}}} = \frac{1}{f_{\text{clk}_{\text{timer}}}} \quad [\mu\text{s}] \quad (9)$$

$$= \frac{1}{\left(\frac{8}{1024}\right)} = \frac{1024}{8} = 128 \quad [\mu\text{s}]$$

Do bộ Timer/Counter0 có thể đếm tối đa đến 255 nên thời gian giới hạn lớn nhất trong trường hợp này là $255 \times T_{\text{clk_timer}} = 32\ 640\ \mu\text{s}$, xấp xỉ 32 ms. Tuy nhiên, chúng ta muốn delay 100ms nên cách tốt nhất là delay 25 ms (25000 μs) trong 4 lần. Vậy giá trị ban đầu của thanh ghi TCNT0 được tính:

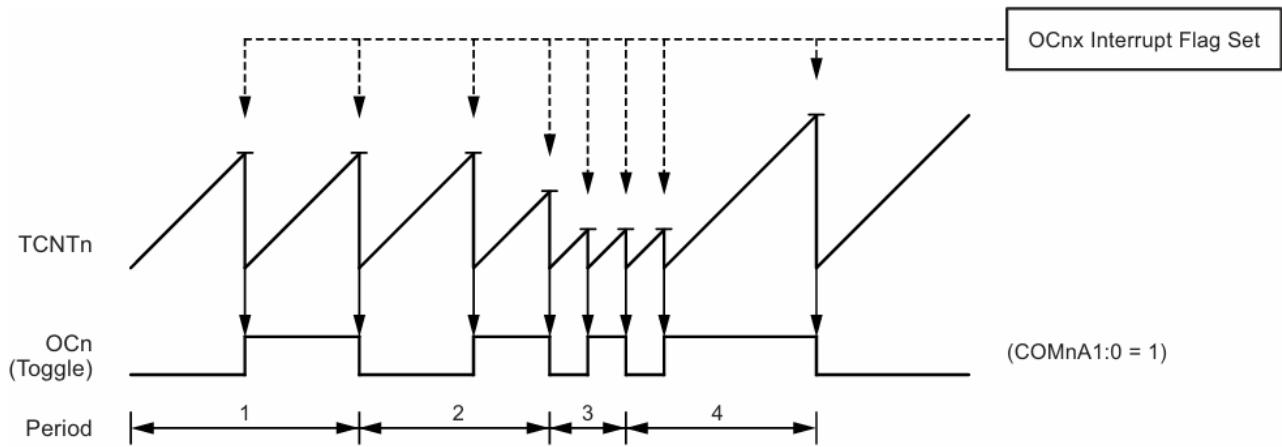
$$\begin{aligned} \text{TCNT0} &= 255 - \frac{25\ 000\ [\mu\text{s}]}{T_{\text{clk_timer}}\ [\mu\text{s}]} \\ &= 255 - \frac{25\ 000}{128} = 60 \end{aligned} \quad (10)$$



Hình 9.5. Quá trình thực hiện hàm delay

❖ Chế độ Clear Timer on Compare Match – CTC

Ở chế độ CTC bộ đếm sẽ được xóa về 0 khi giá trị bộ đếm (TCNT0) bằng giá trị OCR0A, nói cách khác OCR0A xác định giá trị cao nhất của bộ đếm. Đối với Timer/Counter0, chỉ có thể dùng một mode CTC khi thiết lập các bit WGM0[2:0]= 2.



Hình 9.6. Biểu đồ thời gian của chế độ CTC – Timer/Counter0 [2]

Một tín hiệu ngắn có thể xảy ra khi giá trị bộ đếm bằng giá trị TOP thông qua việc sử dụng cờ ngắt OCF0A, trong quá trình này có thể sử dụng để cập nhật giá trị TOP mới. Cần chú ý nếu giá trị OCR0A mới được cập nhật lại thấp hơn giá trị bộ đếm TCNT0 hiện tại thì công việc so sánh ở chu kỳ này không được diễn ra. Bộ đếm sẽ tăng đến giá trị TOP rồi quay về 0 sau đó mới thực hiện việc so sánh với OCR0A mới được cập nhật.

Chân OC0A/OC0B (PD6/PD5) có thể xuất ra một dạng sóng trong chế độ CTC bằng cách thiết lập cơ chế thay đổi mức logic mỗi khi có sự so sánh xảy ra, theo đó phải cấu hình các bit COM0A[1:0] =1. Tần số của sóng được xác định theo công thức:

$$f_{OCnx} = \frac{f_{clkI/O}}{2 \times N \times (1 + OCRnx)} \quad (11)$$

Với $f_{clkI/O}$: Tần số xung nhịp của vi điều khiển.

N: giá trị bộ chia tần số.

OCRnx: giá trị so sánh.

9.2.2. Timer/Counter1

❖ Chế độ thường – Normal Mode

Là một chế độ hoạt động đơn giản của Timer/Counter1 cơ chế hoạt động tương tự như Timer/Counter0, thanh ghi TCNT1 sẽ tăng từ giá trị khởi tạo ban đầu đến giá trị tối đa. Tuy nhiên, giá trị lớn nhất mà thanh ghi này có thể đạt được là 65535 (do bộ T/C1 có đến 16 bit). Để thiết lập cho Timer/Counter1 sử dụng chế độ Normal cần cấu hình 2 bit WGM1[1:0] trong thanh ghi TCCR1A và 2 bit WGM1[3:2] trong thanh ghi TCCR1B về mức logic 0. Cần chú ý thêm đến ba bit CS12, CS11 và CS10 nằm trong thanh ghi TCCR1B, đây là ba bit quan trọng để chọn nguồn xung clock cho T/C1.

Bảng 9.3. Mô tả giá trị bộ chia cho Timer/Counter1 [2]

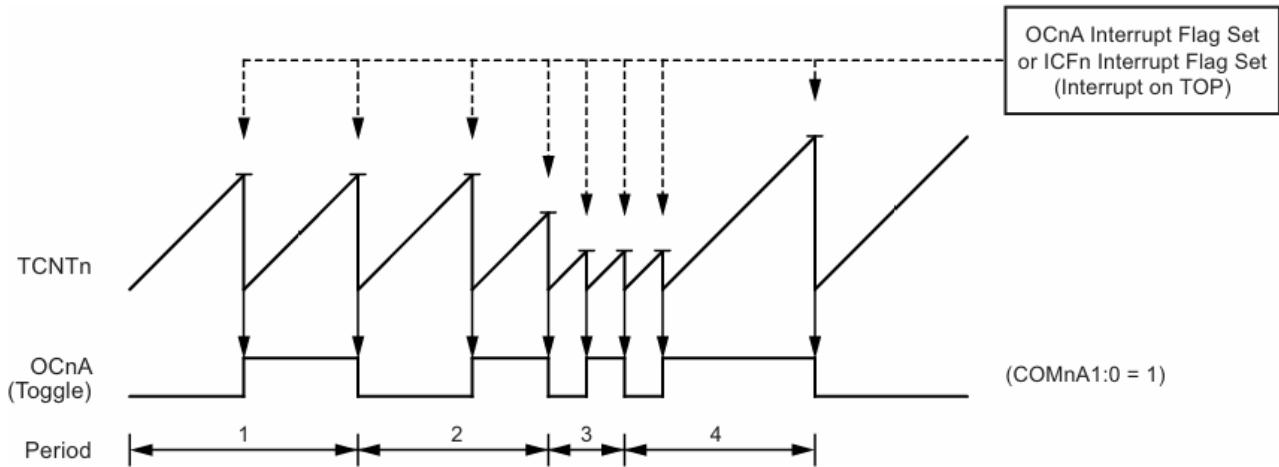
CS12	CS11	CS10	Mô tả
0	0	1	clk _{I/O} /1 (no prescaling)
0	1	0	clk _{I/O} /8 (from prescaler)
0	1	1	clk _{I/O} /64 (from prescaler)
1	0	0	clk _{I/O} /256 (from prescaler)
1	0	1	clk _{I/O} /1024 (from prescaler)

❖ Chế độ Clear Timer on Compare Match – CTC

Ở chế độ này, Timer sẽ bị xóa khi có một so sánh bằng xảy ra. Nói một cách đơn giản, giá trị của bộ đếm TCNT1 được tự động đặt về 0 khi compare match xảy ra. Timer/Counter1 có hai chế độ CTC (mode 4 và mode 12).

Ở mode 12 CTC với TOP là ICR1, khi TCNT1 = ICR1 thì một tín hiệu so sánh xảy ra và bộ đếm được xóa về 0. Đối với mode 4 thì giá trị TOP là OCR1A. Nhìn chung hai chế độ này giống nhau nhưng khi dùng mode 4 thì OCR1A không thể được sử dụng để tạo PWM do nó đã dùng làm giá trị TOP. Chính vì lý do này mà mode 4 phù hợp cho các ứng dụng không cần độ linh hoạt cao, đơn giản; mode 12 có thể dùng để điều khiển xung PWM ở cả hai chân OC1A và OC1B.

Để sử dụng chế độ CTC cần quan tâm đến bit WGM1[3:0] = 4 hoặc 12, đây là các bit giúp chọn mode cần sử dụng. Timer/Counter1 cũng có thể tạo ra một dạng sóng ở chân OC1A/OC1B và tần số cũng được tính theo công thức ở Timer/Counter0.



Hình 9.7. Biểu đồ thời gian của chế độ CTC – Timer/Counter1 [2]

9.2.3. Timer/Counter2

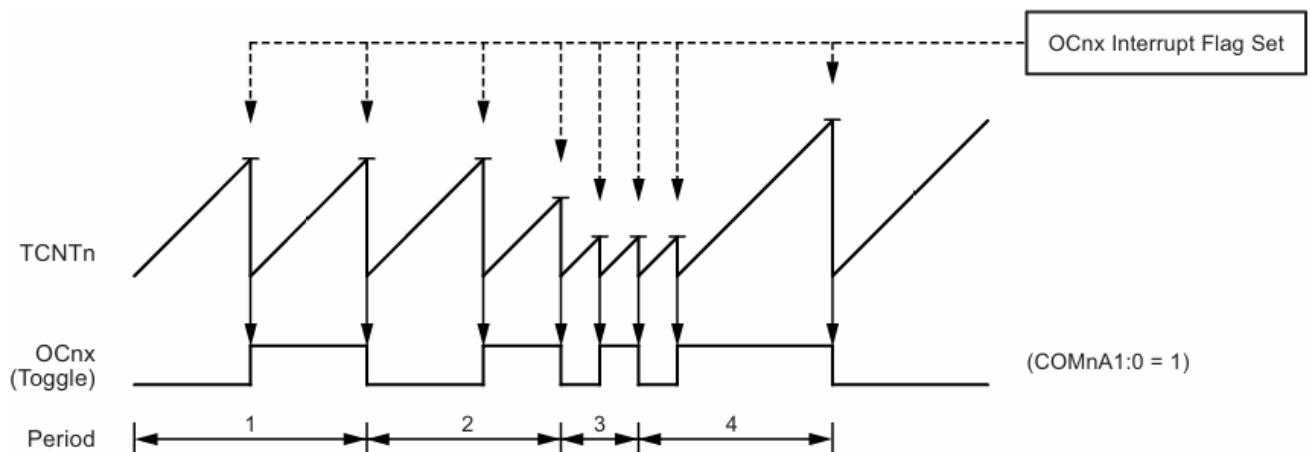
❖ Chế độ thường – Normal Mode

Chế độ Normal của Timer/Counter2 trên ATMEGA328P là chế độ cơ bản nhất, trong đó bộ đếm TCNT2 tăng giá trị từ 0 đến 255 (đối với bộ đếm 8-bit) và tự động tràn về 0 khi đạt giá trị tối đa. Quá trình tràn này sẽ kích hoạt cờ ngắt TOV2 (Timer Overflow Flag), cho phép vi điều khiển xử lý sự kiện thông qua ngắt nếu được cấu hình. Chế độ này thường được sử dụng để tạo trễ thời gian hoặc làm bộ đếm sự kiện khi kết hợp với nguồn xung ngoài trên chân T2. Chế độ Normal rất linh hoạt, dễ cấu hình và là nền tảng để hiểu các chế độ hoạt động phức tạp hơn của Timer/Counter2.

❖ Chế độ Clear Timer on Compare Match – CTC

Chế độ CTC (Clear Timer on Compare Match) của Timer/Counter2 trên ATMEGA328P cho phép bộ đếm TCNT2 tăng từ 0 đến giá trị trong thanh ghi OCR2A. Khi hai giá trị này khớp nhau, bộ đếm tự động xóa về 0, đồng thời cờ OCF2A được bật để tạo ngắt

hoặc xuất tín hiệu tại chân OC2A nếu được cấu hình. Chế độ này tương tự với Timer/Counter0 và Timer/Counter1 ở khả năng so sánh và xóa bộ đếm, nhưng Timer/Counter2 có thêm khả năng sử dụng thạch anh ngoài để tạo nguồn clock độc lập. Điểm khác biệt chính là Timer/Counter0 và Timer/Counter2 là bộ đếm 8-bit, trong khi Timer/Counter1 là 16-bit cho phép đo thời gian dài và chính xác hơn. Chế độ CTC thường được dùng để tạo tín hiệu PWM, tần số cố định hoặc đếm sự kiện trong khoảng thời gian định trước.



Hình 9.8. Biểu đồ thời gian của chế độ CTC – Timer/Counter2 [2]

9.3. Các thanh ghi chức năng Timer/Counter

9.3.1. Timer/Counter0

❖ Thanh ghi điều khiển TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7:6 – COM0A[1:0]: là hai bit điều khiển trạng thái ngõ ra của chân OC0A, tùy thuộc vào cách thiết lập các bit WGM0[2:0] mà có thể chọn chế độ làm việc cho Timer/Counter0.

Bảng 9.4 mô tả chức năng của các bit COM0A[1:0] khi các bit WGM0[2:0] được thiết lập ở chế độ Normal hoặc chế độ CTC.

Bảng 9.4. Cấu hình ngõ ra của chân OC0A [2]

COM0A1	COM0A0	Mô tả
0	0	Hoạt động công thông thường, OC0A ngắt kết nối.
0	1	Đảo trạng thái OC0A khi có so sánh trùng khớp.
1	0	Xóa OC0A khi có so sánh trùng khớp.
1	1	Thiết lập OC0A khi có so sánh trùng khớp.

Bit 5:4 – COM0B[1:0]: có chức năng tương tự như Bit 7:6, giúp điều khiển trạng thái ngõ ra của chân OC0B và tùy thuộc vào cách thiết lập chế độ sử dụng thông qua các bit WGM0[2:0].

Bảng 9.5 mô tả chức năng các bit COM0B[1:0] khi các bit WGM0[2:0] được thiết lập ở chế độ Normal hoặc chế độ CTC.

Bảng 9.5. Cấu hình ngõ ra của chân OC0B [2]

COM0B1	COM0B0	Mô tả
0	0	Hoạt động công thông thường, OC0B ngắt kết nối.
0	1	Đảo trạng thái OC0B khi có so sánh trùng khớp .
1	0	Xóa OC0B khi có so sánh trùng khớp.
1	1	Thiết lập OC0B khi có so sánh trùng khớp.

Bit 1:0 – WGM0[1:0]: Kết hợp với bit WGM02 trong thanh ghi TCCR0B để tạo ra các chế độ làm việc của Timer/Counter0.

Bảng 9.6. Mô tả cấu hình chế độ làm việc của Timer/Counter0 [2]

Mode	WGM02	WGM01	WGM00	Chế độ hoạt động	TOP	Cập nhật OCRx	Cờ tràn TOV
0	0	0	0	Normal	0xFF	Immediate	MAX
2	0	1	0	CTC	OCR0A	Immediate	MAX

Chú ý: Giá trị MAX = 0xFF, BOTTOM = 0x00, TOP là giá trị tối đa của bộ đếm, OCRx là thanh ghi chứa giá trị so sánh, TOV là cờ báo tràn xảy ra khi bộ đếm vượt giá trị TOP.

❖ Thanh ghi điều khiển TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7:6 – FOC0A và FOC0B: đây là hai bit làm thay đổi trạng thái của chân OC0A và OC0B ngay lập tức khi nó được thiết lập với mức logic 1. Tuy nhiên, các bit COM0A[1:0] sẽ quyết định trạng thái mới của chân OC0A và trạng thái này sẽ được giữ cho đến khi có sự kiện khác tác động lên. Khi sử dụng các chế độ PWM thì hai bit 7:6 phải được cấu hình với mức logic 0.

Bit 3 – WGM02: Kết hợp với bit WGM0[1:0] trong thanh ghi TCCR0A để tạo ra các chế độ làm việc của bộ Timer/Counter0 (Tham khảo Bảng 9.6).

Bits 2:0 – CS0[2:0]: Là ba bit xác định nguồn xung clock sử dụng cho Timer/Counter.

Bảng 9.7. Mô tả các bit chọn xung nhịp – Timer/Counter0 [2]

CS02	CS01	CS00	Mô tả
0	0	0	Không có nguồn xung nhịp (Timer/Counter dừng)
0	0	1	clkI/O (Không chia tần số)
0	1	0	clkI/O/8

0	1	1	clkI/O/64
1	0	0	clkI/O/256
1	0	1	clkI/O/1024
1	1	0	Nguồn xung nhịp bên ngoài trên chân T0. Xung nhịp ở cạnh xuống (Falling edge).
1	1	1	Nguồn xung nhịp bên ngoài trên chân T0. Xung nhịp ở cạnh lên (Rising edge).

❖ Thanh ghi TCNT0 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	
0x26 (0x46)	TCNT0 [7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCNT0 là thanh ghi lưu trữ giá trị hiện tại của bộ đếm. Khi giá trị của TCNT0 đạt đến giới hạn (0xFF khi tăng hoặc 0x00 khi giảm), một sự kiện tràn xảy ra, kích hoạt cờ ngắt TOV0 và có thể tạo tín hiệu PWM, đo thời gian hoặc đếm sự kiện. Thanh ghi này sử dụng trong các chế độ hoạt động như Normal Mode, CTC Mode và các chế độ PWM, cung cấp tính linh hoạt cao trong việc xử lý thời gian và tín hiệu.

❖ Thanh ghi OCR0A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
0x27 (0x47)	OCR0A [7:0]								OCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Thanh ghi OCR0A (Output Compare Register 0A) là một thanh ghi 8 bit được sử dụng để lưu giá trị so sánh cho kênh A. Khi giá trị trong bộ đếm TCNT0 bằng với giá trị của OCR0A một sự kiện Compare Match xảy ra. Sự kiện này có thể được cấu hình để thay đổi trạng thái

của chân OC0A, kích hoạt ngắt hoặc thực hiện các thao tác khác tùy thuộc vào các bit COM0A[1:0] trong thanh ghi TCCR0A. Trong chế độ CTC (Clear Timer on Compare Match), thanh ghi OCR0A còn xác định giá trị đếm tối đa, giúp tạo các tín hiệu định thời hoặc sóng vuông. Trong chế độ Fast PWM và Phase Correct PWM, giá trị của OCR0A điều khiển chu kỳ làm việc (duty cycle) của tín hiệu PWM trên chân OC0A, phục vụ các ứng dụng như điều khiển LED, động cơ, hoặc tạo tín hiệu điều chế.

❖ Thanh ghi OCR0B – Output Compare Register B

Bit	7	6	5	4	3	2	1	0	
0x28 (0x48)	OCR0B [7:0]								OCR0B
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Thanh ghi OCR0B (Output Compare Register 0B) cũng là một thanh ghi 8 bit, tương tự như OCR0A, nhưng được sử dụng cho kênh B của bộ Timer/Counter0. Khi giá trị trong TCNT0 bằng với giá trị của OCR0B, một sự kiện Compare Match xảy ra và có thể được cấu hình để thay đổi trạng thái của chân OC0B, kích hoạt ngắt hoặc thực hiện các chức năng khác thông qua các bit COM0B[1:0] trong thanh ghi TCCR0A. Trong các chế độ PWM, thanh ghi OCR0B điều chỉnh chu kỳ làm việc của tín hiệu PWM trên chân OC0B, cho phép điều khiển độc lập hai tín hiệu đầu ra (OC0A và OC0B) từ cùng một bộ Timer. Thanh ghi này rất hữu ích trong các ứng dụng cần điều khiển nhiều kênh như điều chỉnh cường độ sáng của LED hoặc điều khiển nhiều động cơ cùng lúc.

❖ Thanh ghi TIMSK0 – Timer/Counter Interrupt Mask Register

Là thanh ghi được sử dụng để bật/tắt các ngắt có liên quan đến bộ đếm/bộ định thời của Timer/Counter0.

Bit	7	6	5	4	3	2	1	0	
(0x6E)	-	-	-	-	-	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 2:1 – OCIE0B và OCIE0A: Là bit kích hoạt ngắt khi xảy ra sự so sánh giữa bộ đếm và thanh ghi tương ứng OCR0B hoặc OCIE0A.

Bit 0 – TOIE0: Có chức năng kích hoạt ngắt tràn khi bộ đếm đạt đến giá trị giới hạn.

❖ Thanh ghi TIFR0 – Timer/Counter 0 Interrupt Flag Register

Là thanh ghi dùng để lưu trữ trạng thái của các cờ ngắt. Khi một sự kiện cụ thể xảy ra, các bit tương ứng trong thanh ghi này sẽ được thiết lập bởi phần cứng.

Bit	7	6	5	4	3	2	1	0	
0x15(0x35)	-	-	-	-	-	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 2:1 – OCF0B và OCF0A: Hai bit này được thiết lập khi xảy ra sự so sánh giữa bộ đếm với giá trị của thanh ghi tương ứng OCR0B hoặc OCR0A. Trong trường hợp muốn xóa bit này, cần ghi giá trị logic 1 vào chính nó. Ngoài ra phải kết hợp với các bit trong thanh ghi TIMSK0 để tạo ra các ngắt tương ứng.

Bit 0 – TOV0: Khi xảy ra sự kiện tràn của bộ đếm trong Timer/Counter0, bit này sẽ được thiết lập và sẽ tự động được xóa khi thực thi vector ngắt tương ứng.

9.3.2. Timer/Counter1

❖ Thanh ghi TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10	TCCR1A
Read/ Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Các bit COM1A[1:0] và COM1B[1:0] điều khiển trạng thái các chân OC1A và OC1B bằng cách kết hợp với việc thiết lập các bit WGM1[3:0] sẽ tạo ra dạng sóng và chế độ làm việc phù hợp. Bảng 9.8 thể hiện cách thiết lập dành cho COM1A[1:0] và COM1B[1:0] ở chế độ Normal hoặc CTC.

Bảng 9.8. Cấu hình ngõ ra của chân OC1A/OC1B [2]

COM1A1/ COM1B1	COM1A0/ COM1B0	Mô tả
0	0	Hoạt động cổng thông thường, OC1A/OC1B ngắt kết nối.
0	1	Đảo trạng thái OC1A/OC1B khi có so sánh trùng khớp.
1	0	Xóa OC1A/OC1B khi xảy ra so sánh.
1	1	Ghi OC1A/OC1B khi xảy ra so sánh.

Bit 1:0 – WGM1[1:0]: Kết hợp với hai bit 4:3 – WGM1[3:2] trong thanh ghi TCCR1B để tạo ra các chế độ làm việc như Normal, CTC và ba chế độ điều khiển PWM. Ngoài ra còn lựa chọn giá trị TOP, thời điểm cập nhật giá trị so sánh và thời điểm xảy ra ngắt tràn. Bảng 9.9 mô tả cách cấu hình các bit WGM[3:0].

Bảng 9.9. Mô tả cấu hình chế độ làm việc của Timer/Counter1 [2]

Mode	WGM13	WGM12	WGM12	WGM11	Chế độ hoạt động	TOP	Cập nhật OCR1x	Cờ TOV1
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
12	1	1	0	0	CTC	ICR1	Immediate	MAX

❖ Thanh ghi TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0
(0x81)	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
Read/ Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – ICNC1: Kích hoạt bộ lọc nhiễu khi thiết lập bit này với mức logic 1. Bộ lọc sẽ loại bỏ các tín hiệu nhiễu ngắn, hữu ích trong trường hợp đo tín hiệu cảm biến.

Bit 6 – ICES1: Lựa chọn cạnh xung sẽ sử dụng để kích hoạt Input Capture trên chân ICP1. Khi ICES1 = 1 bộ đếm sẽ dựa vào tín hiệu chuyển từ mức LOW sang mức HIGH và ngược lại khi ICES1 = 0 sẽ dựa vào tín hiệu chuyển từ HIGH về LOW để kích hoạt Input Capture.

Bit 4:3 – WGM1[3:2]: Kết hợp với hai bit WGM1[1:0] trong thanh ghi TCCR1A để chọn chế độ hoạt động cho Timer/Counter1 (Tham khảo Bảng 9.9).

Bit 2:0 – CS1[2:0]: Là các bit lựa chọn bộ chia tần số cho Timer/Counter1.

Bảng 9.10. Mô tả các bit chọn xung nhịp – Timer/Counter1 [2]

CS12	CS11	CS10	Mô tả
0	0	0	Không có nguồn xung nhịp (Timer/Counter dừng)
0	0	1	Clk _{I/O} /1

0	1	0	Clk _{I/O} /8
0	1	1	Clk _{I/O} /64
1	0	0	Clk _{I/O} /256
1	0	1	Clk _{I/O} /1024
1	1	0	Nguồn xung nhịp bên ngoài trên chân T1. Xung nhịp ở cạnh xuống (Falling edge).
1	1	1	Nguồn xung nhịp bên ngoài trên chân T1. Xung nhịp ở cạnh lên (Rising edge).

Nếu chế độ nhận xung nhịp bên ngoài được sử dụng, chân T1 sẽ được mặc định là chân nhận tín hiệu ngay cả khi nó được thiết lập là ngõ ra.

❖ Thanh ghi TCCR1C – Timer/Counter1 Control Register C

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	-	-	-	-	-	-	TCCR1C
Read/ Write	R/W	R/W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Hai bit FOC1A và FOC1B chỉ hoạt động trong chế độ ngoài PWM (non-PWM), khi nó được ghi với mức logic 1 cũng giống trường hợp bộ đếm đạt giá trị so sánh, chân OC1A/OC1B sẽ thay đổi trạng thái tùy thuộc vào cấu hình các bit COM1x1:0. Thông thường, FOC1A/FOC1B được ghi với mức logic 0.

❖ Thanh ghi TCNT1H và TCNT1L

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1 [15:8]								TCNT1H
(0x84)	TCNT1 [7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCNT1 là thanh ghi 16-bit được sử dụng để lưu trữ giá trị đếm của bộ Timer/Counter1, đây là bộ đếm đa năng có thể hoạt động ở các chế độ như Normal, CTC và tạo xung PWM. Bao gồm hai byte: TCNT1H và TCNT1L cho phép đếm từ 0 đến 65535. Việc cấu hình và sử dụng TCNT1 cần kết hợp với các thanh ghi điều khiển như TCCR1A, TCCR1B và các thanh ghi so sánh như OCR1A, OCR1B.

❖ Bộ thanh ghi OCR1AH – OCR1AL và OCR1BH – OCR1BL

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A [15:8]								OCR1AH
(0x88)	OCR1A [7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
(0x8B)	OCR1B [15:8]								OCR1BH
(0x8A)	OCR1B [7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bộ thanh ghi này được sử dụng để lưu giá trị so sánh trong các chế độ hoạt động của Timer/Counter1, chẳng hạn như chế độ CTC hoặc tạo xung PWM. Khi giá trị của TCNT1 bằng với giá trị so sánh có thể xảy ra ngắn, thay đổi trạng thái chân OC1A/OC1B hoặc tạo tín hiệu PWM. Việc cài đặt giá trị chính xác cho OCR1AH/ OCR1BH và OCR1AL/ OCR1BL rất quan trọng trong các ứng dụng yêu cầu độ chính xác cao về thời gian hoặc tín hiệu.

❖ Thanh ghi ICR1H và ICR1L – Input Capture Register 1

Bit	7	6	5	4	3	2	1	0	
(0x87)	ICR1 [15:8]								ICR1H
(0x86)	ICR1 [7:0]								ICR1L
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

❖ Thanh ghi TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	-	-	ICIE1	-	-	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 5 – ICIE1: Khi bit này được ghi với mức logic 1 thì ngắt loại Input Capture được kích hoạt.

Bit 2:1 – OCIE1B và OCIE1A: Cho phép kích hoạt ngắt nếu xảy ra sự so sánh giữa bộ đếm và thanh ghi tương ứng OCR1B/OCR1A.

Bit 0 – TOIE1: Quy định ngắt tràn cho bộ Timer/Counter1.

❖ Thanh ghi TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x16(0x36)	-	-	ICF1	-	-	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 5 – ICF1: Là cờ ngắt cho sự kiện Input Capture, khi đó chân ICP1 được ghi ở mức logic 1. Ngoài ra, trong các chế độ làm việc của Timer/Counter1, nếu thanh ghi ICR1 được được cấu hình là giá trị TOP thì bit ICF1 vẫn được ghi ở mức logic 1 nếu bộ đếm đạt giá trị

TOP. ICF1 sẽ tự động được xóa khi vector ngắt được thực thi và có thể chủ động xóa bit này bằng cách ghi mức logic 1 vào nó.

Bit 2:1 – OCF1B và OCF1A: Hai bit này sẽ được thiết lập lên mức logic 1 khi giá trị của bộ đếm TCNT1 trùng với giá trị được lưu trong thanh ghi so sánh OCR1B hoặc OCR1A. Các bit này sẽ tự động được xóa khi vector ngắt tương ứng được thực thi. Ngoài ra, cũng có thể chủ động xóa chúng bằng cách ghi giá trị logic 1 vào các bit này.

Bit 0 –TOV1: Cho phép nhận biết bộ đếm đã tràn và có thể kết hợp với bit TOIE1 trong thanh ghi TIMSK1 để kích hoạt ngắt tràn. Các bit WGM1[3:0] sẽ quyết định thời điểm xảy ra ngắt tràn (Tham khảo Bảng 9.9).

9.3.3. Timer/Counter2

❖ Thanh ghi điều khiển TCCR2A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	-	-	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Chức năng của thanh ghi TCCR2A cũng tương tự như thanh ghi TCCR0A của bộ Timer/Counter0. Các bit COM2A[1:0] và COM2B[1:0] điều khiển dạng tín hiệu ngõ ra ở hai chân tương ứng OC2A và OC2B. Hai bit WGM2[1:0] kết hợp với bit WGM22 của thanh ghi TCCR2B để lựa chọn chế độ làm việc.

Bảng 9.11. Chế độ đầu ra so sánh, không PWM – Timer/Counter0 [2]

COM2A1/ COM2B1	COM2A0/ COM2B0	Mô tả
0	0	Hoạt động như công thông thường, OC2A/OC2B ngắt kết nối.
0	1	Đảo trạng thái OC2A/OC2B khi có so sánh trùng khớp.
1	0	Xóa OC2A/OC2B khi xảy ra so sánh.
1	1	Ghi OC2A/OC2B khi xảy ra so sánh.

Bảng 9.12. Mô tả cấu hình chế độ làm việc của Timer/Counter2 [2]

Mode	WGM22	WGM21	WGM20	Chế độ hoạt động	TOP	Cập nhật OCRx	Cờ tràn TOV
0	0	0	0	Normal	0xFF	Immediate	MAX
2	0	1	0	CTC	OCR2A	Immediate	MAX

❖ Thanh ghi điều khiển TCCR2B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20	TCCR2B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7:6 (FOC2A và FOC2B): đây là hai bit làm thay đổi trạng thái của chân OC2A và OC2B ngay lập tức khi nó được thiết lập với mức logic 1. Tuy nhiên, các bit COM2A[1:0] và COM2B[1:0] sẽ quyết định trạng thái mới của chân OC2A/OC2B và trạng thái này sẽ được giữ cho đến khi có sự kiện khác tác động lên. Khi sử dụng các chế độ PWM thì hai bit 7:6 phải được cấu hình với mức logic 0.

Bit 3 – WGM22: Kết hợp với bit WGM2[1:0] trong thanh ghi TCCR0A để tạo ra các chế độ làm việc của bộ Timer/Counter2 (Tham khảo Bảng 9.12).

Bit 2:0 (CS2[2:0]): Là ba bit xác định nguồn xung clock sử dụng cho Timer/Counter.

Bảng 9.13. Mô tả các bit chọn xung nhịp – Timer/Counter1 [2]

CS02	CS01	CS00	Mô tả
0	0	0	Không có nguồn xung nhịp (Timer/Counter dừng)
0	0	1	clk _{T2S} (Không chia tần số)
0	1	0	clk _{T2S} /8
0	1	1	clk _{T2S} /32

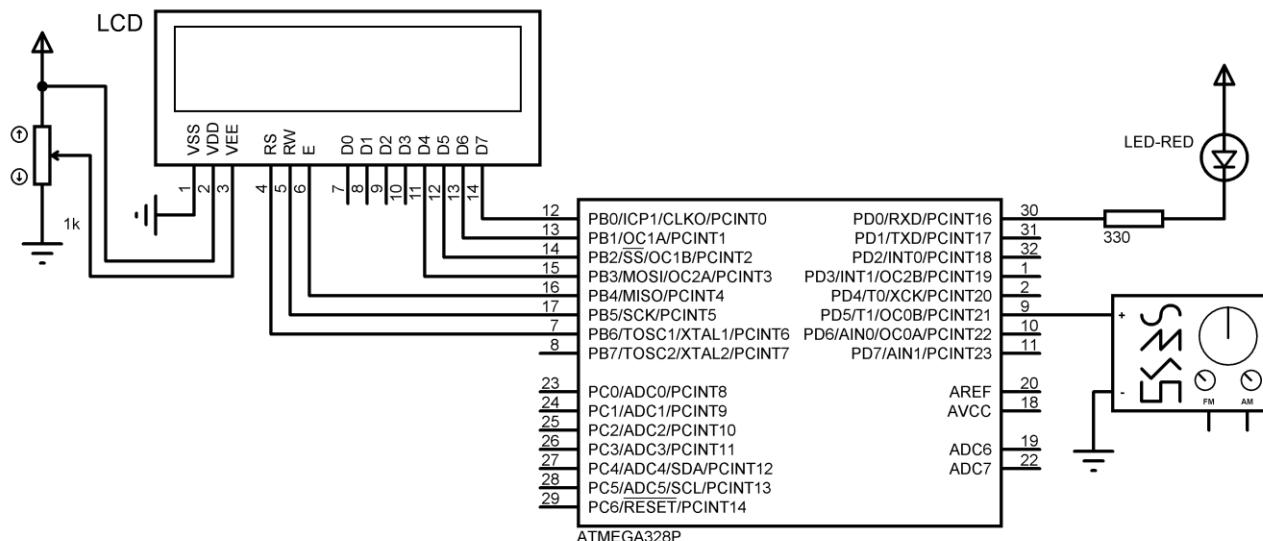
1	0	0	$\text{clk}_{\text{T2S}} / 64$
1	0	1	$\text{clk}_{\text{T2S}} / 128$
1	1	0	$\text{clk}_{\text{T2S}} / 256$
1	1	1	$\text{clk}_{\text{T2S}} / 1024$

Ngoài ra, bộ Timer/Counter2 còn có các thanh ghi khác như TCNT2, OCR2A, OCR2B, TIMSK2 và TIFR2. Tuy nhiên, các bit trong từng thanh ghi có chức năng tương tự như bộ Timer/Counter0.

9.4. Bài thực hành

Yêu cầu bài thực hành: Sử dụng bộ Timer 0 để tạo delay 100ms, nhấp nháy LED ở chân PD1, song song đó sử dụng bộ Timer 1 để đếm xung đầu vào và hiển thị giá trị đếm lên LCD, mỗi khi đếm được 5 xung thì giá trị của biến đếm (counter) sẽ tăng lên một đơn vị. Hai bộ Timer 0 và 1 làm việc độc lập với nhau, xung đếm đi vào nhanh hay chậm đều không ảnh hưởng đến hoạt động nháy LED của Timer 0.

