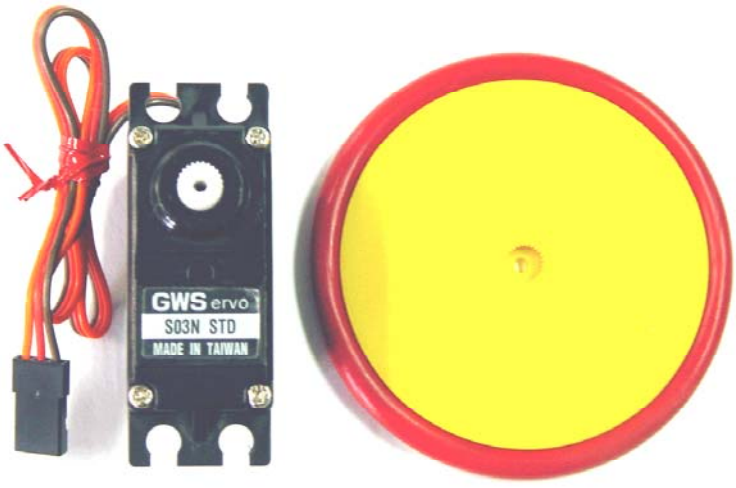
การควบคุมการเคลื่อนที่ของรถหุ่นยนต์

การควบคุมการเคลื่อนที่ของรถหุ่นยนต์ ET-ROBOT RD2

อุปกรณ์สำหรับขับเคลื่อนตัวรถหุ่นยนต์นั้น นับว่าเป็นอุปกรณ์ที่มีความสำคัญมาก ทั้งนี้ก็เนื่องมาจากว่าการที่จะทำให้ตัวรถสามารถขับเคลื่อนไปในทิศทางต่างๆได้นั้น จะต้องอาศัยตัวขับเคลื่อนพาไป สำหรับอุปกรณ์ที่ใช้ในการขับเคลื่อนตัวรถหุ่นยนต์ของ ETROBOT RD2 นั้น จะอาศัยมอเตอร์ แบบ DC SERVO MOTOR เป็นตัวขับเคลื่อน ซึ่งการที่รถจะสามารถเคลื่อนที่ไปในทิศทางต่างๆได้นั้นก็จะขึ้นอยู่กับทิศทางการหมุนของ DC SERVO MOTOR เป็นหลัก ด้วยเหตุนี้เอง ในอันดับแรกก่อนที่จะเริ่มต้นทำการเขียนโปรแกรมเพื่อสั่งงานให้ DC SERVO MOTOR หมุนได้นั้น เราจำเป็นจะต้องทราบและเข้าใจถึงหลักการควบคุมมอเตอร์ชนิดนี้เสียก่อน สำหรับวิธีการควบคุมการทำงานของ DC SERVO MOTOR นั้นมีรายละเอียดดังต่อไปนี้

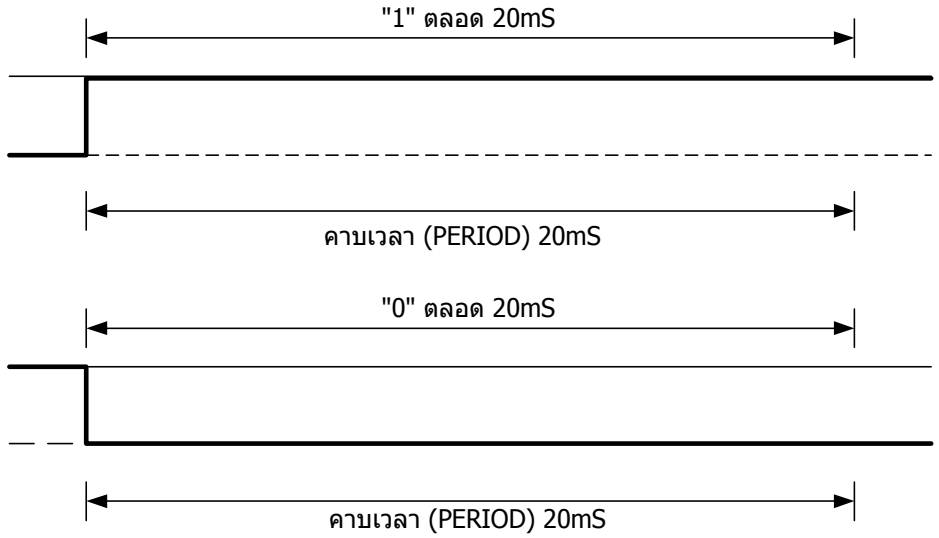
หลักการทำงานของ DC SERVO MOTOR

ตามปรกติทั่วๆไปแล้วนั้น DC SERVO MOTOR นั้น จะสามารถหมุนไปในทิศทางตามเข็มนาฬิกาและทวนเข็มนาฬิกาได้เพียงแค่ 180 องศา หรือ ครึ่งวงกลมเท่านั้น ซึ่งวิธีการสั่งงานให้ SERVO หมุนไปในตำแหน่งใดๆนั้น จะอาศัยลักษณะของสัญญาณ Pulse เป็นตัวบ่งบอก



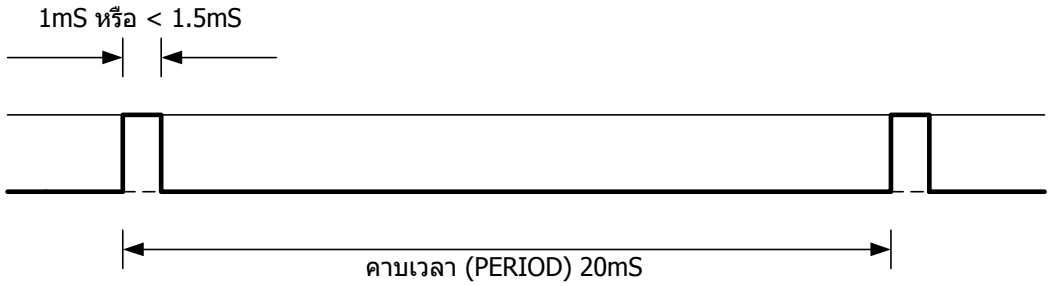
แต่สำหรับ DC SERVO MOTOR ที่เรานำมาใช้ในการขับเคลื่อนล้อเพื่อนำพาตัวรถให้เคลื่อนที่ไปในทิศทางต่างๆได้นั้น จะต้องทำการดัดแปลงหรือ Modify ให้สามารถหมุนได้รอบตัวหรือ 360 องศา เสียก่อน โดย DC SERVO MOTOR ที่ได้จัดให้ในชุดของ ET-ROBOT RD2 นั้น ได้ผ่านการปรับแต่งการทำงานของมอเตอร์ให้สามารถหมุนเป็นวงรอบ (360 องศา) ได้เป็นที่เรียบร้อยแล้วโดยวิธีในการควบคุมให้มอเตอร์ซึ่งทำการดัดแปลงแล้วให้หมุนไปในทิศทางต่างๆนั้น จะมีลักษณะดังนี้

- การควบคุมให้มอเตอร์หยุดหมุน ทำได้โดยการส่งโลจิก “0” หรือ “1” ให้กับมอเตอร์ ตลอดคาบเวลาที่ต้องการให้มอเตอร์หยุดหมุน ซึ่งก็คือการไม่จ่ายสัญญาณ Pulse ให้กับมอเตอร์นั่นเอง



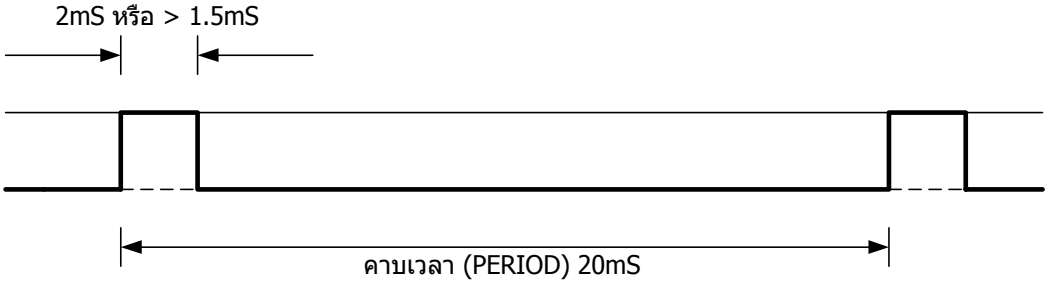
รูปแสดง ลักษณะของ Pulse สำหรับควบคุมให้มอเตอร์หยุดหมุน(STOP)

- การควบคุมให้มอเตอร์หมุนทางด้านซ้ายหรือหมุนตามทิศทางทวนเข็มนาฬิกา จะต้องป้อนสัญญาณ Pulse ที่มีขนาดความกว้างด้านบวก 1 mS หรือ ให้น้อยกว่า 1.5 ms โดยจะต้องป้อนสัญญาณ Pulse นี้ทุกๆ 20 mS (หรือในช่วงประมาณ 20ms – 30ms) เพื่อให้มอเตอร์หมุนต่อเนื่องไปในทิศทางเดิม



รูปแสดง ลักษณะของ Pulse สำหรับควบคุมให้มอเตอร์หมุนทวนเข็มนาฬิกา

- การควบคุมให้มอเตอร์หมุนทางด้านขวาหรือทิศทางตามเข็มนาฬิกา จะต้องป้อนสัญญาณ Pulse ที่มีขนาดความกว้างด้านบวก 2 mS หรือ ไม่ต่ำกว่า 1.5 mS และจะต้องป้อนสัญญาณ Pulseนี้ ทุกๆ 20 mS (หรือในช่วงประมาณ 20ms – 30ms) เพื่อควบคุมให้มอเตอร์หมุนต่อเนื่องไปในทิศทางเดิม



รูปแสดง ลักษณะของ Pulse สำหรับควบคุมให้มอเตอร์หมุนตามเข็มนาฬิกา

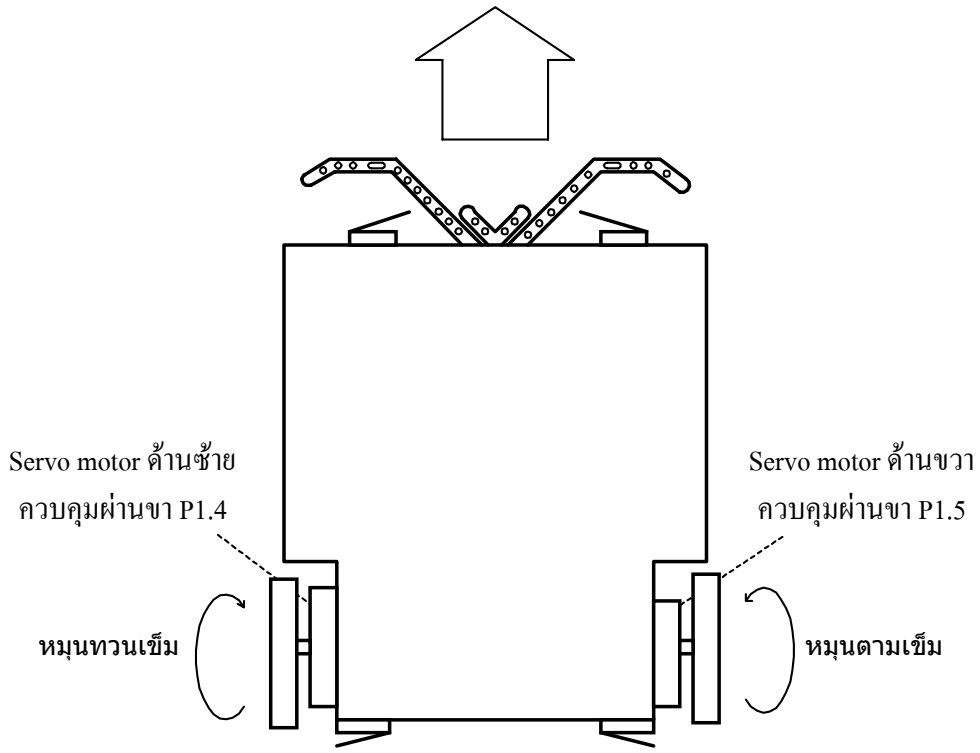
ซึ่งจะเห็นได้ว่าในการควบคุมการทำงานของมอเตอร์นั้นจะใช้วิธีการสร้างสัญญาณ Pulse เพื่อส่งไปบังคับตัวมอเตอร์ให้หมุนไปในทิศทางต่างๆตามต้องการ ซึ่งในกรณีของรถหุ่นยนต์ ET-ROBOT RD2 นั้นจะต้องทำการสร้างสัญญาณ Pulse ซึ่งมีขนาดความกว้างของสัญญาณด้านบวก 1mS หรือ 2mS เพื่อควบคุมให้มอเตอร์หมุนไปในทิศทางทวนเข็มนาฬิกาหรือตามเข็มนาฬิกาตามต้องการ และเมื่อต้องการให้มอเตอร์ทำการขับเคลื่อนล้อเพื่อนำพาตัวรถให้เคลื่อนที่ไปในทิศทางเดิมต่อเนื่องกันไปนั้นก็จะต้องทำการส่งสัญญาณ Pulse แบบเดียวกันซ้ำๆออกไปภายในเวลาประมาณทุกๆ 20mS ด้วยเสมอ