❖ Phản mô phỏng



Hình 9.9. Sơ đồ kết nối bài thực hành bộ định thời và bộ đếm thời gian

❖ Phần lập trình

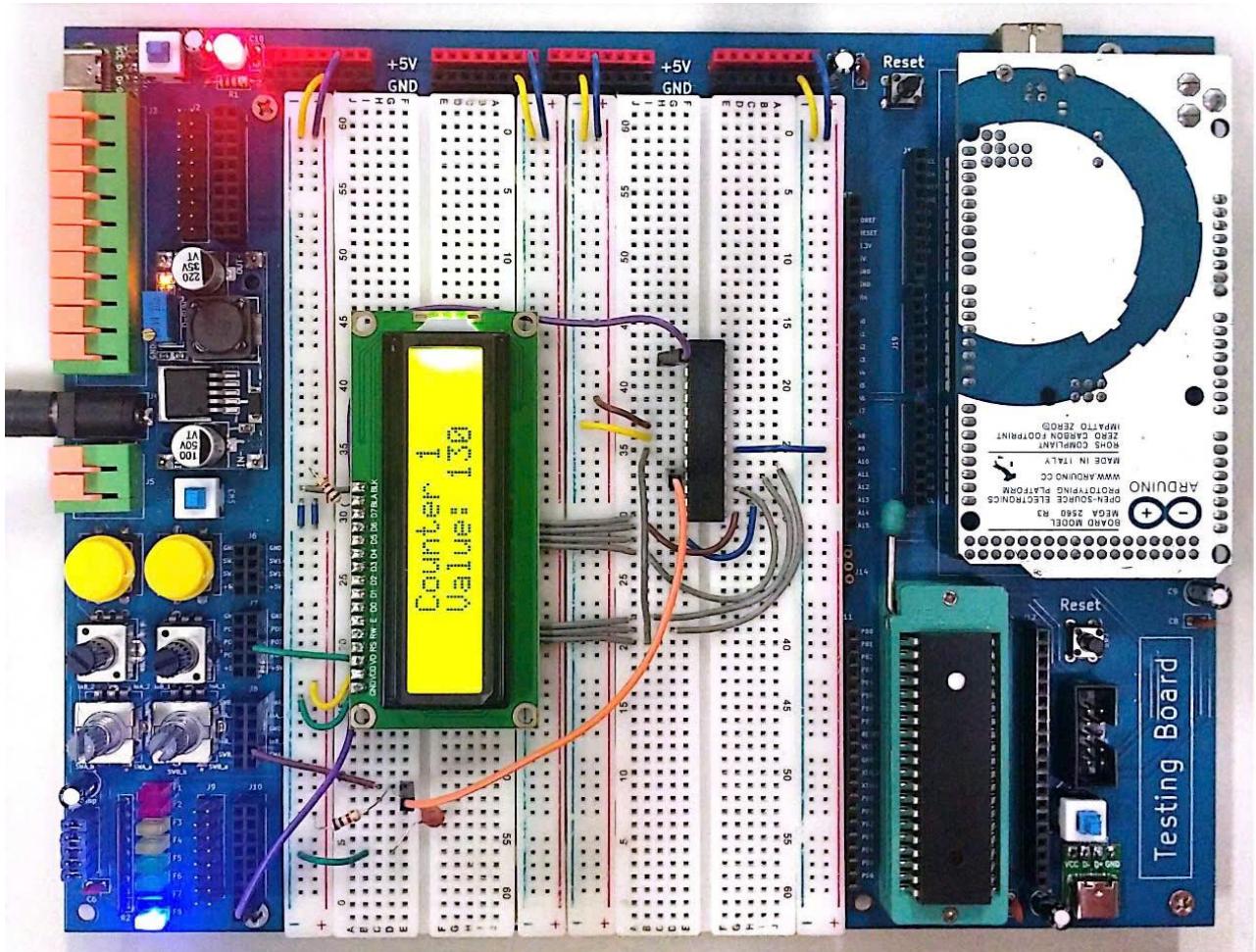
	SourceCode_ATMEGA328P\P9_Timer0_Counter1
	<pre>1 #include <mega328p.h> 2 #include <alcd.h> 3 #include <stdio.h> 4 5 unsigned char counter = 0; 6 int value = 0; 7 char myStr[10]; 8 interrupt [TIM0_OVF] void timer0_overflow_isr(void) 9 { 10 TCNT0 = 60; //dat lai gia tri cho bo dem 11 if(counter > 4){ 12 PORTD ^= (1<<PORTD0); //da trang thai chan LED 13 counter = 0; 14 } 15 else counter++; 16 } 17 interrupt [TIM1_COMPA] void timer1_compareA_isr(void) 18 { 19 value++; 20 sprintf(myStr, "%d", value); 21 lcd_gotoxy(10,1); 22 lcd_puts(myStr); 23 } 24 void main(void) 25 { 26 #pragma optsize- 27 CLKPR =(1<<CLKPCE); 28 CLKPR = 0x00; 29 #ifdef _OPTIMIZE_SIZE_ 30 #pragma optsize+ 31 #endif 32 33 DDRD.0 = 1; //ngo ra dieu khien LED</pre>

```

34 PORTD.0 = 0;      //LED tat
35 //Delay dung Timer 0
36 //che do Normal, bo chia 1024
37 TCCR0A=(0<<WGM01) | (0<<WGM00);
38 TCCR0B=(0<<WGM02) | (1<<CS02) | (0<<CS01) | (1<<CS00);
39 //gia tri ban dau cua bo dem
40 TCNT0=60;
41 TIMSK0 |= (1<<TOIE0); //cho phep ngat khi xay ra tran
42 //Dem xung dung Timer 1
43 //che do CTC, xung ngoai, lay canh xuong
44 DDRD.5 = 0;
45 PORTD.5 = 0;
46 TCCR1A=(0<<WGM11) | (0<<WGM10);
47 TCCR1B=(1<<WGM12) | (1<<CS12) | (1<<CS11) | (0<<CS10);
48 //gia tri ban dau cua bo dem bang 0
49 TCNT1H=0x00;
50 TCNT1L=0x00;
51 //dem den 10 se xay ra ngat
52 OCR1AH=0x00;
53 OCR1AL=0x10;
54 //cho phep ngat khi xay ra so sanh
55 TIMSK1 |= (1<<OCIE1A);
56 lcd_init(16);
57 lcd_gotoxy(3,0);
58 lcd_puts("Counter 1");
59 lcd_gotoxy(3,1);
60 lcd_puts("Value: ");
61 #asm("sei")
62 while (1){}
63 }

```

❖ Phân mô hình

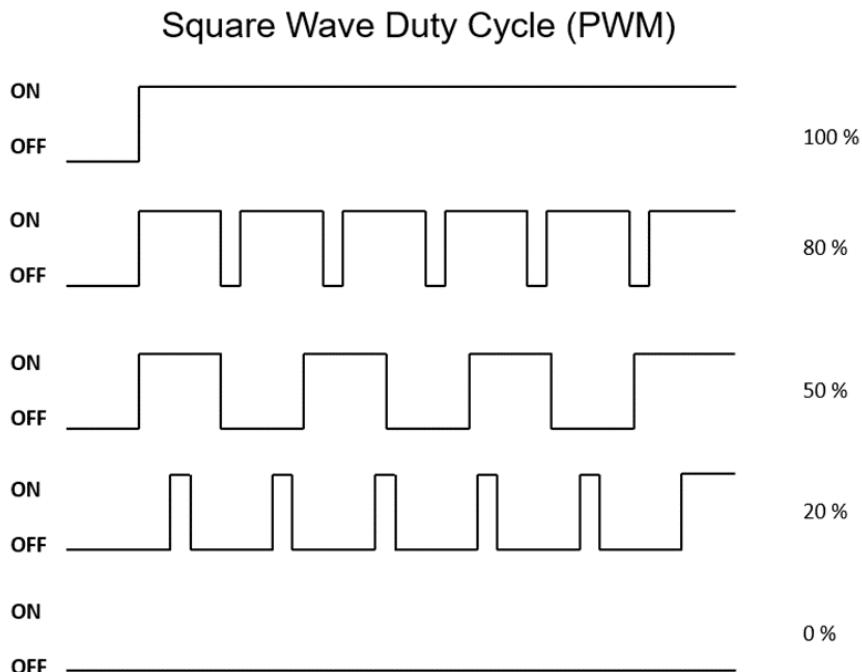


Hình 9.10. Mô hình thực tế bài thực hành bộ định thời và bộ đếm thời gian

CHƯƠNG 10. CẤU TRÚC BỘ TIMER/COUNTER/PWM (tiếp theo)

10.1. Cấu trúc PWM

PWM – Pulse Width Modulation là phương pháp điều chế độ rộng xung dựa trên sự chênh lệch thời gian giữa bật và tắt nguồn điện từ đó làm thay đổi điện áp trung bình cung cấp cho tải.



Hình 10.1. Chu kỳ điều chế độ rộng xung

Duty Cycle là một khái niệm quan trọng, biểu thị tỉ lệ phần trăm thời gian mà tín hiệu PWM ở mức cao so với tổng chu kỳ (tổng thời gian mức cao và mức thấp).

$$\text{Duty Cycle (\%)} = \frac{\text{Thời gian mức cao (ON)}}{\text{Tổng chu kỳ}} \times 100\% \quad (12)$$

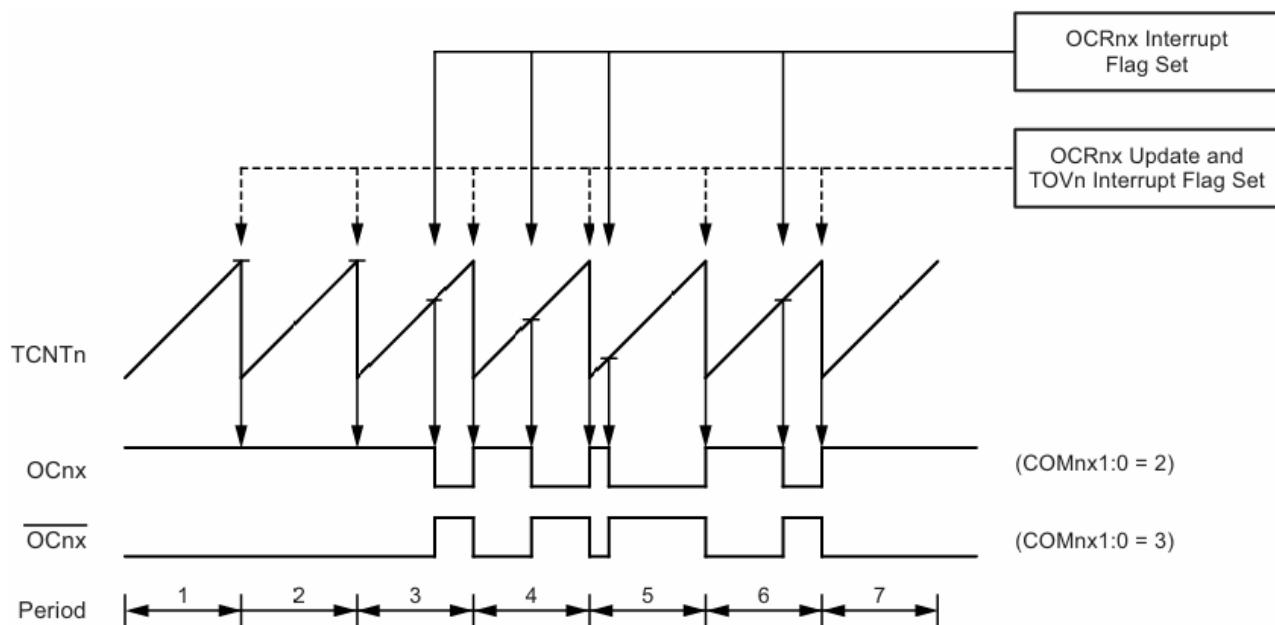
Vì điều khiển sẽ thực hiện nhiệm vụ đóng, ngắt nguồn điện đưa vào tải từ đó tạo ra mức điện áp trung bình để điều khiển các thiết bị như động cơ điện, servo,...

10.2. Các chế độ làm việc PWM

10.2.1. Timer/Counter0

❖ Chế độ Fast PWM

Có hai mode có thể sử dụng khi ở chế độ Fast PWM, bằng cách cấu hình các bit WGM02:0 = 3 hoặc 7. Bộ đếm sẽ bắt đầu từ giá trị BOTTOM và tăng dần đến giá trị TOP sau đó quay lại giá trị BOTTOM. Ở mode 3 (WGM2:0= 3) giá trị TOP luôn là giá trị cố định 0xFF nhưng đối với mode 7 (WGM2:0= 7) thì giá trị này được quyết định bởi giá trị OCR0A.



Hình 10.2. Biểu đồ thời gian của chế độ Fast PWM – Timer/Counter0 [2]

Do sử dụng cơ chế hoạt động single-slope nên tần số của chế độ Fast PWM có thể nhanh gấp đôi các chế độ khác. Giá trị của tần số được tính theo công thức:

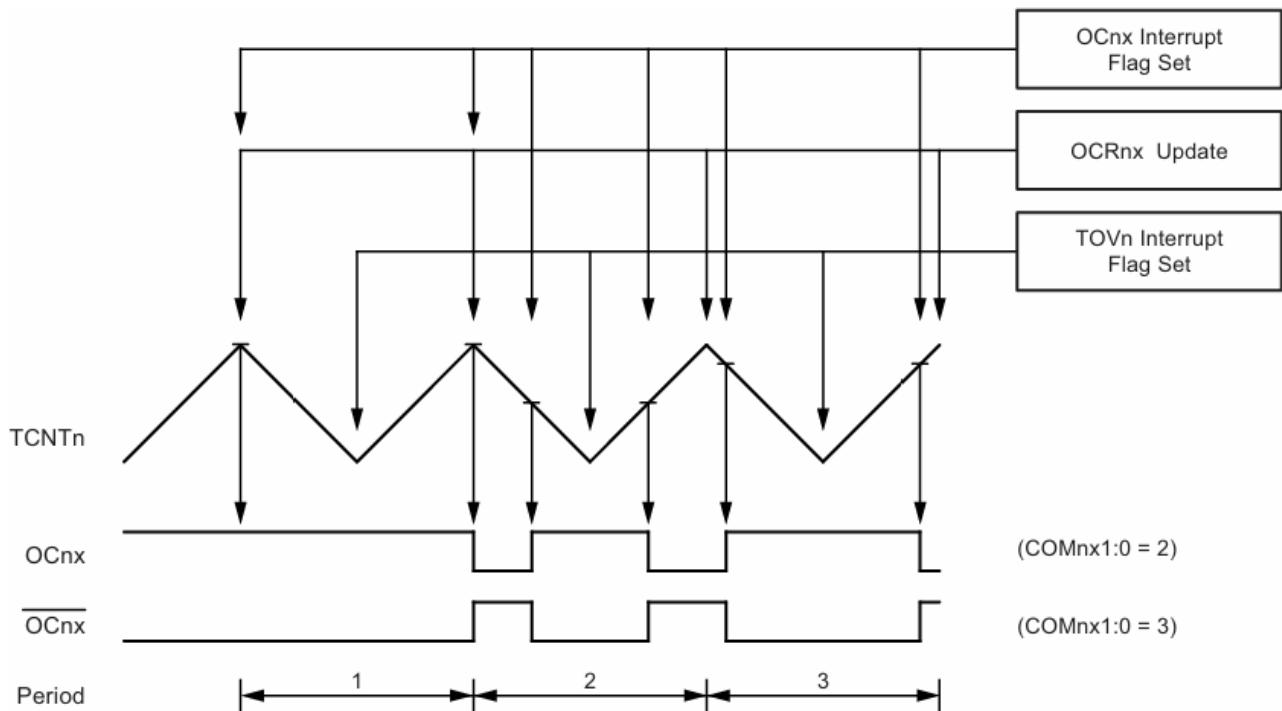
$$f_{\text{PWM}} = \frac{f_{\text{clk I/O}}}{N \times 256} \quad (13)$$

Với $f_{\text{clkI/O}}$: Tần số xung nhịp của vi điều khiển.

N: giá trị bộ chia tần số.

❖ Chế độ Phase Correct PWM.

Chế độ Phase Correct ở Timer/Counter0 có hai mode để sử dụng (WGM2:0= 1 hoặc 5) cung cấp dạng sóng có độ phân giải cao, giá trị TOP của mode 1 là giá trị cố định 0xFF, đối với mode 5 thì giá trị này bằng với OCR0A. Bộ đếm (TCNT0) được tăng lên cho đến khi đạt giá trị so sánh (OCRx), lúc này trạng thái của chân ngõ ra OCnx sẽ được thay đổi.



Một cờ tràn (TOV0) xảy ra mỗi khi bộ đếm quay lại giá trị BOTTOM, đây cũng là thời điểm cập nhật giá trị OCR0A mới. Việc cập nhật tại thời điểm này giúp đảm bảo tín hiệu PWM không bị nhiễu hay gián đoạn trong quá trình đếm và đồng thời duy trì tính đối xứng của sóng. Tần số sóng PWM có thể được tính theo công thức sau:

$$f_{\text{PWM}} = \frac{f_{\text{clk I/O}}}{N \times 510} \quad (14)$$

Với $f_{\text{clkI/O}}$: Tần số xung nhịp của vi điều khiển, N: giá trị bộ chia tần số.

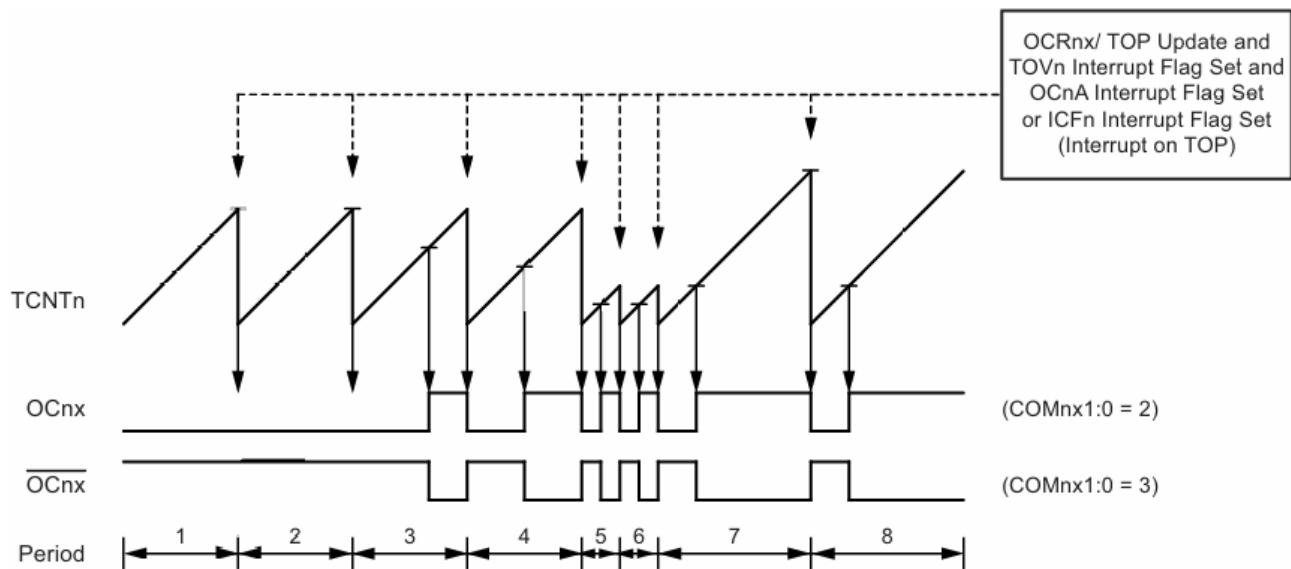
10.2.2. Timer/Counter1

❖ Chế độ Fast PWM

Fast PWM Mode hay chế độ PWM nhanh cho phép tạo sóng PWM ở tần số cao, với nguyên lý hoạt động chỉ một cạnh (single-slope). Bộ đếm TCNT1 sẽ đếm từ giá trị BOTTOM đến TOP và quay trở lại BOTTOM tạo ra chu kỳ hoạt động của chế độ này.

Tùy vào cách thiết lập các bit WGM13:0 trong thanh ghi TCCR1A và TCCR1B kết hợp với việc chọn ngõ ra tại chân OC1x (chân PB1 hoặc PB2) sẽ tạo ra dạng sóng PWM với tần số cao.

Do hoạt động với cơ chế single-slope, chế độ Fast PWM có thể đạt tần số hoạt động gấp đôi so với các chế độ Phase Correct PWM và Phase and Frequency Correct PWM sử dụng cơ chế dual-slope. Vì thế Fast PWM rất phù hợp cho các ứng dụng như điều chỉnh công suất, chỉnh lưu, chuyển đổi DAC.

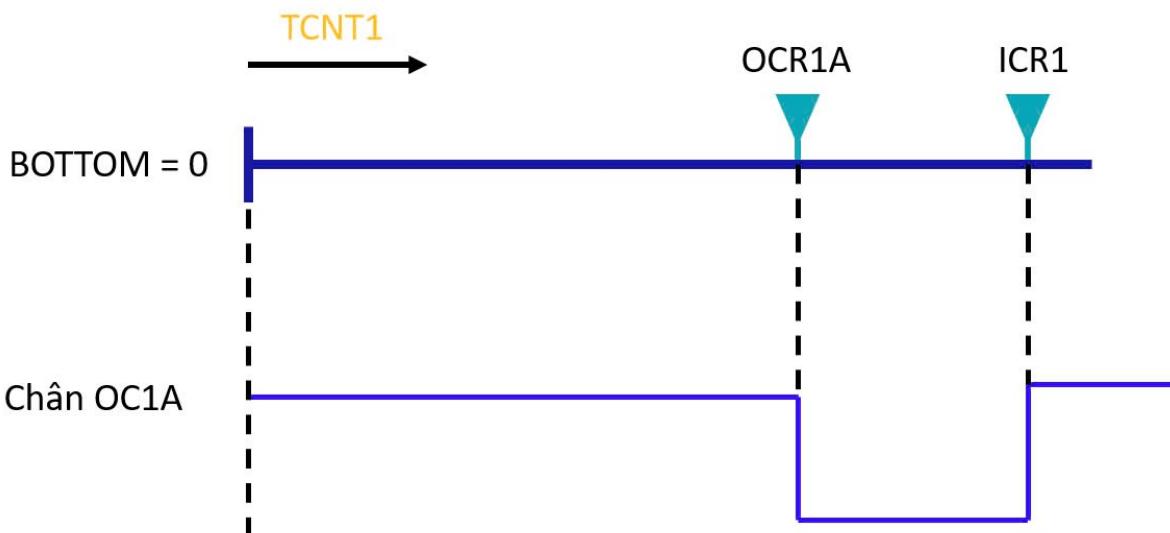


Hình 10.4. Biểu đồ thời gian ở chế độ Fast PWM [2]

Hình 10.3 thể hiện chế độ Fast PWM khi OCR1A/ICR1 được dùng để xác định giá trị TOP, OCR1x được xem là giá trị so sánh của cả 5 chế độ của Fast PWM (mode 5, 6, 7, 14,

15). Trong trường hợp sử dụng mode 15, OCR1A đồng thời cũng là giá trị TOP vì thế khi TCNT1 đạt giá trị OCR1A sẽ không làm thay đổi trạng thái của chân Ocnx mà chỉ xảy ra ngắt tràn và TCNT1 được trả về 0, tương ứng với giai đoạn 1 và 2 trên Hình 10.3. Chính vì thế mode 15 chỉ phù hợp cho các ứng dụng yêu cầu tần số cố định, tín hiệu đồng bộ hoặc ngắt định kỳ.

Đối với mode 14, giá trị TOP được quyết định bởi ICR1, thanh ghi đếm TCNT1 vẫn tăng dần đến khi đạt giá trị so sánh (OCR1x), chân Ocnx (chân PB1/PB2) sẽ thay đổi trạng thái xuất tín hiệu tùy thuộc vào các bit COMnx[1:0]. Tuy nhiên, TCNT1 vẫn tiếp tục đếm cho đến khi đạt giá trị TOP (ICR1) và xảy ra tràn, kết thúc một chu kỳ làm việc.



Hình 10.5. Mode 14 trong chế độ Fast PWM [12]

Tần số xung PWM ở chế độ Fast PWM có thể được xác định theo công thức:

$$f_{\text{PWM}} = \frac{f_{\text{clkI/O}}}{N \times (1 + \text{TOP})} \quad (15)$$

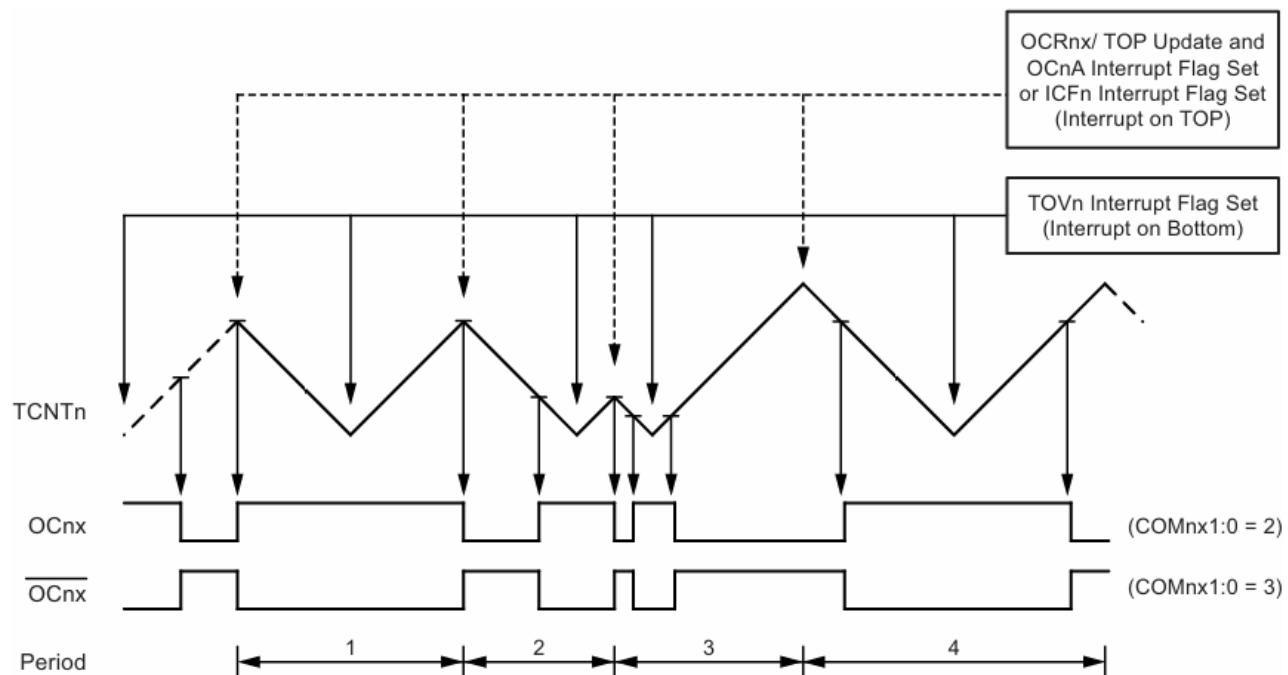
Với $f_{\text{clkI/O}}$: Tần số xung nhịp của vi điều khiển.

N: giá trị bộ chia tần số.

TOP: giá trị tối đa của bộ đếm (Tham khảo Bảng 9.10)

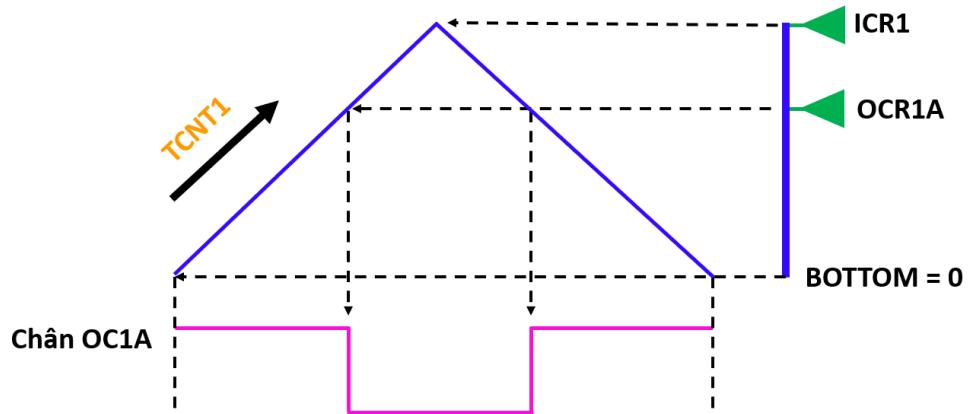
❖ Chế độ Phase Correct PWM

Chế độ Phase Correct giúp tạo xung PWM có độ phân giải cao và chính xác về pha, dựa trên cơ chế hai cạnh – dual slope. Khác với Fast PWM, thanh ghi TCNT1 tăng từ BOTTOM đến TOP, sau đó từ TOP giảm dần về BOTTOM và lặp lại chu kỳ.



Hình 10.6. Biểu đồ thời gian ở chế độ Phase Correct PWM [2]

Khi thiết lập các bit WGM13:0 = 1, 2 hoặc 3, giá trị lớn nhất mà thanh ghi đếm có thể đạt được lần lượt là 0x00FF, 0x001F hoặc 0x03FF, đối với WGM13:0 = 10 và WGM13:0 = 11 thì giá trị này phụ thuộc vào ICR1 và OCR1A. Giá trị so sánh và giá trị TOP mới sẽ được cập nhật khi thanh ghi đếm đạt giá trị TOP, vì thế chế độ này phù hợp cho các ứng dụng với độ phân giải cao nhưng không yêu cầu đối xứng hoàn hảo chẳng hạn như điều chỉnh độ sáng LED, tạo tín hiệu DAC.



Hình 10.7. Mode 10 trong chế độ Phase Correct PWM [12]

Hình 10.6 thể hiện một cách trực quan về cách đếm, so sánh giá trị và tạo xung PWM tại chân OC1A (chân PB1). Tần số ngõ ra PWM được xác định theo công thức:

$$f_{\text{PWM}} = \frac{f_{\text{clk I/O}}}{2 \times N \times \text{TOP}} \quad (16)$$

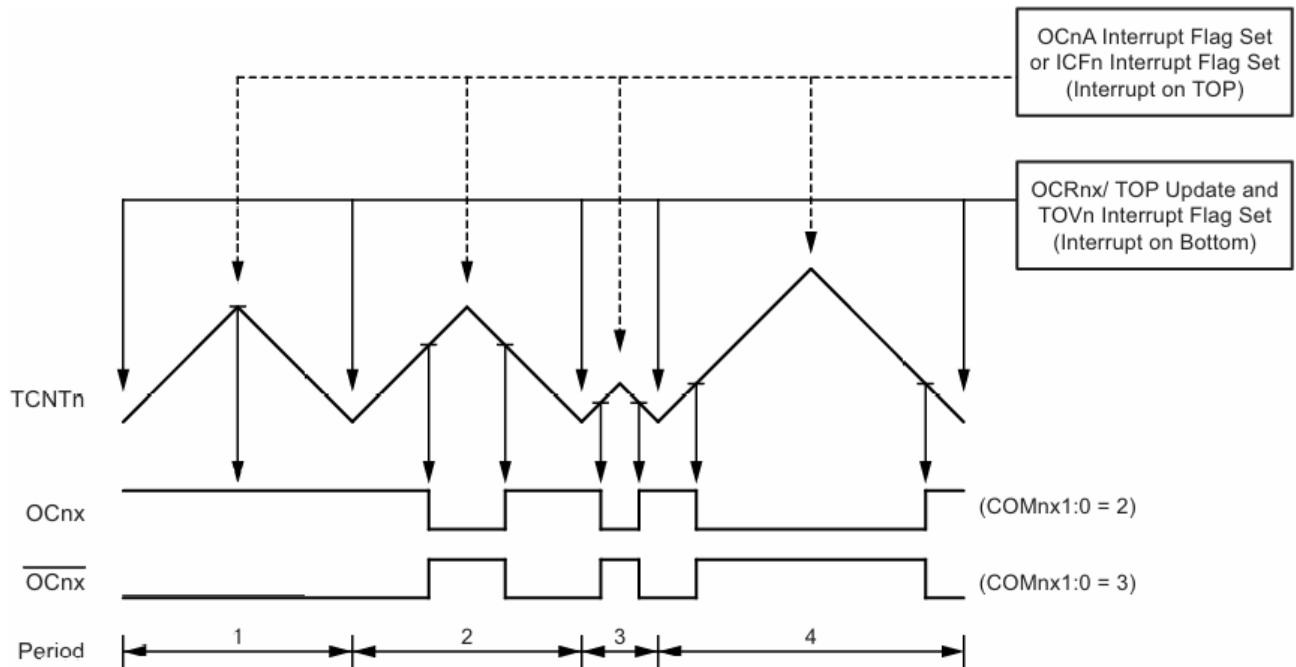
Với $f_{\text{clk I/O}}$: Tần số xung nhịp của vi điều khiển.

N: giá trị bộ chia tần số.

TOP: giá trị tối đa của bộ đếm (Tham khảo Bảng 9.10)

❖ Chế độ Phase and Frequency Correct PWM

Nguyên lý hoạt động của chế độ Phase and Frequency Correct PWM tương tự như chế độ Phase Correct PWM, với cơ chế dual-slope cho tín hiệu PWM ổn định và chính xác.



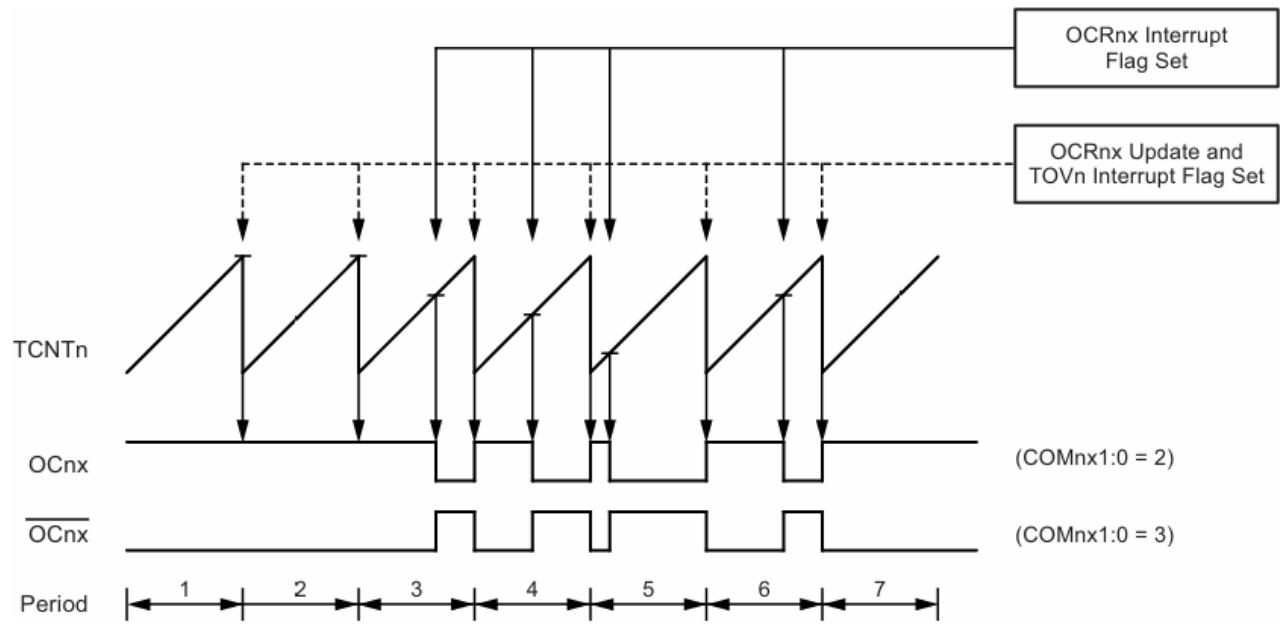
Hình 10.8. Biểu đồ thời gian ở chế độ Phase and Frequency Correct PWM [2]

Giá trị TOP của chế độ này chỉ có thể là một trong hai thanh ghi OCR1A hoặc ICR1 được thiết lập thông qua các bit WGM13:0 = 8 hoặc 9. Khác với Phase Correct PWM, giá trị thanh ghi OCR1x được cập nhật khi TCNT1 giảm về BOTTOM, điều này giúp giá trị so sánh và giá trị TOP luôn được cố định trước khi bắt đầu chu kỳ mới, từ đó tạo xung đầu ra luôn đối xứng và tần số chính xác. Công thức tính tần số PWM của chế độ này cũng giống như ở chế độ Phase Correct PWM.