ซึ่งจากคุณสมบัติของสัญญาณ Pulse ดังกล่าวข้างต้นนั้น ก็คือ ลักษณะของสัญญาณ Pulse Width Modulation หรือ PWM นั่นเอง ดังนั้นจะได้ว่าเมื่อต้องการให้มอเตอร์หมุนก็จะต้องทำการส่งสัญญาณ Pulse Width Modulation หรือ PWM ซึ่งมีคาบเวลา 20mS ออกไปให้กับมอเตอร์ ซึ่งก็คือค่า Period ของสัญญาณ PWM นั่นเอง ส่วนทิศทางการหมุนของมอเตอร์นั้นจะกำหนดจากความกว้างของสัญญาณ Pulse ในขณะที่เป็นบวกอยู่ ซึ่งก็คือค่า Duty Cycle ของสัญญาณ PWM นั่นเอง ดังนั้นเมื่อต้องการควบคุมการหมุนของ DC SERVO MOTOR ด้วยสัญญาณ PWM ก็สามารถสรุปได้ว่า

- เมื่อต้องการให้มอเตอร์หมุนไปในทิศทางตามเข็มนาฬิกา ก็จะต้องสร้างสัญญาณ PWM ที่มีค่า Period ขนาด 20mS โดยมีค่า Duty Cycle ของสัญญาณ 1mS
- เมื่อต้องการให้มอเตอร์หมุนไปในทิศทางทวนเข็มนาฬิกา ก็จะต้องสร้างสัญญาณ PWM ที่มีค่า Period ขนาด 20mS โดยมีค่า Duty Cycle ของสัญญาณ 2mS
- เมื่อต้องการให้มอเตอร์หยุดหมุนก็จะต้องทำการหยุดการสร้างสัญญาณ PWM ซึ่งทำได้โดยการส่งสัญญาณที่มีสภาวะเป็นโลจิก “0” หรือ “1” ให้กับตัวมอเตอร์ตลอดเวลาก็ได้

การควบคุมการเคลื่อนที่ของตัวรถหุ่นยนต์

เราได้ศึกษาถึงวิธีการควบคุมการทำงานของ DC SERVO MOTOR ในการบังคับล้อให้หมุนไปในทิศทางตามเข็มนาฬิกา หรือ ทวนเข็มนาฬิกา หรือหยุดหมุน กันมาแล้ว ซึ่งลำดับต่อไปนี้เราจะมาทำความเข้าใจเกี่ยวกับวิธีการควบคุมทิศทางการเคลื่อนที่ของตัวรถหุ่นยนต์ให้เคลื่อนที่ไปในทิศทางต่างๆไม่ว่าจะเป็นเดินหน้า ถอยหลัง เลี้ยวซ้าย หรือเลี้ยวขวา เป็นต้น



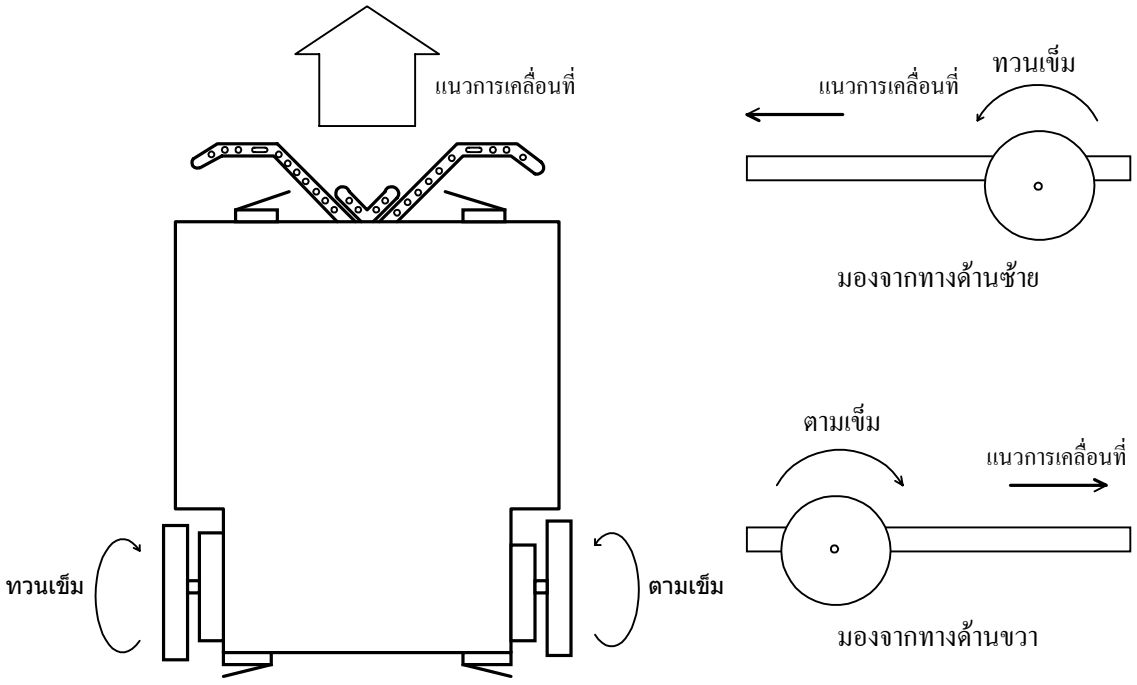
รูปแสดง ลักษณะการเคลื่อนที่ของรถหุ่นยนต์ โดยการขับเคลื่อนจาก DC SERVO MOTOR

จากรูปจะเห็นได้ว่า ลักษณะการติดตั้งตัว DC SERVO MOTOR เข้ากับล้อของรถหุ่นยนต์นั้น จะสังเกตุเห็นได้ว่า ตัวมอเตอร์จะถูกจัดวางอยู่ในทิศทางที่ตรงกันข้าม ดังนั้นในการควบคุมการเคลื่อนที่ของตัวรถหุ่นยนต์ให้เคลื่อนที่ไปในทิศทางเดียวกันนั้น ตัวมอเตอร์ที่อยู่ทางด้ายซ้ายและขวาจะต้องทำงานในทิศทางที่ตรงกันข้ามกัน โดยมอเตอร์ที่ติดตั้งอยู่ทางด้านล้อขวาจะใช้วิธีการควบคุมแบบปรกติ ส่วนมอเตอร์ที่ติดตั้งอยู่กับล้อด้านซ้ายจะใช้วิธีการควบคุมแบบกลับทางหรือตรงกันข้ามกับทิศทางการเคลื่อนที่ที่ต้องการ ตัวอย่างเช่น เมื่อต้องการให้ตัวรถขับเคลื่อนไปข้างหน้าก็จะต้องบังคับให้มอเตอร์ตัวที่บังคับการหมุนของล้อทางด้านขวาหมุนไปในทิศทางตามเข็มนาฬิกาเพื่อบังคับให้ล้อขวาหมุนไปข้างหน้า ส่วนมอเตอร์ตัวที่ทำหน้าที่บังคับการหมุนของล้อด้านซ้ายก็จะต้องถูกบังคับให้หมุนไปในทิศทางตรงกันข้ามคือทวนเข็มนาฬิกาเพื่อบังคับให้ล้อด้านซ้ายหมุนไปข้างหน้าเป็นต้น

สำหรับวิธีการบังคับเลี้ยวของรถหุ่นยนต์นั้นจะแตกต่างจากการบังคับเลี้ยวของรถยนต์ที่เราพบเห็นอยู่ทั่วไปในชีวิตประจำวัน เนื่องจากรถหุ่นยนต์ไม่มีระบบพวงมาลัยในการบังคับล้อให้เลี้ยวซ้ายหรือขวาตามต้องการเหมือนรถยนต์ทั่วๆไป แต่อย่างไรก็ตามเราสามารถทำการบังคับให้รถหุ่นยนต์ทำการเลี้ยวไปในทิศทางที่เราต้องการได้เช่นเดียวกัน โดยใช้วิธีการควบคุมการหมุนของล้อหลังซึ่งเป็นตัวขับเคลื่อนตัวรถ เช่นเมื่อต้องการให้รถหุ่นยนต์เลี้ยวไปทางด้านซ้าย ก็สามารถทำได้โดยการบังคับให้ล้อซ้ายหยุดหมุนส่วนล้อขวาก็ให้เคลื่อนที่ไปข้างหน้าตามปรกติ หรือถ้าต้องการให้รถเลี้ยวซ้ายอย่างรวดเร็วก็สามารถทำได้โดยการบังคับให้ล้อซ้ายหมุนถอยหลังส่วนล้อขวาก็ให้หมุนไปข้างหน้าอย่างนี้เป็นต้น

การบังคับให้รถหุ่นยนต์เคลื่อนที่ไปข้างหน้า

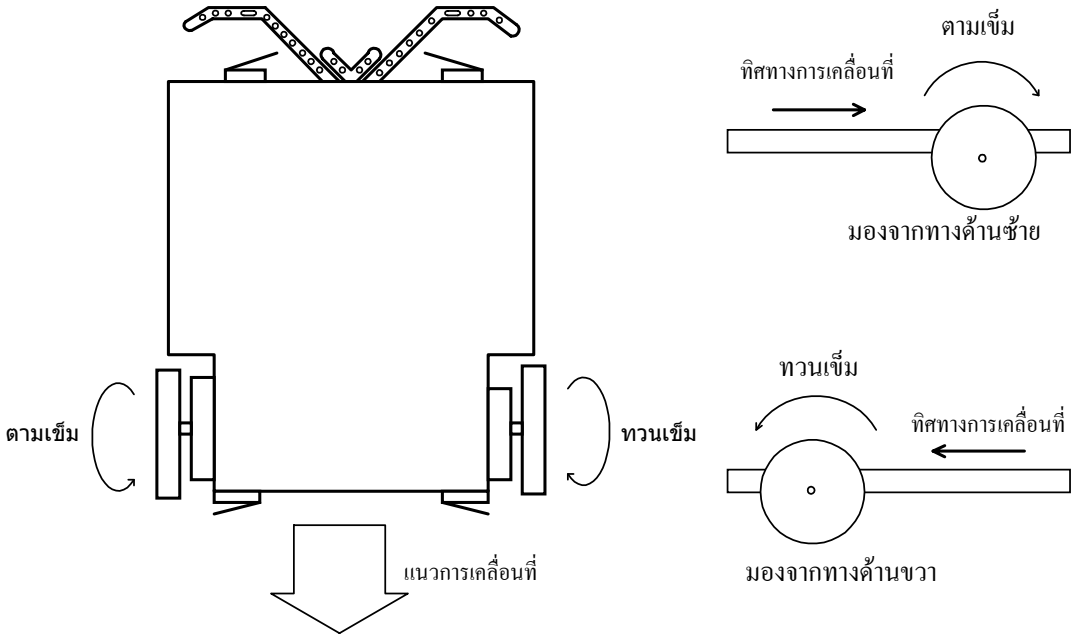
สำหรับวิธีการบังคับให้รถหุ่นยนต์เคลื่อนที่ไปข้างหน้านั้น สามารถทำได้โดยการบังคับให้มอเตอร์ที่ใช้ควบคุมการหมุนของล้อซ้ายหมุน แบบทวนเข็มนาฬิกาหรือถอยหลัง (ส่ง Pulse ความกว้าง 2 mS) ส่วนมอเตอร์ที่ใช้ควบคุมการหมุนของล้อขวาก็ต้องบังคับให้หมุนแบบตามเข็มนาฬิกาหรือเดินหน้า (ส่ง Pulse ความกว้าง 1 mS) ดังรูป



รูปแสดงการบังคับให้รถหุ่นยนต์เคลื่อนที่ไปข้างหน้า(เดินหน้า)

การบังคับให้รถหุ่นยนต์เคลื่อนที่ไปข้างหลัง

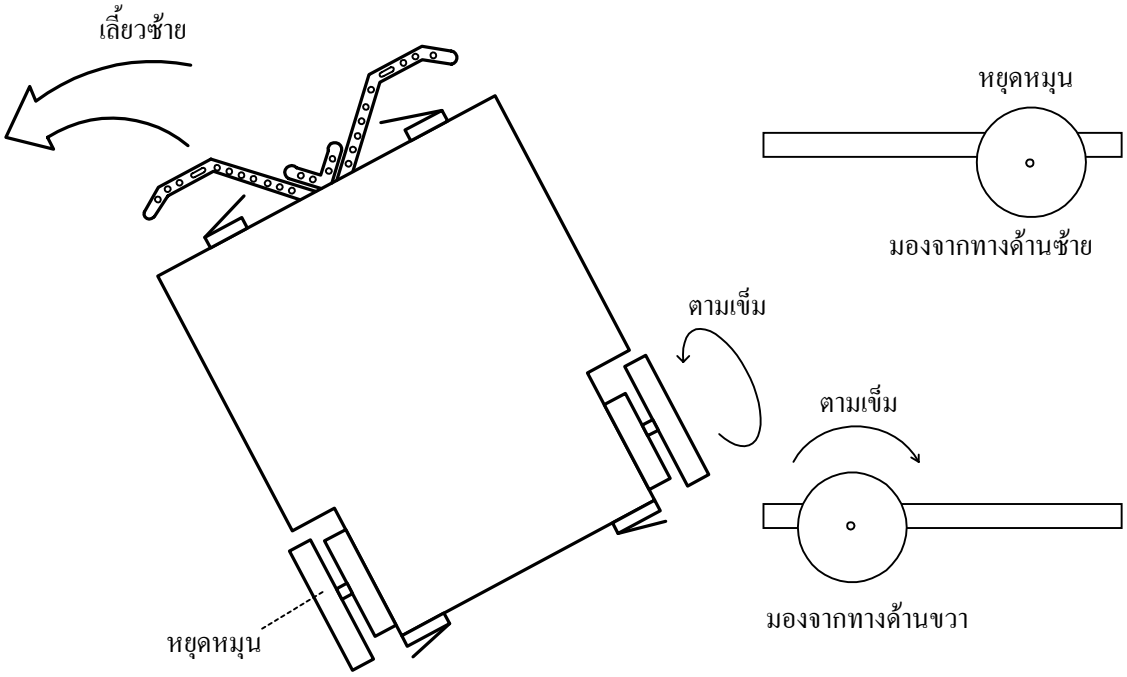
สำหรับวิธีการบังคับให้รถหุ่นยนต์เคลื่อนที่ไปข้างหลังนั้นจะมีวิธีการตรงกันข้ามกับการเคลื่อนที่ไปข้างหน้า ซึ่งก็คือการกลับทิศทางการหมุนของมอเตอร์ทั้งสองตัวให้เป็นตรงข้ามกับการเคลื่อนที่ไปข้างหน้านั่นเอง ซึ่งสามารถทำได้โดยการบังคับให้มอเตอร์ที่ใช้ควบคุมการหมุนของล้อซ้ายหมุน แบบตามเข็มนาฬิกาหรือเดินหน้า (ส่ง Pulse ความกว้าง 1 mS) ส่วนมอเตอร์ที่ใช้ควบคุมการหมุนของล้อขวาก็ต้องบังคับให้หมุนแบบทวนเข็มนาฬิกาหรือถอยหลัง (ส่ง Pulse ความกว้าง 2 mS)



รูปแสดงการบังคับให้รถหุ่นยนต์เคลื่อนที่ไปข้างหลัง(ถอยหลัง)

การบังคับให้รถหุ่นยนต์เลี้ยวซ้ายแบบปรกติ

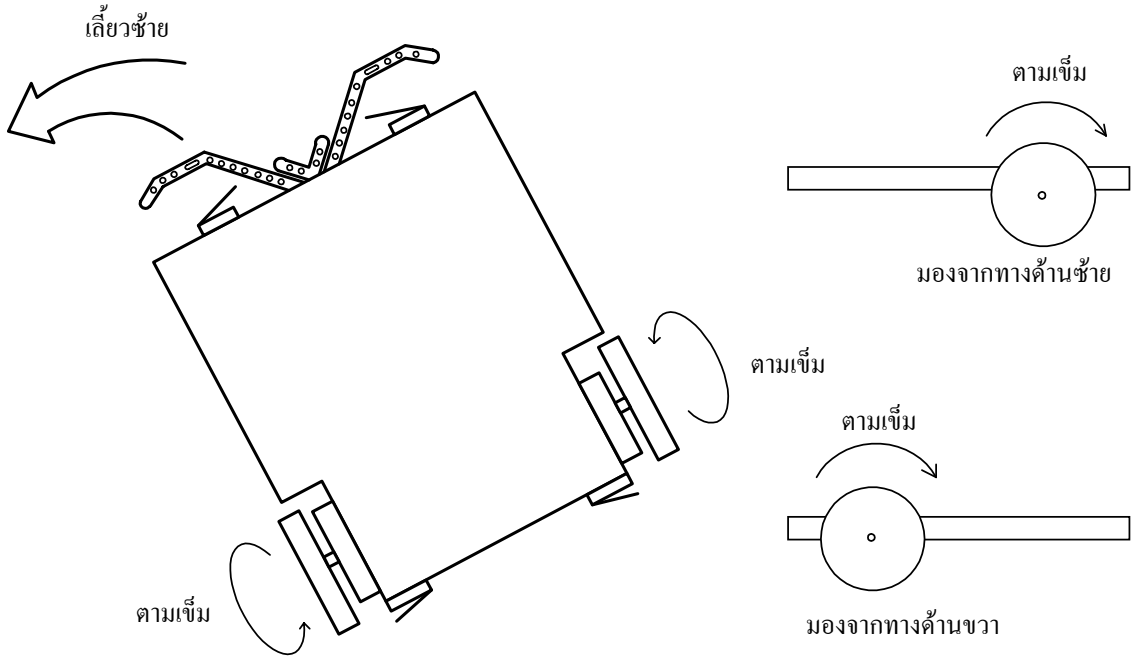
สำหรับวิธีการบังคับให้ตัวรถหุ่นยนต์เลี้ยวซ้ายแบบนี้ จะอาศัยล้อซ้ายเป็นจุดหมุนของการเลี้ยวซ้าย โดยวิธีการในการบังคับให้รถหุ่นยนต์เลี้ยวซ้ายแบบนี้ จะต้องทำการหยุดการหมุนของล้อทางด้านซ้าย ส่วนล้อทางด้านขวาก็ปล่อยให้หมุนไปข้างหน้าตามปรกติ ดังรูป



รูปแสดงการบังคับให้รถหุ่นยนต์เลี้ยวซ้ายโดยการเคลื่อนที่ของล้อขวา อย่างเดียว

การบังคับให้รถหุ่นยนต์เลี้ยวซ้ายแบบรวดเร็ว

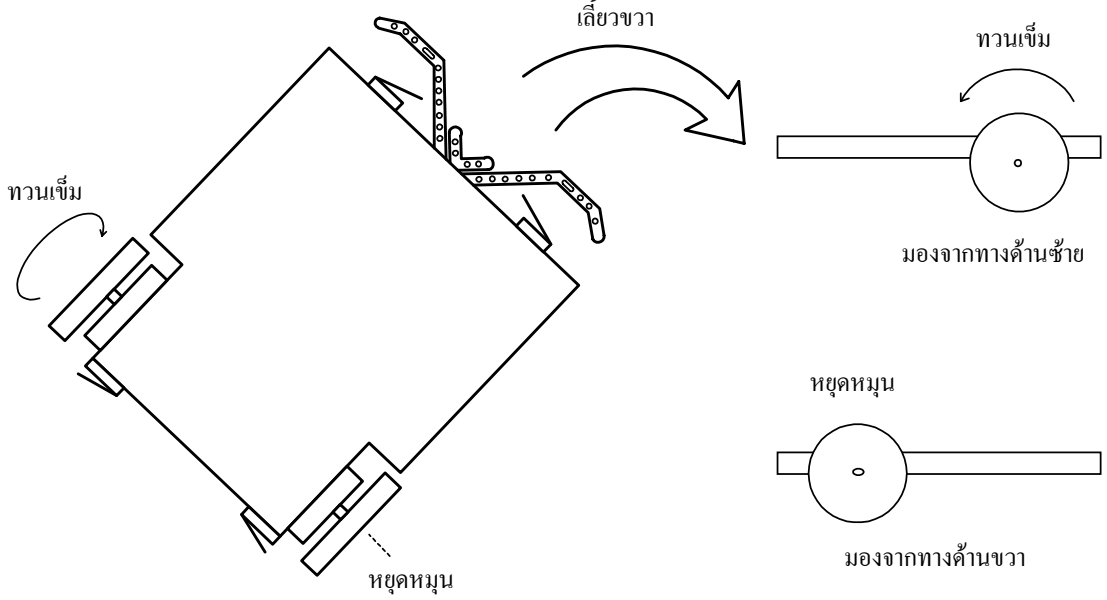
สำหรับวิธีการบังคับเลี้ยวแบบนี้ สามารถกระทำได้โดยการบังคับให้ล้อทางด้านซ้ายและขวาหมุนไปในทิศทางที่ตรงกันข้าม กล่าวคือ ล้อทางด้านซ้ายจะถูกบังคับให้หมุนถอยหลัง ส่วนล้อทางด้านขวาจะถูกบังคับให้เดินไปทางหน้า ดังรูป