10.2.3. Timer/Counter2

❖ Chế độ Fast PWM

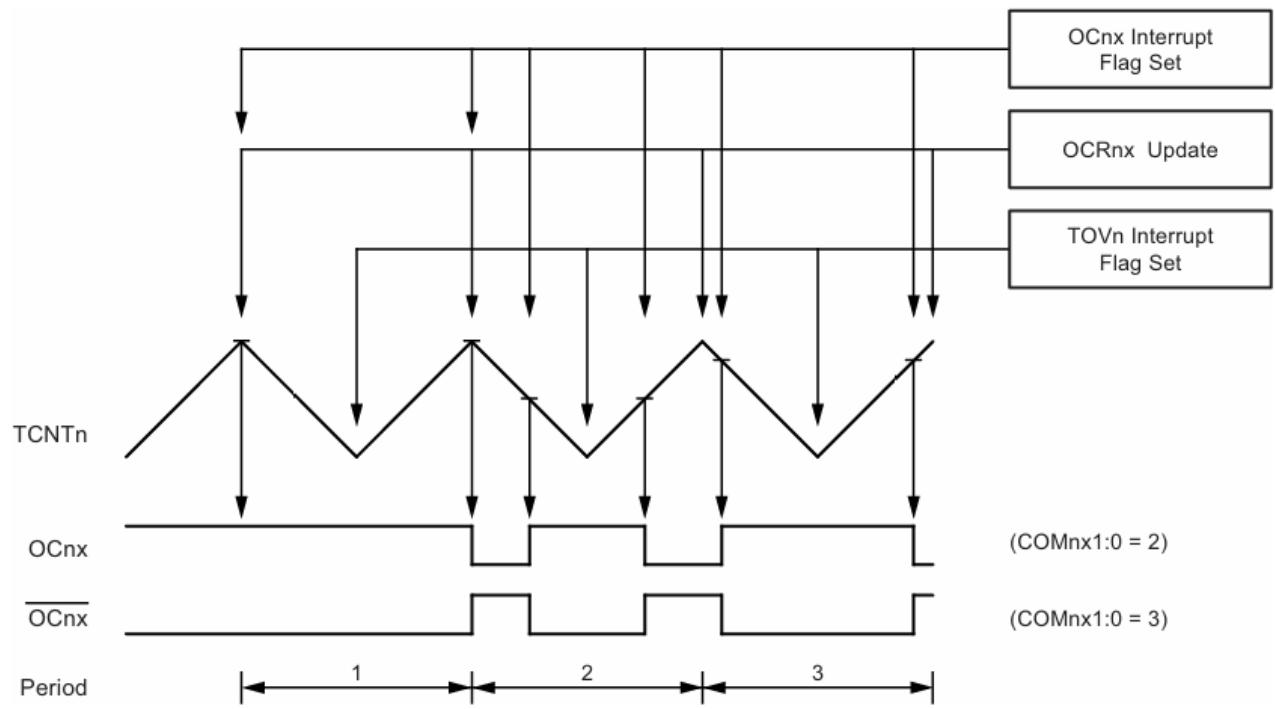
Là một chế độ tạo xung PWM với tần số cao, tương tự như Timer/Counter0, có hai mode có thể sử dụng trong chế độ này khi thiết lập WGM22:0= 3 hoặc 7. Giá trị TOP ở mode 3 là giá trị cố định 0xFF, ở mode 7 thì giá trị này sẽ bằng với OCR2A. Tuy nhiên, Timer/Counter2 có thể sử dụng thêm nguồn thạch anh ngoài để tạo tần số độc lập và bộ chia tần số (Prescaler) có thêm lựa chọn 32 và 128.



Hình 10.9. Biểu đồ thời gian của chế độ Fast PWM – Timer/Counter2 [2]

❖ Chế độ Phase Correct PWM

Nguyên lý hoạt động của chế độ Phase Correct PWM tương tự như Timer/Counter0, giá trị TOP có thể là 0xFF hoặc OCR2A tùy theo cấu hình các bit WGM22:0= 1 hoặc 5. Tần số PWM thấp hơn so với chế độ Fast PWM do bộ đếm phải thực hiện cả hai quá trình tăng và giảm trong mỗi chu kỳ.



Hình 10.10. Biểu đồ thời gian của chế độ Phase Correct PWM – Timer/Counter2 [2]

10.3. Các thanh ghi chức năng PWM

Các thanh ghi sử dụng cho chế độ Normal hay chế độ CTC cũng được sử dụng để thiết lập cho tất cả các chế độ làm việc PWM của bộ Timer/Counter 0, 1 và 2. Tuy nhiên, tùy thuộc vào cách cấu hình các bit trong từng thanh ghi sẽ tạo xung PWM khác nhau.

10.3.1. Timer/Counter0

❖ Thanh ghi điều khiển TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7:6 – COM0A[1:0]: Điều khiển trạng thái ngõ ra của chân OC0A, tùy thuộc vào cách thiết lập các bit WGM0[2:0].

Bảng 10.1. Ngõ ra chân OC0A với chế độ Fast PWM – Timer/Counter0 [2]

COM0A1	COM0A0	Mô tả
0	0	Như cổng thông thường, OC0A ngắt kết nối.
0	1	Khi WGM02 = 0: Như cổng thông thường, OC0A ngắt kết nối. Khi WGM02 = 1: Đảo trạng thái OC0A khi có so sánh.
1	0	Xóa OC0A khi có so sánh và thiết lập OC0A tại BOTTOM.
1	1	Thiết lập OC0A khi có so sánh và xóa OC0A tại BOTTOM.

Bảng 10.2. Ngõ ra chân OC0A với chế độ Phase Correct PWM – Timer/Counter0 [2]

COM0A1	COM0A0	Mô tả
0	0	Như cổng thông thường, OC0A ngắt kết nối.
0	1	Khi WGM02 = 0: Như cổng thông thường, OC0A ngắt kết nối. Khi WGM02 = 1: Đảo trạng thái OC0A khi có so sánh.
1	0	Xóa OC0A khi đang đếm lên và xảy ra so sánh. Thiết lập OC0A khi đang đếm xuống và xảy ra so sánh.
1	1	Thiết lập OC0A khi đang đếm lên và xảy ra so sánh. Xóa OC0A khi đang đếm xuống và xảy ra so sánh.

Bit 5:4 (COM0B[1:0]): Điều khiển trạng thái ngõ ra của chân OC0B, tùy thuộc vào cách thiết lập chế độ sử dụng thông qua các bit WGM0[2:0].

Bảng 10.3. Ngõ ra chân OC0B với chế độ Fast PWM – Timer/Counter0 [2]

COM0B1	COM0B0	Mô tả
0	0	Hoạt động cổng thông thường, OC0B ngắt kết nối.
0	1	Không sử dụng
1	0	Xóa OC0B khi có so sánh và thiết lập OC0B tại BOTTOM.
1	1	Thiết lập OC0B khi có so sánh và xóa OC0B tại BOTTOM.

Bảng 10.4. Ngõ ra chân OC0B với chế độ Phase Correct PWM – Timer/Counter0 [2]

COM0B1	COM0B0	Mô tả
0	0	Như cổng thông thường, OC0A ngắt kết nối.
0	1	Không sử dụng
1	0	Xóa OC0B khi đang đếm lên và xảy ra so sánh. Thiết lập OC0B khi đang đếm xuống và xảy ra so sánh.
1	1	Thiết lập OC0B khi đang đếm lên và xảy ra so sánh. Xóa OC0B khi đang đếm xuống và xảy ra so sánh.

Bit 1:0 (WGM0[1:0]): Kết hợp với bit WGM02 trong thanh ghi TCCR0B để tạo ra các chế độ làm việc của Timer/Counter0.

Bảng 10.5. Mô tả cấu hình các chế độ làm việc PWM – Timer/Counter0 [2]

Mode	WGM02	WGM01	WGM00	Chế độ hoạt động	TOP	Cập nhật OCRx	Cờ tràn TOV
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

❖ Thanh ghi điều khiển TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	TCCR0B
0x25 (0x45)	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00	
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Trong các chế độ làm việc PWM, thanh ghi TCCR0B giúp cấu hình bộ chia tần số xung nhịp và quyết định các mode sẽ được sử dụng. Cụ thể, ba bit CS0[2:0] quyết định tốc độ bộ đếm TCNT0, chọn nguồn xung nhịp của hệ thống hay xung ngoài (Tham khảo Bảng 9.7) và bit WGM02 kết hợp với hai bit WGM0[1:0] trong thanh ghi TCCR0A để chọn ra mode tạo xung PWM phù hợp (Tham khảo Bảng 10.5).

10.3.2. Timer/Counter1

❖ Thanh ghi TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10	TCCR1A
Read/ Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Thanh ghi TCCR1A có liên hệ chặt chẽ với các chế độ làm việc PWM, giúp kiểm soát và điều chỉnh các tín hiệu xung đầu ra, điều khiển các thiết bị như động cơ, servo. Các bit COM1A[1:0] và COM1B[1:0] dùng để thiết lập các dạng xung ngõ ra của chân OC1A/OC1B và hai bit WGM1[1:0] kết hợp với các bit WGM của thanh ghi TCCR1B giúp xác định các chế độ làm việc.

Bảng dưới đây thể hiện cách cấu hình các bit trong thanh ghi TCCR1A và mô tả chi tiết xung ngõ ra, chế độ làm việc PWM của bộ Timer/Counter1.

Bảng 10.6. Ngõ ra chân OC1A/OC1B với chế độ Fast PWM – Timer/Counter1 [2]

COM1A1/ COM1B1	COM1A0/ COM1B0	Mô tả
0	0	Như cổng thông thường, OC1A/OC1B ngắt kết nối.
0	1	Nếu WGM1[3:0] = 14 hoặc 15: Đảo trạng thái OC1A khi xảy ra so sánh, OC1B ngắt kết nối. Các trường hợp còn lại: Như cổng thông thường, OC1A/OC1B ngắt kết nối.

1	0	Xóa OC1A/OC1B khi xảy ra so sánh. Thiết lập OC1A/OC1B tại BOTTOM.
1	1	Thiết lập OC1A/OC1B khi xảy ra so sánh. Xóa OC1A/OC1B tại BOTTOM.

Bảng 10.7. Ngõ ra chân OC1A/OC1B với chế độ Phase Correct và Phase and Frequency Correct PWM – Timer/Counter1 [2]

COM1A1/ COM1B1	COM1A0/ COM1B0	Mô tả
0	0	Như cổng thông thường, OC1A/OC1B ngắt kết nối.
0	1	Nếu WGM1[3:0] = 9 hoặc 11: Đảo trạng thái OC1A khi xảy ra so sánh, OC1B ngắt kết nối. Các trường hợp còn lại: Như cổng thông thường, OC1A/OC1B ngắt kết nối.
1	0	Xóa OC1A/OC1B khi xảy ra so sánh. Thiết lập OC1A/OC1B tại BOTTOM.
1	1	Thiết lập OC1A/OC1B khi xảy ra so sánh. Xóa OC1A/OC1B tại BOTTOM.

Bảng 10.8. Mô tả cấu hình chế độ làm việc PWM của Timer/Counter1 [2]

Mode	WGM13	WGM12	WGM12	WGM11	Chế độ hoạt động	TOP	Cập nhật OCR1x	Cờ TOV1
1	0	0	0	1	Phase Correct PWM, 8 bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	Phase Correct PWM, 9 bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	Phase Correct PWM, 10 bit	0x03FF	TOP	BOTTOM
5	0	1	0	1	Fast PWM, 8 bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9 bit	0x01FF	BOTTOM	TOP

7	0	1	1	1	Fast PWM, 10 bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	Phase Correct PWM	ICR1	TOP	BOTTOM
11	1	0	1	1	Phase Correct PWM	OCR1A	TOP	BOTTOM
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

❖ Thanh ghi TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/ Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Nhóm các bit CS1[2:0] xác định nguồn xung clock và bộ chia tần số (Prescaler) cho các chế độ làm việc PWM, cho phép tần số đầu vào từ nguồn xung hệ thống hoặc xung ngoài (Tham khảo Bảng 9.10). Lựa chọn các chế độ làm việc PWM thông qua cấu hình các bit WGM1[3:0] (Tham khảo Bảng 10.8). Hai bit ICNC1 và ICES1 không ảnh hưởng đến hoạt động của các chế độ PWM.

10.3.3. Timer/Counter2

❖ Thanh ghi điều khiển TCCR2A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	-	-	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Thanh ghi TCCR2A chứa các bit có chức năng điều khiển dạng tín hiệu của ngõ ra OC2A/OC2B và lựa chọn các mode làm việc tương tự như ở bộ Timer/Counter0. Bảng dưới đây mô tả chi tiết cách cấu hình các bit COM2A[1:0], COM2B[1:0] và WGM2[2:0] (bao gồm bit WGM22 trong thanh ghi TCCR2B).

Bảng 10.9. Ngõ ra chân OC2A với chế độ Fast PWM – Timer/Counter2 [2]

COM2A1	COM2A0	Mô tả
0	0	Như cổng thông thường, OC2A ngắt kết nối.
0	1	Khi WGM22 = 0: Như cổng thông thường, OC2A ngắt kết nối. Khi WGM22 = 1: Đảo trạng thái OC2A khi có so sánh.
1	0	Xóa OC2A khi có so sánh và thiết lập OC2A tại BOTTOM.
1	1	Thiết lập OC2A khi có so sánh và xóa OC2A tại BOTTOM.

Bảng 10.10. Ngõ ra chân OC2A với chế độ Phase Correct PWM–Timer/Counter2 [2]

COM2A1	COM2A0	Mô tả
0	0	Như cổng thông thường, OC2A ngắt kết nối.
0	1	Khi WGM22 = 0: Như cổng thông thường, OC2A ngắt kết nối. Khi WGM22 = 1: Đảo trạng thái OC2A khi có so sánh.
1	0	Xóa OC2A khi đang đếm lên và xảy ra so sánh. Thiết lập OC2A khi đang đếm xuống và xảy ra so sánh.
1	1	Thiết lập OC2A khi đang đếm lên và xảy ra so sánh. Xóa OC2A khi đang đếm xuống và xảy ra so sánh.

Bảng 10.11. Ngõ ra chân OC2B với chế độ Fast PWM – Timer/Counter2 [2]

COM2B1	COM2B0	Mô tả
0	0	Hoạt động cổng thông thường, OC2B ngắt kết nối.
0	1	Không sử dụng
1	0	Xóa OC2B khi có so sánh và thiết lập OC2B tại BOTTOM.
1	1	Thiết lập OC2B khi có so sánh và xóa OC2B tại BOTTOM.

Bảng 10.12. Ngõ ra chân OC2B với chế độ Phase Correct PWM–Timer/Counter0 [2]

COM2B1	COM2B0	Mô tả
0	0	Như cổng thông thường, OC2A ngắt kết nối.
0	1	Không sử dụng
1	0	Xóa OC2B khi đang đếm lên và xảy ra so sánh. Thiết lập OC0B khi đang đếm xuống và xảy ra so sánh.
1	1	Thiết lập OC0B khi đang đếm lên và xảy ra so sánh. Xóa OC0B khi đang đếm xuống và xảy ra so sánh.

Bảng 10.13. Mô tả cấu hình các chế độ làm việc PWM – Timer/Counter2 [2]

Mode	WGM22	WGM21	WGM20	Chế độ hoạt động	TOP	Cập nhật OCRx	Cờ tràn TOV
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

❖ Thanh ghi điều khiển TCCR2B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20	TCCR2B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Là thanh ghi cấu hình chế độ làm việc và lựa chọn bộ chia tần số, quyết định tốc độ hoạt động của Timer/Counter2. Cần có sự kết hợp giữa bit WGM22 và hai bit WGM2[1:0] trong thanh ghi TCCR2A để chọn chế độ làm việc PWM (Tham khảo Bảng 10.13). Tham khảo Bảng 9.13 để xem cách cấu hình các bit CS2[2:0] và chi tiết các giá trị bộ chia cho bộ Timer/Counter2.

10.4. Bài thực hành

Yêu cầu bài thực hành: Điều chế độ rộng xung PWM để điều khiển góc quay của servo, ban đầu servo giữ ở góc 0° và góc quay của servo sẽ thay đổi khi nhấn nút. Loại sóng PWM được sử dụng là chế độ Fast PWM (mode 14) của bộ Timer 1.

Servo thường được điều khiển với tần số 50Hz tương ứng với chu kỳ là 20 ms, để điều khiển servo quay góc 0° thì thời gian xung mức cao phải chiếm 1 ms trên tổng chu kỳ (đây chính là Duty Cycle).

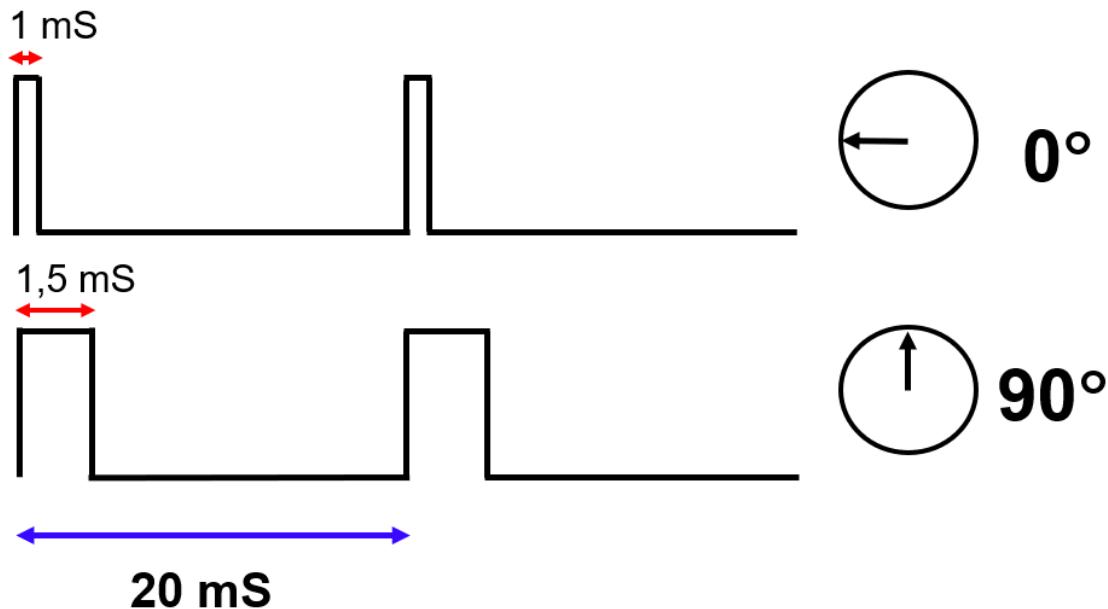
Trong bài thực hành này đang sử dụng chế độ Fast PWM và bộ chia 8 nên giá trị TOP (trong trường hợp này là thanh ghi ICR1) sẽ được tính toán dựa theo công thức (11) :

$$TOP = ICR1 = \frac{f_{clkI/O}}{f_{PWM} \cdot N} - 1 = \frac{8 \cdot 10^6}{50 \cdot 8} - 1 = 19\ 999$$

Giá trị bộ đếm OCR1A được tính toán dựa trên góc quay mong muốn:

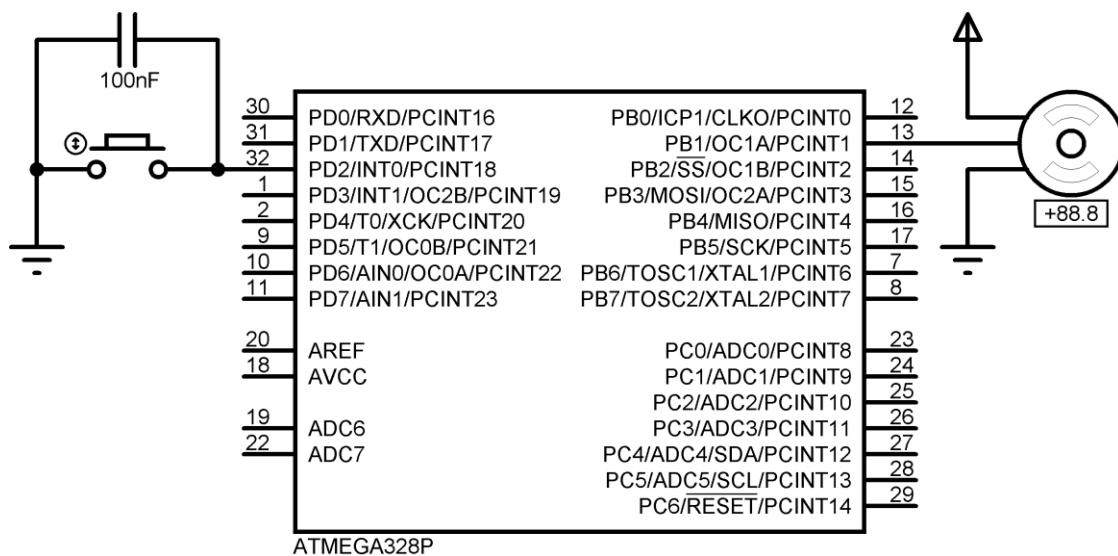
$$OCR1A = \frac{T_{pulse}}{T_{all}} \times ICR1 = \frac{1}{20} \times 19\ 999 \approx 999$$

Với: T_{pulse} là độ rộng xung mong muốn (1ms), T_{all} là tổng chu kỳ (20ms).



Hình 10.11. Độ rộng xung điều khiển servo

❖ Phản mô phỏng



Hình 10.12. Sơ đồ kết nối bài thực hành xung PWM

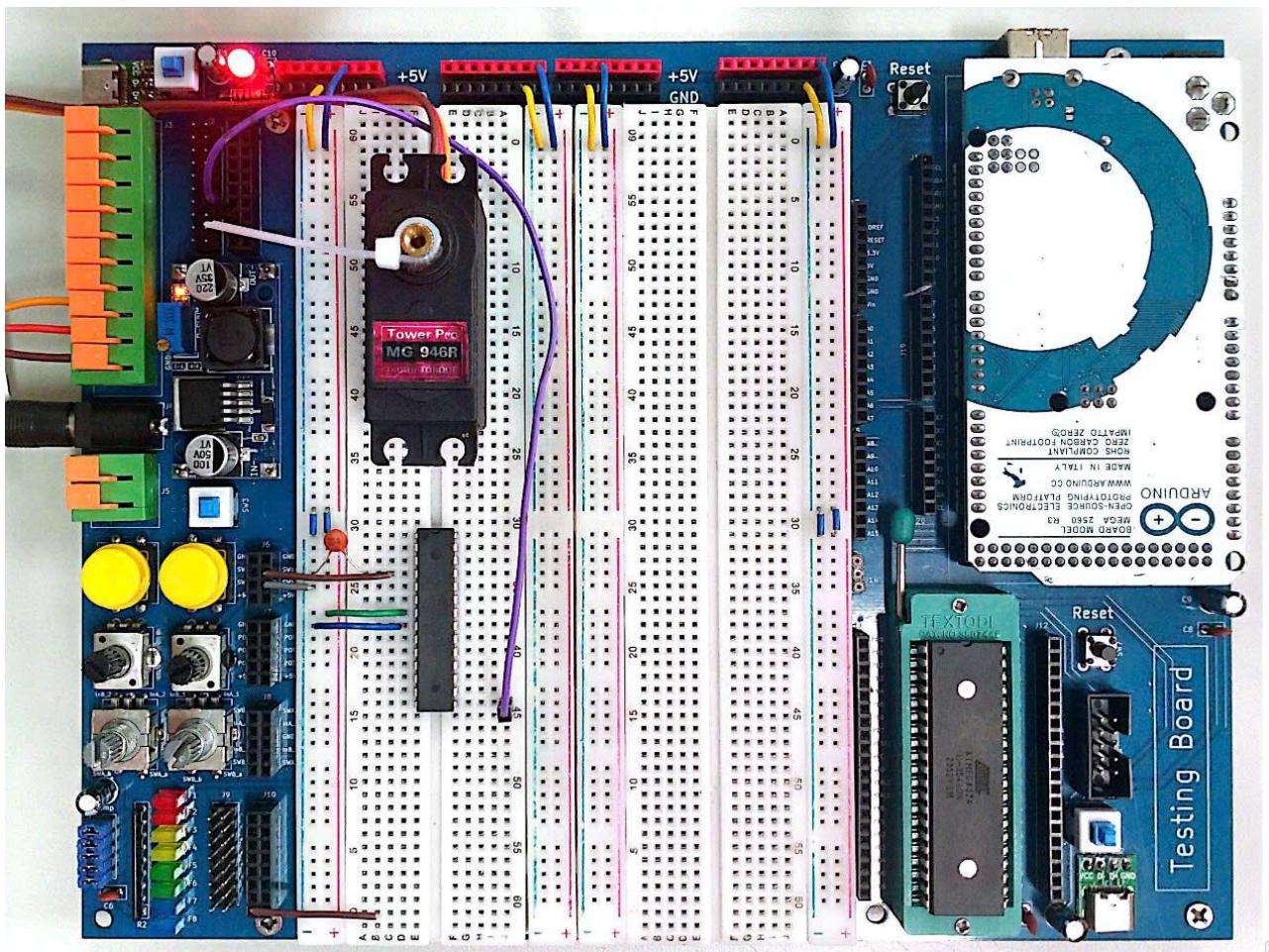
❖ Phần lập trình

	SourceCode_ATMEGA328P\P10_PWM
	<pre>1 #include <mega328p.h> 2 3 unsigned long int TOP = 19999; 4 unsigned long int STA_VAL = 999; //gia tri ban dau cua bo dem 5 interrupt [EXT_INT0] void ext_int0_isr(void) 6 { 7 // neu la 0 do thi keo len 90 do va nguoc lai 8 if(((OCR1AH << 8) OCR1AL)==999){ 9 OCR1AH = (1999 >> 8); 10 OCR1AL = 1999 & 0xFF; 11 } 12 else{ 13 OCR1AH = (999 >> 8); 14 OCR1AL = 999 & 0xFF; 15 } 16} 17 void main(void) 18 { 19 #pragma optsize- 20 CLKPR = (1<<CLKPCE); 21 CLKPR = 0x00; 22 #ifdef _OPTIMIZE_SIZE_ 23 #pragma optsize+ 24 #endif 25 DDRB.1 = 1; //ngo ra chan OC1A, dieu khien servo 26 PORTD.2 = 1; //chan PD2 la ngo vao, co dien tro keo len 27 TCCR1A=(1<<COM1A1) (1<<WGM11); 28 TCCR1B=(1<<WGM13) (1<<WGM12) (1<<CS11); 29 // do ICR1 la thanh ghi 16bit nen phai chia ra de luu tru 30 ICR1H = (TOP >> 8); 31 ICR1L = TOP & 0xFF; 32 // gia tri ban dau - 0 do 33 OCR1AH = (STA_VAL >> 8);</pre>

```

34     OCR1AL = STA_VAL & 0xFF;
35     //ngat 0, lay canh xuong
36     EICRA |= (1<<ISC01);
37     EIMSK |= (1<<INT0);
38     #asm("sei")
39     while (1){}
40 }
```

❖ Phản mô hình



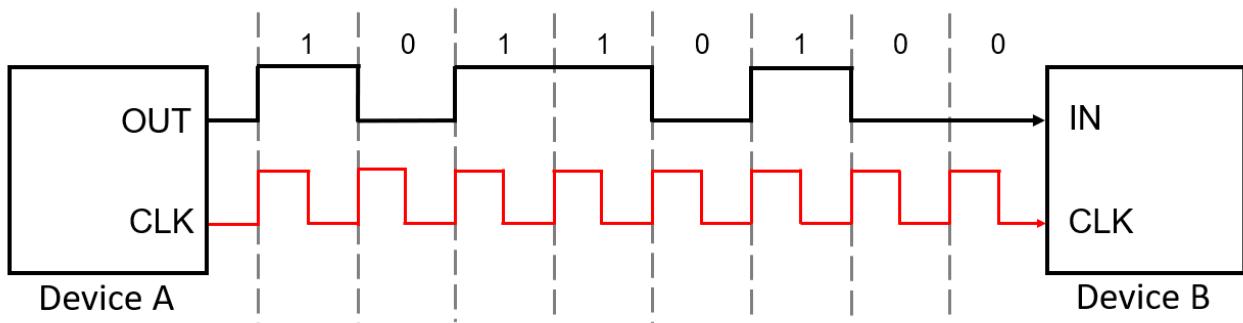
Hình 10.13. Mô hình thực tế bài thực hành xung PWM

CHƯƠNG 11. THIẾT KẾ GIAO TIẾP

11.1. Giao thức truyền thông nối tiếp

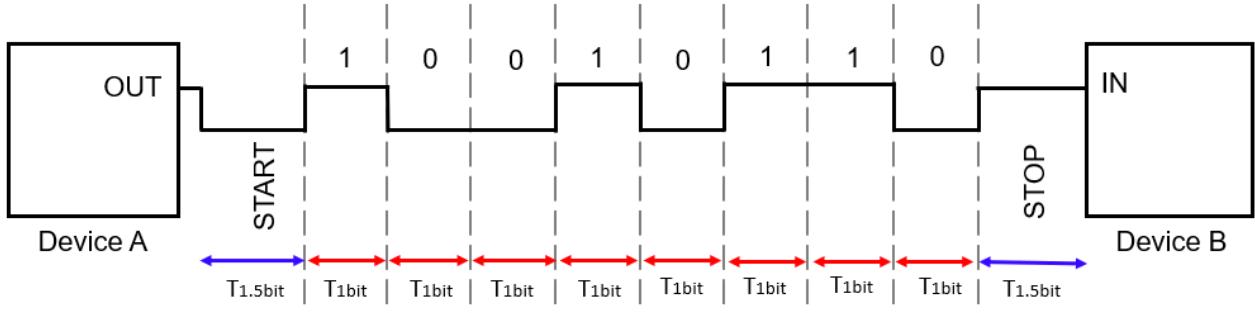
11.1.1. Tổng quan

Là một phương pháp giao tiếp, trao đổi dữ liệu giữa máy tính hoặc vi điều khiển và thiết bị ngoại vi, tuân thủ các nguyên tắc bảo mật và độ tin cậy cao. Giao thức cho phép cấu hình khung truyền với 5, 6, 7, 8 hoặc 9 bit dữ liệu và 1 hoặc 2 bit stop, đồng thời hỗ trợ kiểm tra và tạo bit chẵn lẻ (odd hoặc even parity). Các tính năng khác bao gồm: phát hiện tràn dữ liệu, lỗi khung, lọc nhiễu. Giao thức truyền thông nối tiếp bao gồm hai loại: USART – nối tiếp đồng bộ và UART – nối tiếp bất đồng bộ.



Hình 11.1. Truyền thông nối tiếp đồng bộ - USART

Hình 11.1 thể hiện cách truyền một byte dữ liệu (0b10110100) từ thiết bị Master (Device A) đến thiết bị Slave (Device B). Trong quá trình truyền dữ liệu, Device A gửi thêm một xung clock để giữ nhịp, giúp Device B nhận biết thời điểm bắt đầu giao tiếp, thời gian lấy mẫu (mỗi chu kỳ của xung clock), đồng bộ hóa tốc độ truyền và nhận của hai thiết bị nhằm tránh sai sót trong quá trình truyền thông.



Hình 11.2. Truyền thông nối tiếp bất đồng bộ - UART

Hình 11.2 thể hiện phương thức truyền dữ liệu ở chế độ bất đồng bộ, ban đầu đường bus dữ liệu sẽ ở trạng thái lý tưởng (mức cao), bộ truyền bắt đầu một START bit bằng cách kéo đường bus xuống mức thấp trong chu kỳ 1.5 bit, tiếp theo là 8 bit dữ liệu và cuối cùng là bit parity (kiểm tra lỗi của khung truyền), đường bus sẽ được trả về trạng thái lý tưởng sau khi quá trình truyền thông kết thúc. Khác với chế độ truyền đồng bộ, UART sử dụng tốc độ Baud để đồng bộ thời điểm lấy mẫu giữa Master và Slave.

11.1.2. Các thành phần quan trọng của UART

❖ Tốc độ Baud

Là thông số quan trọng khi sử dụng giao thức truyền thông nối tiếp ở chế độ bất đồng bộ, tốc độ Baud quy định số bit dữ liệu truyền qua mỗi giây, khi giá trị này tăng thì quá trình truyền – nhận dữ liệu sẽ giảm đi. Tốc độ Baud sẽ được tính toán dựa trên giá trị của thanh ghi UBRR_n và tùy theo cách thiết lập tốc độ cao hay thấp.

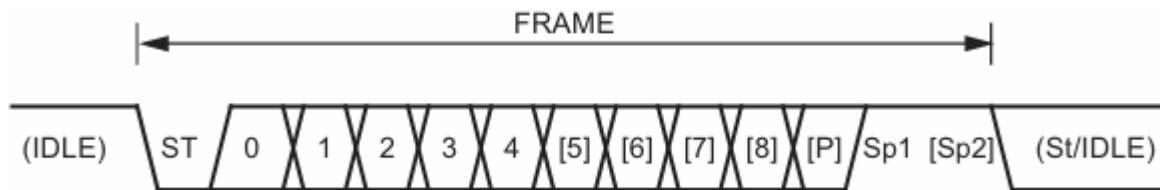
Bảng 11.1. Công thức tính tốc độ Baud [2]

Chế độ hoạt động	Công thức tính tốc độ Baud
Asynchronous normal (U2Xn = 0)	$BAUD = \frac{f_{osc}}{16(UBRR_n + 1)}$
Asynchronous double speed (U2Xn = 1)	$BAUD = \frac{f_{osc}}{8(UBRR_n + 1)}$

Synchronous master

$$BAUD = \frac{f_{osc}}{2(UBRR_n + 1)}$$

❖ Định dạng khung dữ liệu



Hình 11.3. Cấu tạo khung dữ liệu của giao thức UART [2]

Một khung dữ liệu bao gồm 1 bit Start đánh dấu quá trình truyền dữ liệu sẽ bắt đầu; tiếp theo là các bit dữ liệu chính, đây là các bit chứa thông tin của Master cần truyền đến Slave, có thể chứa 5, 6, 7, 8 hoặc 9 bit. Sau đó là Parity bit, đây là bit kiểm tra tính bảo toàn của dữ liệu, phát hiện sai sót khi truyền và có thể bỏ qua bit này đối với các ứng dụng đơn giản. Cuối cùng là một hoặc hai bit Stop để kết thúc.

❖ Parity bit

Là một cơ chế phát hiện lỗi đơn giản, có thể cấu hình là Even Parity hoặc Odd Parity. Nếu sử dụng cấu hình Even Parity mà tổng số lượng bit 1 trong gói dữ liệu là số chẵn thì Parity bit được đặt là 0, ngược lại bit này là 1. Nếu sử dụng cấu hình Odd Parity mà tổng số lượng bit 1 trong gói dữ liệu là số lẻ thì Parity bit được đặt là 0 và ngược lại đặt là 1. Tuy nhiên, phương pháp kiểm tra bit chẵn lẻ không thể phát hiện lỗi nếu có nhiều hơn một bit bị thay đổi.

11.1.3. Các thanh ghi của USART

❖ Thanh ghi dữ liệu UDRn

Bit	7	6	5	4	3	2	1	0	
	RXB [7:0]								UDRn (Read)
	TXB [7:0]								UDRn (Write)
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Thanh ghi bộ đếm dữ liệu truyền và nhận cùng chia sẻ một địa chỉ, khi ghi dữ liệu vào địa chỉ thanh ghi UDRn, dữ liệu sẽ được lưu vào bộ đếm dữ liệu truyền TXB và ngược lại đọc dữ liệu của UDRn qua bộ đếm RXB. Đối với khung dữ liệu sử dụng 5, 6 hoặc 7 bit thì các bit cao của thanh ghi UDRn không được sử dụng.

❖ Thanh ghi UCSRnA – USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	Fen	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – RXCn: Là bit cho biết quá trình nhận đã hoàn tất và sẵn sàng đọc dữ liệu từ thanh ghi UDRn, bit này được ghi với mức logic 1 khi có dữ liệu mới được nhận và được xóa về 0 khi không có dữ liệu trong bộ đếm nhận.

Bit 6 – TXCn: Là bit cho biết quá trình truyền dữ liệu đã hoàn tất, thanh ghi UDRn đã trống, khi đó TXCn được cấu hình với mức logic 1 và ngược lại nếu quá trình truyền dữ liệu đang diễn ra thì TXCn được cấu hình với mức logic 0.

Bit 5 – UDREn: Khi thanh ghi dữ liệu UDRn đang trống thì bit này được thiết lập bằng 1. Trước khi thực hiện quá trình truyền dữ liệu nên kiểm tra xem bit UDREn có bằng 1 hay không để tránh sai sót.

Bit 4 – Fen: Khi bit này được thiết lập bằng 1 đồng nghĩa với việc có lỗi xảy ra trong quá trình truyền thông.

Bit 3 – DORn: Bit này được thiết lập bằng 1 khi xảy ra quá tải, bộ đệm nhận đã đầy trong khi dữ liệu mới tiếp tục được nhận.

Bit 2 – UPEn: Là bit thông báo xảy ra lỗi Parity trong quá trình nhận và quá trình kiểm tra sẽ được kích hoạt nếu bit UPMn1 = 1 (Thanh ghi UCSRnC).

Bit 1 – U2Xn: Khi bit này được ghi bằng 1, bộ chia của tốc độ Baud sẽ giảm từ 16 về 8, đồng nghĩa với tốc độ truyền dữ liệu sẽ được tăng gấp đôi. Tốc độ này chỉ có thể hoạt động với chế độ bất đồng bộ, đối với chế độ đồng bộ thì bit này được gán bằng 0.

Bit 0 – MPCMn: Kích hoạt chế độ Multi-processor Communication (truyền thông đa xử lý). Mục đích để giảm tải xử lý trong các ứng dụng có nhiều hơn một thiết bị được kết nối chung một đường bus.

❖ Thanh ghi UCSRnB – USART Control and Status Register B

Bit	7	6	5	4	3	2	1	0	UCSRnB
	RXCIEn	TXCIEn	UDRIEEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7:6 – RXCIEn và TXCIEn: Là hai bit kích hoạt ngắt khi hoàn thành quá trình nhận hoặc truyền dữ liệu.

Bit 5 – UDRIEEn: Kích hoạt ngắt khi thanh ghi dữ liệu UDRn đang trống.

Bit 4:3 – RXENn và TXENn: Kích hoạt bộ nhận và truyền dữ liệu.

Bit 2 – UCSZn2: Kết hợp với Bit 2:1 UCSZn[1:0] trong thanh ghi UCSRnC để thiết lập kích thước dữ liệu của bộ truyền – nhận.