รูปแสดงการบังคับให้รถหุ่นยนต์เลี้ยวซ้ายโดยให้ล้อซ้ายถอยหลังส่วนล้อขวาเดินหน้า

การบังคับให้รถหุ่นยนต์เลี้ยวขวาแบบปรกติ

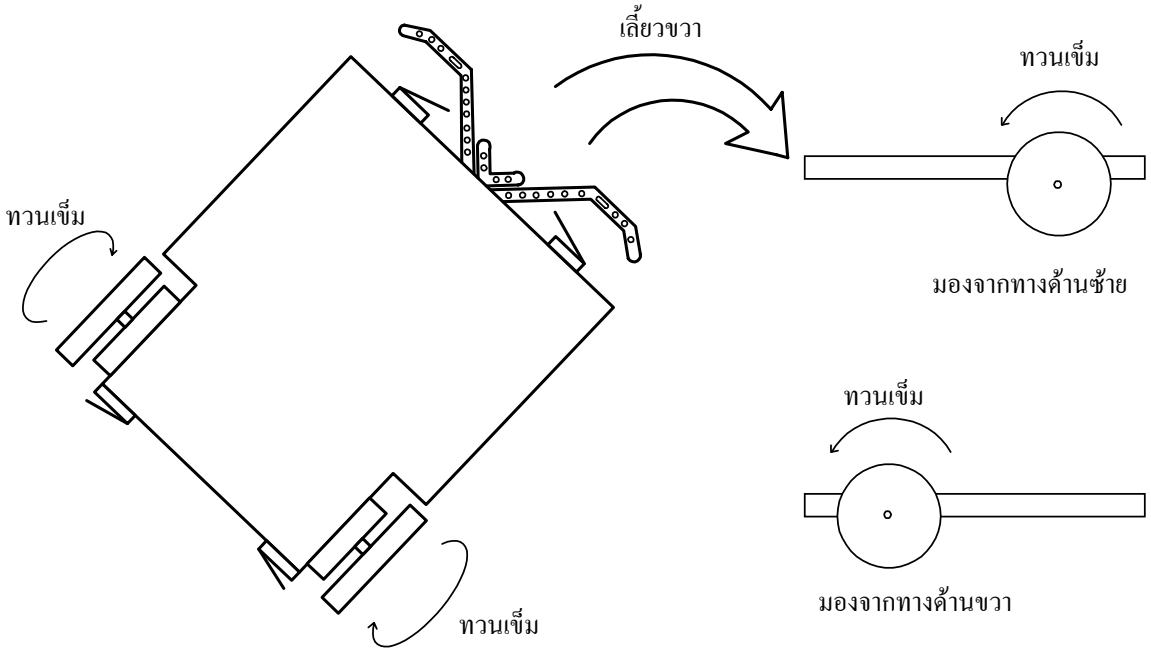
สำหรับวิธีการบังคับให้ตัวรถหุ่นยนต์เลี้ยวขวาแบบนี้ จะอาศัยล้อขวาเป็นจุดหมุนของการเลี้ยว โดยวิธีการในการบังคับให้รถหุ่นยนต์เลี้ยวขวาแบบนี้ จะต้องทำการหยุดการหมุนของล้อทางด้านขวา ส่วนล้อทางด้านซ้ายก็ปล่อยให้หมุนไปข้างหน้าตามปรกติ ดังรูป



รูปแสดงการบังคับให้รถหุ่นยนต์เลี้ยวขวาโดยการเคลื่อนที่ของล้อซ้ายอย่างเดียว

การบังคับให้รถหุ่นยนต์เลี้ยวขวาแบบรวดเร็ว

สำหรับวิธีการบังคับเลี้ยวแบบนี้ สามารถกระทำได้โดยการบังคับให้ล้อทางด้านซ้ายและขวาหมุนไปในทิศทางที่ตรงกันข้าม กล่าวคือ ล้อทางด้านขวาจะถูกบังคับให้หมุนถอยหลัง ส่วนล้อทางด้านซ้ายจะถูกบังคับให้เดินไปทางหน้า ดังรูป



รูปแสดงการบังคับให้รถหุ่นยนต์เลี้ยวขวาโดยให้ล้อขวาถอยหลังส่วนล้อซ้ายเดินหน้า

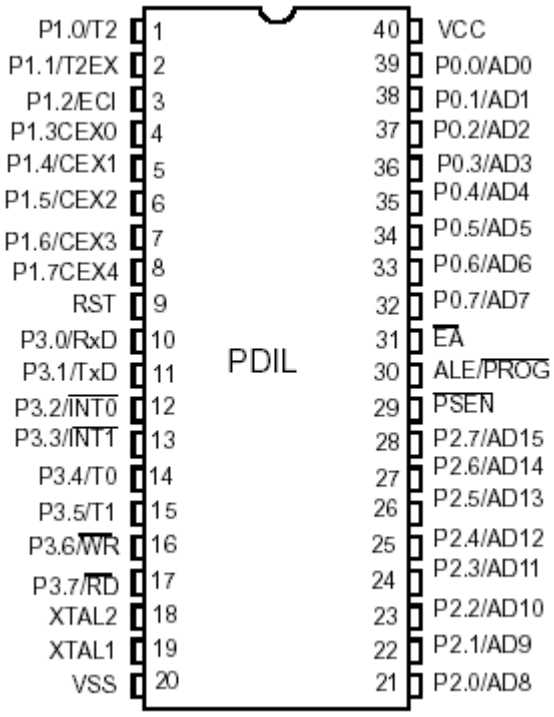
ทิศทางการเคลื่อนที่ของล้อด้านซ้าย			ทิศทางการเคลื่อนที่ของล้อด้านขวา			ทิศทางการเคลื่อนที่ของ
ล้อซ้าย	SERVO	PULSE	**ล้อขวา**	SERVO	PULSE	**ตัวรถหุ่นยนต์**
เดินหน้า	ทวนเข็ม	2 mS	เดินหน้า	ตามเข็ม	1 mS	เคลื่อนที่ไปข้างหน้า (เดินหน้า)
ถอยหลัง	ตามเข็ม	1 mS	ถอยหลัง	ทวนเข็ม	2 mS	เคลื่อนที่ไปข้างหลัง (ถอยหลัง)
หยุดหมุน	-	“0”	เดินหน้า	ตามเข็ม	1 mS	เดินหน้า + เลี้ยวซ้ายแบบปรกติ
ถอยหลัง	ตามเข็ม	1 mS	เดินหน้า	ตามเข็ม	1 mS	เดินหน้า + เลี้ยวซ้ายแบบรวดเร็ว
เดินหน้า	ทวนเข็ม	2 mS	หยุดหมุน	-	“0”	เดินหน้า + เลี้ยวขวาแบบปรกติ
เดินหน้า	ทวนเข็ม	2 mS	ถอยหลัง	ทวนเข็ม	2 mS	เดินหน้า + เลี้ยวขวาแบบรวดเร็ว
หยุดหมุน	-	“0”	ถอยหลัง	ทวนเข็ม	2 mS	ถอยหลัง + เลี้ยวซ้ายแบบปรกติ
เดินหน้า	ทวนเข็ม	2 mS	ถอยหลัง	ทวนเข็ม	2 mS	ถอยหลัง + เลี้ยวซ้ายแบบรวดเร็ว
ถอยหลัง	ตามเข็ม	1 mS	หยุดหมุน	-	“0”	ถอยหลัง + เลี้ยวขวาแบบปรกติ
ถอยหลัง	ตามเข็ม	1 mS	เดินหน้า	ตามเข็ม	1 mS	ถอยหลัง + เลี้ยวขวาแบบรวดเร็ว

ตาราง สรุปวิธีการควบคุมการเคลื่อนที่ของรถหุ่นยนต์แบบต่าง ๆ

(SERVO **ของ** GWS **รุ่น** S03T STD)

การสร้างสัญญาณ Pulse เพื่อควบคุมการหมุนของ DC SERVO MOTOR

สำหรับวิธีการสร้างสัญญาณ Pulse สำหรับควบคุมการหมุนของ DC SERVO MOTOR นั้น สามารถทำได้หลายวิธี ขึ้นอยู่กับเทคนิคการเขียนโปรแกรมของแต่ละคน เช่น การใช้วิธีการหน่วงเวลาในการ SET และ RESET สถานะของสัญญาณที่ Port Pin ของ CPU เพื่อให้ได้สัญญาณ Pulse ตามค่าเวลาที่กำหนดไว้จากโปรแกรม หรืออาจใช้ Timer สำหรับทำการนับเวลาเพื่อสร้างเป็นความถี่ของสัญญาณ Pulse ตามต้องการ แต่สำหรับ CPU เบอร์ P89C51RD2 ของ Philips หรือ T89C51RD2 ของ ATMEL นั้น จะมีวงจรสำหรับสร้างสัญญาณ PWM บรรจุไว้ภายในตัว CPU ด้วยอยู่แล้ว โดยในส่วนของวงจรของบอร์ดที่ได้ออกแบบไว้นั้นจะกำหนดให้ขาสัญญาณ P1.4 ซึ่งเป็นขา Output ของวงจรสร้างสัญญาณ PWM ช่องที่1 (CEX1) สำหรับใช้ควบคุมการหมุนของ DC SERVO MOTOR ที่ใช้ควบคุมการหมุนของล้อรถด้านซ้าย (SERVO-L) และกำหนดให้ขาสัญญาณ P1.5 ซึ่งเป็นขา Output ของวงจรสร้างสัญญาณ PWM ช่องที่2 (CEX2) สำหรับใช้ควบคุมการหมุนของ DC SERVO MOTOR ที่ใช้ควบคุมการหมุนของล้อรถด้านขวา (SERVO-R) โดยวงจรการเชื่อมต่อสามารถดูเพิ่มเติมได้จากส่วนของ Circuit ของบอร์ด



รูปแสดงการจัดขาสัญญาณของ P89C51RD2/T89C51RD2

ซึ่งการใช้วงจร Pulse Width Modulation หรือ PWM ในการสร้างสัญญาณ เพื่อควบคุมการหมุนของ DC SERVO MOTOR นั้น นับว่าเป็นวิธีการและแนวทางที่ง่ายและมีประสิทธิภาพมากที่สุด เนื่องจากสามารถลดความยุ่งยากในการเขียนโปรแกรมเพื่อสร้างเป็นสัญญาณ Pulse ไปได้เป็นอย่างมาก และ CPU เองก็จะไม่ต้องสูญเสียเวลาในการไปประมวลผลเพื่อคำนวณค่าเวลาในการควบคุม Port Pin เพื่อสร้างความถี่ให้ยุ่งยาก เนื่องจากระบบฮาร์ดแวร์ PWM นั้น เมื่อเราทำการกำหนดค่าต่างๆให้กับ รีจิสเตอร์ ไปแล้ว วงจร PWM ก็จะสามารถสร้างสัญญาณ PWM ออกมาได้อย่างต่อเนื่อง โดยไม่รบกวนการทำงานของ CPU อีกเลย แต่เมื่อต้องการเปลี่ยนแปลงค่าความถี่หรือคาบเวลาของ PWM ก็เพียงแต่ทำการกำหนดค่าให้กับรีจิสเตอร์ใหม่เท่านั้น ซึ่งจะเห็นได้ว่าการใช้วงจร PWM ในการสร้างสัญญาณ Pulse เพื่อควบคุมการทำงานของมอเตอร์นั้น สามารถลดความยุ่งยากในการเขียนโปรแกรมไปได้มากเลยทีเดียว ซึ่งในที่นี้จะขอแนะนำถึงวิธีการใช้งานระบบ Timer ของ PCA ในการสร้างสัญญาณ PWM แบบพอสังเขปเพื่อเป็นแนวทางประกอบการใช้งานดังนี้

การทำงานของวงจร Programmable Counter Array (PCA)

วงจร Programmable Counter Array หรือ PCA เป็นส่วนเพิ่มเติมของ ระบบ Timer ใน MCS51 ซึ่งวงจรส่วนนี้ ตามปรกติแล้วจะไม่มีอยู่ในโครงสร้างมาตรฐานของ CPU ในตระกูล MCS51 ทั่วๆไป ซึ่งระบบการทำงานของวงจร Timer/Counter แบบ PCA นี้จะมีขีดความสามารถที่สูงกว่าระบบการทำงานของ Timer/Counter ปรกติของ MCS51 ซึ่งจะช่วยให้ลดความยุ่งยากของโปรแกรมในการสั่งงาน CPU และช่วยลดภาระการทำงานของ CPU ได้เป็นอย่างมาก โดยลักษณะโครงสร้างของวงจร PCA จะประกอบไปด้วย ส่วนของวงจร Timer/Counter หลัก (PCA Timer) และเชื่อมต่อเข้ากับวงจรของ Capture/Compare (PCA Module) จำนวน 5 ชุด โดยเชื่อมต่อกันแบบเรียงลำดับ (Array) ซึ่งการทำงานของ PCA Module แต่ละชุดจะอ้างอิงความถี่จากวงจร PCA Counter ชุดเดียวกัน หรือใช้ฐานเวลาร่วมกัน