Bit 1:0 – RXB8n và TXB8n: Là bit thứ 9 của thanh ghi dữ liệu, do giao thức truyền thông nối tiếp có thể truyền – nhận với tối đa 9 bit dữ liệu nên 8 bit đầu (từ bit 0 đến bit 7) sẽ được chứa trong thanh ghi UDRn và bit RXB8n/ TXB8n sẽ chứa dữ liệu cuối cùng.

❖ Thanh ghi UCSRnC – USART Control and Status Register C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7:6 – UMSELn[1:0]: Là hai bit thiết lập chế độ hoạt động của giao thức truyền thông nối tiếp.

Bảng 11.2. Cấu hình các chế độ hoạt động của UART [2]

UMSELn1	UMSELn0	Mode
0	0	Bất đồng bộ
0	1	Đồng bộ
1	0	-
1	1	Master SPI

Bit 5:4 – UPMn[1:0]: Giúp kích hoạt và chọn chế độ kiểm tra Parity bit. Nếu được kích hoạt, bộ truyền sẽ tự động tạo và gửi Parity bit này trong mỗi khung dữ liệu. Bộ nhận sẽ tạo mỗi giá trị Parity cho dữ liệu nhận được và so sánh với cấu hình UPMn, nếu phát hiện sự không trùng khớp thì cờ UPEn trong thanh ghi UCSRnA sẽ được thiết lập bằng 1.

Bảng 11.3. Cấu hình sử dụng Parity Bit [2]

UPMn1	UPMn0	Parity Mode
0	0	Không kích hoạt
1	0	Parity chẵn
1	1	Parity lẻ

Bit 3 – USBSn: Lựa chọn số lượng bit dừng của bộ truyền, bộ nhận không cần phải thiết lập bit này.

Bảng 11.4. Lựa chọn độ dài Stop Bit [2]

USBSn	Stop Bit
0	1 bit
1	2 bit

Bit 2:1 – UCSZn[1:0]: Kết hợp với bit UCSZn2 để lựa chọn kích thước của dữ liệu mà bộ truyền và bộ nhận sử dụng.

Bảng 11.5. Cấu hình lựa chọn độ dài dữ liệu của bộ truyền [2]

UCSZn2	UCSZn1	UCSZn0	Kích thước
0	0	0	5 bit
0	0	1	6 bit
0	1	0	7 bit
0	1	1	8 bit
1	1	1	9 bit

Bit 0 – UCPOLn: Trong chế độ USART (đồng bộ), bit này giúp xác định cạnh xung nhịp sẽ được sử dụng để lấy mẫu, mục đích đồng bộ hóa dữ liệu giữa thiết bị truyền và nhận.

Bảng 11.6. Cấu hình bit UCPOLn [2]

UCPOLn	Thiết bị truyền	Thiết bị nhận
0	Cạnh lên	Cạnh xuống
1	Cạnh xuống	Cạnh lên

❖ Thanh ghi UBRRnL và UBRRnH – Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	-	-	-	-	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Là thanh ghi 12 bit chứa tốc độ Baud của giao thức truyền thông nối tiếp. UBRRnH chứa bốn bit có trọng số cao và UBRRnL chứa tám bit trọng số thấp, khi ghi giá trị vào UBRRnL thì lập tức giá trị tốc độ Baud sẽ bị thay đổi vì thế cần ghi giá trị vào thanh ghi UBRRnH trước. Lưu ý giá trị của thanh ghi UBRR chỉ gián tiếp xác định tốc độ Baud, công thức tính tốc độ Baud được xác định theo công thức ở Bảng 11.1.

Bảng 11.7. Cấu hình thanh ghi UBRR và tốc độ Baud mẫu

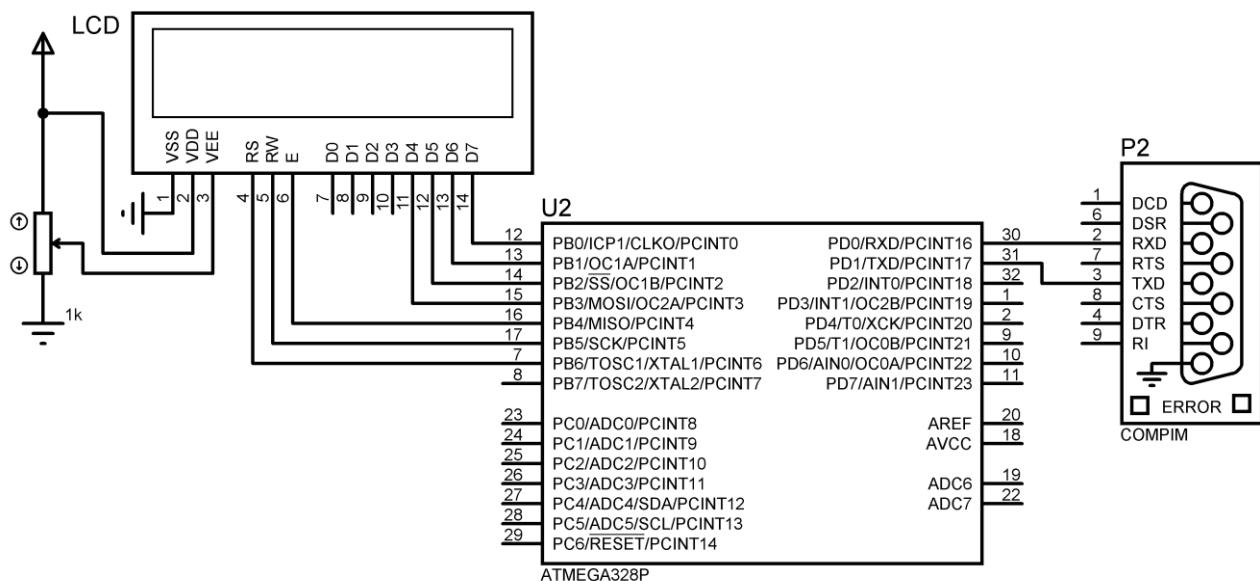
Baud Rate (bps)	$f_{osc} = 8 \text{ MHz}$				
	U2Xn = 0		U2Xn = 1		
	UBRRn	Error (%)	UBRRn	Error (%)	
2400	207	0.2	416	-0.1	
4800	103	0.2	207	0.2	
9600	51	0.2	103	0.2	
14.4K	34	-0.8	68	0.6	
19.2K	25	0.2	51	0.2	
28.8K	16	2.1	34	-0.8	
38.4K	12	0.2	25	0.2	
57.6K	8	-3.5	16	2.1	
76.8K	6	-7.0	12	0.2	

115.2K	3	8.5	8	-3.5
230.4K	1	8.5	3	8.5
250K	1	0.0	3	0.0
0.5M	0	0.0	1	0.0
1M	-	-	0	0.0
Max	0.5Mbps		1Mbps	

11.1.4. Bài thực hành

Yêu cầu bài thực hành: Giao tiếp giữa máy tính với vi điều khiển thông qua giao thức UART, gửi từng ký tự thông qua phần mềm “Hercules” đến module giao tiếp đến vi điều khiển và hiển thị lên màn hình LCD. Trong trường hợp mô phỏng, cần phải thông qua phần mềm “VSPE” để tạo cổng COM ảo mới có thể kết nối với phần mềm “Hercules”.

❖ Phần mô phỏng



Hình 11.4. Sơ đồ kết nối bài thực hành giao thức UART

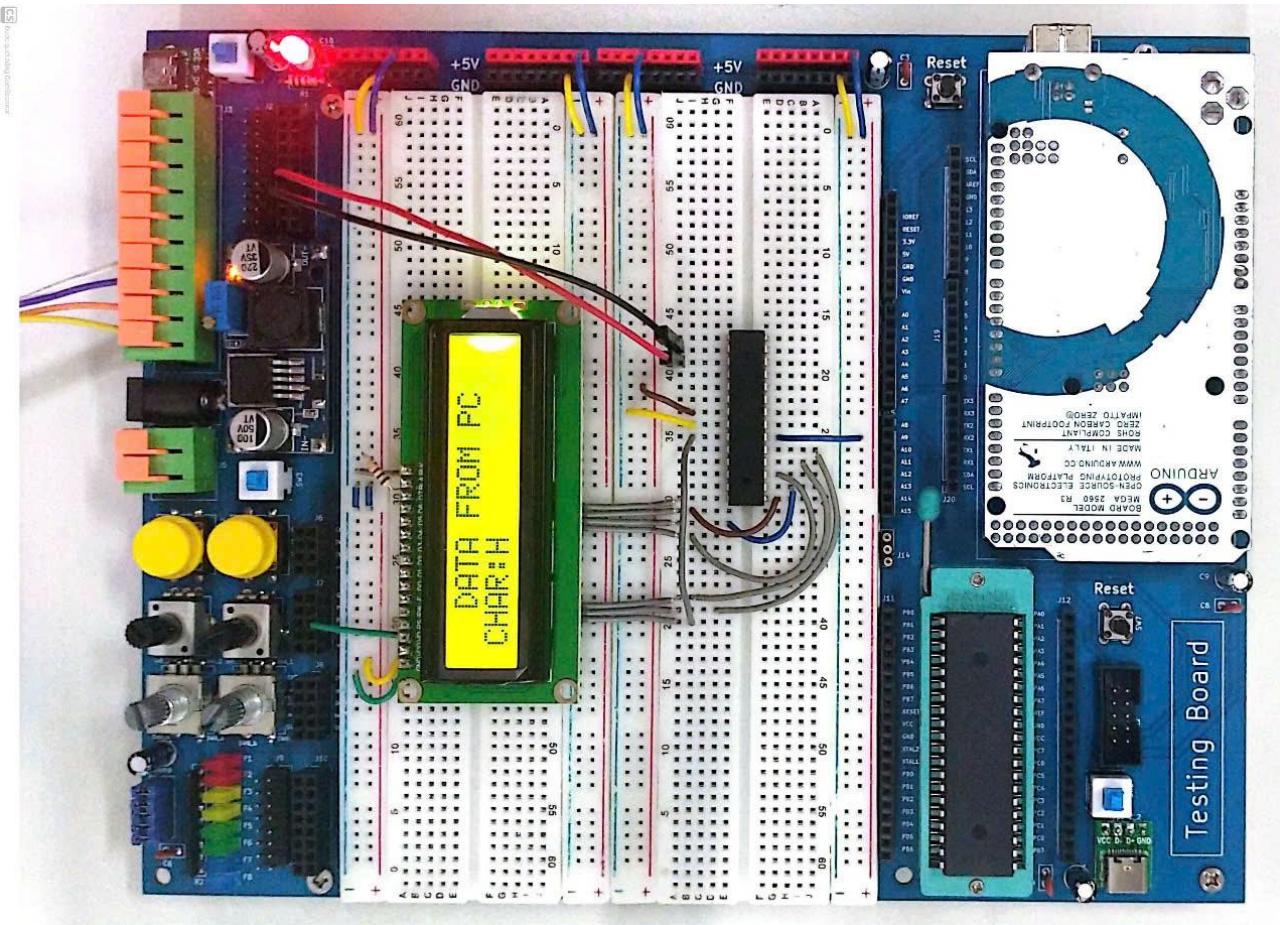
❖ Phân lập trình

	SourceCode_ATMEGA328P\P11_1_UART_Protocol
	<pre>1 #include <mega328p.h> 2 #include <alcd.h> 3 volatile char receivedData; 4 interrupt [USART_RXC] void USART_RX_Complete(void){ 5 //Lay du lieu tu thanh ghi UDR0 6 receivedData = UDR0; 7 lcd_gotoxy(5,1); 8 lcd_putchar(receivedData); 9 } 10 void main(void) 11 { 12 #pragma optsize- 13 CLKPR = (1<<CLKPCE); 14 CLKPR = 0x00; 15 #ifdef _OPTIMIZE_SIZE_ 16 #pragma optsize+ 17 #endif 18 19 //kich hoat nhan du lieu 20 //cho phep ngat khi nhan 21 UCSR0B = (1<<RXEN0) (1<<RXCIE0); 22 //cau hinh du lieu: 8 bit 23 UCSR0C = (1<<UCSZ01) (1<<UCSZ00); 24 UCSR0A = 0x00; 25 26 //chon toc do Baud Rate, 9600bps 27 UBRR0H=0x00; 28 UBRR0L=0x33; 29 30 lcd_init(16); 31 lcd_clear(); 32 lcd_gotoxy(2,0); 33 lcd_putsf("DATA FROM PC");</pre>

```

34     lcd_gotoxy(0,1);
35     lcd_putsf("CHAR:");
36     #asm("sei")
37 while (1){}
38 }
```

❖ Phần mô hình

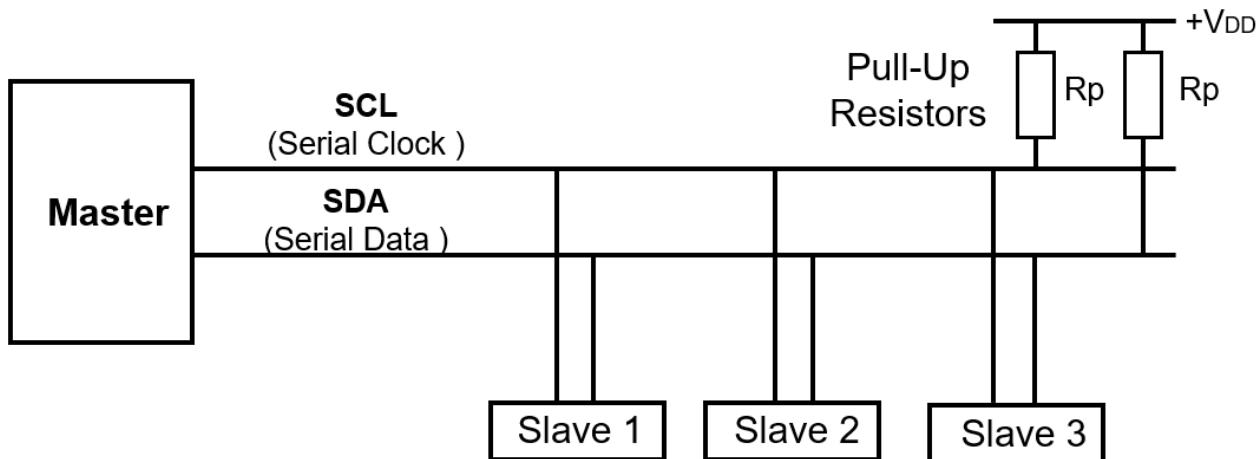


Hình 11.5. Mô hình thực tế bài thực hành UART

11.2. Giao tiếp I2C

11.2.1. Tổng quan về giao tiếp I2C

I2C (Inter- Integrated Circuit) là một giao thức giao tiếp nối tiếp đồng bộ được phát triển bởi Philips Semiconductors để truyền dữ liệu giữa một bộ xử lý trung tâm với nhiều IC mà chỉ sử dụng hai đường truyền tín hiệu.



Hình 11.6 Mạng TWI (I2C) với nhiều thiết bị

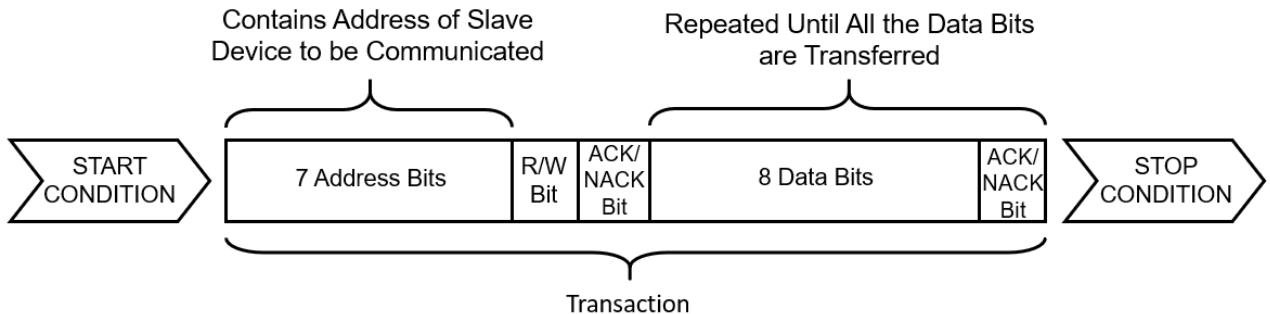
I2C chỉ sử dụng hai dây để truyền dữ liệu giữa các thiết bị:

- SDA (Serial Data): Đây là đường truyền dữ liệu nối tiếp, nơi tất cả thông tin về địa chỉ và dữ liệu được gửi đi lần lượt từng bit một. Trong giao thức I2C, bit có trọng số lớn nhất (MSB) sẽ được truyền trước, điều này khác với chuẩn giao tiếp UART.

- SCL (Serial Clock): là đường tạo xung nhịp nối tiếp. Giao thức TWI (I2C) là một chuẩn truyền thông nối tiếp đồng bộ, yêu cầu một đường xung nhịp để điều phối quá trình truyền và nhận dữ liệu. Mỗi xung nhịp trên SCL sẽ điều khiển việc lấy mẫu (sample) một bit dữ liệu trên SDA. Dữ liệu trên SDA được lấy mẫu khi SCL ở mức cao trong chu kỳ xung nhịp. Do đó, SDA không được phép thay đổi trạng thái khi SCL đang ở mức cao, ngoại trừ trong các điều kiện START và STOP. Trạng thái của SDA có thể được thay đổi khi SCL ở mức thấp.

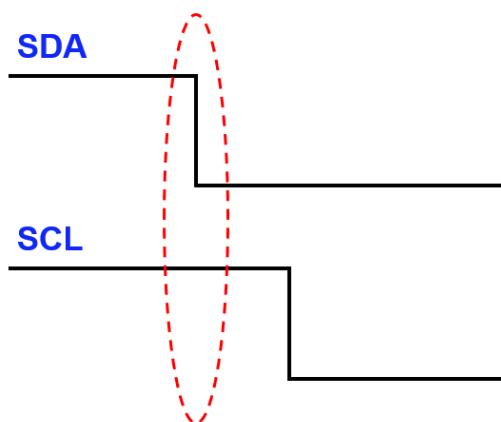
Các bit dữ liệu sẽ được truyền từng bit một theo một đường duy nhất (SDA) theo các khoảng thời gian đều đặn được thiết lập bởi 1 tín hiệu clock (SCL).

Dữ liệu được truyền giữa thiết bị Master và các thiết bị Slave thông qua một đường dữ liệu SDA duy nhất, các chuỗi có cấu trúc gồm các số 0 và 1 (bit). Mỗi chuỗi số 0 và 1 được gọi là giao dịch (transaction) và dữ liệu trong mỗi giao dịch có cấu trúc như sau:



Hình 11.7. Cấu tạo một khung truyền của giao thức I2C

- START Condition – Điều kiện bắt đầu: Trong trạng thái nghỉ, khi cả SDA và SCL đều ở mức cao, nếu Master muốn khởi tạo một giao tiếp, nó sẽ kéo chân SDA từ mức điện áp cao xuống mức điện áp thấp trước khi đường SCL chuyển từ cao xuống thấp. Khi điều kiện bắt đầu được gửi bởi thiết bị Master, tất cả các thiết bị Slave đều hoạt động ngay cả khi chúng ở chế độ ngủ (sleep mode) và sẽ sẵn sàng chờ nhận bit địa chỉ.



Hình 11.8. Start Condition

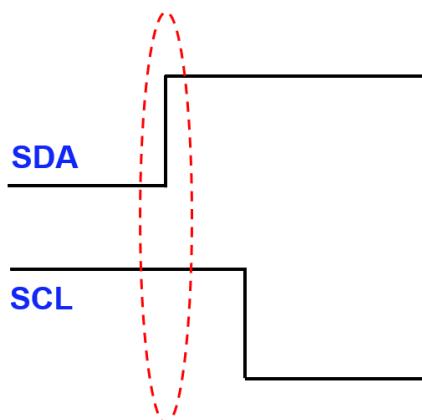
- Address Bits – Bits địa chỉ: Thường quá trình truyền nhận sẽ diễn ra với rất nhiều thiết bị Slave với nhau. Do đó để phân biệt các thiết bị Slave này, chúng sẽ được gắn 1 địa chỉ 7 bits cố định để xác định Slave khi Master muốn giao tiếp với nó.

- Read/Write Bit – Bit đọc/ghi: Bit này để xác định hướng truyền dữ liệu. Nếu thiết bị Master cần gửi dữ liệu đến thiết bị Slave, bit này được thiết lập là ‘0’ (mức điện áp thấp). Nếu Master cần nhận dữ liệu từ thiết bị Slave, bit này được thiết lập là ‘1’ (mức điện áp cao).

- ACK/NACK Bit – Bit xác nhận/không xác nhận: Mỗi khung trong một tin nhắn được theo sau bởi một bit xác nhận / không xác nhận. Nếu một khung địa chỉ hoặc khung dữ liệu được nhận thành công thì bit ACK sẽ được đặt bằng ‘0’ và ngược lại thì mặc định bằng ‘1’.

- Data Bits – Bits dữ liệu: Mỗi khung dữ liệu có độ dài 8 bit và được truyền với bit quan trọng nhất. Sau mỗi khung dữ liệu, sẽ có một bit ACK hoặc NACK để xác nhận rằng khung đã được nhận thành công. Bit ACK phải được Master hoặc Slave (tùy thuộc vào thiết bị đang truyền dữ liệu) nhận trước khi khung dữ liệu tiếp theo được gửi.

- STOP Condition - Điều kiện kết thúc: Khi quá trình truyền hoặc nhận dữ liệu đã hoàn tất và Master muốn kết thúc giao tiếp, nó sẽ tạo ra một điều kiện kết thúc (STOP condition). Điều này được thực hiện bằng cách kéo chân SDA lên mức cao trong khi chân SCL đang ở mức cao. Điều kiện kết thúc chỉ được tạo ra sau khi việc truyền hoặc nhận địa chỉ và dữ liệu đã hoàn thành.



Hình 11.9. Stop Condition

11.2.2. Cơ chế hoạt động

TWI trên AVR được thiết kế theo hướng byte (byte-oriented) và dựa trên ngắt (interrupt-based). Mỗi sự kiện xảy ra trong quá trình truyền hoặc nhận TWI đều có thể kích hoạt một ngắt TWI. Nhờ đó, TWI trên AVR hoạt động khá độc lập với vi điều khiển. Tuy nhiên, việc sử dụng ngắt cần được triển khai hợp lý. Chẳng hạn, đối với Master, ngắt không thực sự cần thiết vì chip này hoàn toàn kiểm soát quá trình truyền và nhận. Ngược lại, với Slave, sử dụng ngắt là cần thiết để đảm bảo không bỏ lỡ các yêu cầu giao tiếp.

Mỗi vi điều khiển AVR trong mạng TWI đều có thể hoạt động ở chế độ Master hoặc Slave, và cả Master lẫn Slave đều có khả năng truyền và nhận dữ liệu. Vì vậy, TWI trên AVR có tổng cộng bốn chế độ hoạt động, bao gồm: Master Transmitter, Master Receiver, Slave Receiver và Slave Transmitter .

Các ký hiệu thường được sử dụng trong tài liệu datasheet của các vi điều khiển AVR:

- S: START CONDITION – điều kiện bắt đầu
- Rs: REPEAT START – bắt đầu lặp lại
- R: READ BIT – bit này bằng 1 được gửi kèm với gói địa chỉ
- W: WRITE BIT – bit này bằng 0 được gửi kèm với gói địa chỉ
- ACK: ACKNOWLEDGE – xác nhận, SDA được kéo xuống 0 ở xung thứ 9
- NACK: NOT ACKNOWLEDGE – không xác nhận, SDA ở mức cao ở bit thứ 9
- P: STOP CONDITION – điều kiện kết thúc.
- SLA: SLAVE ADDRESS – địa chỉ của Slave cần giao tiếp.

❖ Master Transmitter mode – Chế độ Master truyền dữ liệu

Ở chế độ này, Master sẽ gửi một hoặc nhiều byte dữ liệu tới một hoặc nhiều Slave. Đầu tiên, Master tạo ra một tín hiệu START trên đường SDA. Nếu đường truyền đang ở trạng thái rảnh, Master tiếp tục gửi địa chỉ của Slave cần giao tiếp kèm theo bit WRITE , theo định dạng SLA+W. Nếu Slave phản hồi bằng tín hiệu ACK, Master sẽ tiếp tục truyền một hoặc nhiều

byte dữ liệu lên đường SDA. Sau mỗi byte dữ liệu, Master sẽ kiểm tra tín hiệu ACK từ Slave. Nếu Slave gửi tín hiệu NACK hoặc Master không có ý định truyền thêm dữ liệu, nó sẽ gửi tín hiệu STOP hoặc REPEAT START. Khi tín hiệu STOP được gửi thì quá trình giao tiếp sẽ kết thúc. Còn nếu tín hiệu REPEAT START được phát thì một phiên giao tiếp mới sẽ bắt đầu với địa chỉ của Slave khác ngay sau Rs.

❖ Master Receiver mode – Chế độ Master nhận dữ liệu

Ở chế độ này, Master sẽ nhận một hoặc nhiều byte dữ liệu từ một Slave. Để bắt đầu, Master tạo tín hiệu START trên đường SDA. Nếu đường truyền đang ở trạng thái rảnh, Master tiếp tục gửi địa chỉ của Slave cần giao tiếp kèm theo bit READ, theo định dạng SLA+R. Khi Slave phản hồi bằng tín hiệu ACK, Master bắt đầu lấy mẫu dữ liệu từ đường SDA. Sau mỗi byte dữ liệu, nếu Master muốn nhận thêm byte khác, nó sẽ tiếp tục gửi tín hiệu ACK để thông báo cho Slave. Khi Master muốn dừng quá trình nhận dữ liệu, nó sẽ gửi tín hiệu NACK sau byte dữ liệu cuối cùng, sau đó gửi tín hiệu STOP để kết thúc giao tiếp, hoặc gửi tín hiệu REPEAT START nếu cần tiếp tục giao tiếp với các Slave khác.

❖ Slave Receiver mode – Chế độ Slave nhận dữ liệu

Chế độ Slave nhận dữ liệu xảy ra khi Master thực hiện giao tiếp truyền dữ liệu (SLA+W). Slave chỉ nhận biết được giao tiếp này khi địa chỉ nhận được trùng khớp với địa chỉ của nó (chế độ địa chỉ riêng) hoặc khi Master thực hiện một cuộc gọi chung. Khi đó, bit TWINT trong Slave sẽ được đặt lên mức 1. Nếu tính năng ngắt TWI được kích hoạt (bit TWIE trong thanh ghi TWCR đã được bật), một ngắt sẽ được tạo ra để báo hiệu sự kiện này. Nếu có một cuộc gọi địa chỉ riêng và Slave đã phản hồi Master bằng tín hiệu ACK, Slave sẽ bắt đầu nhận dữ liệu trên đường SDA. Sau mỗi byte dữ liệu, Slave cần gửi tín hiệu ACK để tiếp tục nhận dữ liệu. Nếu Slave không thể nhận thêm, tín hiệu NACK sẽ được gửi sau byte dữ liệu cuối cùng. Giao tiếp kết thúc khi Slave nhận được tín hiệu STOP. Quá trình giao tiếp trong trường hợp gọi chung diễn ra tương tự cuộc gọi địa chỉ riêng, chỉ khác ở mã code được sử dụng. Khi lập trình chế độ nhận dữ liệu cho Slave, cần đảm bảo xử lý cả hai trường hợp: cuộc gọi địa chỉ riêng và cuộc gọi chung.

❖ Slave Transmitter mode – Chế độ Slave truyền dữ liệu

Chế độ Slave truyền dữ liệu xảy ra khi Master yêu cầu nhận dữ liệu từ Slave thông qua một cuộc gọi đọc dữ liệu (SLA+R). Slave nhận biết cuộc gọi này khi địa chỉ nhận được trùng với địa chỉ của nó (trong chế độ địa chỉ riêng). Khi đó, bit TWINT của Slave sẽ được thiết lập lên mức 1. Nếu Slave đáp lại bằng tín hiệu ACK, nó sẽ bắt đầu gửi dữ liệu qua đường SDA. Sau mỗi byte dữ liệu, Master sẽ gửi tín hiệu ACK nếu muốn tiếp tục nhận thêm. Ngược lại, nếu Master không cần thêm dữ liệu, nó sẽ phát tín hiệu NACK, đánh dấu việc kết thúc truyền dữ liệu từ Slave. Một tình huống đặc biệt xảy ra khi bit TWEA (ACK) trong thanh ghi TWCR của Slave được đặt về 0 trước khi dữ liệu được truyền. Điều này báo hiệu rằng byte dữ liệu sắp gửi sẽ là byte cuối cùng. Khi Master nhận byte này, dù có gửi tín hiệu ACK (do không biết đây là byte cuối cùng), Slave vẫn tự động kết thúc quá trình truyền mà không cần chờ thêm tín hiệu từ Master.

11.2.3. Các thanh ghi của I2C

❖ **Thanh ghi TWBR - TWI Bit Rate Register:** là thanh ghi quy định tốc độ phát xung giữ nhịp trên đường SCL của chip Master.

Bit	7	6	5	4	3	2	1	0	TWBR
(0xB8)	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

Bit 7:0 – Thanh ghi tốc độ bit: TWBR chọn hệ số chia cho bộ tạo tốc độ bit. Bộ tạo tốc độ bit là một bộ chia tần số, có nhiệm vụ tạo ra tần số xung nhịp SCL trong các chế độ Master.

Tốc độ phát xung giữ nhịp được tính theo công thức:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16+2(\text{TWBR}), 4\text{TWPS}} \quad (13)$$

Trong đó: CPU Clock frequency là tần số hoạt động chính của AVR; TWBR là giá trị thanh ghi TWBR và TWPS là giá trị của 2 bit TWPS1 và TWPS0 nằm trong thanh ghi trạng thái TWSR. Hai bit này được gọi là bit prescaler, thường set là TWPS1 = 0 và TWPS0 = 0 để chọn prescaler là 1.

❖ Thanh ghi TWCR - TWI Control Register

Bit	7	6	5	4	3	2	1	0	
(0xBC)	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – TWINT (TWI Interrupt Flag): Là một cờ báo quan trọng trong TWI, được tự động thiết lập lên mức 1 khi TWI hoàn thành một quá trình nào đó (ví dụ như phát/nhận tín hiệu START, hoặc phát/nhận địa chỉ,...). Khác với nhiều cờ báo trong các module khác, TWINT không được phần cứng tự động xóa. Vì vậy, khi lập trình điều khiển TWI, cần xóa bit TWINT trước khi bắt đầu bất kỳ quá trình nào. Đặc biệt, TWINT được xóa bằng cách ghi giá trị 1 vào nó. Trong lập trình TWI, thông thường chúng ta sẽ ghi 1 vào TWINT để xóa nó, sau đó liên tục kiểm tra trạng thái của bit này. Khi TWINT trở lại mức 1, điều đó cho thấy quá trình đã hoàn tất.

Bit 6 – TWEA (TWI Enable Acknowledge Bit): Bit này được sử dụng để kích hoạt tín hiệu xác nhận (ACK). Đối với Slave, nếu bit này được bật, tín hiệu ACK sẽ được gửi trong các trường hợp sau: địa chỉ phát bởi Master khớp với địa chỉ của Slave; Slave chấp nhận tham gia một cuộc gọi chung; hoặc dữ liệu từ Master đã được Slave nhận thành công. Do đó, khi cấu hình chip ở chế độ Slave, cần kích hoạt bit này để đảm bảo Slave có thể phản hồi lại Master khi được gọi. Đối với Master, tín hiệu ACK được gửi trong trường hợp Master nhận dữ liệu từ Slave. Lúc này, Master phát tín hiệu ACK để xác nhận đã nhận dữ liệu và yêu cầu Slave tiếp tục truyền thêm.

Bit 5 – TWSTA (TWI START Condition Bit): Bit này được sử dụng để khởi tạo điều kiện START. Khi một chip muốn đóng vai trò Master để bắt đầu giao tiếp, cần kích hoạt bit này. Khi đó, điều kiện START sẽ được tạo trên đường truyền nếu đường truyền đang ở trạng thái rảnh. Nếu đường truyền bận, TWI sẽ đợi đến khi phát hiện điều kiện STOP, sau đó tiếp tục gửi điều kiện START. Lưu ý rằng, sau khi điều kiện START được phát, bit này cần được xóa thủ công bằng phần mềm bằng cách ghi giá trị 0 vào nó.

Bit 4 – TWSTO (TWI STOP Condition Bit): Bit này được sử dụng để khởi tạo điều kiện STOP trong giao tiếp TWI. Khi Master muốn chấm dứt quá trình giao tiếp, nó thực hiện việc phát tín hiệu STOP bằng cách đặt giá trị 1 vào bit TWSTO. Tương tự, trong trường hợp Slave gặp lỗi trong quá trình truyền nhận, Slave có thể ghi giá trị 1 vào bit TWSTO để đưa đường truyền trở về trạng thái rảnh ban đầu.

Bit 3 – TWWC (TWI Write Collision Flag): Khi cờ TWINT ở mức thấp, điều đó biểu thị rằng TWI đang trong trạng thái đang bận. Nếu dữ liệu được ghi vào thanh ghi TWDR trong tình huống này, một lỗi sẽ xảy ra và bit TWWC sẽ tự động được thiết lập lên mức 1. Vì vậy, trong quá trình truyền dữ liệu, cần đảm bảo rằng bit TWINT đang ở mức cao trước khi ghi dữ liệu vào TWDR và chỉ tiến hành xóa bit này sau khi dữ liệu đã sẵn sàng.

Bit 2 – TWEN (TWI Enable Bit): Là bit kích hoạt TWI trên AVR. Khi bit này được thiết lập lên mức 1, TWI sẽ được bật và sẵn sàng hoạt động.

Bit 0 – TWIE (TWI Interrupt Enable Bit): Bit này cho phép kích hoạt ngắt TWI. Khi bit này được thiết lập ở mức 1 và đồng thời bit I trong thanh ghi trạng thái chung cũng được thiết lập, một ngắt TWI sẽ xảy ra khi bit TWINT được phần cứng đưa lên mức cao. Ngắt TWI có thể xuất hiện sau bất kỳ hoạt động nào liên quan đến TWI, vì vậy cần quản lý việc sử dụng ngắt một cách cẩn thận. Trong hầu hết các trường hợp, ngắt chủ yếu được áp dụng cho Slave, trong khi Master thường không cần sử dụng ngắt vì Master có thể chủ động khởi tạo các cuộc truyền dữ liệu.

Lưu ý rằng các bit trong thanh ghi TWCR không nhất thiết phải được thiết lập cùng lúc. Tùy theo từng bước trong quá trình giao tiếp TWI, các bit có thể được thiết lập riêng lẻ.

❖ **Thanh ghi TWSR - TWI Status Register:** là thanh ghi mà trong đó có 5 bit chứa code trạng thái của TWI và 2 bit chọn prescaler.

Bit	7	6	5	4	3	2	1	0	
(0xB9)	TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

Bit 7:3 – Trạng thái TWI: 5 bit này phản ánh trạng thái của logic TWI và bus nối tiếp 2 dây. Lưu ý rằng giá trị đọc từ thanh ghi TWSR bao gồm cả giá trị trạng thái 5 bit và giá trị bộ chia tần số (prescaler) 2 bit. Nên gán giá trị 0 cho các bit bộ chia tần số khi kiểm tra các bit trạng thái, giúp việc kiểm tra trạng thái không bị phụ thuộc vào cấu hình bộ chia tần số.

Bit 1:0 – Các Bit bộ chia TWI: Các bit này có thể đọc và ghi, đồng thời điều khiển bộ chia tần số của tốc độ bit.

Bảng 11.8. Bộ chia tần số tốc độ bit TWI [2]

TWPS1		TWPS0		Prescaler Value			
0		0		1			
0		1		4			
1		0		16			
1		1		64			

❖ **Thanh ghi TWDR – TWI Data Register**

Bit	7	6	5	4	3	2	1	0	
(0xBB)	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W								
Initial Value	1	1	1	1	1	1	1	1	

Chế độ truyền dữ liệu (Transmit Mode) : Trong chế độ truyền, thanh ghi TWDR chứa byte dữ liệu tiếp theo sẽ được truyền đi.

Chế độ nhận dữ liệu (Receive Mode): Trong chế độ nhận, thanh ghi TWDR chứa byte dữ liệu cuối cùng đã nhận được. Thanh ghi này có thể ghi dữ liệu khi TWI không đang trong quá trình dịch chuyển một byte. Điều này xảy ra khi cờ ngắt TWI (TWINT) được phần cứng đặt. Lưu ý rằng thanh ghi dữ liệu không thể được khởi tạo bởi người dùng trước khi ngắt đầu tiên xảy ra. Dữ liệu trong TWDR vẫn ổn định miễn là TWINT được đặt. Khi dữ liệu được dịch chuyển ra ngoài, dữ liệu trên bus sẽ được dịch chuyển vào đồng thời. TWDR luôn chứa byte cuối cùng có mặt trên bus, ngoại trừ trường hợp TWI thoát khỏi chế độ ngủ (sleep mode) bởi ngắt TWI, khi đó nội dung của TWDR là không xác định.

Trong trường hợp mất quyền điều khiển bus (lost bus arbitration) : Không có dữ liệu nào bị mất trong quá trình chuyển đổi từ chế độ Master sang Slave. Việc xử lý bit ACK được điều khiển tự động bởi logic TWI và CPU không thể truy cập trực tiếp vào bit ACK.

Bit 7:0 – Thanh ghi dữ liệu TWI (TWI Data Register) : 8 bit này tạo thành byte dữ liệu tiếp theo sẽ được truyền đi, hoặc byte dữ liệu mới nhất đã nhận trên bus nối tiếp hai dây.

❖ Thanh ghi TWAR – TWI (Slave) Address Register

Bit	7	6	5	4	3	2	1	0	
(0xBA)	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W								
Initial Value	1	1	1	1	1	1	1	0	

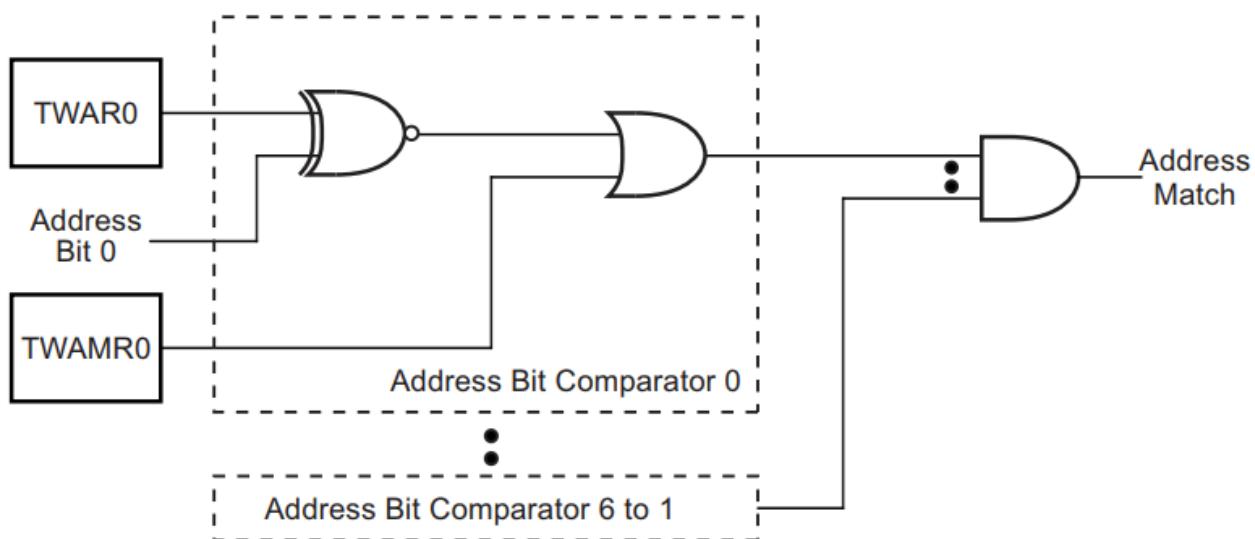
Bit 7:1 – Thanh ghi địa chỉ Slave của TWI: 7 bit này tạo thành địa chỉ của thiết bị Slave trong đơn vị TWI.

Bit 0 – Bit kích hoạt nhận diện địa chỉ gọi chung TWI: Khi bit này được đặt, nó cho phép nhận diện địa chỉ gọi chung trên bus nối tiếp hai dây. Một bộ so sánh địa chỉ kèm sẽ kiểm tra địa chỉ Slave (hoặc địa chỉ gọi chung nếu được kích hoạt) trong địa chỉ nối tiếp nhận được. Nếu có sự khớp, một yêu cầu ngắt sẽ được tạo ra.

❖ Thanh ghi TWAMR – TWI (Slave) Address Mask Register

Bit	7	6	5	4	3	2	1	0	
(0xBD)	TWAM [6:0]							-	TWAMR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R	
Initial	0	0	0	0	0	0	0	0	
Value									

Bits 7:1 – Mặt nạ địa chỉ TWI (TWI Address Mask) : Thanh ghi TWAMR có thể được nạp một mặt nạ địa chỉ Slave 7 bit. Mỗi bit trong TWAMR có thể được sử dụng để vô hiệu hóa bit địa chỉ tương ứng trong thanh ghi địa chỉ TWI (TWAR). Nếu một bit mặt nạ được đặt là 1, thì logic so sánh địa chỉ sẽ bỏ qua việc so sánh giữa bit địa chỉ nhận được và bit tương ứng trong TWAR.



Hình 11.10. Sơ đồ logic so sánh địa chỉ TWI [2]

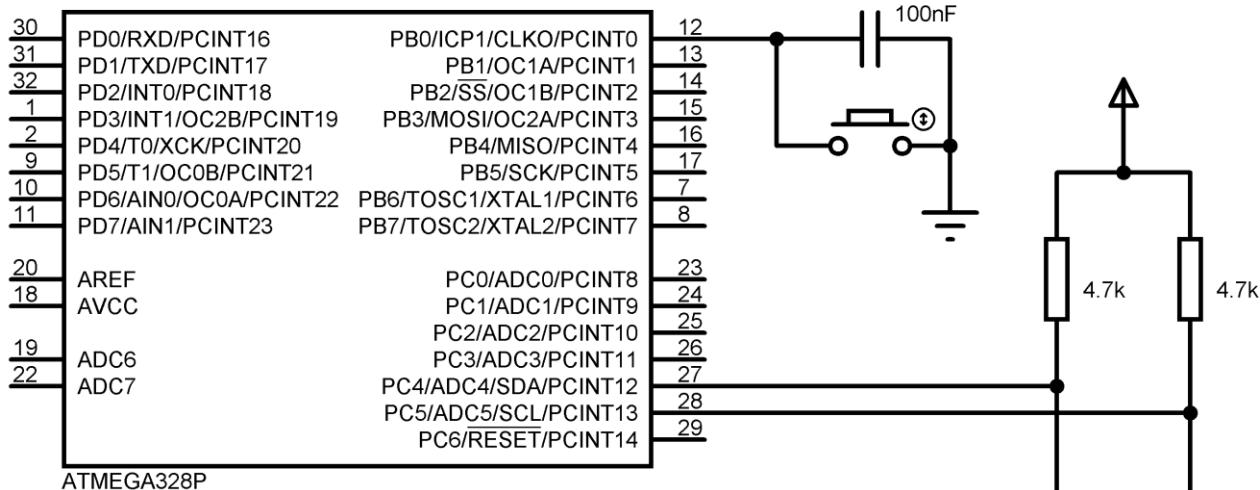
Bit 0 – Reserved Bit: Đây là một bit dự trù, không sử dụng trong ATMEGA328P và sẽ luôn đọc được giá trị 0

11.2.4. Bài thực hành

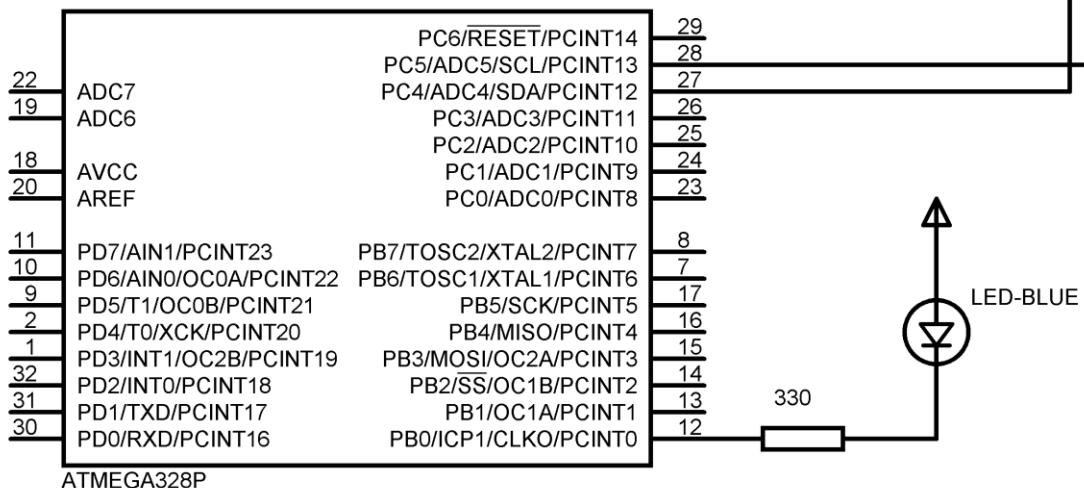
Yêu cầu bài thực hành: Giao tiếp giữa hai vi điều khiển thông qua giao thức I2C, khi nhấn BUTTON sẽ làm thay đổi trạng thái đèn LED.

❖ Phản mô phỏng

MASTER



SLAVE



Hình 11.11. Sơ đồ kết nối bài thực hành giao thức I2C

❖ Phản lập trình

- Chương trình dành cho MASTER.

	SourceCode_ATmega328P\P11_2_I2C_Protocol\I2C_MASTER
1	#include <mega328p.h>
2	#include <io.h>
3	#include <delay.h>
4	

```

5 //dia chi cua SLAVE muon giao tiep
6 #define SLAVE_ADDRESS 0x20
7
8 void I2C_START() {
9     //gui tin hieu START
10    TWCR = (1 << TWSTA) | (1 << TWEN) | (1 << TWINT);
11    while (!(TWCR & (1 << TWINT)));
12 }
13 void I2C_STOP() {
14     //gui tin hieu STOP
15    TWCR = (1 << TWSTO) | (1 << TWEN) | (1 << TWINT);
16    while (TWCR & (1 << TWSTO));
17 }
18 void I2C_WRITE(unsigned int data) {
19     TWDR = data;
20     TWCR = (1 << TWEN) | (1 << TWINT);
21     while (!(TWCR & (1 << TWINT)));
22 }
23 void main(void)
24 {
25     #pragma optsize-
26     CLKPR = (1<<CLKPCE);
27     CLKPR = 0x00;
28     #ifdef _OPTIMIZE_SIZE_
29     #pragma optsize+
30     #endif
31     //nut nhan chan PA2
32     DDRB &= ~(1<<DDB0);
33     PORTB |= (1<<PORTB0);
34
35     //toc do giao tiep 100 kHz
36     TWSR = 0x00;
37     TWBR = 32;
38     while (1) {
39         if (!(PINB & (1 << PINB0))) {
40             delay_ms(50); //delay chong doi nut nhan

```

```

41     if (!(PINB & (1 << PINB0))) {
42         PORTB ^= (1<<7);
43         I2C_START();
44         //MASTER gui dia chi SLAVE muon giao tiep
45         I2C_WRITE(SLAVE_ADDRESS << 1);
46         //MASTER gui gia tri 1
47         I2C_WRITE(0x01);
48         I2C_STOP();
49         delay_ms(200);
50     }
51 }
52 }
53 }
```

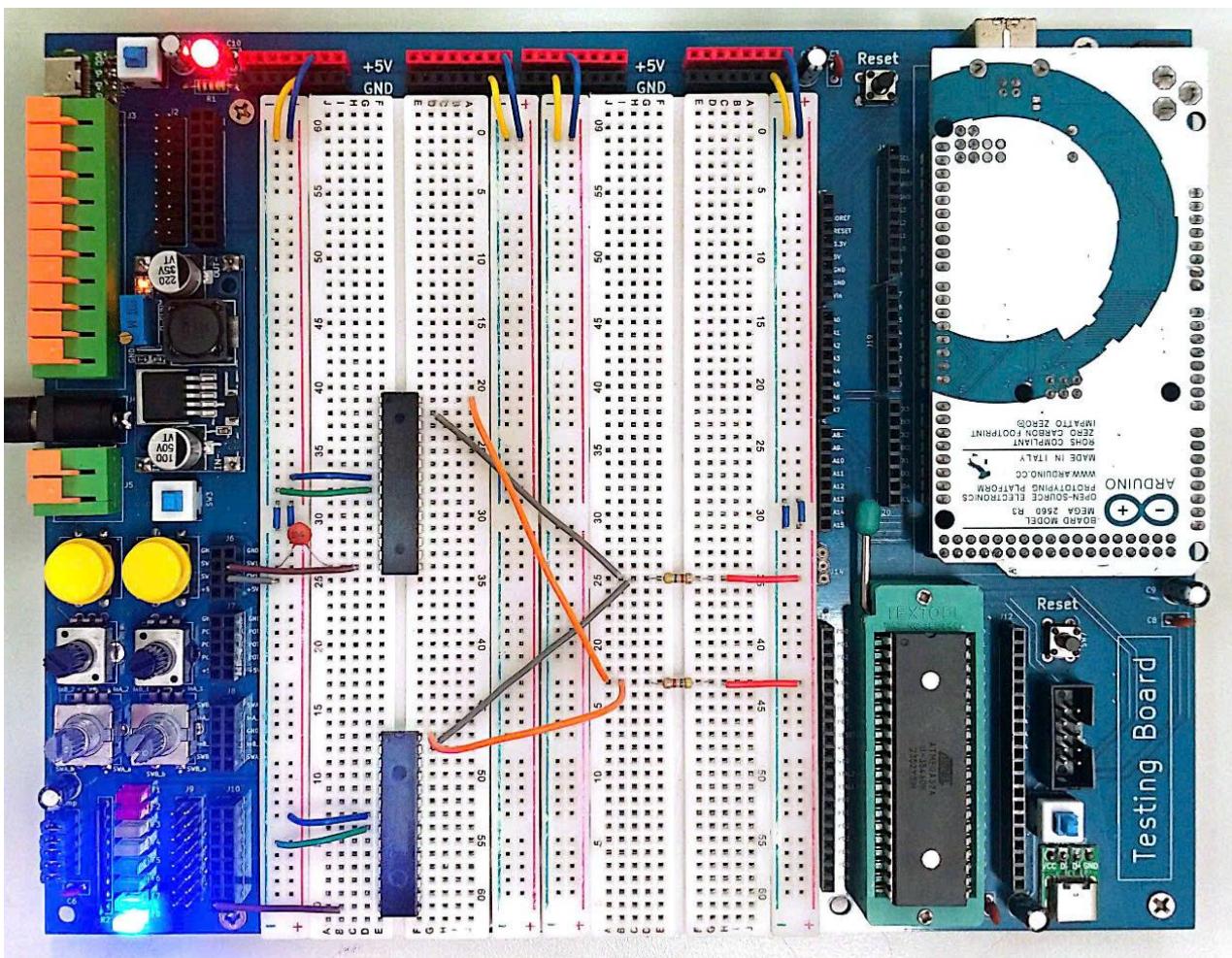
- Chương trình dành cho SLAVE.

	SourceCode_ATMEGA328P\P11_2_I2C_Protocol\I2C_SLAVE
	<pre> 1 #include <mega328p.h> 2 #include <io.h> 3 #include <delay.h> 4 5 #define SLAVE_ADDRESS 0x20 6 unsigned int received_data = 0; 7 void I2C_INIT_SLAVE() { 8 //gan dia chi vao thanh ghi TWAR 9 TWAR = (SLAVE_ADDRESS << 1); 10 //kich hoat I2C, bit ACK 11 TWCR = (1 << TWEN) (1 << TWEA) (1 << TWINT); 12 } 13 unsigned int I2C_WAIT() { 14 while (!(TWCR & (1 << TWINT))); 15 return TWSR & 0xF8; //tra ve 5 bit cao cua TWSR 16 } 17 unsigned int I2C_RECEIVE() { 18 return TWDR; 19 } 20 void I2C_ACKNOWLEDGE() {</pre>

```

21     TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWINT);
22 }
23 void I2C_RESET() {
24     TWCR = (1 << TWEN) | (1 << TWINT) | (1 << TWEA);
25 }
26 void main() {
27     #pragma optsize-
28     CLKPR=(1<<CLKPCE);
29     CLKPR= 0x00;
30     #ifdef _OPTIMIZE_SIZE_
31     #pragma optsize+
32     #endif
33
34     //khai bao chan PB0 dieu khien LED
35     DDRB |= (1 << DDB0);
36     PORTB &= ~(1 << PORTB0);
37
38     I2C_INIT_SLAVE();
39     while (1) {
40         if (I2C_WAIT() == 0x60) {
41             I2C_ACKNOWLEDGE();
42         }
43         if (I2C_WAIT() == 0x80) {
44             received_data = I2C_RECEIVE();
45             I2C_ACKNOWLEDGE();
46         }
47         if (I2C_WAIT() == 0xA0) {
48             I2C_RESET();
49         }
50         if (received_data == 0x01) {
51             PORTB ^= (1 << PORTB0);
52             delay_ms(50);
53             received_data = 0;
54         }
55     }
56 }
```

❖ Phân mô hình



Hình 11.12. Mô hình thực tế bài thực hành I2C

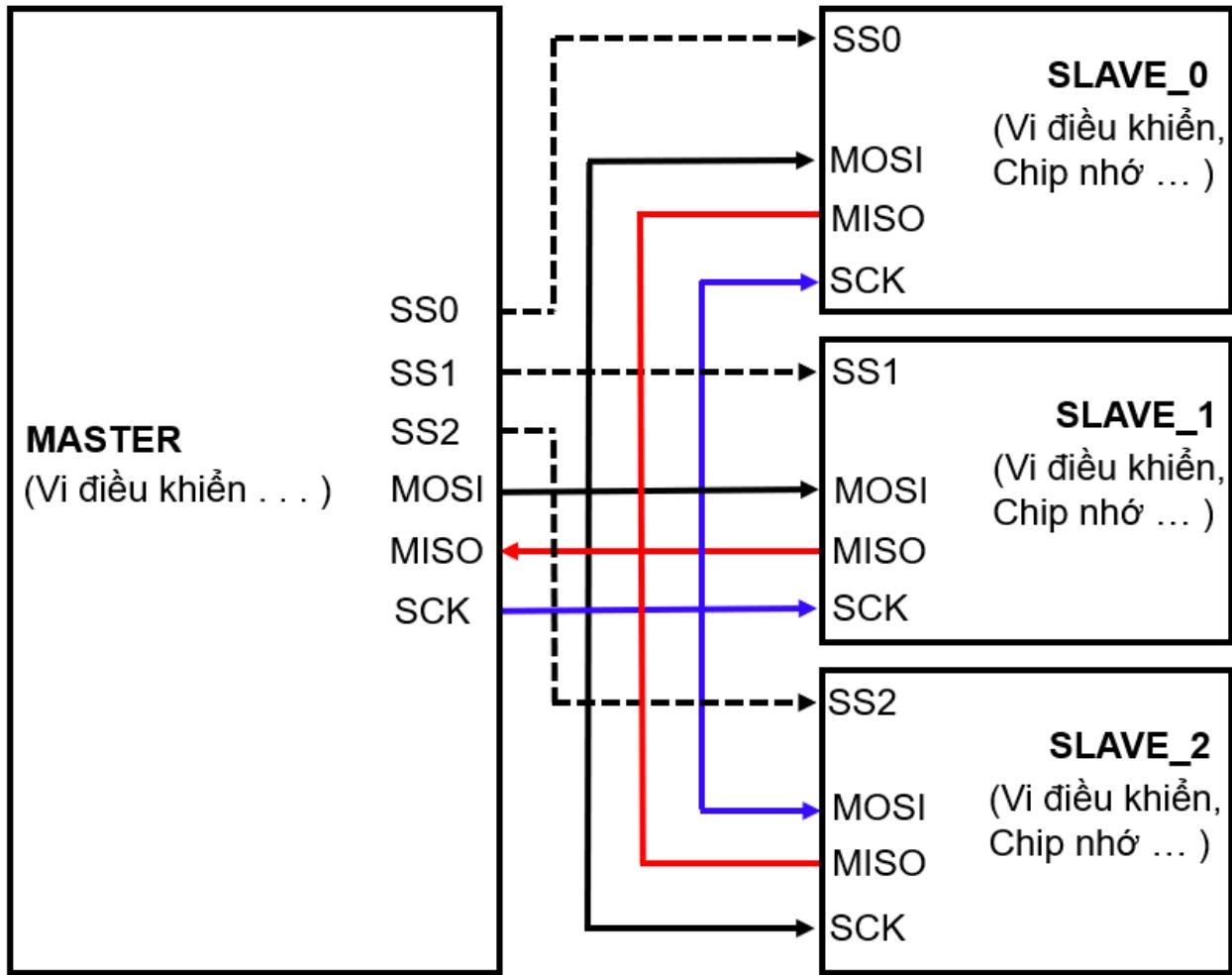
11.3. Giao tiếp SPI

11.3.1. Tổng quan về giao tiếp SPI

SPI (Serial Peripheral Bus) là một chuẩn giao tiếp nối tiếp tốc độ cao được Motorola phát triển. Đây là một kiểu giao tiếp Master-Slave, trong đó một chip Master đảm nhiệm việc điều phối quá trình truyền thông, và các chip Slave sẽ được điều khiển bởi Master, do đó giao tiếp chỉ diễn ra giữa Master và Slave. SPI hỗ trợ truyền song công (full duplex), nghĩa là quá trình truyền và nhận dữ liệu có thể diễn ra đồng thời tại cùng một thời điểm. SPI còn được gọi là giao thức "4 dây" vì sử dụng 4 đường tín hiệu chính, bao gồm SCK (Serial Clock), MISO

(Master Input Slave Output), MOSI (Master Output Slave Input), và SS (Slave Select). Hình bên dưới minh họa một kết nối SPI giữa một chip Master và 3 chip Slave thông qua 4 đường tín hiệu này.

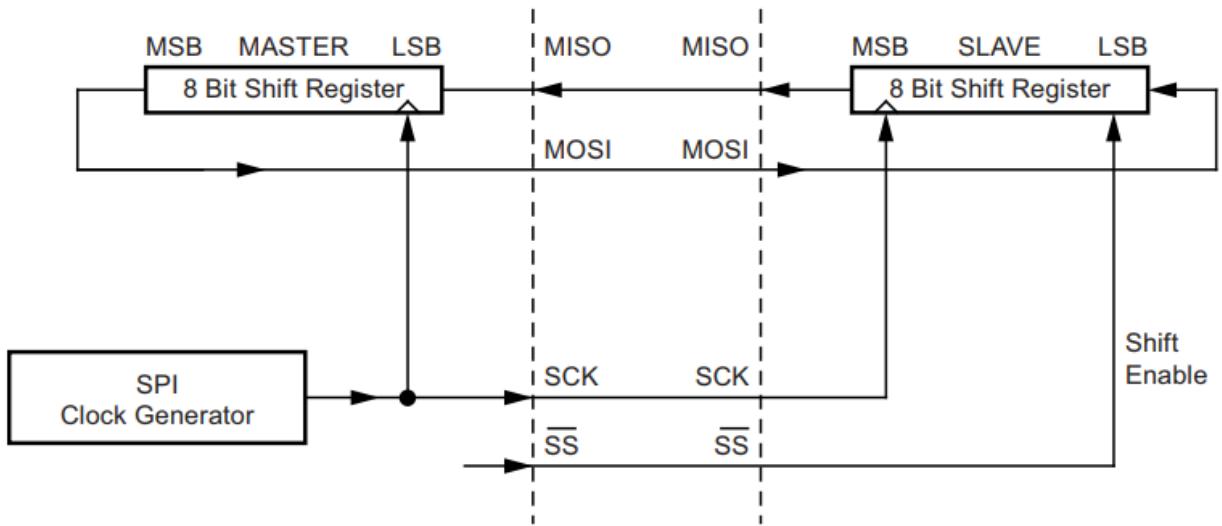
- SCK – Serial Clock là tín hiệu xung nhịp trong giao tiếp SPI, đóng vai trò đồng bộ dữ liệu. Do SPI là chuẩn truyền dữ liệu đồng bộ, nên cần có đường xung nhịp để mỗi chu kỳ trên chân SCK tương ứng với một bit dữ liệu được truyền hoặc nhận. Điều này khác với truyền thông không đồng bộ như chuẩn UART. Việc sử dụng chân SCK giúp giảm thiểu lỗi trong quá trình truyền, cho phép SPI đạt được tốc độ truyền rất cao. Lưu ý rằng xung nhịp chỉ được tạo ra bởi chip Master.
- MISO – Master Input / Slave Output : đối với chip Master, đây là đường Input, trong khi với chip Slave, MISO sẽ là đường Output. Các chân MISO của Master và các Slave được kết nối trực tiếp với nhau.
- MOSI – Master Output / Slave Input: đối với chip Master, đây là đường Output, còn với chip Slave, MOSI đóng vai trò là đường Input. Các chân MOSI của Master và các Slave cũng được nối trực tiếp với nhau.
- SS – Slave Select: Đây là chân được sử dụng để chọn Slave cần giao tiếp. Ở các chip Slave, chân SS sẽ duy trì ở mức cao khi không hoạt động. Khi chip Master kéo chân SS của một Slave xuống mức thấp, việc giao tiếp giữa Master và Slave đó sẽ bắt đầu. Mỗi Slave chỉ có một chân SS, nhưng chip Master có thể có nhiều chân SS để điều khiển, tùy thuộc vào cách thiết kế của người dùng.



Hình 11.13. Giao diện SPI [13]

11.3.2. Cơ chế hoạt động

Mỗi chip Master hoặc Slave đều có một thanh ghi dữ liệu 8 bit. Trong mỗi xung nhịp được Master tạo ra trên đường SCK, một bit dữ liệu từ thanh ghi dữ liệu của Master sẽ được gửi tới Slave thông qua đường MOSI, đồng thời một bit dữ liệu từ thanh ghi dữ liệu của Slave cũng được truyền ngược lại cho Master qua đường MISO. Do dữ liệu giữa hai chip được trao đổi đồng thời theo cả hai chiều, quá trình này được gọi là "truyền song công". Hình 11.13 minh họa việc truyền một gói dữ liệu được thực hiện bởi module SPI trong AVR, với chip Master ở bên trái và chip Slave ở bên phải.



Hình 11.14. Truyền dữ liệu SPI [13]

Cực của xung giữ nhịp (CPOL - Clock Polarity), để cập đến trạng thái của chân SCK khi ở trạng thái nghỉ. Trong trạng thái nghỉ (Idle), chân SCK có thể được giữ ở mức cao (CPOL=1) hoặc mức thấp (CPOL=0). Pha (CPHA) biểu thị cách dữ liệu được lấy mẫu (sample) dựa trên xung giữ nhịp. Dữ liệu có thể được lấy mẫu tại cạnh lên của SCK (CPHA=0) hoặc tại cạnh xuống (CPHA=1). Sự kết hợp giữa CPOL và CPHA tạo ra bốn chế độ hoạt động khác nhau cho SPI. Việc lựa chọn một trong bốn chế độ này thường không ảnh hưởng đến chất lượng truyền thông, miễn là đảm bảo sự tương thích giữa Master và Slave.

Bảng 11.9. Chế độ SPI

SPI Mode	Conditions	Leading Edge	Trailing Edge
0	CPOL=0, CPHA=0	Sample(rising)	Setup (falling)
1	CPOL=0, CPHA=1	Setup (rising)	Sample(falling)
2	CPOL=1, CPHA=0	Sample(falling)	Setup (rising)
3	CPOL=1, CPHA=1	Setup (falling)	Sample(rising)

11.3.3. Các thanh ghi của SPI

❖ **Thanh ghi SPCR – SPI Control Register :** là một thanh ghi 8 bit điều khiển tất cả các hoạt động của SPI.

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – Bit cho phép ngắt SPI (SPI Interrupt Enable): Bit này cho phép thực hiện ngắt SPI nếu bit SPIF trong thanh ghi SPSR được đặt và bit cho phép ngắt toàn cục (Global Interrupt Enable) trong thanh ghi SREG cũng được đặt.

Bit 6 – Bit kích hoạt SPI (SPI Enable): Khi bit SPE được ghi là 1, giao diện SPI sẽ được kích hoạt. Bit này phải được đặt để thực hiện bất kỳ hoạt động nào liên quan đến SPI.

Bit 5 – Thứ tự dữ liệu (Data Order): Khi bit DORD được ghi là 1, bit có trọng số thấp nhất (LSB) của từ dữ liệu sẽ được truyền đi trước. Khi bit DORD được ghi là 0, bit có trọng số cao nhất (MSB) của từ dữ liệu sẽ được truyền đi trước.

Bit 4 – Lựa chọn Master/Slave (Master/Slave Select): Khi bit này được ghi là 1, chế độ Master SPI được chọn. Còn khi bit này được ghi là 0, chế độ Slave SPI được chọn. Nếu chân SS được cấu hình làm đầu vào và bị kéo xuống mức thấp trong khi MSTR được đặt, MSTR sẽ tự động bị xóa, và bit SPIF trong SPSR sẽ được đặt. Người dùng cần đặt lại MSTR để kích hoạt lại chế độ Master SPI.

Bit 3 – Cực của xung nhịp (Clock Polarity): Khi bit này được ghi là 1, tín hiệu xung nhịp SCK sẽ ở mức cao khi ở trạng thái chờ. Khi bit này được ghi là 0, tín hiệu SCK sẽ ở mức thấp khi ở trạng thái chờ.

Bảng 11. 10. Chức năng của CPOL [2]

CPOL	Leading	Trailing Edge
1	Rising	Falling
0	Falling	Rising

Bit 2 – Pha xung nhịp (Clock Phase): Thiết lập bit pha xung nhịp (CPHA) xác định liệu dữ liệu được lấy mẫu ở cạnh đầu (cạnh thứ nhất) hay cạnh sau (cạnh cuối cùng) của xung nhịp SCK.

Bảng 11.11. Chức năng của CPHA [2]

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

Bit 1:0 – Lựa chọn tốc độ xung nhịp SPI (SPI Clock Rate Select 1 và 0): Hai bit này khi kết hợp với bit SPI2X trong thanh ghi SPSR, được sử dụng để thiết lập tốc độ giao tiếp SPI. Tốc độ này được xác định dựa trên tốc độ của nguồn xung clock chia cho một hệ số chia.

Bảng 11. 12. Mối quan hệ giữa tốc độ xung nhịp SCK và tần số dao động (fosc) [2]

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	fosc/4
0	0	1	fosc/16
0	1	0	fosc/64
0	1	1	fosc/128
1	0	0	fosc/2
1	0	1	fosc/8
1	1	0	fosc/32
1	1	1	fosc/64

❖ Thanh ghi SPSR – SPI Status Register

Bit	7	6	5	4	3	2	1	0	
0x2D (0x4D)	SPIF	WCOL	-	-	-	-	-	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – Cờ ngắt SPI (SPI Interrupt Flag): Khi một quá trình truyền dữ liệu nối tiếp hoàn tất, cờ SPIF sẽ được đặt. Một ngắt sẽ được tạo nếu bit SPIE trong thanh ghi SPCR được đặt và các ngắt toàn cục được kích hoạt. Nếu chân SS được cấu hình làm đầu vào và bị kéo xuống mức thấp trong khi SPI ở chế độ Master, cờ SPIF cũng sẽ được đặt. Cờ SPIF được xoá bởi phần cứng khi thực thi trình xử lý ngắt tương ứng. Ngoài ra, cờ SPIF cũng có thể được xoá bằng cách đọc thanh ghi trạng thái SPI (SPSR) khi SPIF được đặt, sau đó truy cập thanh ghi dữ liệu SPI (SPDR).

Bit 6 – Cờ xung đột ghi (Write Collision Flag): Bit WCOL sẽ được đặt nếu thanh ghi dữ liệu SPI (SPDR) bị ghi đè trong khi một quá trình truyền dữ liệu đang diễn ra. Cờ WCOL (và cờ SPIF) được xóa bằng cách đọc thanh ghi trạng thái SPI (SPSR) khi WCOL được đặt, sau đó truy cập thanh ghi dữ liệu SPI (SPDR).

Bit 0 – Bit nhân đôi tốc độ SPI (Double SPI Speed Bit) : Khi bit này được đặt là 1, tốc độ SPI (tần số SCK) sẽ được nhân đôi khi SPI ở chế độ Master. Điều này có nghĩa là chu kỳ tối thiểu của SCK sẽ bằng hai chu kỳ xung nhịp CPU. Khi SPI được cấu hình ở chế độ Slave, SPI chỉ được đảm bảo hoạt động ở tần số $f_{osc}/4$ hoặc thấp hơn.

❖ Thanh ghi SPDR – SPI Data Register

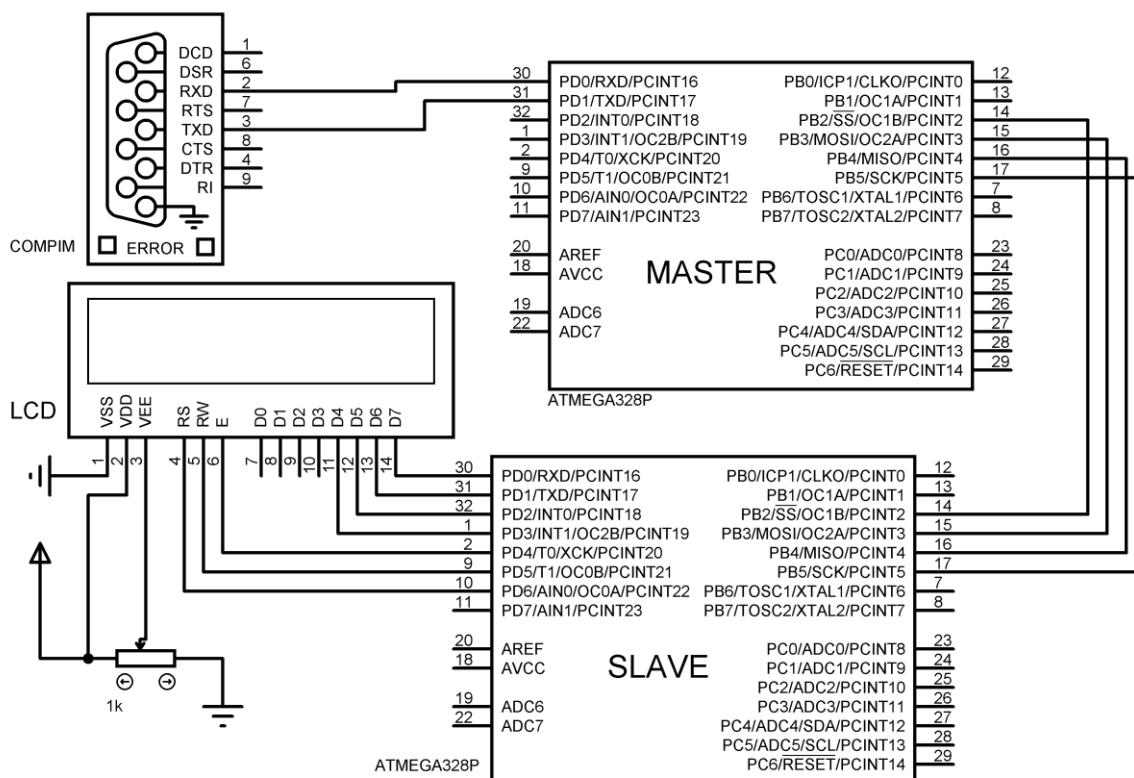
Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

Thanh ghi dữ liệu SPI là một thanh ghi có khả năng đọc/ghi được sử dụng để trao đổi dữ liệu giữa tập thanh ghi và thanh ghi dịch SPI (SPI Shift Register). Khi ghi dữ liệu vào thanh ghi này, quá trình truyền dữ liệu sẽ được khởi động. Việc đọc từ thanh ghi sẽ đọc dữ liệu từ bộ đệm nhận (Receive Buffer) của thanh ghi dịch SPI. Đối với chip Master, ghi một giá trị vào thanh ghi SPDR sẽ bắt đầu quá trình giao tiếp SPI. Đối với chip Slave, dữ liệu nhận được từ Master sẽ lưu trong thanh ghi SPDR, dữ liệu được lưu sẵn trong SPDR sẽ được gửi cho Master.

11.3.4. Bài thực hành

Yêu cầu bài thực hành: Vi điều khiển ATMEGA32 sẽ nhận dữ liệu gửi từ máy tính thông qua giao thức UART, sau đó truyền dữ liệu này đến ATMEGA328P (Arduino Uno) thông qua giao thức SPI và hiển thị lên màn hình LCD.

❖ Phản mô phỏng



Hình 11.15. Sơ đồ kết nối bài thực hành giao thức SPI

❖ Phần lập trình

- Chương trình dành cho MASTER.