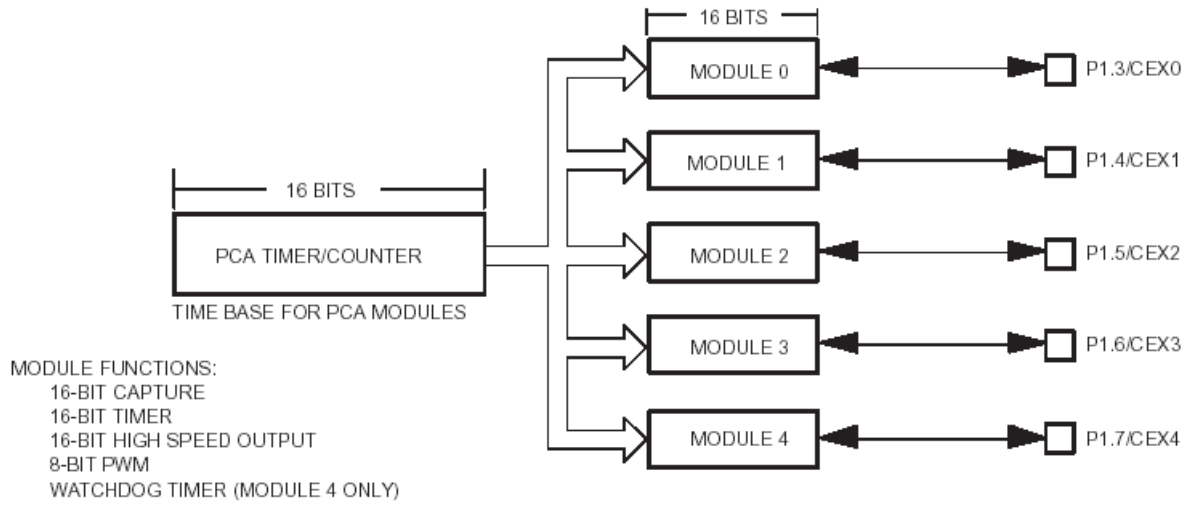
ซึ่งระบบฐานเวลาหลัก (PCA Timer) ของวงจร PCA นั้น จะสามารถทำการโปรแกรมหรือกำหนดเงื่อนไขในการนับได้ว่าจะให้นับจากสัญญาณนาฬิกา Input แบบใด โดยสามารถกำหนดได้ 4 แบบ โดยการเลือกกำหนดจากบิต CPS1:CPS0 ของรีจิสเตอร์ CMOD ดังนี้คือ

- กำหนดให้นับจากความถี่สัญญาณนาฬิกา Oscillator หาร 12 (หาร 6 ในโหมด X2)
- กำหนดให้นับจากความถี่สัญญาณนาฬิกา Oscillator หาร 4 (หาร 2 ในโหมด X2)
- กำหนดให้นับจากการ Overflow ของวงจร Timer0
- กำหนดให้นับจากสัญญาณนาฬิกา Input ภายนอกที่ป้อนให้กับขาสัญญาณ ECI หรือ P1.2

ส่วนการทำงานของวงจร PCA Module ทั้ง 5 ชุด นั้น แต่ละ โมดูล สามารถกำหนดหน้าที่การทำงานของวงจรได้อย่างอิสระ โดยแต่ละ โมดูล เองก็ยังสามารถ โปรแกรมการทำงานได้หลายๆหน้าที่ ดังนี้

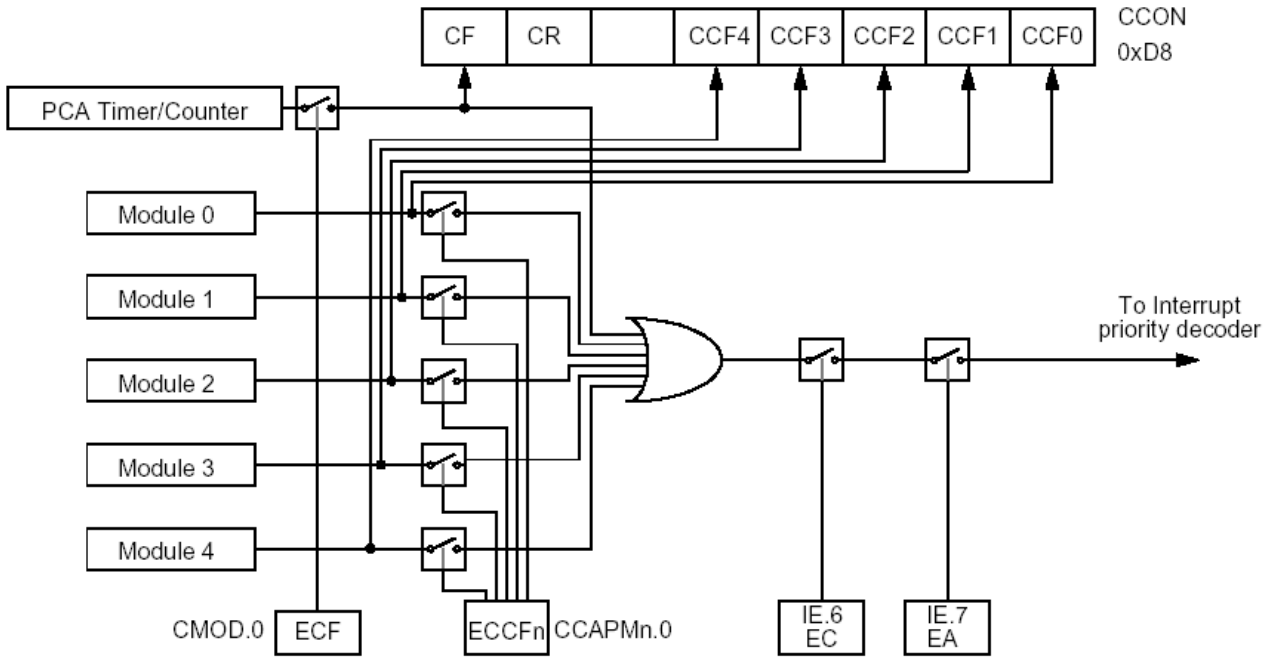
- กำหนดให้ทำการตรวจจับสัญญาณ Input (Capture) โดยสามารถเลือกตรวจจับได้ทั้งสัญญาณที่เป็นแบบขอบขาขึ้น (Rising Edge) และ ขอบขาลง(Falling Edge)
- กำหนดให้ทำการสร้างสัญญาณ Output ความเร็วสูง แบบ High Speed Output
- กำหนดให้ทำการสร้างสัญญาณ Pulse Width Moduration หรือ PWM
- กำหนดให้ทำหน้าที่เป็น Timer สำหรับนับสัญญาณนาฬิกา (Software Timer)

นอกจากนี้แล้ว ในส่วนของโมดูล PCA ชุดที่4 นั้น ยังมีความสามารถพิเศษ กว่าโมดูลอื่นๆ กล่าวคือ สามารถทำการโปรแกรมหน้าที่ให้เป็น Watch-Dog Timer ได้อีกด้วย



รูปแสดง ลักษณะของระบบ PCA Timer

โดยเมื่อวงจร PCA แต่ละโมดูล ถูกโปรแกรมให้ทำหน้าที่แบบ Input Capture หรือ Software Timer หรือ High Speed Output นั้น การทำงานของวงจร PCA แต่ละชุด สามารถส่งสัญญาณไปร้องขอการ Interrupt จาก CPU ได้ด้วย โดยที่โมดูลของ PCA ทั้ง 5 ชุดนี้จะใช้สัญญาณ Interrupt ร่วมกัน ซึ่งไม่ว่าจะเกิดการร้องขอการ Interrupt จากโมดูลชุดใด ก็จะมีตำแหน่ง Vector ในการบริการ Interrupt ที่เหมือนกัน แต่ผู้ใช้งานสามารถทำการตรวจสอบแหล่งที่มาของการร้องขอการ Interrupt ได้ว่ามาจากโมดูลชุดใดจากบิต CCFx ในรีจิสเตอร์ CCON โดยลักษณะการสร้างสัญญาณ Interrupt ของ PCA Timer เป็นดังรูป



รูปแสดง แผนผันการสร้างสัญญาณ Interrupt ของ PCA Timer

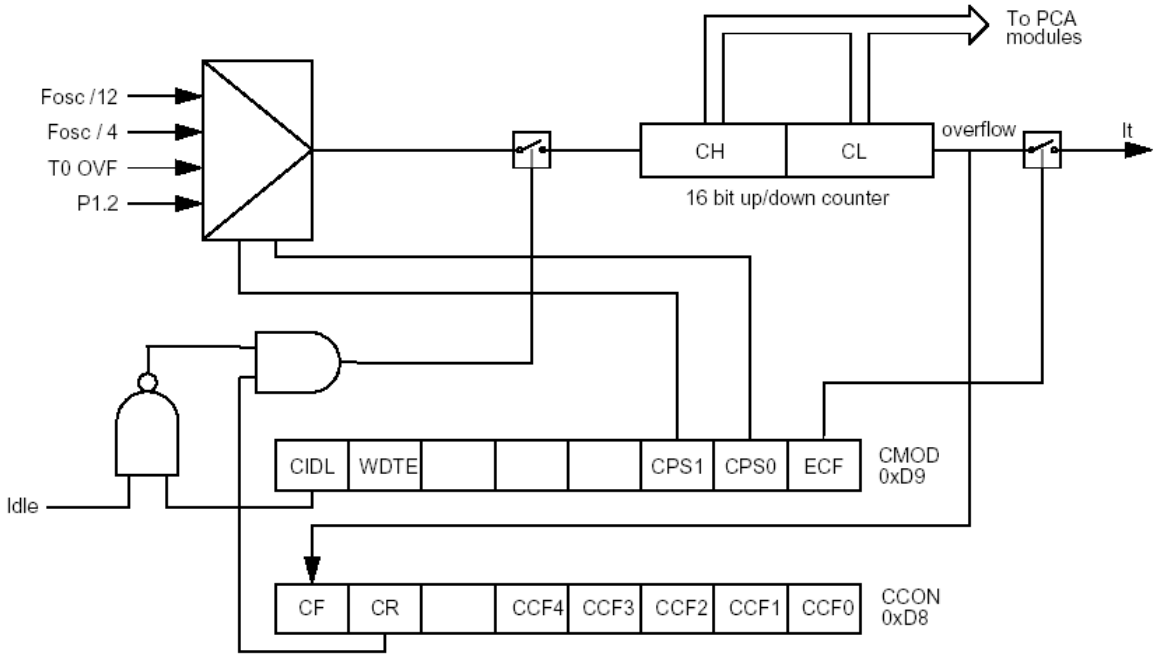
สำหรับการทำงานของ PCA ทั้ง 5 ชุดนั้น จะใช้สัญญาณจาก พอร์ต P1 (P1.2-P1.7) เป็นจุดผ่านของสัญญาณในการทำงาน ซึ่งเมื่อไม่มีการโปรแกรมการทำงานให้กับวงจร PCA แล้ว ขาสัญญาณของพอร์ต P1 ก็ยังสามารถนำไปใช้งานเป็น Input หรือ Output ทั่วๆไปได้ตามต้องการ แต่ในกรณีที่มีการโปรแกรมการทำงานของวงจร PCA ไว้ ขาสัญญาณของพอร์ต P1 ก็จะถูกควบคุมการทำงานโดยวงจร PCA แต่ละชุด โดยสามารถแสดงให้เห็นได้ดังนี้คือ

การทำงานของวงจร PCA	**ขาสัญญาณ พอร์ต P1**
16 บิต Counter	P1.2 / ECI
16 บิต Module0	P1.3 / CEX0
16 บิต Module1	P1.4 / CEX1
16 บิต Module2	P1.5 / CEX2
16 บิต Module3	P1.6 / CEX3
16 บิต Module4	P1.7 / CEX4