	SourceCode_ATMEGA328P\P11_3_SPI_Protocol\SPI_MASTER
1	#include <mega328p.h>
2	#define MOSI 3 //chan PB3
3	#define MISO 4 //chan PB4
4	#define SCK 5 //chan PB5
5	#define SS 2 //chan PB2
6	
7	volatile char receivedData;
8	void UART_Init(){
9	//kich hoat nhan du lieu
10	//cho phep ngat khi nhan
11	UCSR0B = (1<<RXEN0) (1<<RXCIE0);
12	//cau hinh du lieu: 8 bit
13	UCSR0C = (1<<UCSZ01) (1<<UCSZ00);
14	UCSR0A = 0x00;
15	
16	//chon toc do Baud Rate, 9600bps
17	UBRR0H=0x00;
18	UBRR0L=0x33;
19	}
20	void SPI_MASTER_Init(){
21	//Khoi tao SPI MASTER
22	//MOSI, SCK, SS = Output
23	DDRB = (1<<MOSI) (1<<SCK) (1<<SS);
24	DDRB &= ~(1<<MISO); //MISO = Input
25	PORTB = (1<<SS); //dien tro keo len
26	
27	//SPI Enable, Master, F/128
28	SPCR = (1<<SPE) (1<<MSTR) (1<<SPR1) (1<<SPR0);
29	}
30	void SPI_MASTER_Transmit(unsigned char data) {
31	//ghi du lieu vao thanh ghi du lieu
32	SPDR = data;

```

33     while (!(SPSR & (1<<SPIF)));
34 }
35 void main(void)
36 {
37     #pragma optsize-
38     CLKPR = (1<<CLKPCE);
39     CLKPR = 0x00;
40     #ifdef _OPTIMIZE_SIZE_
41     #pragma optsize+
42     #endif
43     UART_Init();
44     SPI_MASTER_Init();
45     #asm("sei")
46     while (1){}
47 }
48 //Nhan duoc Data tu PC, gui sang SLAVE thong qua giao thuc SPI
49 interrupt [USART_RXC] void USART_RX_Complete(void) {
50     receivedData = UDR0;
51
52     //keo chan SS xuong muc thap de truyen
53     PORTB &= ~(1<<SS);
54     SPI_MASTER_Transmit(receivedData);
55     //keo chan SS len khi ket thuc truyen
56     PORTB |= (1<<SS);
57 }
```

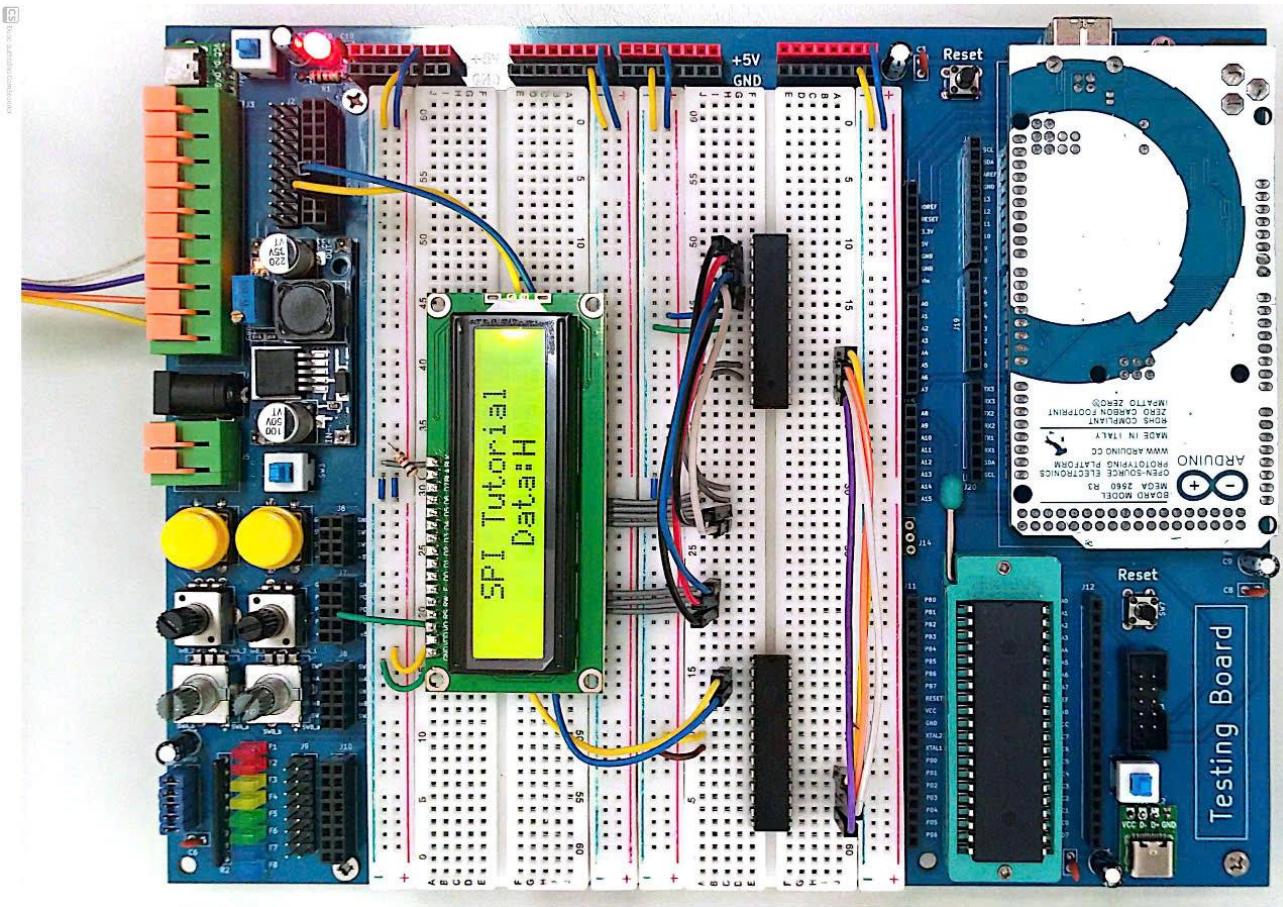
- Chương trình dành cho SLAVE.

	SourceCode_ATMEGA328P\P11_3_SPI_Protocol\SPI_SLAVE
1	#include <mega328p.h>
2	#include <alcd.h>
3	
4	#define MOSI 3
5	#define MISO 4
6	#define SCK 5
7	#define SS 2

```

8 int count;
9 void SPI_Slave_Init(){
10     SPCR = (1<<SPE);
11     //MOSI, SCK, SS= Input
12     //MISO = Output
13     DDRB &= ~ ((1<<MOSI)|(1<<SCK)|(1<<SS));
14     DDRB |= (1<<MISO);
15 }
16 char SPI_Slave_Receive(){
17     while(!(SPSR & (1<<SPIF))); //Hoan thanh nhan
18     return (SPDR); //tra ve gia tri nhan
19 }
20 void main(void)
21 {
22     #pragma optsize-
23     CLKPR = (1<<CLKPCE);
24     CLKPR = 0x00;
25     #ifdef _OPTIMIZE_SIZE_
26     #pragma optsize+
27     #endif
28     lcd_init(16);
29
30     SPI_Slave_Init();
31     lcd_gotoxy(2,0);
32     lcd_putsf("SPI Tutorial");
33     lcd_gotoxy(5,1);
34     lcd_putsf("Data:");
35     while (1)
36     {
37         count = SPI_Slave_Receive();
38         lcd_gotoxy(5,1);
39         lcd_putsf("Data:");
40         lcd_putchar(count);
41     }
42 }
```

❖ Phần mô hình



Hình 11.16. Mô hình thực tế bài thực hành SPI

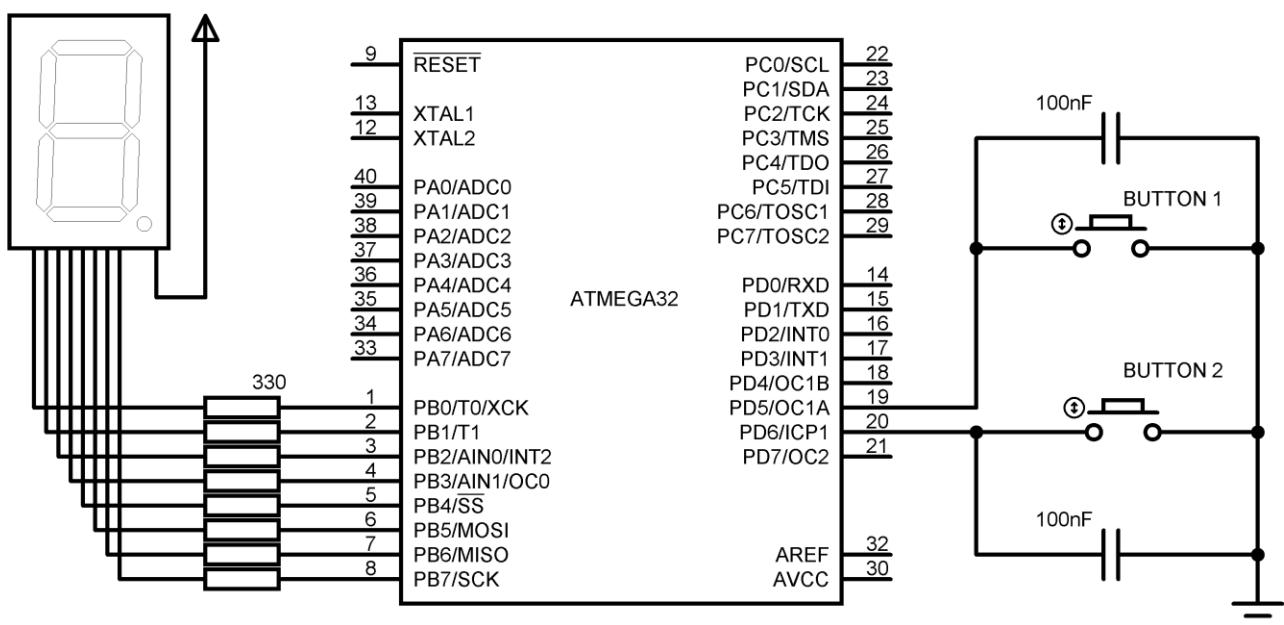
CHƯƠNG 12: LẬP TRÌNH VỚI VI ĐIỀU KHIỂN ATMEGA32A

12.1. Lập trình hiển thị

12.1.1. Lập trình hiển thị LED 7 đoạn

Nội dung bài thực hành: Lập trình hiển thị LED 7 đoạn loại dương chung, khi nhấn “BUTTON 1” LED 7 đoạn sẽ hiển thị tăng lên một đơn vị và ngược lại khi nhấn “BUTTON 2” giá trị hiển thị sẽ giảm đi một đơn vị.

❖ Phản mô phỏng



Hình 12.1. Sơ đồ nối dây bài thực hành LED 7 đoạn

❖ Phản lập trình

	SourceCode_ATMega32A\Segment_Led_Button
1	#include <mega32.h>
2	#include <delay.h>
3	
4	char Led_7Seg[] = {
5	0b01000000,
6	0b01111001,

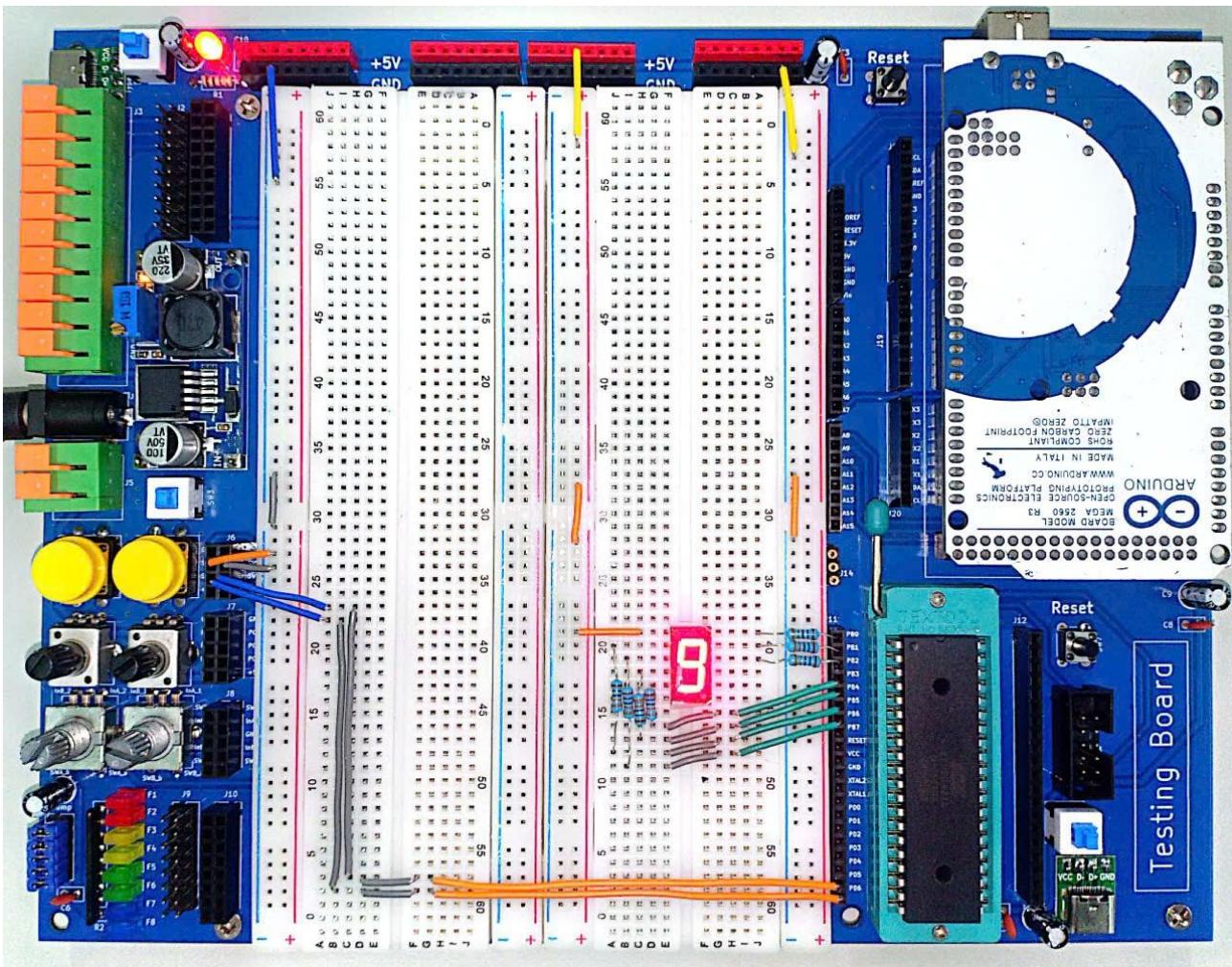
```

7      0b00100100,
8      0b00110000,
9      0b00011001,
10     0b00010010,
11     0b00000010,
12     0b01111000,
13     0b00000000,
14     0b00010000
15 };
16
17 char current_button_1, previous_button_1;
18 char current_button_2, previous_button_2;
19 char counter = 0;
20
21 void main(void)
22 {
23 // dat muc logic cac chan PORTB voi muc thap
24 PORTB = 0b11111111;
25 // cau hinh cac chan PORTB la chan xuat
26 DDRB = 0b11111111;
27
28 // cau hinh dien tro keo len ben trong MCU
29 PORTD = 0xFF; //PORTA = 0b11111111;
30 // cau hinh cac chan PORTA la chan nhan
31 DDRD = 0x00;
32
33 while (1)
34 {
35     previous_button_1 = current_button_1;
36     current_button_1 = PIND.5;
37
38     previous_button_2 = current_button_2;
39     current_button_2 = PIND.6;
40
41     if(PIND.5 == 0 && previous_button_1 != 0)
42     {

```

```
43     if(counter < 9) counter++;
44     else counter = 0;
45 }
46 else if(PIND.6 == 0 && previous_button_2 != 0)
47 {
48     if(counter > 0) counter--;
49     else counter = 9;
50 }
51 PORTB = Led_7Seg[counter];
52 delay_ms(50);
53 }
54 }
```

❖ Phản mô hình

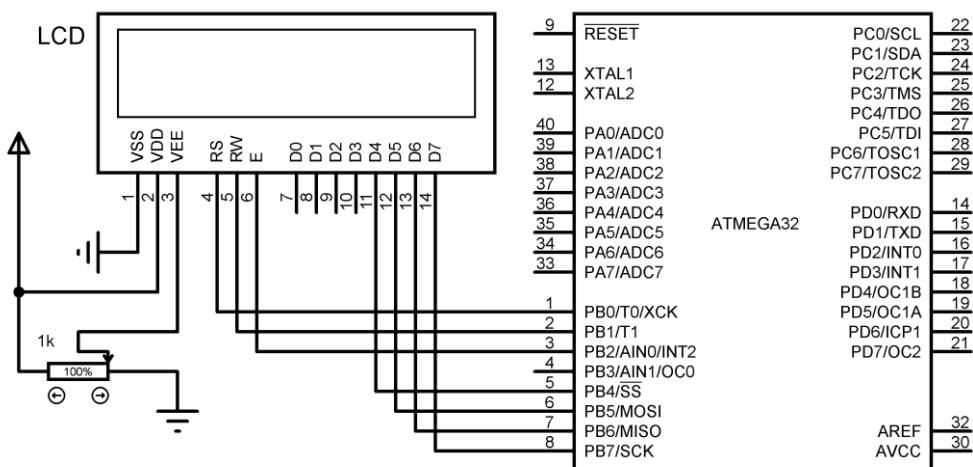


Hình 12.2. Mô hình thực tế bài thực hành LED 7 đoạn

12.1.2. Lập trình hiển thị LCD

Nội dung bài thực hành: Sử dụng LCD 16x2 để hiển thị đoạn ký tự, tạo hiệu ứng chạy dòng chữ “HELLO WORLD” từ trái sang phải.

❖ Phản mô phỏng



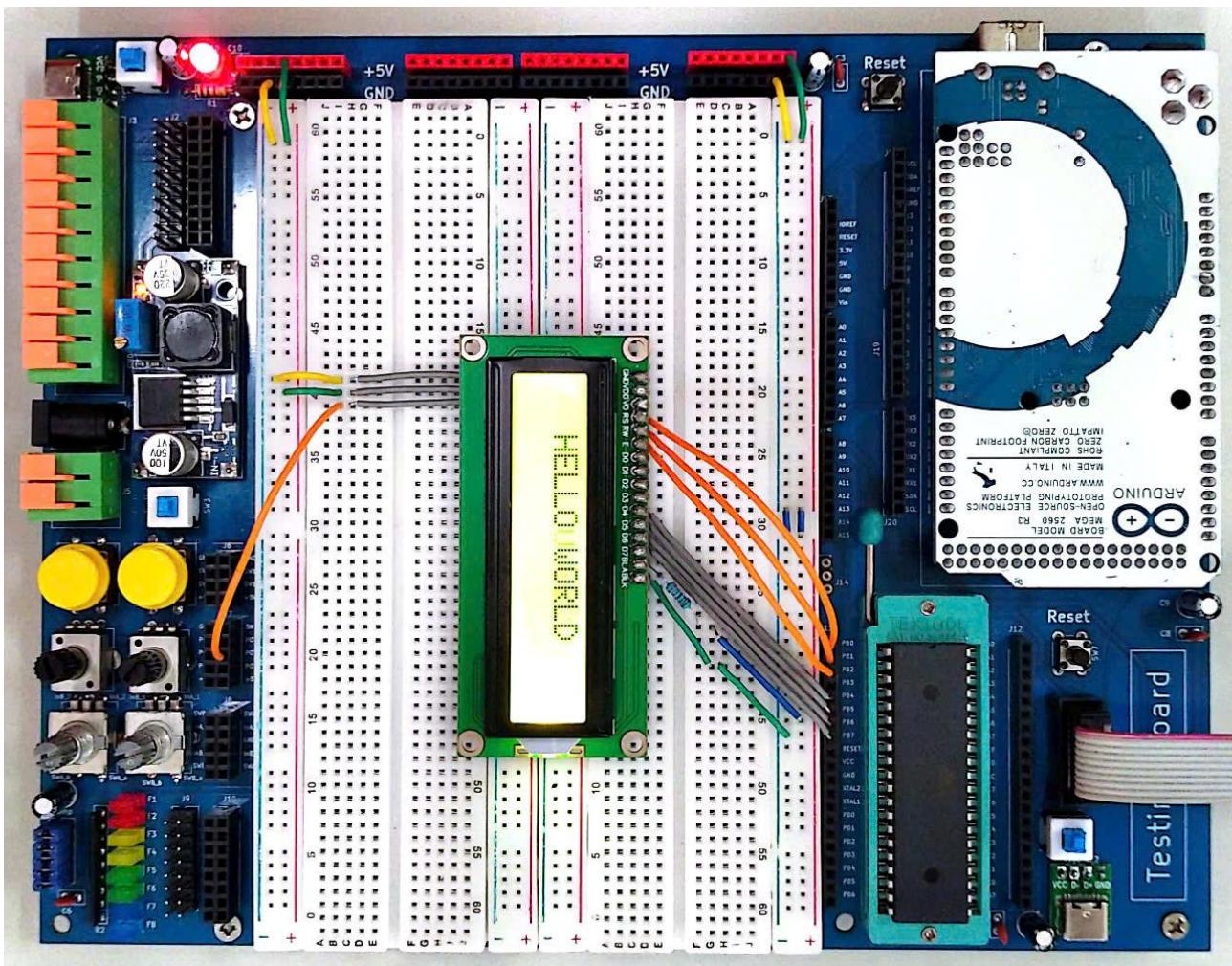
Hình 12.3. Sơ đồ nối dây bài thực hành với LCD

❖ Phản lập trình

	SourceCode_ATMega32A\LCD_16x2
1	#include <mega32.h>
2	#include <delay.h>
3	#include <alcd.h>
4	
5	//mang chua doan ky tu can hien thi
6	//myStr gom (16+11+16) ky tu
7	char myStr[] = " HELLO WORLD ";
8	int length = sizeof(myStr) - 1; //tru 1 ky tu ket thuc
9	int lcd_width = 16;
10	
11	void main(void)
12	{
13	lcd_init(16);
14	lcd_clear();

```
15    lcd_gotoxy(2, 0);
16    lcd_putsf("Demo of the");
17    lcd_gotoxy(0, 1);
18    lcd_putsf("2x16 LCD Display");
19    delay_ms(2000);
20    lcd_clear();
21    while (1)
22    {
23        int i, j;
24        for (i = length - lcd_width; i >= 0; i--) {
25            lcd_clear();
26            for (j = 0; j < lcd_width; j++) {
27                lcd_gotoxy(j, 0);
28                lcd_putchar(myStr[i + j]);
29            }
30            delay_ms(500);
31        }
32    }
33 }
```

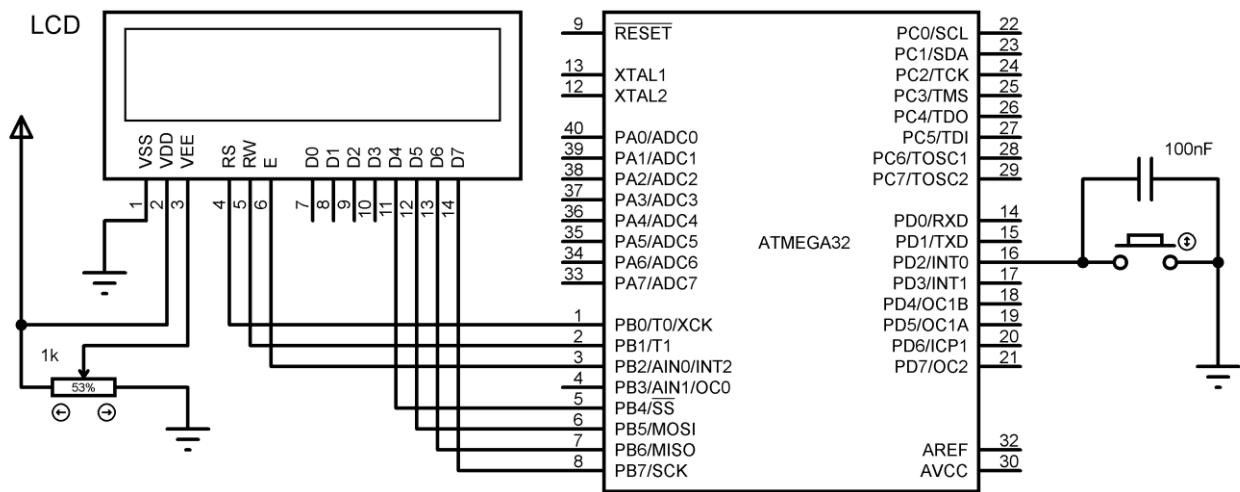
❖ Phân mô hình



Hình 12.4. Mô hình thực tế bài thực hành hiển thị LCD

12.2. Cấu trúc ngắt

❖ Phần mô phỏng



Hình 12.5. Sơ đồ nối dây bài thực hành ngắt ngoài

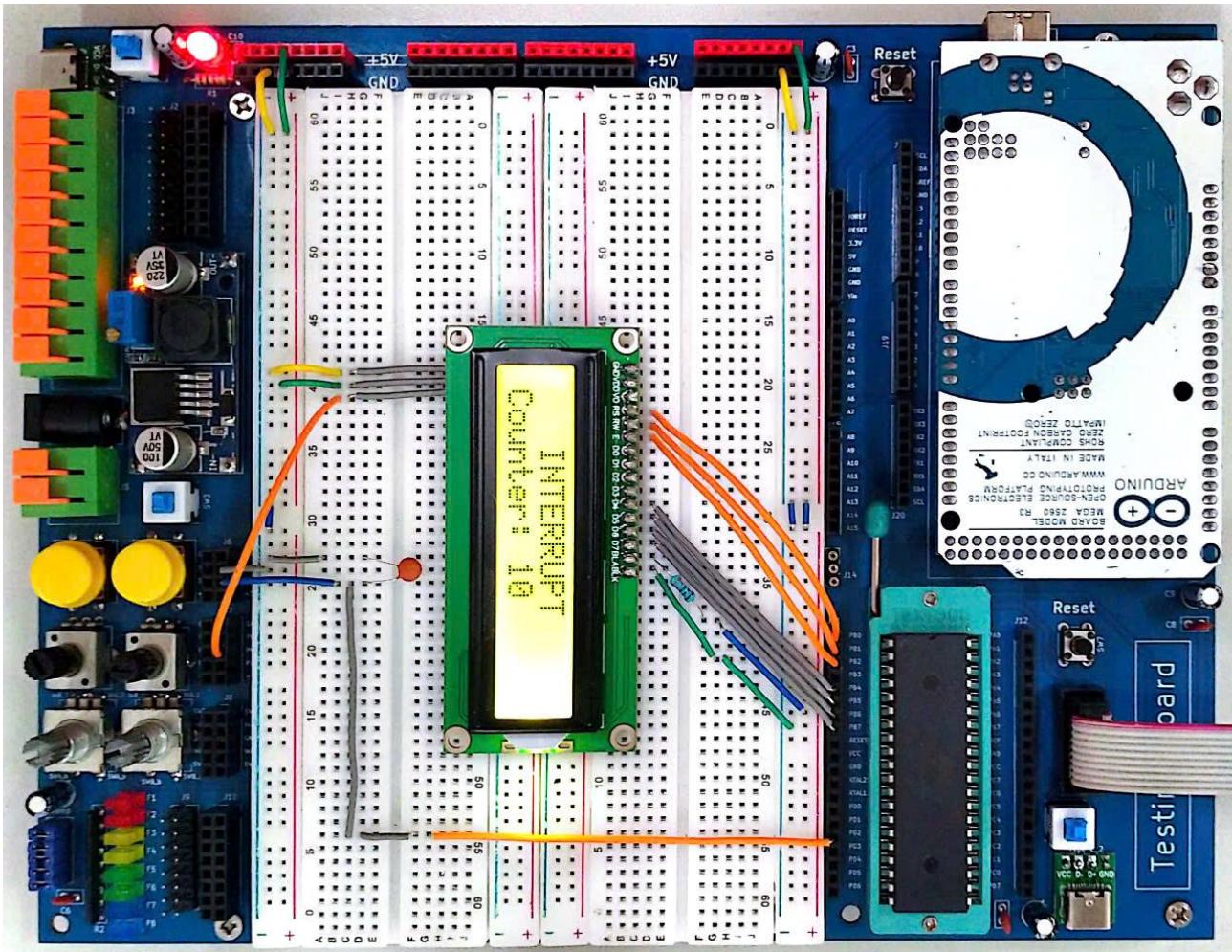
❖ Phần lập trình

SourceCode_ATMega32A\Interrupt_LCD	
1	#include <mega32.h>
2	#include <delay.h>
3	#include <alcd.h>
4	#include <stdlib.h>
5	
6	unsigned char counter = 0;
7	unsigned char myStr[3];
8	
9	interrupt [EXT_INT0] void ext_int0_isr(void)
10	{
11	counter++; //dem len sau moi lan ngat
12	itoa(counter, myStr);
13	lcd_gotoxy(0,1);
14	lcd_puts("Counter:");
15	lcd_puts(myStr);
16	}
17	

```

18 void main(void)
19 {
20     lcd_init(16);
21     lcd_gotoxy(3,0);
22     lcd_puts("INTERRUPT");
23     lcd_gotoxy(0,1);
24     lcd_puts("Counter:");
25     itoa(counter, myStr);
26     lcd_puts(myStr);
27
28     // cau hinh cac chan PORTC la chan xuat
29     DDRC = 0b01111111;
30     // dat muc logic cac chan PORTC voi muc thap
31     PORTC = 0b01111111;
32
33     // cau hinh dien tro keo len ben trong MCU
34     PORTD = 0xFF;      //PORTA = 0b11111111;
35     // cau hinh cac chan PORTA la chan nhan
36     DDRD = 0x00;
37
38     // cau hinh cac thanh ghi ngat
39     GICR|=(1<<INT0);
40     MCUCR=(1<<ISC11) | (1<<ISC01);
41     GIFR=(1<<INTF0);
42
43     #asm("sei")
44     while (1){}
45 }
```

❖ Phân mô hình



Hình 12.6. Mô hình thực tế bài thực hành ngắn ngoài

12.3. Bộ chuyển đổi ADC

Nội dung bài thực hành: Đọc giá trị cảm biến nhiệt độ LM35 thông qua ADC và hiển thị lên màn hình LCD. LM35 được kết nối với chân PA0, đây là chân ADC0 của vi điều khiển và bộ ADC được sử dụng với độ phân giải 10 bit, ngoài ra cần nối chân AVCC của vi điều khiển với điện áp tham chiếu 5V.

- Giá trị ADC được chuyển đổi thành điện áp đầu ra của LM35 theo công thức:

$$V_{out} = \frac{ADC_{value} \times V_{AVCC}}{1024} [mV] \quad (1)$$

Với: V_{AVCC} là điện áp tham chiếu ở chân AVCC ($5V = 5000mV$)

Do sử dụng độ phân giải 10 bit nên phải chia cho 1024 (từ 0 đến 1023)

ADC_value là giá trị mà ADC đọc được sau mỗi frame

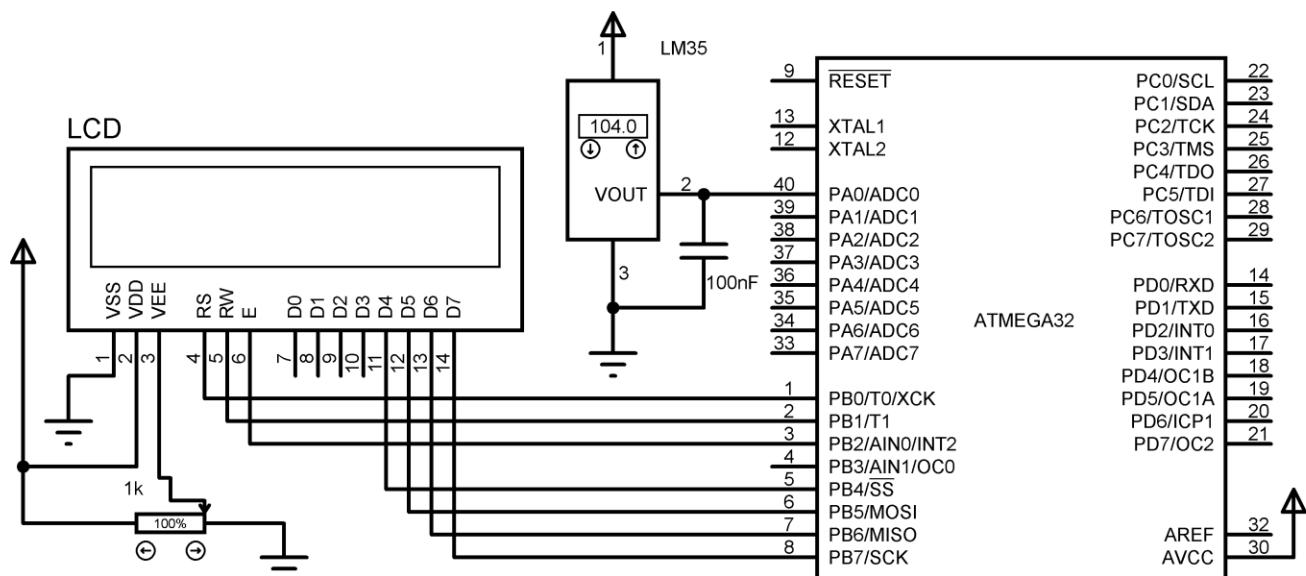
- Nhiệt độ được tính toán dựa trên công thức:

$$T = \frac{V_{out}}{10} = \frac{ADC_{value} \times V_{AVCC}}{1024} \times \frac{1}{10} [^{\circ}\text{C}] \quad (2)$$

do 10 mV tương đương với 1°C nên từ đó kết luận:

$$T = \frac{ADC_{value} \times 5000}{1024} \times \frac{1}{10} = \frac{ADC_{value} \times 125}{256} \quad (3)$$

❖ Phản mô phỏng



Hình 12.7. Sơ đồ nối dây bài thực hành với LM35

❖ Phản lập trình

	SourceCode_ATMEGA32A\ADC_LM35
1	#include <mega32.h>
2	#include <delay.h>
3	#include <alcd.h>

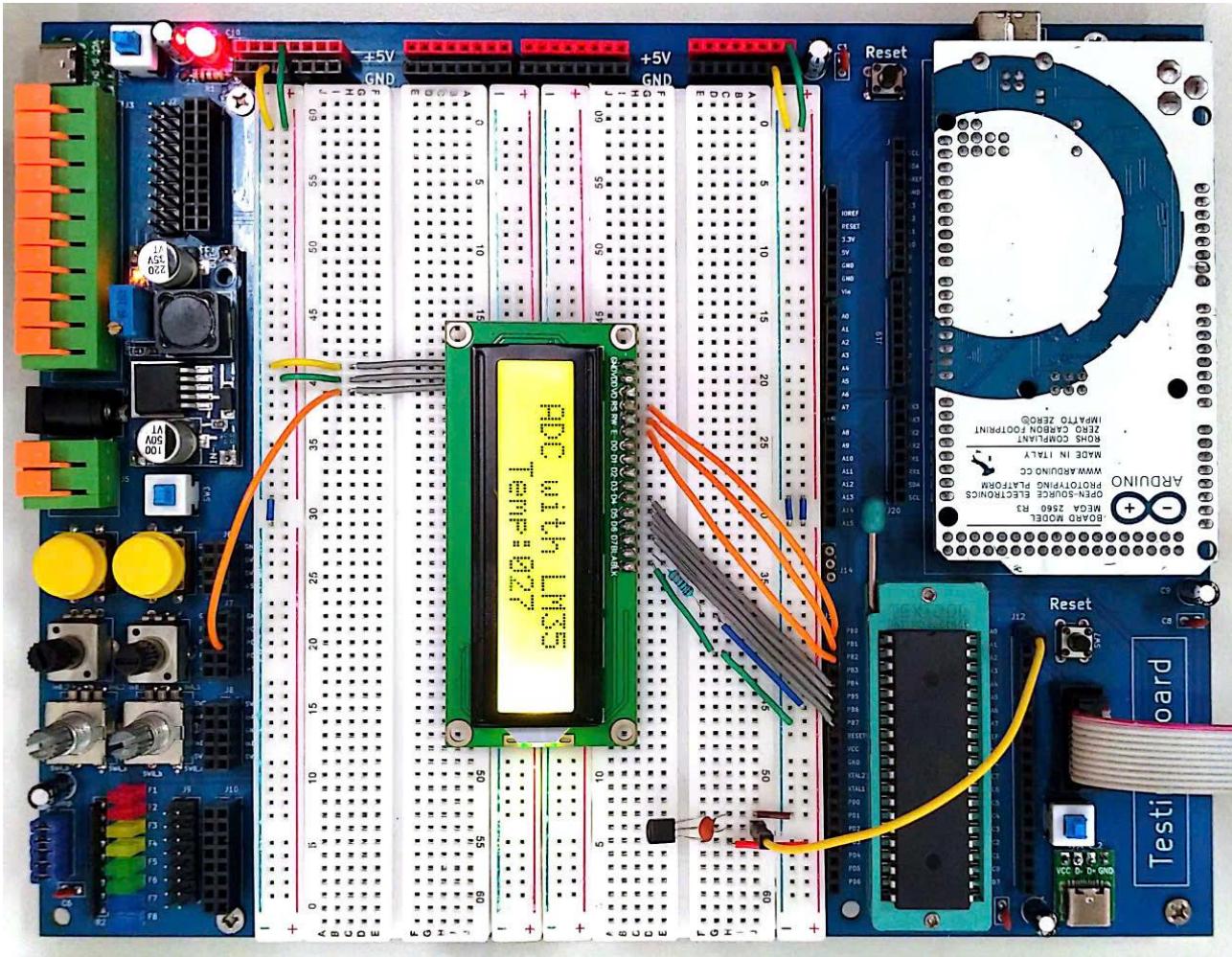
```

4
5 void Display_Value(unsigned int number);
6
7 //chon dien ap tham chieu AVCC
8 #define ADC_VREF_TYPE ((0<<REFS1) | (1<<REFS0) | (0<<ADLAR))
9
10 unsigned int resultADC0;
11 unsigned int Temp;
12 // Ham doc gia tri ADC
13 unsigned int read_adc(unsigned char adc_input)
14 {
15     ADMUX = adc_input | ADC_VREF_TYPE;
16     delay_us(10);
17     ADCSRA |= (1<<ADSC);
18     while ((ADCSRA & (1<<ADIF)) == 0);
19     ADCSRA |= (1<<ADIF);
20     return ADCW;
21 }
22
23 void main(void)
24 {
25     //cau hinh chan doc ADC la ngo vao
26     DDRA.0 = 0;
27     PORTA.0 = 0;
28
29     ADMUX = ADC_VREF_TYPE;
30     ADCSRA = (1<<ADEN) | (1<<ADPS2);
31     SFIOR = (0<<ADTS2) | (0<<ADTS1) | (0<<ADTS0);
32
33     lcd_init(16);
34     lcd_gotoxy(1,0);
35     lcd_puts("ADC with LM35 ");
36     delay_ms(500);
37     while (1){
38         resultADC0 = read_adc(0);
39         Temp = (resultADC0*125/256);

```

```
40     lcd_gotoxy(4,1);
41     lcd_putsf("Temp:");
42     Display_Value(Temp);
43 }
44 }
45 //Ham hien thi gia tri len LCD
46 void Display_Value(unsigned int number)
47 {
48     int tram, chuc, donvi;
49     tram = (number/100)%10;
50     chuc = (number/10)%10;
51     donvi = number%10;
52     lcd_putchar(tram+48);
53     lcd_putchar(chuc+48);
54     lcd_putchar(donvi+48);
55 }
```

❖ Phần mô hình



Hình 12.8. Mô hình thực tế bài thực hành với LM35

12.4. Lập trình điều khiển kim phun nhiên liệu

12.4.1. Giới thiệu

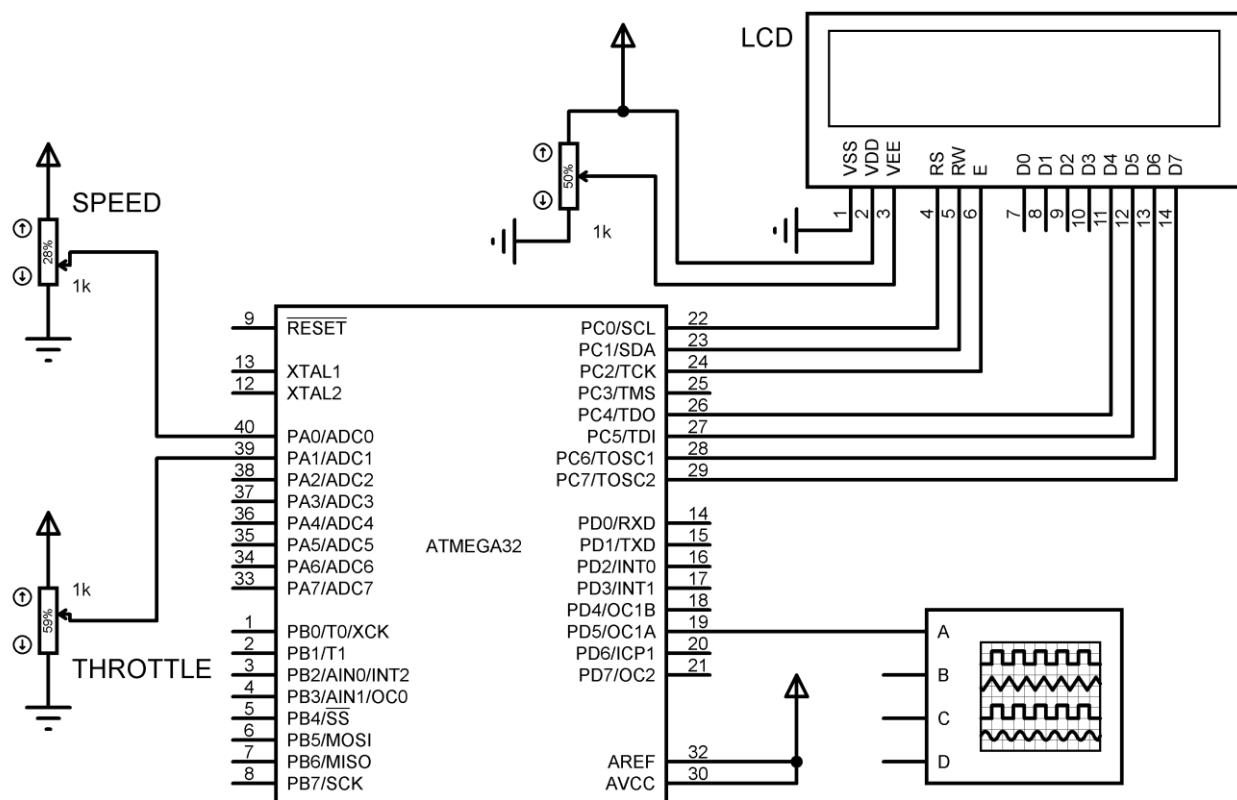
Điều khiển kim phun nhiên liệu là một ứng dụng tiêu biểu trong lĩnh vực ô tô, thể hiện sự tiến bộ vượt bậc của công nghệ trong việc tối ưu hóa hiệu suất động cơ. Với vai trò chính là kiểm soát lượng nhiên liệu được phun vào buồng đốt, hệ thống này sử dụng vi điều khiển để xử lý các tín hiệu từ cảm biến như áp suất, nhiệt độ không khí, tốc độ động cơ và vị trí bướm ga. Dựa trên dữ liệu này, vi điều khiển tính toán và điều chỉnh thời điểm cũng như lượng phun nhiên liệu nhằm đảm bảo tỷ lệ hòa khí tối ưu, từ đó tăng hiệu suất đốt cháy, giảm tiêu

thu nhiên liệu và khí thải. Sự chính xác và linh hoạt của hệ thống điều khiển kim phun không chỉ cải thiện hiệu suất vận hành mà còn đóng vai trò quan trọng trong việc đáp ứng các tiêu chuẩn khí thải ngày càng nghiêm ngặt.

12.4.2. Thực hành

Nội dung bài thực hành: Sử dụng hai biến trở để giả lập tín hiệu tốc độ động cơ và tín hiệu góc mở bướm ga, từ đó tính toán để đưa ra thời gian mở kim phun.

❖ Phần mô phỏng



Hình 12.9. Sơ đồ kết nối bài thực hành điều khiển kim phun

❖ Phần lập trình

	SourceCode_ATMEGA32A\Fuel_Injector
1	#include <mega32.h>
2	#include <delay.h>
3	#include <alcd.h>

```

4 #include <stdio.h>
5
6 #define ADC_VREF_TYPE ((0<<REFS1) | (1<<REFS0) | (0<<ADLAR))
7
8 #define CYLINDER      1    // so luong xy lanh
9 #define STROKE        4    // so ky
10 #define INJECTOR_COUNT 1   // so luong kim phun
11 #define CAPACITY      0.3 // dung tich dong co(lit)
12 #define M_AIR          // khoi luong khong khi(mg/lit)
13 unsigned int speed, throttle;
14 float H_charge = 0;           // hieu suat nap
15 float lamda = 0;             // he so du luong khong khi
16
17 // the tich nap cua xy lanh / lan nap (lit/100)
18 float V_cylinder = (float)(CAPACITY / CYLINDER);
19 float m_air_in;              // khoi luong kk nap thuc te
20
21 //the tich khong khi nap vao xy lanh/phut (lit)
22 float V_air_1min;
23 //khoi luong kk nap vao xy lanh/phut
24 float m_air_1min;
25 //khoi luong nhien lieu vao xy lanh/phut
26 float m_fuel_1min;
27 //luong nhien lieu phun cho 1 chu trinh (mg)
28 float m_fuel_1cycle;
29 //thoi gian mo kim phun cho 1 chu trinh (ms)
30 float t_injector_1cycle;
31 //so lan phun tren 1ph va 1s
32 float spray_1_min, spray_1_sec;
33 float T_cycle; //thoi gian cua 1 chu trinh phun
34
35 unsigned int Top_Timer1;
36
37 unsigned int read_adc(unsigned char adc_input)
38 {
39     ADMUX=adc_input | ADC_VREF_TYPE;

```

```

40 delay_us(10);
41 ADCSRA|=(1<<ADSC);
42 while ((ADCSRA & (1<<ADIF))==0);
43 ADCSRA|=(1<<ADIF);
44 return ADCW;
45 }

46
47 //Ham hien thi
48 void Display_Value(unsigned int number)
49 {
50     int nghin, tram, chuc, donvi;
51
52     nghin = number/1000%10;
53     tram = number/100%10;
54     chuc = (number/10)%10;
55     donvi = number%10;
56
57     lcd_putchar(nghin+48);
58     lcd_putchar(tram+48);
59     lcd_putchar(chuc+48);
60     lcd_putchar(donvi+48);
61 }
62 void Read_Speed_Throttle(){
63     unsigned int ADC0, ADC1;
64     ADC0 = read_adc(0);
65     ADC1 = read_adc(1);
66     //speed = ADC0*6000/1023
67     speed = (unsigned long)ADC0*200/1023*20;
68     throttle = (ADC1*10/3*10/341);
69 }
70 void Lamda_Efficiency(){
71     if(throttle < 7 || speed<1000){
72         H_charge = 0.0;
73         lamda = 0.0;
74     }
75     else if(throttle>=7 && throttle<14){

```

```

76     if(speed>=1000 && speed < 2499){
77         H_charge = 0.6;
78         lamda = 1;
79     }
80     else if(speed>=2499 && speed <= 4000){
81         H_charge = 0.6;
82         lamda = 1.35;
83     }
84 }
85 else if(throttle>=14 && throttle<21){
86     if(speed>=1000 && speed < 2499){
87         H_charge = 0.6;
88         lamda = 1;
89     }
90     else if(speed>=2499 && speed <= 4000){
91         H_charge = 0.6;
92         lamda = 1.12;
93     }
94 }
95 else if(throttle>=21 && throttle<28){
96     if(speed>=1000 && speed < 2499){
97         H_charge = 0.6;
98         lamda = 1;
99     }
100    else if(speed>=2499 && speed <= 4000){
101        H_charge = 0.6;
102        lamda = 1;
103    }
104 }
105 else if(throttle>=28 && throttle<35){
106     if(speed>=1000 && speed < 2499){
107         H_charge = 0.6;
108         lamda = 1;
109     }
110     else if(speed>=2499 && speed <= 4000){
111         H_charge = 0.6;

```

```

112         lamda = 1;
113     }
114 }
115 else if(throttle>=35 && throttle<42){
116     if(speed>=1000 && speed < 2499){
117         H_charge = 0.6;
118         lamda = 1;
119     }
120 }
121 else if(throttle>=42 && throttle<49){
122     if(speed>=1000 && speed < 2499){
123         H_charge = 0.7;
124         lamda = 1;
125     }
126     else if(speed>=2499 && speed <= 4000){
127         H_charge = 0.7;
128         lamda = 1;
129     }
130 }
131 else if(throttle>=49 && throttle<56){
132     if(speed>=1000 && speed < 2499){
133         H_charge = 0.7;
134         lamda = 1;
135     }
136     else if(speed>=2499 && speed <= 4000){
137         H_charge = 0.7;
138         lamda = 1;
139     }
140 }
141 else if(throttle>=56 && throttle<63){
142     if(speed>=1000 && speed < 2499){
143         H_charge = 0.7;
144         lamda = 0.97;
145     }
146     else if(speed>=2499 && speed <= 4000){
147         H_charge = 0.7;

```

```

148         lamda = 1;
149     }
150 }
151 else if(throttle>=63 && throttle<70){
152     if(speed>=1000 && speed < 2499){
153         H_charge = 0.7;
154         lamda = 0.91;
155     }
156 }
157 else if(throttle>=70 && throttle<77){
158     if(speed>=1000 && speed < 2499){
159         H_charge = 0.7;
160         lamda = 0.91;
161     }
162     else if(speed>=2499 && speed <= 4000){
163         H_charge = 0.7;
164         lamda = 0.91;
165     }
166 }
167 else if(throttle>=77 && throttle<84){
168     if(speed>=1000 && speed < 2499){
169         H_charge = 0.85;
170         lamda = 0.91;
171     }
172     else if(speed>=2499 && speed <= 4000){
173         H_charge = 0.85;
174         lamda = 0.89;
175     }
176 }
177 else if(throttle>=84 && throttle<91){
178     if(speed>=1000 && speed < 2499){
179         H_charge = 0.85;
180         lamda = 0.97;
181     }
182     else if(speed>=2499 && speed <= 4000){
183         H_charge = 0.85;

```

```

184         lamda = 0.88;
185     }
186 }
187 else if(throttle>=91){
188     if(speed>=1000 && speed < 2499){
189         H_charge = 0.85;
190         lamda = 0.91;
191     }
192     else if(speed>=2499 && speed <= 4000){
193         H_charge = 0.85;
194         lamda = 0.88;
195     }
196 }
197 }
198 void Timer1_Init(){
199     DDRD |= (1 << DDD5);
200     //Timer1 bo chia 8, mode 14
201     TCCR1A = (1<<COM1A1)|(1<<WGM11)|(1<<COM1A0);
202     TCCR1B = (1<<WGM13)|(1<<WGM12)|(1<<CS11);
203 }
204 void Calculate(){
205     //khoi luong kk nap thuc te (mg)
206     // = V_cylinder*H_charge*khoi luong rieng kk(1130)/14,7
207     m_air_in = V_cylinder * H_charge * 1130.5 * 83/1470 ;
208     //the tich khong khi nap vao xylanh/phut (lit)
209     V_air_1min = V_cylinder*H_charge* speed/2;
210     //khoi luong kk nap vao xylanh/phut
211     m_air_1min = V_air_1min*1130.5;
212     //khoi luong nhien lieu vao xylanh/phut
213     m_fuel_1min = m_air_1min/(14.7*lamda);
214     //luong nhien lieu phun cho 1 chu trinh (mg)
215     m_fuel_1cycle = m_fuel_1min/(speed/2);
216     //thoi gian mo kim phun cho 1 chu trinh (ms)
217     t_injector_1cycle = 0.33*m_fuel_1cycle-0.57;
218     //so chu ky lam viec tren phut va giay
219     spray_1_min = speed/2;

```

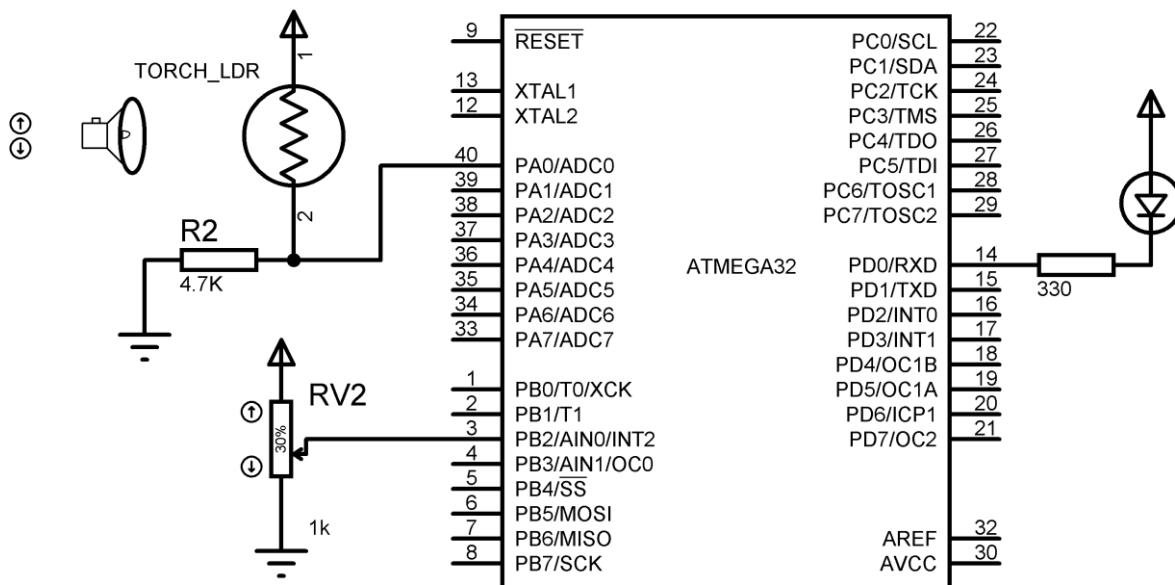
```

220     spray_1_sec = spray_1_min/60;
221     //thoi gian cho 1 chu trinh phun
222     //bang 1000ms chia so lan phun tren giay
223     T_cycle = 1000/spray_1_sec;
224 }
225 void main(void)
226 {
227     Timer1_Init();
228     ADMUX=ADC_VREF_TYPE;
229     ADCSRA=(1<<ADEN) | (1<<ADPS1) | (1<<ADPS0);
230     lcd_init(16);
231     while (1)
232     {
233         Read_Speed_Throttle();
234         Lamda_Efficiency();
235         Calculate();
236         Top_Timer1 = (int)T_cycle*1000;
237         ICR1H = (Top_Timer1 >> 8);
238         ICR1L = (Top_Timer1 & 0xFF);
239         OCR1A = (int)(T_cycle - t_injector_1cycle)*1000;
240         //hien thi toc do
241         lcd_gotoxy(0,0);
242         lcd_puts("Sp:");
243         Display_Value(speed);
244         //hien thi goc mo buom ga
245         lcd_gotoxy(0,1);
246         lcd_puts("Thr:");
247         Display_Value(throttle);
248     }
249 }
```

12.5. Bộ so sánh tương tự

Nội dung bài thực hành: So sánh điện áp giữa chân PA0 (kết nối với quang điện trở) với chân PB2 (kết nối với biến trở). Khi không có ánh sáng chiếu vào quang điện trở, lập tức đèn LED ở chân PD0 sẽ sáng, biến trở dùng để điều chỉnh độ nhạy khi có ánh sáng thay đổi.

❖ Phản mô phỏng



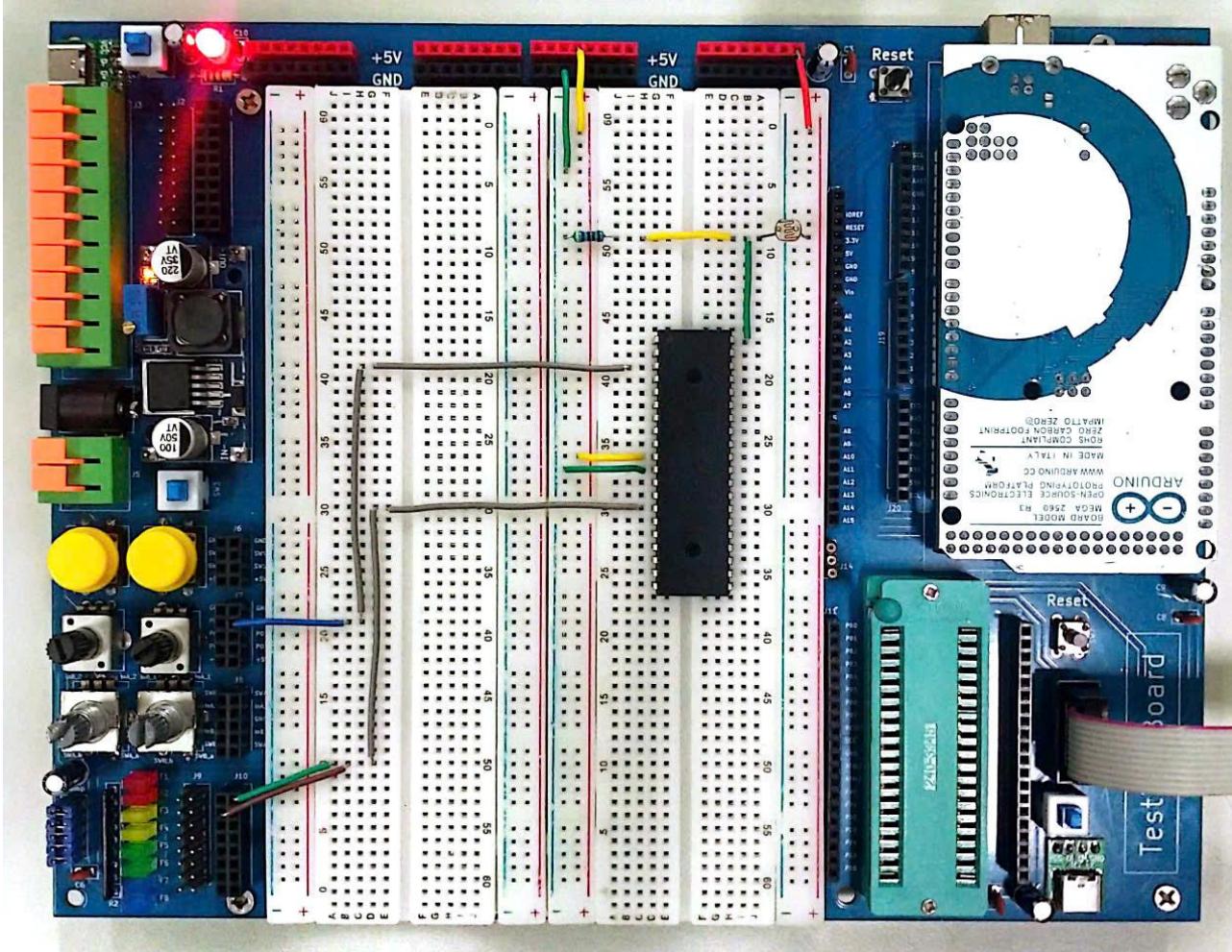
Hình 12.10. Sơ đồ kết nối bài thực hành bộ so sánh tương tự

❖ Phần lập trình

	SourceCode_ATMEGA32A\Analog_Comparator
1	#include <mega32.h>
2	void main(void)
3	{
4	//chan PD0 dung dieu khien LED
5	DDRD = (1<<DDD0); // DDRD.0 = 1
6	//chan PB2 la chan doc ADC
7	DDRB &= ~(1<<DDB2);
8	//tat che do ADC
9	ADCSRA &= ~(1<<ADEN);
10	SFIOR = (1<<ACME);
11	ACSR = 0x00;
12	while (1)
13	{
14	if(ACSR & (1<<AC0)) //neu V_AIN0 > V_AIN1
15	PORTD.0 = 0; //bat den LED
16	else

17	PORTD. 0 = 1;	<i>//tat den LED</i>
18	}	
19	}	

❖ Phản mô hình



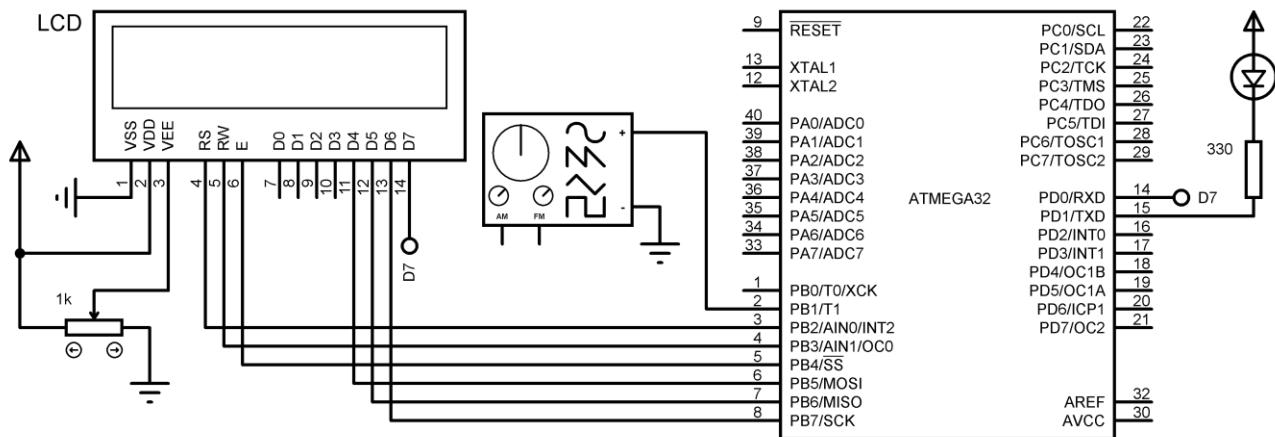
Hình 12.11. Mô hình thực tế bài thực hành bộ so sánh tương tự

12.6. Cấu trúc bộ Timer/Counter

Nội dung bài thực hành: Sử dụng bộ Timer 0 để tạo delay 100ms, nhấp nháy LED ở chân PD1, song song đó sử dụng bộ Timer 1 để đếm xung đầu vào và hiển thị giá trị đếm lên LCD, mỗi khi đếm được 5 xung thì giá trị của biến đếm (counter) sẽ tăng lên một đơn vị. Hai bộ

Timer 0 và 1 làm việc độc lập với nhau, xung đếm đi vào nhanh hay chậm đều không ảnh hưởng đến hoạt động nháy LED của Timer 0.

❖ Phần mô phỏng



Hình 12.12. Sơ đồ kết nối bài thực hành bộ định thời và bộ đếm thời gian

❖ Phần lập trình

SourceCode_ATmega32A\Delay0_Counter1	
1	#include <mega32.h>
2	#include <alcd.h>
3	#include <stdio.h>
4	#include <stdlib.h>
5	
6	unsigned char counter = 0;
7	int value = 0;
8	char myStr[5];
9	interrupt [TIM0_OVF] void timer0_overflow_isr(void)
10	{
11	TCNT0 = 60; // dat lai gia tri cho bo dem
12	if(counter > 4){
13	PORTD ^= (1<<PORTD1); // doi trang thai chan LED
14	counter = 0;

```

15    }
16    else{
17        counter++;
18    }
19}
20 interrupt [TIM1_COMPA] void timer1_compareA_isr(void)
21{
22    value++;
23    sprintf(myStr, "%d", value);
24    lcd_gotoxy(10,1);
25    lcd_puts(myStr);
26}
27 void main(void)
28{
29 lcd_init(16);
30 lcd_gotoxy(3,0);
31 lcd_puts("Counter 1");
32 lcd_gotoxy(3,1);
33 lcd_puts("Value: ");
34
35 //Delay dung Timer 0
36 DDRD.1 = 1;           // ngo ra dieu khien LED
37 PORTD.1 = 1;          // LED tat
38 //che do Normal, bo chia 1024
39 TCCR0 = (0<<WGM01)|(0<<WGM00);
40 TCCR0 = (1<<CS02)|(0<<CS01)|(1<<CS00);
41 TCNT0 = 60;           //gia tri ban dau cua bo dem
42 TIMSK |= (1<<TOIE0); // cho phep ngat khi xay ra tran
43
44 //Dem xung dung Timer 1
45 DDRB.0 = 0;           // cau hinh chan PB0 la chan nhan
46 PORTB.0 = 0;          // co dien tro keo len
47 // chon che do CTC 4, xung ngoai, xet canh xuong
48 TCCR1B = (1<<WGM12)|(1<<CS12)|(1<<CS11);

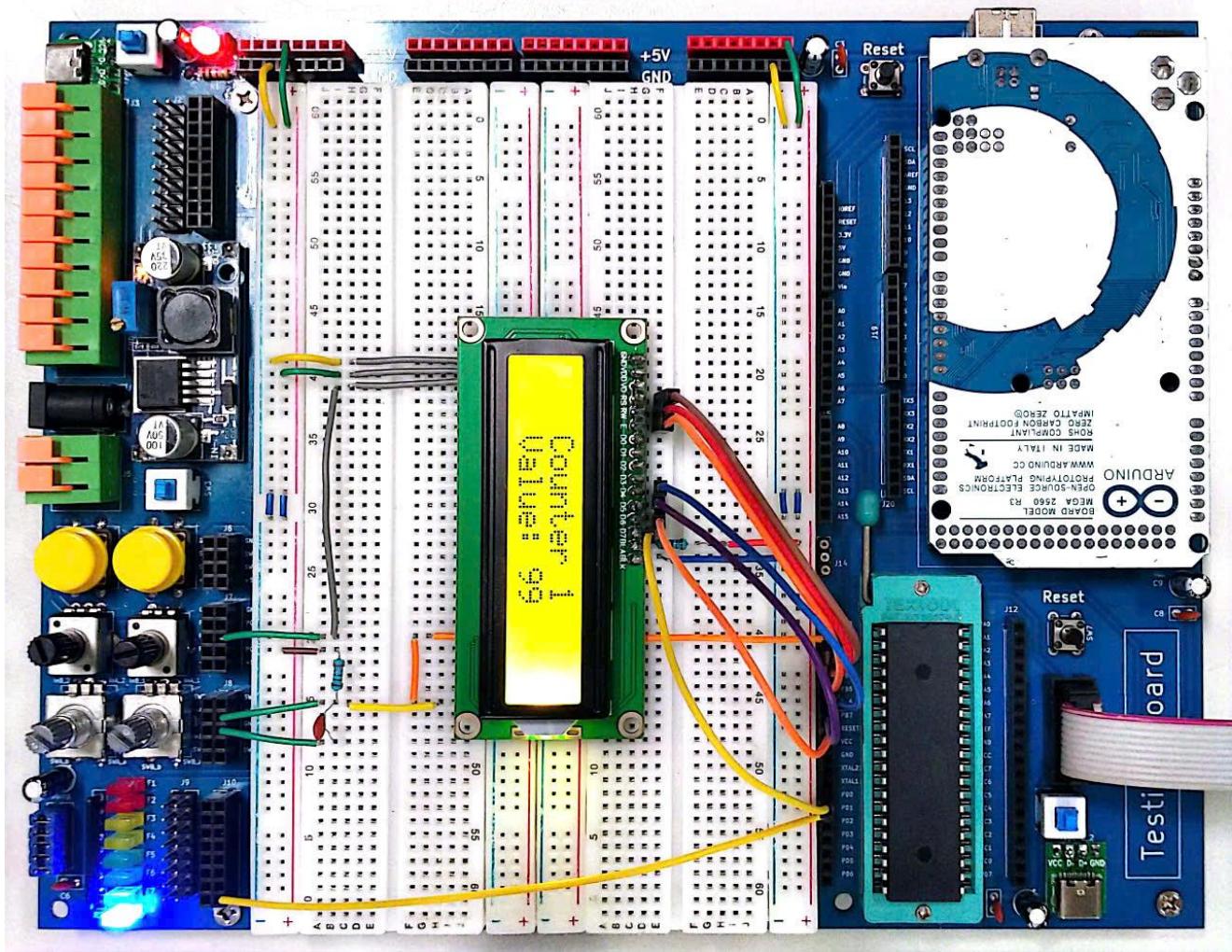
```

```

49  OCR1A = 5;                      // gia tri TOP
50  TIMSK |= (1<<OCIE1A);        // ngat khi dat gia tri so sanh
51
52 #asm("sei")                     // set bit ngat
53 while (1){}
54 }

```

❖ Phân mô hình



Hình 12.13. Mô hình thực tế bài thực hành bộ định thời và bộ đếm thời gian

12.7. Cấu trúc PWM

Nội dung bài thực hành: Điều chế độ rộng xung PWM để điều khiển góc quay của servo, ban đầu servo giữ ở góc 0° và góc quay của servo sẽ thay đổi khi nhấn nút. Loại sóng PWM được sử dụng là chế độ Fast PWM (mode 14) của bộ Timer 1.

Servo thường được điều khiển với tần số 50Hz tương ứng với chu kỳ là 20 ms, để điều khiển servo quay góc 0° thì thời gian xung mức cao phải chiếm 1 ms trên tổng chu kỳ (đây chính là Duty Cycle).

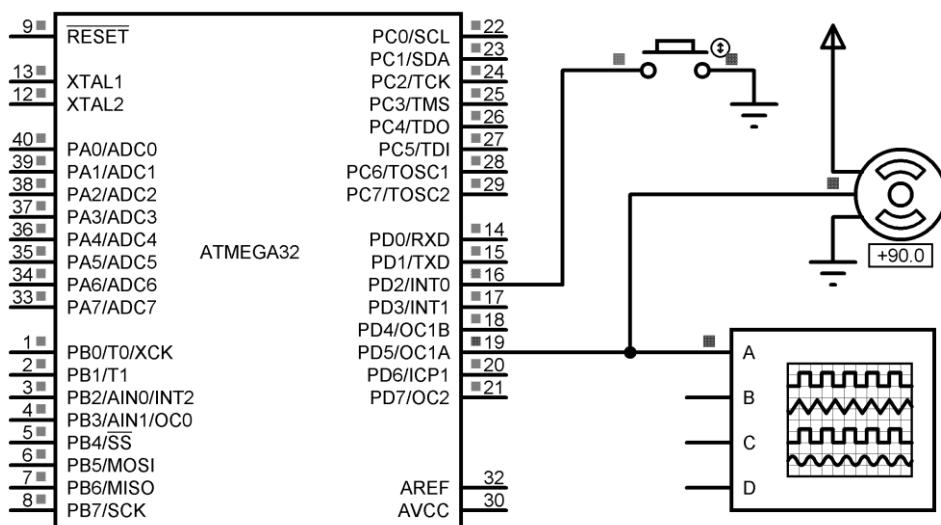
Trong bài thực hành này đang sử dụng chế độ Fast PWM và bộ chia 8 nên giá trị TOP (trong trường hợp này là thanh ghi ICR1) sẽ được tính toán dựa theo công thức (11) :

$$TOP = ICR1 = \frac{\frac{f_{clkI}}{0}}{f_{PWM} \cdot N} - 1 = \frac{8 \cdot 10^6}{50 \cdot 8} - 1 = 19\,999$$

Giá trị bộ đếm OCR1A được tính toán dựa trên góc quay mong muốn:

$$OCR1A = \frac{T_{pulse}}{T_{all}} \times ICR1 = \frac{1}{20} \times 19\,999 \approx 999$$

❖ Phản mô phỏng

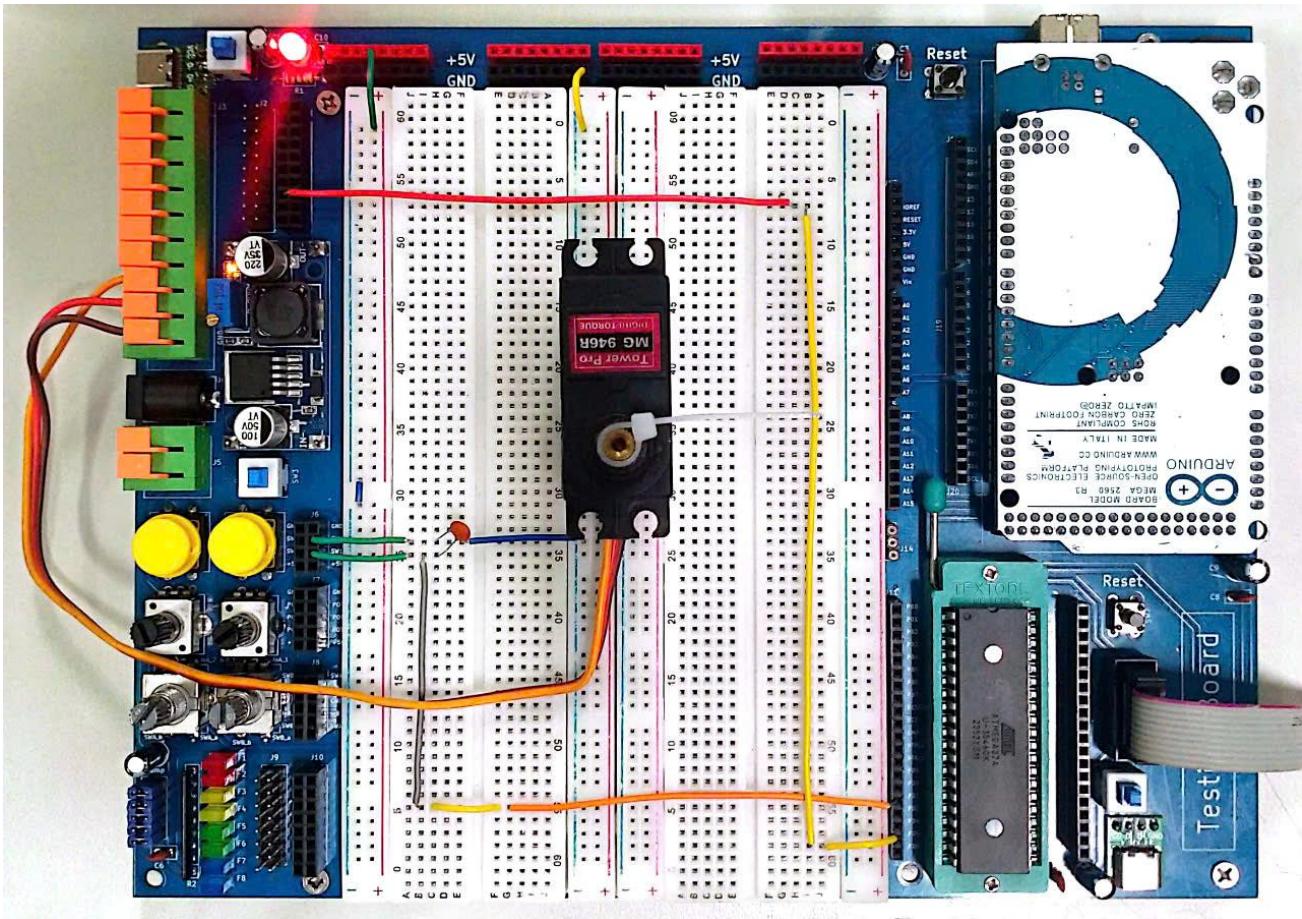


Hình 12.14. Sơ đồ kết nối bài thực hành xung PWM

❖ Phần lập trình

	SourceCode_ATMEGA32A\PWM_Timer1
1	#include <mega32.h>
2	#include <delay.h>
3	
4	unsigned long int TOP = 19999; //dat gia tri TOP cho PWM
5	interrupt [EXT_INT0] void ext_int0_isr(void)
6	{
7	// neu la 0 do thi keo len 90 do va nguoc lai
8	if(OCR1A == 999) OCR1A = 1999;
9	else OCR1A = 999;
10	}
11	void main(void)
12	{
13	DDRD = (1<<DDD5); // chan PD5 la ngo ra
14	PORTD = (1<<PORTD2); // dien tro keo len chan PD2
15	
16	// ngat 0, canh xuong
17	MCUCR = (1<<ISC01);
18	GICR = (1<<INT0);
19	
20	// chon che do 14 Fast PWM, bo chia 8
21	TCCR1A = (1<<COM1A1) (1<<WGM11);
22	TCCR1B = (1<<WGM13) (1<<WGM12) (1<<CS11);
23	
24	// do ICR1 la thanh ghi 16bit nen phai chia ra de luu tru
25	ICR1H = (TOP >> 8);
26	ICR1L = TOP & 0xFF;
27	OCR1A = 999; // gia tri ban dau - 0 do
28	#asm("sei")
29	while (1)
30	{
31	delay_ms(100);
32	}
33	}

❖ Phần mô hình



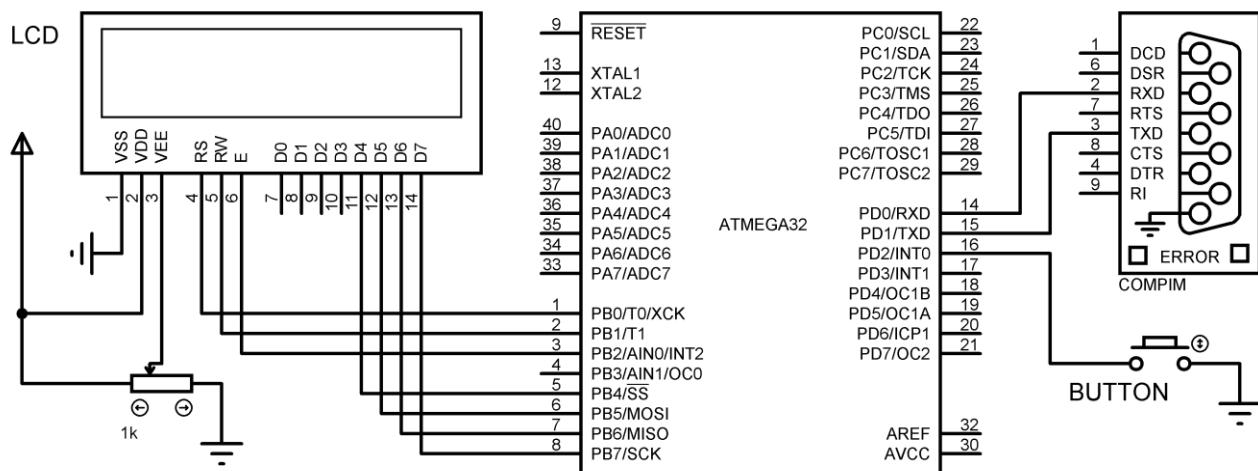
Hình 12.15. Mô hình thực tế bài thực hành xung PWM

12.8. Thiết kế giao tiếp

12.8.1. Giao tiếp UART

Nội dung bài thực hành: Giao tiếp giữa máy tính với vi điều khiển thông qua giao thức UART, gửi từng ký tự thông qua phần mềm “Hercules” đến module giao tiếp đến vi điều khiển và hiển thị lên màn hình LCD. Trong trường hợp mô phỏng, cần phải thông qua phần mềm “VSPE” để tạo cổng COM ảo mới có thể kết nối với phần mềm “Hercules”.