ตาราง แสดง การจัดสรรขาสัญญาณพอร์ต P1 สำหรับใช้งานร่วมกับ PCA

การทำงานของ PCA Timer

วงจร PCA Timer จัดเป็นวงจรนับ ขนาด 16 บิต ทำหน้าที่สำหรับส่งสัญญาณ Output ที่ได้จากการนับไปยังโมดูลของ PCA ทั้ง 5 ชุด โดยทำงานของวงจร PCA Timer นั้น จะถูกควบคุมการทำงานจากรีจิสเตอร์ CMOD และ CCON ซึ่งวงจร PCA โมดูลทั้ง 5 ชุด นั้น จะใช้ฐานเวลาจากวงจร PCA Timer ชุดเดียวกันทั้งหมด โดยการนับของ PCA Timer นั้น จะกำหนดลักษณะของสัญญาณ Input ได้ 4 แบบ โดยกำหนดผ่านทางบิต CPS0:CPS1 ของรีจิสเตอร์ CMOD ซึ่งลักษณะโครงสร้างของวงจร PCA Timer นั้นสามารถแสดงให้เห็นได้ดังรูป



รูปแสดง ลักษณะโครงสร้างของวงจร PCA Timer

จากรูปจะเห็นได้ว่า บิต CPS1:CPS0 ของรีจิสเตอร์ CMOD จะถูกใช้งานร่วมกันสำหรับทำหน้าที่เลือกแหล่งกำเนิดสัญญาณนาฬิกาสำหรับป้อนเป็น Input การนับของวงจร PCA Timer (CH:CL) โดยการทำงานของวงจร PCA Timer นั้นจะเริ่มต้นทำงานได้ก็ต่อเมื่อทำการเซ็ตบิต CR ในริจิสเตอร์ CCON ให้มีค่าเป็น “1” โดยผลการนับของวงจร PCA Timer นั้น เมื่อเกิดการ Overflow ขึ้น สามารถที่จะควบคุมให้การ Overflow ของวงจรนับเพื่อให้ส่งสัญญาณร้องขอการ Interrupt ไปยัง CPU ได้จากควบคุมของบิต ECF ในรีจิสเตอร์ CMOD โดยลักษณะการนับของวงจร PCA Timer นั้น จะเป็นวงจรนับแบบอิสระ ไม่สามารถจะทำการโปรแกรมค่าการนับให้กับวงจรได้โดยตรง กล่าวคือ เมื่อทำการอนุญาติให้วงจร PCA Timer เริ่มต้นทำงาน (เซ็ตบิต CR ใน CCON เป็น “1”) การนับของ PCA Timer ก็จะเริ่มต้นนับต่อเนื่องกันไปไม่รู้จบ ตามค่า Input ที่ได้รับ จนกว่าจะมีการสั่งหยุดการทำงาน โดยผลการนับของวงจร PCA Timer

นั้น จะอยู่ในรีจิสเตอร์ CH:CL แต่อย่างไรก็ตามจำนวนค่าการนับของวงจร PCA Timer นั้นเราสามารถทำการโปรแกรมจำนวนครั้งในการนับให้กับวงจร PCA Time ได้ด้วยวิธีการทางอ้อม โดยใช้เงื่อนไขของการ Overflow เป็นตัวกำหนด ตัวอย่างเช่น ถ้าต้องการให้วงจร PCA Timer ทำการนับจำนวน 500 ครั้ง ก็สามารถทำได้โดยการกำหนดค่าเริ่มต้นให้กับ CH:CL ด้วยค่าการนับสูงสุดหักลบออกจากจำนวนครั้งที่ต้องการคือ 500 (65536-500) และจะต้องการกำหนดค่านี้ให้กับ CH:CL ทุกๆครั้งที่เกิดการ Overflow ขึ้น ไม่เช่นนั้นแล้วการนับของ PCA Timer จะกลับไปเริ่มต้นใหม่จากศูนย์แทน

คุณสมบัติของรีจิสเตอร์ CMOD

สำหรับรีจิสเตอร์ CMOD เป็นรีจิสเตอร์ขนาด 8 บิต ใช้สำหรับกำหนดโหมดการทำงานของ PCA Timer โดยรีจิสเตอร์ CMOD นี้จะจัดไว้ในส่วนของ SFR (Special Function Register) ตำแหน่งแอดเดรส D9H ซึ่งในการเข้าถึงรีจิสเตอร์ตัวนี้สามารถเข้าถึงได้แบบ Byte ด้วยวิธีการแบบ Direct หรือการอ้างตำแหน่งแอดเดรสของรีจิสเตอร์โดยตรงเพียงอย่างเดียวเท่านั้น ไม่สามารถใช้วิธีการเข้าถึงแบบระดับบิตได้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
CIDL	WDTE	-	-	-	CPS1	CPS0	ECF

รูปแสดง โครงสร้างของ CMOD Register

- **CIDL** เป็นบิต Counter Idle Control ใช้สำหรับควบคุมการทำงานของ PCA Counter เมื่อ CPU เข้าทำงานในโหมดประหยัดพลังงาน (Idle Mode) โดยถ้ากำหนดให้บิต CIDL นี้มีค่าเป็น “0” จะเป็นการปล่อยให้วงจร PCA Counter ทำงานต่อเนื่องไปตลอด ถึงแม้ว่า CPU จะหยุดทำงานแล้วก็ตาม แต่ถ้ากำหนดให้บิต CIDL นี้มีค่าเป็น “1” จะหมายถึง การบังคับให้วงจร PCA หยุดการทำงานทันที เมื่อ CPU เริ่มต้นเข้าทำงานใน Idle โหมด
- **WDTE** เป็นบิต Watchdog Timer Enable ใช้สำหรับควบคุมการทำงานของวงจร Watchdog ที่สร้างจากวงจร PCA โมดูลที่4 โดยถ้ากำหนดให้บิตนี้มีค่าเป็น “0” จะเป็นการปิดการทำงานของ PCA Watchdog แต่ถ้ากำหนดให้บิต WDTE นี้มีค่าเป็น “1” จะเป็นการเปิดการทำงานของวงจร PCA Watchdog

- **CPS0:CPS1** เป็นบิต PCA Count Pulse Select ใช้ร่วมกันสำหรับเลือกแหล่งของสัญญาณ Input ที่จะส่งให้กับวงจร PCA Timer ทำการนับ โดยมีคุณสมบัติดังนี้คือ

CPS 1	CPS 0	**สัญญาณที่ป้อนให้กับ PCA Counter**
0	0	Internal Clock หรือ $F_{osc}/12$ ($F_{osc}/6$ ใน X2 Mode)
0	1	Internal Clock หรือ $F_{osc}/4$ ($F_{osc}/2$ ใน X2 Mode)
1	0	การ Overflow ของ Timer-0
1	1	External Clock ที่ป้อนให้กับขาสัญญาณ ECI หรือ P1.2 (ความถี่สูงสุด $F_{osc}/8$)

- **ECF** เป็นบิต PCA Enable Counter Overflow Interrupt ใช้สำหรับควบคุมการร้องขอการ Interrupt ของวงจร PCA โดยถ้ากำหนดให้บิต ECF นี้มีค่าเป็น “0” จะหมายถึง เป็นการปิดการร้องขอการ Interrupt จากวงจร PCA Timer แต่ถ้ากำหนดให้บิต ECF นี้มีค่าเป็น “1” จะเป็นการอนุญาติให้บิต CF ในรีจิสเตอร์ CCON ทำการร้องขอการ Interrupt ไปยัง CPU เมื่อการทำงานของโมดูล PCA Timer เกิดการ Overflow ขึ้น

คุณสมบัติของรีจิสเตอร์ CCON

สำหรับรีจิสเตอร์ CCON เป็นรีจิสเตอร์ขนาด 8 บิต ใช้สำหรับแสดงสถานะและควบคุมการทำงานของวงจร PCA Timer โดยรีจิสเตอร์ CCON นี้จะจัดไว้ในส่วนของ SFR (Special Function Register) ตำแหน่งแอดเดรส D8H ซึ่งในการเข้าถึงรีจิสเตอร์ตัวนี้สามารถเข้าถึงได้แบบ Byte ด้วยวิธีการแบบ Direct หรือการอ้างตำแหน่งแอดเดรสของรีจิสเตอร์โดยตรงเพียงอย่างเดียวเท่านั้น ไม่สามารถใช้วิธีการเข้าถึงแบบระดับบิตได้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
CF	CR	-	CCF4	CCF3	CCF2	CCF1	CCF0

รูป แสดง โครงสร้างของ CCON Register

- **CF** เป็นบิต PCA Counter Overflow Flag ใช้แสดงสถานะการทำงานของ PCA Counter โดยบิตนี้จะมีค่าเป็น “1” เมื่อการทำงานของ PCA Counter เกิดการ Overflow ขึ้น โดยถ้ามีการกำหนดให้บิต ECF ในรีจิสเตอร์ CMOD มีค่าเป็น “1” ไว้ด้วย เมื่อบิต CF นี้ถูกเซ็ตเป็น “1” จะมีการร้องขอการ Interrupt ไปยัง CPU ด้วยเสมอ โดยเมื่อบิต CF นี้ถูกเซ็ตเป็น “1” แล้วต้องทำการรีเซ็ตหรือสั่งทำการเคลียร์ให้กลับเป็น “0” ด้วยโปรแกรมเพียงอย่างเดียวเท่านั้น
- **CR** เป็นบิต PCA Counter Run Control ใช้สำหรับควบคุมการทำงานของ PCA Timer โดยเมื่อกำหนดให้บิตนี้มีค่าเป็น “1” จะเป็นการสั่งให้ PCA Timer เริ่มต้นทำงาน และเมื่อกำหนดให้บิตนี้มีค่าเป็น “0” จะเป็นการสั่งหยุดการทำงานของ PCA Timer
- **CCFx** เป็นบิต PCA Modulex Interrupt Flag ใช้สำหรับแสดงสถานะการทำงานของ PCA Module แต่ละชุด ตามค่าบิต x เช่น CCF4 ใช้แสดงสถานะของ PCA โมดูลที่4 ส่วน CCF1 ก็ใช้แสดงสถานะการทำงานของ PCA โมดูลที่1 เป็นต้น ซึ่งบิต CCFx นี้จะถูกเซ็ตให้มีค่าเป็น “1” เมื่อการทำงานของโมดูลนั้นๆตรงตามเงื่อนไขที่กำหนดไว้ และเมื่อบิต CCFx ถูกเซ็ตเป็น “1” แล้วจะต้องสั่งรีเซ็ตหรือเคลียร์ ให้กลับเป็น “0” ด้วยโปรแกรมเพียงอย่างเดียวเท่านั้น

การทำงานของ PCA Module

วงจร PCA Module ใน CPU เบอร์ P89C51RD2/T89C51RD2 จะมีอยู่ด้วยกันทั้งหมด 5 ชุดด้วยกัน โดยแต่ละชุด สามารถที่จะโปรแกรมหน้าที่การทำงานของแต่ละโมดูล แยกออกจากกัน ได้อย่างอิสระ ซึ่งแต่ละโมดูลของ PCA นั้น สามารถโปรแกรมหน้าที่การทำงานได้หลายหน้าที่ดังนี้คือ

- วงจรตรวจจับ Input ขนาด 16 บิตแบบขอบขาขึ้น (Capture แบบ Positive-Edge Trigger)
- วงจรตรวจจับ Input ขนาด 16 บิตแบบขอบขาลง (Capture แบบ Negative-Edge Trigger)
- วงจรตรวจจับ Input ขนาด 16 บิตแบบทั้ง2ขอบ (Capture แบบ Positive และ Negative Edge Trigger)
- วงจรนับขนาด 16บิต (16 Bit Software Timer)
- วงจรสร้างสัญญาณ Output ความเร็วสูงขนาด 16บิต (16 Bit High Speed Output)
- วงจรสร้างสัญญาณ Pulse Width Modulation ขนาด 8บิต (8 Bit PWM)

หมายเหตุ สำหรับ PCA โมดูลที่4 นั้น นอกจากจะสามารถทำการโปรแกรมหน้าที่การทำงานทั้ง 6 แบบ ข้างต้น แล้ว ยังสามารถทำการโปรแกรมให้ทำหน้าที่เป็นวงจร Watchdog Timer ได้อีกด้วย