❖ Phân mô phỏng



Hình 12.16. Sơ đồ kết nối bài thực hành giao thức UART

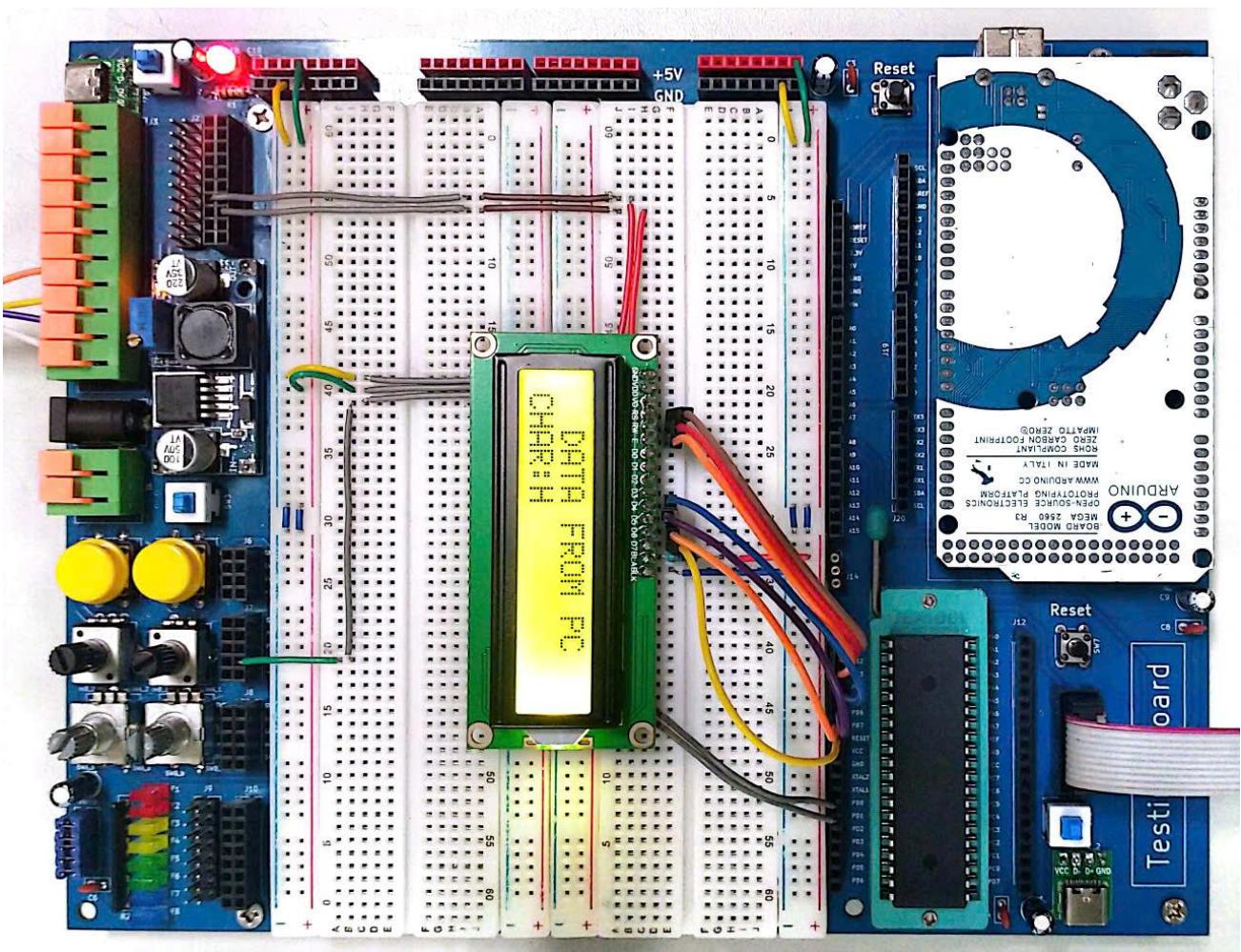
❖ Phân lập trình

	SourceCode_ATMEGA32A\UART_Protocol
1	#include <mega32.h>
2	#include <alcd.h>
3	#include <delay.h>
4	
5	volatile char receivedData;
6	void UART_CHAR_TX(unsigned char value){
7	while(!UCSRA & (1<<UDRE)){};
8	UDR = value;
9	}
10	void UART_Init() {
11	//chon toc do Baud = 9600
12	UBRRH = 0;
13	UBRRL = 51;
14	
15	//kich hoat Receiver, Transmitter, cho phep ngat khi nhan
16	UCSRB = (1 << RXEN) (1 << RXCIE) (1<<TXEN);
17	//du lieu 8 bit
18	UCSRC = (1 << URSEL) (1 << UCSZ1) (1 << UCSZ0);
19	//nut nhan tai chan INT0

```

20     GICR |= (1<<INT0);
21     MCUCR |= (1<<ISC01);
22     GIFR |= (1<<INTF0);
23 }
24 void main() {
25     DDRD &= ~(1<<DDD2);
26     PORTD |= (1<<PORTD2);
27
28     UART_Init();
29     lcd_init(16);
30
31     lcd_gotoxy(2,0);
32     lcd_putsf("DATA FROM PC");
33     lcd_gotoxy(0,1);
34     lcd_putsf("CHAR:");
35     #asm("sei")
36     while (1) {}
37 }
38 interrupt [USART_RXC] void USART_RX_Complete(void) {
39     receivedData = UDR;
40     lcd_gotoxy(5,1);
41     lcd_putchar(receivedData);
42 }
43 interrupt [EXT_INT0] void ext_int0_isr(void)
44 {
45     //neu nhan nut se gui ky tu '1' den may tinh
46     UART_CHAR_TX('1');
47 }
```

❖ Phần mô hình

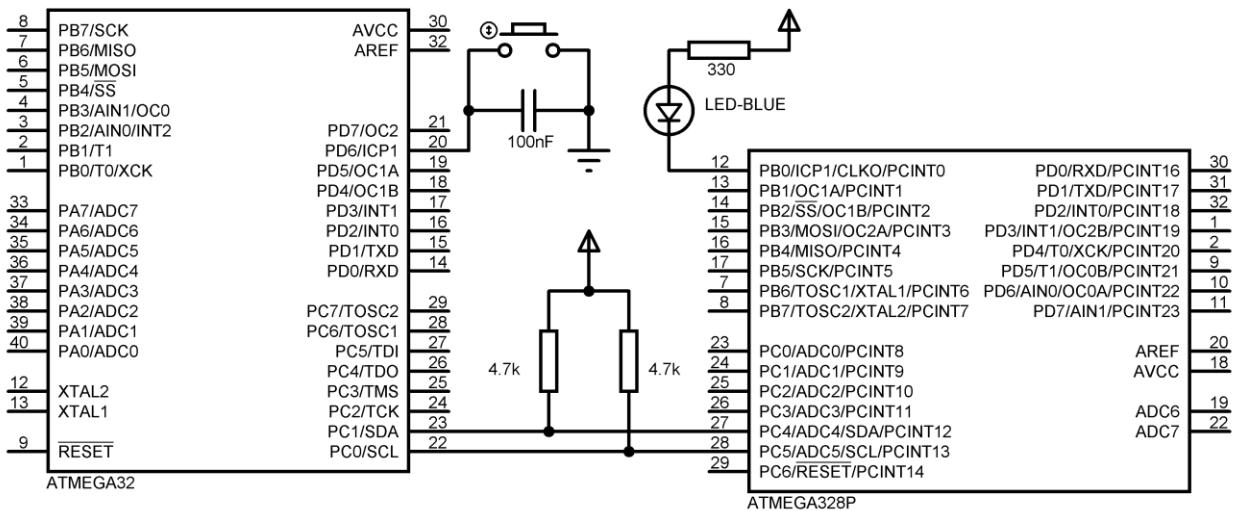


Hình 12.17. Mô hình thực tế bài thực hành UART

12.8.2. Giao tiếp I2C

Nội dung bài thực hành: Giao tiếp giữa hai vi điều khiển thông qua giao thức I2C, khi nhấn BUTTON sẽ làm thay đổi trạng thái đèn LED.

❖ Phân mô phỏng



Hình 12.18. Sơ đồ kết nối bài thực hành giao thức I2C

❖ Phân lập trình

- Chương trình dành cho MASTER.

	SourceCode_ATMEGA32A\I2C_Protocol\I2C_MASTER
1	#include <mega32.h>
2	#include <iio.h>
3	#include <delay.h>
4	
5	//dia chi cua SLAVE
6	#define SLAVE_ADDRESS 0x20
7	
8	void I2C_START() {
9	//gui tin hieu START
10	TWCR = (1 << TWSTA) (1 << TWEN) (1 << TWINT);
11	while (!(TWCR & (1 << TWINT)));
12	}
13	void I2C_STOP() {
14	//gui tin hieu STOP
15	TWCR = (1 << TWSTO) (1 << TWEN) (1 << TWINT);
16	while (TWCR & (1 << TWSTO));
17	}

```

18 void I2C_WRITE(unsigned int data) {
19     TWDR = data;
20     TWCR = (1 << TWEN) | (1 << TWINT);
21     while (!(TWCR & (1 << TWINT)));
22 }
23 void main() {
24     //nut nhan chan PD6
25     DDRD.6 = 0;
26     PORTD.6 = 1;
27     //toc do giao tiep 100 kHz
28     TWSR = 0x00;
29     TWBR = 32;
30     while (1) {
31         if (!(PIND & (1 << PIND6))) {
32             delay_ms(50); //delay chong doi nut nhan
33             if (!(PIND & (1 << PIND6))) {
34                 PORTB ^= (1<<7);
35                 I2C_START();
36                 //MASTER gui dia chi SLAVE muon giao tiep
37                 I2C_WRITE(SLAVE_ADDRESS << 1);
38                 //MASTER gui gia tri 1
39                 I2C_WRITE(0x01);
40                 I2C_STOP();
41                 delay_ms(200);
42             }
43         }
44     }
45 }
```

- Chương trình dành cho SLAVE.

	SourceCode_ATMEGA32A\I2C_Protocol\I2C_SLAVE
1	#include <mega32p.h>
2	#include <io.h>
3	#include <delay.h>
4	

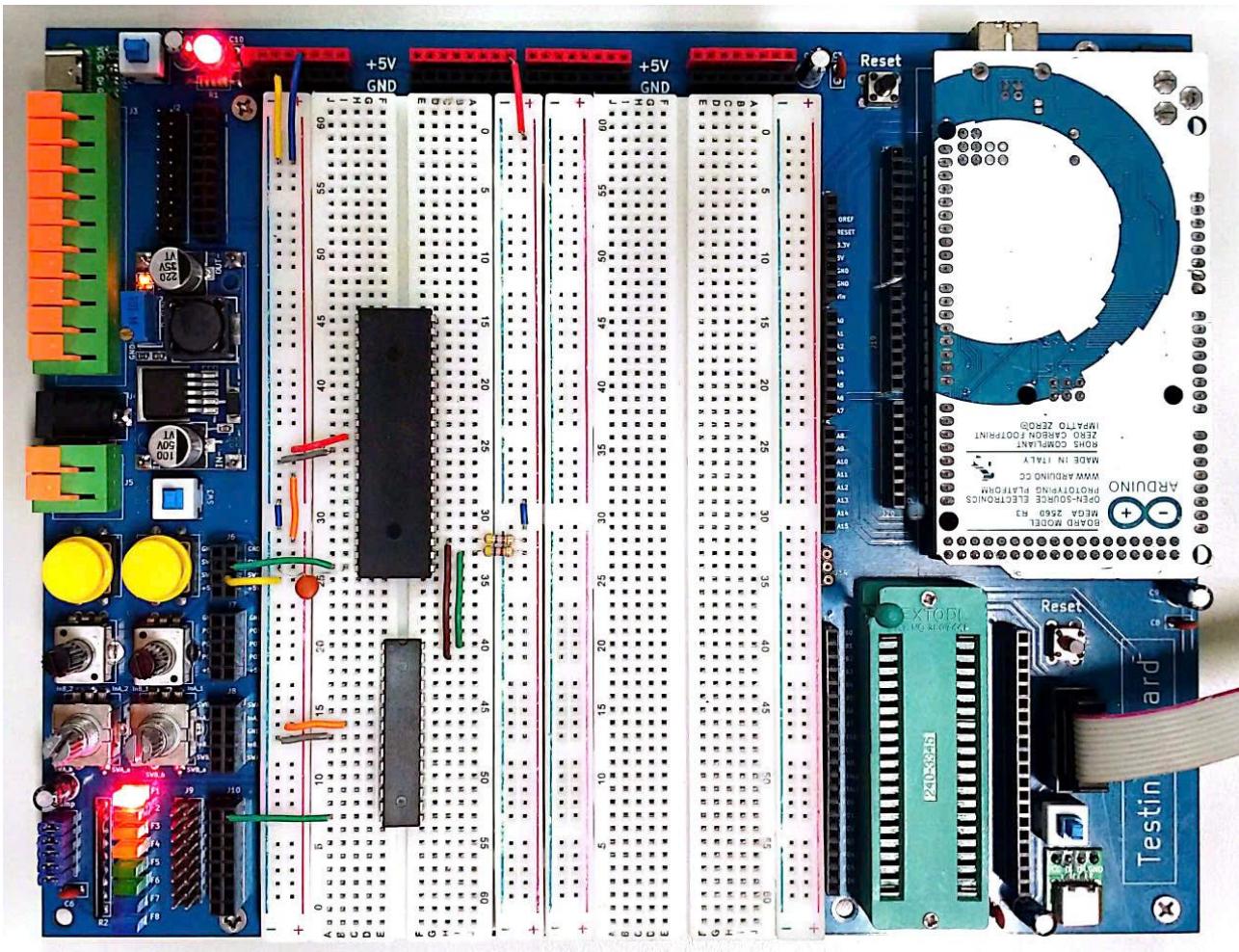
```

5 #define SLAVE_ADDRESS 0x20
6 unsigned int received_data = 0;
7 void I2C_INIT_SLAVE() {
8     //gan dia chi vao thanh ghi TWAR
9     TWAR = (SLAVE_ADDRESS << 1);
10    //kich hoat I2C, bit ACK
11    TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWINT);
12 }
13 unsigned int I2C_WAIT() {
14     while (!(TWCR & (1 << TWINT)));
15     return TWSR & 0xF8; //tra ve 5 bit cao cua TWSR
16 }
17 unsigned int I2C_RECEIVE() {
18     return TWDR;
19 }
20 void I2C_ACKNOWLEDGE() {
21     TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWINT);
22 }
23 void I2C_RESET() {
24     TWCR = (1 << TWEN) | (1 << TWINT) | (1 << TWEA);
25 }
26 void main() {
27     // Crystal Oscillator division factor: 1
28     #pragma optsize-
29     CLKPR=(1<<CLKPCE);
30     CLKPR= 0x00;
31     #ifdef _OPTIMIZE_SIZE_
32     #pragma optsize+
33     #endif
34     //khai bao chan PB0 dieu khien LED
35     DDRB |= (1 << DDB0);
36     PORTB &= ~(1 << PORTB0);
37     I2C_INIT_SLAVE();
38     while (1) {
39         if (I2C_WAIT() == 0x60) {
40             I2C_ACKNOWLEDGE();

```

```
41      }
42      if (I2C_WAIT() == 0x80) {
43          received_data = I2C_RECEIVE();
44          I2C_ACKNOWLEDGE();
45      }
46      if (I2C_WAIT() == 0xA0) {
47          I2C_RESET();
48      }
49      if (received_data == 0x01) {
50          PORTB ^= (1 << PORTB0);
51          delay_ms(50);
52          received_data = 0;
53      }
54  }
55 }
```

❖ Phần mô hình

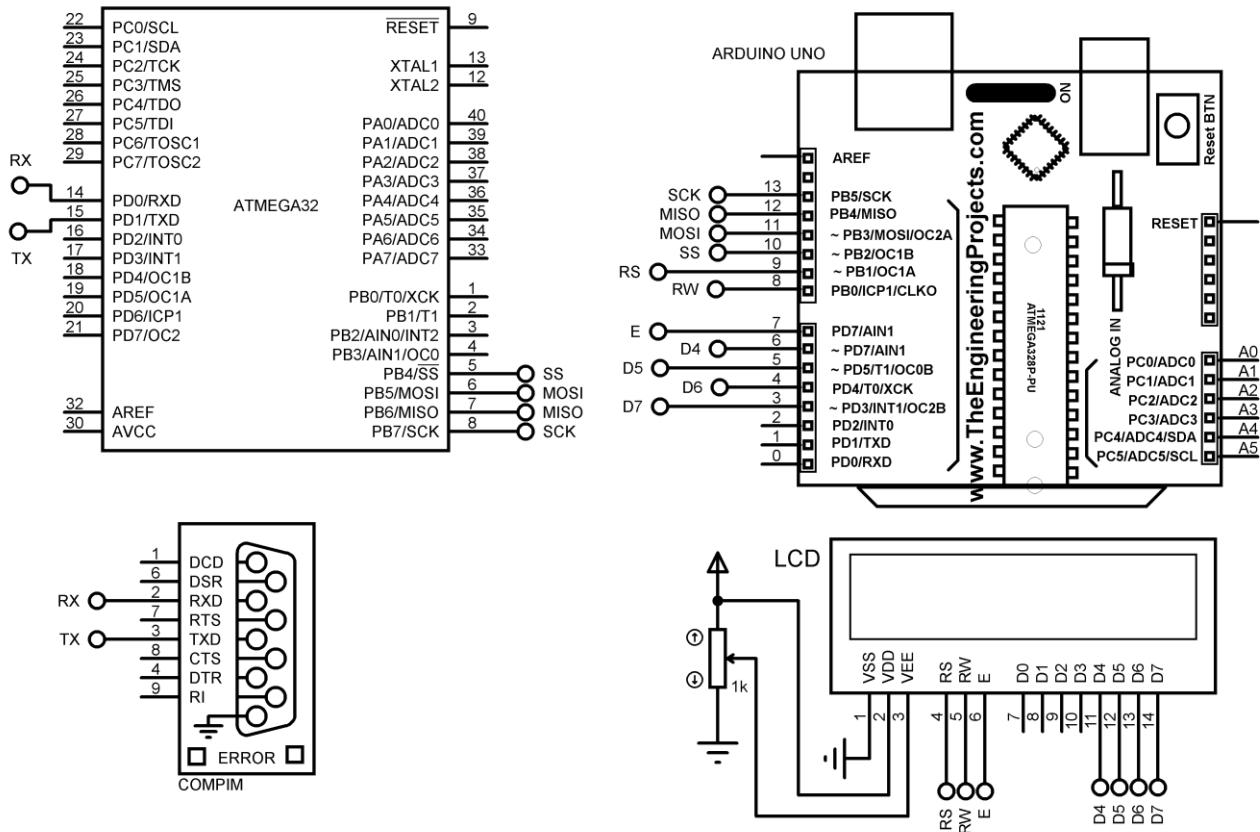


Hình 12.19. Mô hình thực tế bài thực hành I2C

12.8.3. Giao tiếp SPI

Nội dung bài thực hành: Vi điều khiển ATMEGA32 sẽ nhận dữ liệu gửi từ máy tính thông qua giao thức UART, sau đó truyền dữ liệu này đến ATMEGA328P (Arduino Uno) thông qua giao thức SPI và hiển thị lên màn hình LCD.

❖ Phần mô phỏng



Hình 12.20. Sơ đồ kết nối bài thực hành giao thức SPI

❖ Phần lập trình

- Chương trình dành cho MASTER.

	SourceCode_ATMEGA32A\SPI_Protocol\SPI_MASTER
1	#include <mega32.h>
2	#include <alcd.h>
3	
4	//cac chan cua SPI
5	#define MOSI 5
6	#define MISO 6
7	#define SCK 7
8	#define SS 4

```

9
10 //khai bao mot bien nhan du lieu
11 volatile char receivedData;
12 //viet ham khai bao SPI
13 void SPI_Master_Init(){
14     DDRB |= (1<<MOSI)|(1<<SCK)|(1<<SS); //MOSI,SCK,SS = Output
15     DDRB &= ~(1<<MISO); //MISO = Input
16     PORTB |= (1<<SS); //co dien tro keo len chan SS
17
18     //Kich hoat SPI, loai MASTER, bo chia 64
19     SPCR |= (1<<SPE)|(1<<MSTR)|(1<<SPR1)|(1<<SPR0);
20 }
21 //ham truyen du lieu cua Master qua giao thuc SPI
22 void SPI_Master_Transmit(unsigned char data){
23     SPDR = data; //ghi data vao thanh ghi du lieu
24     while( !(SPSR&(1<<SPIF)) ); //doi cho den khi gui xong
25 }
26 //viet ham khai tao UART
27 void UART_Init(){
28     //kich hoat Receiver, cho phep ngat khi nhan
29     UCSRB = (1<<RXEN)|(1<<RXCIE);
30     UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0); //du lieu 8 bit
31     //chon toc do Baud = 9600
32     UBRRH = 0;
33     UBRL = 51;
34 }
35 void main(void)
36 {
37     UART_Init();
38     SPI_Master_Init();
39     #asm("sei")
40     while (1){}
41 }
42 //ham ngat khi ngat duoc du lieu
43 interrupt [USART_RXC] void USART_RX_Complete(void){
44     receivedData = UDR;

```

45	//keo chan SS xuong truoc khi truyen data
46	PORTB &= ~(1<<SS);
47	//truyen du lieu di qua SPI
48	SPI_Master_Transmit(receivedData);
49	//keo chan SS len muc 1
50	PORTB = (1<<SS);
51	}

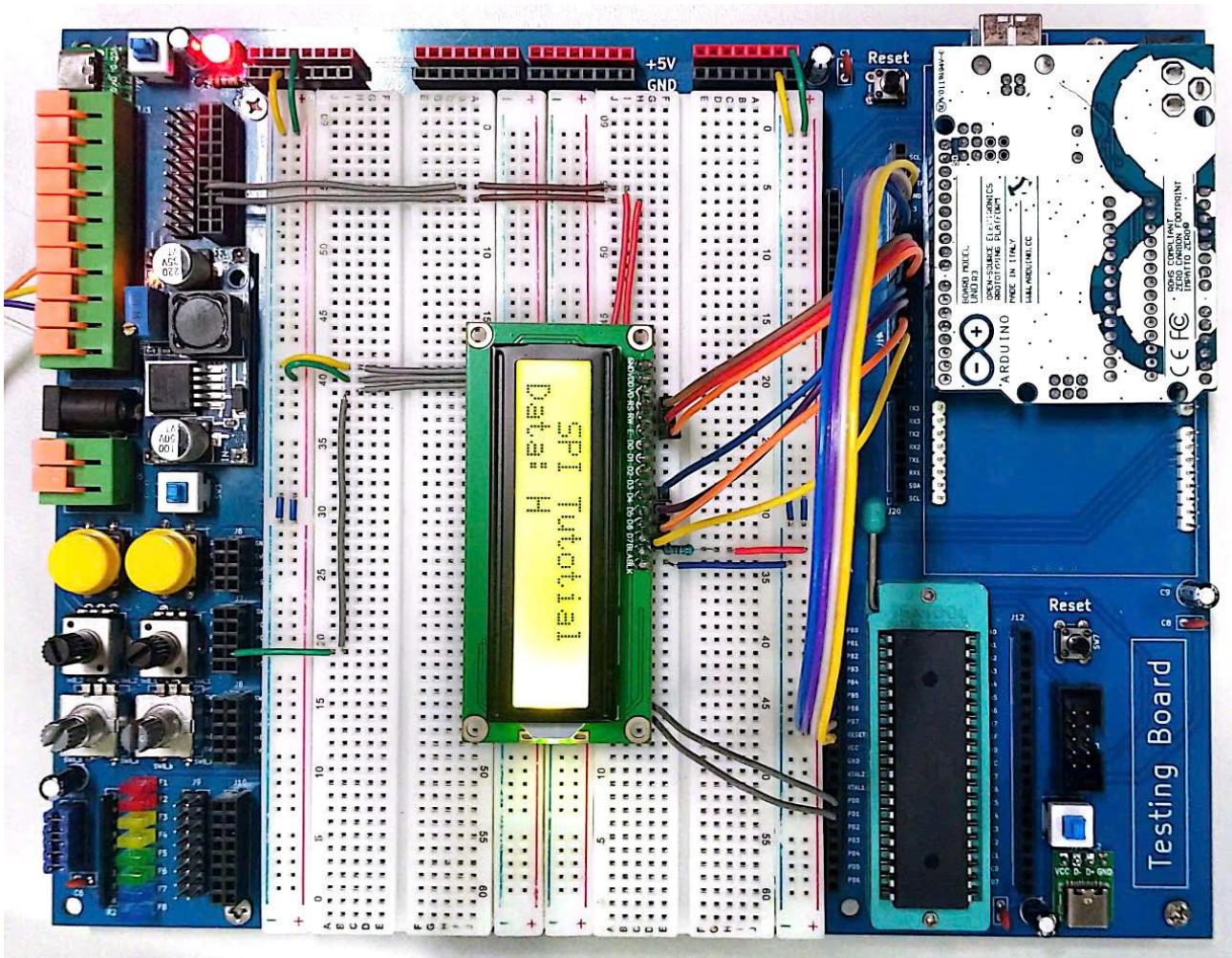
- Chương trình dành cho SLAVE.

	SourceCode_ATMEGA32A\SPI_Protocol\SPI_SLAVE
1	#include <mega328p.h>
2	#include <alcd.h>
3	
4	//cac chan dung SPI cho ATMEGA328P
5	#define MOSI 3
6	#define MISO 4
7	#define SCK 5
8	#define SS 2
9	char count = 0;
10	//ham khai tao SPI cho SLAVE
11	void SPI_Slave_Init(){
12	SPCR = (1<<SPE);
13	//khai bao cac chan SPI tuong tu Master
14	//MOSI, SCK, SS= Input
15	//MISO = Output
16	DDRB &= ~ ((1<<MOSI) (1<<SCK) (1<<SS));
17	DDRB = (1<<MISO);
18	}
19	//ham nhan du lieu
20	char SPI_Slave_Receive(){
21	//cho den khi hoan thanh nhan
22	while(!(SPSR & (1<<SPIF)));
23	return SPDR; //tra ve gia tri nhan
24	}
25	void main(void)
26	{

```

27 #pragma optsize-
28 CLKPR=(1<<CLKPCE);
29 CLKPR=(0<<CLKPCE)|(0<<CLKPS3);
30 CLKPR=(0<<CLKPS2)|(0<<CLKPS1)|(0<<CLKPS0);
31 #ifdef _OPTIMIZE_SIZE_
32 #pragma optsize+
33 #endif
34     SPI_Slave_Init();
35     lcd_init(16);
36     lcd_gotoxy(2,0);
37     lcd_putsf("SPI Tutotial");
38     while (1)
39     {
40         //gia tri nhan duoc tu MASTER
41         count = SPI_Slave_Receive();
42         lcd_gotoxy(0,1);
43         lcd_putsf("Data: ");
44         lcd_putchar(count);
45     }
46 }
```

❖ Phân mô hình



Hình 12.21. Mô hình thực tế bài thực hành SPI

CHƯƠNG 13: KẾT LUẬN VÀ KIẾN NGHỊ

13.1. Kết luận

13.1.1. Kết quả đạt được

Sau hơn 4 tháng thực hiện đồ án tốt nghiệp, nhóm đã hoàn thiện việc biên soạn tài liệu giảng dạy môn học vi điều khiển ứng dụng, nhằm hỗ trợ cho việc giảng dạy trong chuyên ngành Công nghệ Kỹ thuật Ô tô. Tài liệu được xây dựng với sự kết hợp hài hòa giữa lý thuyết nền tảng và các bài thực hành thực tế, giúp cho sinh viên dễ dàng tiếp cận và nắm vững kiến thức hơn trong quá trình học. Ngoài ra, nhóm còn quay các video bài giảng và một bộ KIT thực hành. Nội dung của các video là hướng dẫn thực hiện các bài thực hành của từng chương trong tài liệu giảng dạy và bộ KIT được sử dụng để hỗ trợ thực hiện các bài thực hành, tạo điều kiện cho sinh viên học tập một cách trực quan và sinh động hơn.

13.1.2. Hạn chế của đề tài

Mặc dù đề tài “Biên soạn tài liệu giảng dạy môn học vi điều khiển ứng dụng cho chuyên ngành Công nghệ Kỹ thuật Ô tô – Phần thực hành” đã đạt được nhiều kết quả tích cực, nhưng vẫn tồn tại một số hạn chế cần khắc phục. Trước hết, nội dung của tài liệu chủ yếu tập trung vào các kiến thức cơ bản, chưa có đi sâu vào các bài thực hành lập trình điều khiển trên xe ô tô.Thêm vào đó, bộ KIT thực hành được thiết kế chỉ đáp ứng các bài tập cơ bản và có thể chưa phù hợp với các bài phức tạp hơn. Hơn nữa, thời gian thực hiện có hạn, khiến việc thử nghiệm và đánh giá tính hiệu quả của tài liệu giảng dạy trong môi trường thực tế chưa được tiến hành một cách toàn diện.

13.2. Kiến nghị và hướng phát triển

Để hoàn thiện và phát triển đề tài hơn nữa, cần bổ sung các nội dung về mạng CAN và các giao thức truyền thông khác được sử dụng phổ biến trên ô tô. Đồng thời, nên tăng cường tính ứng dụng thực tiễn bằng cách thực hành nhóm, cho phép sinh viên lập trình trên hệ thống điều khiển thực tế trên ô tô. Ngoài ra, việc kèm theo tài liệu hướng dẫn chi tiết về xử lý lỗi và giải quyết các tình huống thường gặp trong quá trình thực hành sẽ hỗ trợ hiệu quả cho sinh

viên trong việc tự học và nâng cao kỹ năng. Để đáp ứng tốt hơn nhu cầu đào tạo, cần tiếp tục cài tiến bộ KIT thực hành, không chỉ để thực hiện các bài tập nâng cao mà còn để mở rộng khả năng sử dụng với nhiều dòng vi điều khiển khác nhau, góp phần tăng tính linh hoạt và hiệu quả trong giảng dạy.

DANH MỤC TÀI LIỆU THAM KHẢO

- [1] “ATmega328 với Arduino Optiboot (Arduino Uno) - Open Circuit.” [Online]. Available: <https://opencircuit.fr/produit/atmega328-with-arduino-optiboot-arduino-uno>
- [2] Atmel, *ATmega328P 8-bit AVR Microcontroller*. 1998.
- [3] Microchip, *ATmega328/P AVR ® Microcontroller with picoPower ® Technology*. 2018.
- [4] “Cấu Trúc AVR | Cùng học AVR (AVR tutorial).” [Online]. Available: <https://www.hocavr.com/2018/06/bai-2-cau-truc-avr.html>
- [5] “C cho AVR | Cùng học AVR (AVR tutorial).” [Online]. Available: <https://www.hocavr.com/2018/06/c-cho-avr.html>
- [6] “Interfacing to 7-Segment Displays.” [Online]. Available: <https://canadu.com/lp/doc/7seg/7seg.html>
- [7] “CGROM - Crystalfontz LCD Glossary.” [Online]. Available: <https://www.crystalfontz.com/blog/glossary/cgrom/>
- [8] “Text LCD | Cùng học AVR (AVR tutorial).” [Online]. Available: <https://www.hocavr.com/2018/06/text-lcd.html>
- [9] “Ngắt ngoài | Cùng học AVR (AVR tutorial).” [Online]. Available: <https://www.hocavr.com/2018/06/bai-3-ngat-ngoai.htm>
- [10] “Bộ điều khiển PID là gì? Nguyên lý hoạt động của PID.” [Online]. Available: <https://binhduongaec.com.vn/tin-tuc/bo-dieu-khien-pid/>
- [11] “CLT engine temperature sensor: a) view, b) characteristics of... | Download Scientific Diagram.” [Online]. Available: https://www.researchgate.net/figure/CLT-engine-temperature-sensor-a-view-b-characteristics-of-resistance-changes-under_fig3_330394009
- [12] “Timer - Counter | Cùng học AVR (AVR tutorial).” [Online]. Available: <https://www.hocavr.com/2018/06/bai-4-timer-counter.html>
- [13] “Giao tiếp SPI | Cùng học AVR (AVR tutorial).” [Online]. Available: <https://www.hocavr.com/2018/06/bai-7-giao-tiep-spi.html>