โดยการกำหนดโหมดการทำงานของวงจร PCA แต่ละโมดูลนั้นจะใช้รีจิสเตอร์ CCAPMx ในการควบคุมสั่งงาน แต่ละโมดูล โดยที่รีจิสเตอร์ CCAPM จะมีทั้งหมด 5 ชุด คือ CCAPM0-CCAPM4 โดยที่รีจิสเตอร์ CCAPM ทุกๆตัว จะมีคุณสมบัติที่เหมือนกันทุกประการ เพียงแต่ ว่าจะทำหน้าที่ในการควบคุมการทำงานของวงจร PCA แต่ละโมดูลตามลำดับ โดย รีจิสเตอร์ CCAPM0 ก็จะใช้ควบคุมวงจร PCA โมดูลที่0 ส่วนรีจิสเตอร์ CCAPM1 ก็จะใช้สำหรับควบคุมการทำงานของวงจร PCA โมดูลที่1 และในทำนองเดียวกันรีจิสเตอร์ CCAPM4 ก็จะใช้สำหรับควบคุมการทำงานของวงจร PCA โมดูลที่4 เป็นต้น

คุณสมบัติของรีจิสเตอร์ CCAPMx

สำหรับรีจิสเตอร์ CCAPMx นั้นจัดเป็นรีจิสเตอร์ขนาด 8 บิต ใช้สำหรับเลือกหน้าที่การทำงานให้กับโมดูล CPA โดยที่รีจิสเตอร์นี้ จะมีอยู่ด้วยกันจำนวน 5 ชุด คือ CCAPM0,CCAPM1,CCAPM2,CCAPM3 และ CCAPM4 โดยคุณสมบัติของรีจิสเตอร์แต่ละตัว จะเหมือนกันทุกประการ ต่างกันเพียงแต่ละตัวจะใช้สำหรับควบคุมการทำงานของโมดูล PCA ต่างชุดกัน กล่าวคือ รีจิสเตอร์ CCAPM0 ก็จะใช้สำหรับควบคุมการทำงานของ วงจร PCA โมดูล0 เป็นต้น

แต่เนื่องจากหน้าที่การทำงานของรีจิสเตอร์ทั้ง 5 ชุดนี้ มีลักษณะเหมือนกัน ดังนั้นในที่นี้จึงขอกล่าวอธิบายรวมกัน โดยอ้างชื่อเป็น CCAPMx แทน ซึ่งค่า x ก็จะหมายถึงโมดูลชุดที่x ซึ่งมีค่า 0-4 นั่นเอง

โดยรีจิสเตอร์ CCAPMx นี้จะจัดไว้ในส่วนของ SFR (Special Function Register) และมีตำแหน่งแอดเดรส อยู่ระหว่าง DAH-DEH ตามลำดับ ซึ่งในการเข้าถึงรีจิสเตอร์ตัวนี้สามารถเข้าถึงได้แบบ Byte ด้วยวิธีการแบบ Direct หรือการอ้างตำแหน่งแอดเดรสของรีจิสเตอร์โดยตรงเพียงอย่างเดียวเท่านั้น ไม่สามารถใช้วิธีการเข้าถึงแบบระดับบิตได้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
-	ECOMx	CAPPx	CAPNx	MATx	TOGx	PWMx	ECCFx

รูป แสดง โครงสร้างของ CCAPMx Register

- **ECOMx** เป็นบิต Enable Comparator ใช้สำหรับเลือกการทำงานของโมดูล PCA ให้ทำหน้าที่เป็นวงจร Comparator โดยถ้ากำหนดให้บิตนี้มีค่าเป็น “1” จะเป็นการเปิดการทำงานฟังก์ชั่น Comparator ของโมดูล PCA ที่ถูกควบคุมโดยรีจิสเตอร์ชุดนั้นๆ เช่น ถ้าทำการกำหนดให้บิต ECOM4 ของรีจิสเตอร์ CCAPM4 ก็จะเป็นการเปิดการทำงานฟังชั่น Comparator ของ PCA โมดูลที่4 เป็นต้น
- **CAPPx** เป็นบิต Capture Positive โดยถ้ากำหนดให้บิตนี้มีค่าเป็น “1” จะเป็นการกำหนดให้วงจร PCA ทำการตรวจจับสัญญาณ Input (Capture) ในขณะที่สัญญาณเป็นขอบขาขึ้น (Rising Edge)
- **CAPNn** เป็นบิต Capture Negative โดยถ้ากำหนดให้บิตนี้มีค่าเป็น “1” จะเป็นการกำหนดให้วงจร PCA ทำการตรวจจับสัญญาณ Input (Capture) ในขณะที่สัญญาณเป็นขอบขาลง (Falling Edge)

- **MATn** เป็นบิต Match ใช้สำหรับแสดงสถานะการทำงานของวงจร PCA โดยบิตนี้จะถูกเซ็ตให้มีค่าเป็น “1” เมื่อการทำงานของ โมดูล PCA นับแล้วมีค่าตรงกับค่าของ PCA Counter
- **TOGx** เป็นบิต Toggle ใช้สำหรับแสดงสถานะการทำงานของโมดูล PCA
- **PWMx** เป็นบิต Pulse Width Modulation Mode ใช้สำหรับ เลือกการทำงานของโมดูล PCA ให้ทำหน้าที่เป็นวงจรสร้างสัญญาณ PWM โดยเมื่อกำหนดให้บิตนี้มีค่าเป็น “1” จะเป็นการเลือกกำหนดให้ โมดูล PCA ที่ถูกเลือกทำหน้าที่สร้างสัญญาณ PWM ออกที่ขาสัญญาณ CEXn
- **ECCFx** เป็นบิต Enable CCF Interrupt ใช้สำหรับควบคุมการร้องขอ Interrupt ของโมดูล PCA โดยถ้ากำหนดให้บิตนี้มีค่าเป็น “1” จะเป็นการสั่งอนุญาติให้โมดูล PCA ทำการร้องขอการ Interrupt ถ้าบิต CCFx ในรีจิสเตอร์ CCON ของโมดูล PCA นั้นๆ ถูกกำหนดให้มีค่าเป็น “1” ไว้ก่อนแล้ว

ECOMx	CAPPx	CAPNx	MATx	TOGx	PWMx	ECCFx	**โหมดการทำงาน**
0	0	0	0	0	0	0	ไม่ทำงาน
X	1	0	0	0	0	X	16 Bit Capture ขอบขาขึ้น
X	0	1	0	0	0	X	16 Bit Capture ขอบขาลง
X	1	1	0	0	0	X	16 Bit Capture ทั้ง 2 ขอบ
1	0	0	1	0	0	X	16 Bit Timer/Compare
1	0	0	1	1	0	X	16 Bit High Speed Output
1	0	0	1	0	1	0	8 Bit PWM
1	0	0	1	X	0	X	Watchdog Timer (PCA-4)

ตาราง แสดงการกำหนดค่าสำหรับเลือกโหมดการทำงนของโมดูล PCA

จากตารางจะเห็นได้ว่าโมดูล PCA นั้น สามารถจะโปรแกรมหน้าที่การทำงานในรูปแบบต่างๆ ได้ถึง 5 โหมดการทำงานด้วยกัน คือ

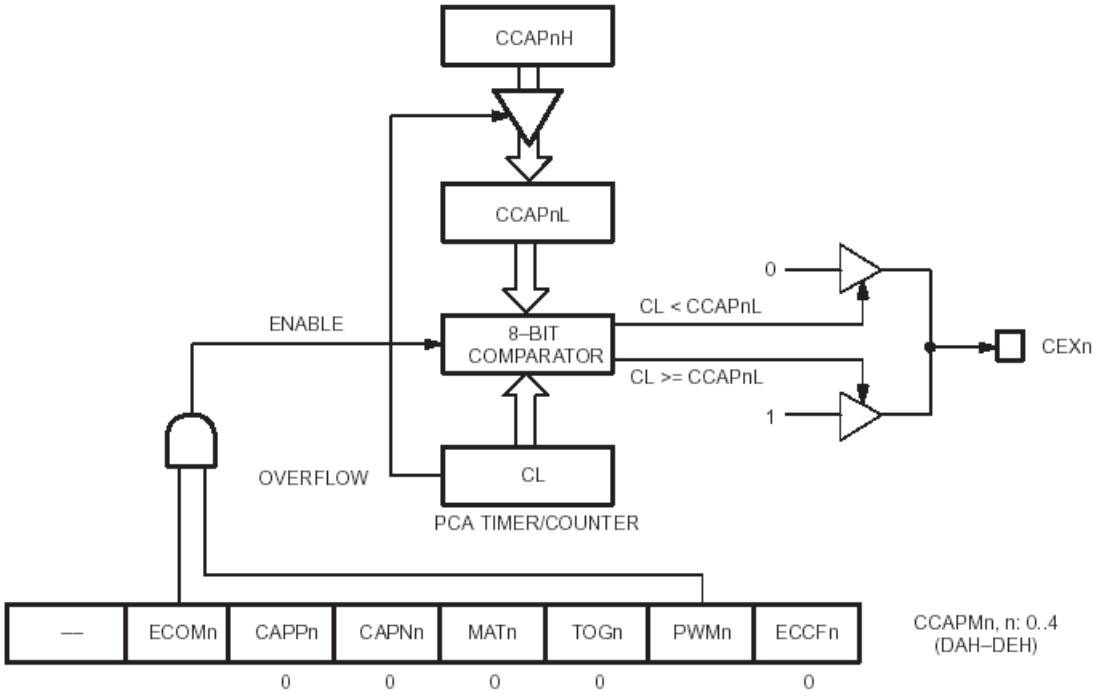
- 16 บิต Capture ในโหมดนี้ การทำงานของโมดูล PCA จะทำหน้าที่คอยตรวจจับการเปลี่ยนแปลงของสัญญาณ Input โดยสามารถกำหนดลักษณะการตรวจจับให้กับโมดูล PCA ได้ 3 ลักษณะคือ กำหนดให้คอยตรวจจับเมื่อสัญญาณเปลี่ยนแปลงสภาวะเป็นช่วงขอบขาขึ้น (Rising Edge หรือ Positive Edge) ขอบขาลง (Falling Edge หรือ Negative) หรืออาจกำหนดให้คอยทำการตรวจจับการเปลี่ยนแปลงทั้งช่วงที่เป็นขอบขาขึ้นและขอบขาลงเลยก็ได้

- 16 บิต Software Timer/Compare ในโหมดนี้ โมดูล PCA จะทำหน้าที่คอยเปรียบเทียบการนับของ โมดูล PCA และ PCA Counter โดยเมื่อการนับของวงจรทั้ง2 มีค่าเท่ากันจะเกิดการ Interrupt ขึ้น
- 16 บิต High Speed Output ในโหมดนี้ โมดูล PCA จะทำการเปรียบเทียบการนับของ PCA และโมดูล PAC Counter แต่จะมีการกลับสถานะของ Pin Port ของแต่ละโมดูลด้วย เมื่อผลการเปรียบเทียบตรงตามที่กำหนดไว้
- 8 บิต Pulse Width Modulation ในโหมดนี้ โมดูล PCA จะทำหน้าที่สร้างสัญญาณ PWM ออกไปยังขา CEX ของแต่ละโมดูล PCA โดยการทำงานของโมดูล PCA ในโหมดนี้จะสามารถควบคุมให้สร้างสัญญาณ PWM ขนาด 8 บิตโดยสามารถ ทำการกำหนดหรือเปลี่ยนแปลงค่า Duty Cycle ของสัญญาณ PWM ได้ตามต้องการ
- Watchdog Timer ในโหมดนี้ จะใช้ได้กับโมดูล PCA4 เพียงชุดเดียวเท่านั้น ส่วน โมดูล PCA อื่นๆจะไม่สามารถกำหนดการทำงานให้ทำหน้าที่ในโหมดนี้ได้

เนื่องจากหน้าที่การทำงานของโมดูล PCA นั้น มีอยู่ด้วยกันหลายโหมดดังได้กล่าวไปแล้วในข้างต้น แต่สำหรับในที่นี้จะขอกล่าวรายละเอียดการใช้งานของ PCA โมดูล เฉพาะส่วนการทำงานของ PCA โมดูล ในโหมดของการสร้างสัญญาณ PWM เท่านั้น เนื่องจากเป็นส่วนที่เกี่ยวข้องกับการสร้างสัญญาณ Pulse สำหรับใช้ควบคุมการทำงานของ DC SERVO MOTOR ส่วนการทำงานของ PCA ในโหมดอื่นๆ นั้น ขอให้ผู้ใช้ศึกษาเพิ่มเติมได้จากส่วนของ Data Sheet ของ CPU เอง

การทำงานของ PCA ในโหมด Pulse Width Modulation

ในโหมดนี้ โมดูล PCA จะทำหน้าที่สร้างสัญญาณ PWM ออกไปยังขา CEX ของโมดูล PCA โดยการทำงานของโมดูล PCA ในโหมดนี้จะสามารถควบคุมให้สร้างสัญญาณ PWM ขนาด 8 บิต โดยสามารถทำการกำหนดหรือเปลี่ยนแปลงค่า Duty Cycle ของสัญญาณ PWM ได้ตามต้องการ



รูปแสดง โครงสร้างการทำงานของ PCA ในโหมด PWM

สำหรับหลักการทำงานของ PCA Timer ในการสร้างสัญญาณ PWM นั้นจะอาศัยผลการเปรียบเทียบค่าการนับแบบ 8 บิต ระหว่าง PCA Timer (ซึ่งตามปรกติมีขนาด 16 บิต แต่ในโหมดนี้จะใช้การนับแบบ 8 บิต) โดยค่าการนับของ PCA Timer นั้นจะใช้รีจิสเตอร์ CL เป็นตัวเก็บค่าการนับ โดยผลการนับของ CL จะถูกนำไปเปรียบเทียบกับค่าที่กำหนดไว้ในรีจิสเตอร์ CCAPnL โดยถ้าค่าการนับของ PCA Timer ในรีจิสเตอร์ CL มีค่าน้อยกว่าค่าที่กำหนดไว้ใน รีจิสเตอร์ CCAPnL แล้วจะได้ผลลัพธ์ของสัญญาณมีค่าเป็น “0” ที่ Pin Port ของ CPU ที่ทำหน้าที่เป็น Output Pin ของ PCA ช่องนั้นๆ(CEXn = “0”) แต่ถ้าค่าการนับของรีจิสเตอร์ CL มีค่าเท่ากับหรือมากกว่าค่าที่กำหนดไว้ในรีจิสเตอร์ CCAPnL แล้วจะได้ผลลัพธ์เป็น “1” ที่ Pin Port ของ CPU ที่ทำหน้าที่เป็น Output Pin ของ PCA ช่องนั้นๆ(CEXn = “1”)

โดยจะเห็นว่ารีจิสเตอร์ CL ซึ่งถูกใช้สำหรับทำการนับนั้น มีขนาดเป็น 8 บิต ดังนั้นค่าผลการนับของรีจิสเตอร์ CL จึงมีค่าการนับแบบวนรอบอยู่ระหว่าง 00H-FFH (0-255) หรือ 256 ค่าพอดี โดยทุกๆครั้งที่ค่าการนับของรีจิสเตอร์ CL เกิดค่าเกิน หรือ Overflow หรือค่าการนับเกินกว่า FFH แล้ว ค่าการนับของรี

จิสเตอร์ CL จะวนกลับไปเริ่มต้นที่ 00H ใหม่เสมอ ดังนั้นค่า Period หรือ ค่าคาบเวลาของสัญญาณ PWM นั้นจะขึ้นอยู่กับผลคูณของ 256 กับ ค่าความถี่ของสัญญาณนาฬิกาที่ป้อนให้กับวงจร PCA เสมอ ตัวอย่าง เช่น สัญญาณนาฬิกาที่ป้อนให้กับ PCA เพื่อทำการนับมีค่าความถี่ 1Hz ค่าคาบเวลาทั้งหมดของสัญญาณ ญ PWM ก็จะมีค่าเป็น 256 Hz โดยปริยาย ส่วนค่า Duty Cycle หรือค่าคาบเวลาที่เป็นซีกบวกและซีกลบของสัญญาณ PWM นั้นได้จากการกำหนดค่าอ้างอิงการเปรียบเทียบที่กำหนดให้กับรีจิสเตอร์ CCAPnL แทน

โดยค่า Duty Cycle ของสัญญาณ PWM จะกำหนดโดยค่าใน CCAPnL โดยถ้ากำหนดให้ค่าของ CCAPnL มีค่าน้อยจะได้ลักษณะของสัญญาณ PWM ที่มีคาบเวลาด้านบวกมาก แต่ถ้ากำหนดให้ค่าของ CCAPnL มีค่ามากจะทำให้ได้ลักษณะของสัญญาณ Pulse ที่มีคาบเวลาซีกบวกน้อย โดยค่าที่จะกำหนดให้กับรีจิสเตอร์ CCAPnL นั้น จะไม่ได้กำหนดให้กับรีจิสเตอร์นี้โดยตรง แต่จะกำหนดไว้ในรีจิสเตอร์ CCAPnH แทน โดยจะเห็นได้ว่าในทุกๆครั้งที่ค่าการนับของรีจิสเตอร์ CLนั้นเกิด Overflow หรือเกินกว่า FFH จะมีการส่งสัญญาณไปทำการ Reload ค่าในรีจิสเตอร์ CCAPnH มายังรีจิสเตอร์ CCAPnL ทุกครั้งด้วยเสมอ ตัวอย่างเช่น ถ้ามีการกำหนดค่าให้กับรีจิสเตอร์ CCAPnL (กำหนดผ่าน CCAPnH) ด้วยค่า 10 จะได้ว่า ลักษณะของสัญญาณ PWM ที่ได้จะมีคาบเวลาของสัญญาณซีกลบ (“0”) เป็นเวลา 10/256 หน่วยของคาบเวลาทั้งหมด และก็จะมีคาบเวลาในซีกบวก(“1”) เป็นเวลา 246/256หน่วยของคาบเวลาทั้งหมด เป็นต้น

```
Config Timer0 = Timer , Gate = Internal , Mode = 2 ' Timer0 = 8Bit Auto Reload
Load Timer0 , 240          'กำหนดค่า Reload ให้ Timer0 = 240x325.52ns = 78.125uS
Disable Timer0             'ห้ามไม่ให้เกิดการ Interrupt จากการทำงานของ Timer0
Start Timer0               'เริ่มการทำงานของ Timer0 เพื่อสร้างสัญญาณ Trig PCA Counter
Cmod = &B00000100          'กำหนดค่าบิต CPS(1:0) = 10 (PCA นับจากการ Overflow ของ
Timer0)
Ccapm1 = &B01000010        'Set Bit ECOM1,PWM1 (Enable PWM1)
Ccapm2 = &B01000010        'Set Bit ECOM2,PWM2 (Enable PWM2)
Ccap1h = 0                 'กำหนดค่าคาบเวลา Period ของ PWM1  = 20mS
Ccap2h = 0                 'กำหนดค่าคาบเวลา Period ของ PWM2  = 20mS
Ccon = &B01000000          'Set Bit CR (เริ่มการทำงานของระบบ PCA Timer)
```

แสดง ส่วนของโปรแกรมการ Initial วงจร PCA Timer ในการสร้าง PWM ของ BASCOM-8051

จากตัวอย่างโปรแกรมข้างต้น เป็นการแสดงให้เห็นวิธีการเขียนโปรแกรมเพื่อควบคุมให้วงจร PCA Timer ทำการสร้างสัญญาณ PWM สำหรับใช้ควบคุมการทำงานของ DC SERVO MOTOR โดยตัวอย่างโปรแกรมนี้จะอ้างอิงจากระบบฮาร์ดแวร์ของบอร์ด ET-ROBOT RD2 โดยใช้ CPU เบอร์ P89C51RD2 ของ Philips ซึ่งทำงานใน X2 Mode หรือ 6 Clock / Machine Cycle โดยอ้างอิงความถี่ของสัญญาณนาฬิการะบบที่ 18.432 MHz ซึ่งแนวคิดและวิธีการในการคำนวณค่าพารามิเตอร์ต่างสำหรับสั่งงานให้ระบบ PCA Timer สร้างสัญญาณ PWM ที่มีคาบเวลา 20mS และมีค่า Duty Cycle ของสัญญาณซีกบวกขนาด 1mS และ 2mS สำหรับควบคุมการทำงานของ DC SERVO MOTOR สามารถแสดงให้เห็นได้ดังขั้นตอนต่อไปนี้

ต้องการคาบเวลาของ PWM(Period)	=	20mS	
วงรอบการนับของ PCA Timer (8 Bit)	=	256 Cycle	
ต้องป้อนสัญญาณนาฬิกาให้ PCA Timer	=	20mS / 256 Cycle	= 78.125 uS
ค่าความถี่ XTAL ที่ป้อนให้กับ CPU	=	18.432 MHz	
คาบเวลาของสัญญาณนาฬิกา 1 Cycle	=	1/18.432 MHz	
	=	54.253 nS	
1 Machine Cycle ของ CPU (X2 Mode)	=	54.253 nS x 6 Clock	
	=	325.520 nS	
1 Machine Cycle ของ Timer0 (X2 Mode)	=	54.253 nS x 6 Clock	= 325.520 nS
ต้องการให้ Timer Overflow ที่ 78.125uS	=	78.125uS / 325.520nS	
	=	240 Cycle	
ต้องกำหนดค่า Auto Reload ให้ Timer0	=	256-240 Cycle	
	=	16 Cycle	
ต้องการคาบเวลาของ PWM ช่วงบวก 1mS	=	1mS / 78.125uS	
	=	12.8 Cycle	= 13 Cycle
	=	256-13 Cycle	= 243 Cycle
ต้องกำหนดค่าเปรียบเทียบให้ CCAPnH	=	243	
ต้องการคาบเวลาของ PWM ช่วงบวก 2mS	=	2mS / 78.125uS	
	=	25.6 Cycle	= 26 Cycle
	=	256-26 Cycle	= 230 Cycle
ต้องกำหนดค่าเปรียบเทียบให้ CCAPnH	=	230	

แสดงวิธีการคำนวณค่าพารามิเตอร์ต่าง ๆสำหรับใช้ในการสร้างสัญญาณ PWM

จากตัวอย่างโปรแกรมข้างต้นจะเห็นว่า จะมีการแบ่งแสดงส่วนของโปรแกรมต่างๆ ออกเป็น 3 ส่วน ด้วยกัน โดยโปรแกรมในส่วนแรกจะเป็นส่วนของการ Initial ค่ารีจิสเตอร์ต่างๆของ PCA Timer และส่วนที่เกี่ยวข้องเพื่อ กำหนดหน้าที่การทำงานของ PCA โมดูลที่ 1 และ 2 (CEX1 หรือ P1.4 และ CEX2 หรือ P1.5) สำหรับทำหน้าที่ในโหมด PWM โดยในตัวอย่างโปรแกรมจะเลือกใช้ Timer0 และ PCA Timer ร่วมกันในการสร้างสัญญาณ PWM ซึ่งระบบ Timer0 จะถูกกำหนดให้ทำหน้าที่สำหรับสร้างฐานเวลาการนับ (Input Clock) เพื่อป้อนให้กับวงจรนับของ PCA Timer ส่วน PCA Timer นั้นจะส่งผลค่าการนับให้กับโมดูล PCA ทั้ง 5 ชุด แต่ในตัวอย่างนั้น จะมีการโปรแกรมการใช้งานโมดูล PCA เพียง 2 ชุด คือ PCA โมดูล1 และ PCA โมดูล 2 สำหรับวิธีการคำนวณค่าต่างที่ต้องกำหนดให้กับรีจิสเตอร์สามารถอธิบายให้เข้าใจพอเป็นแนวทางในการใช้งานได้ดังนี้

จากวิธีการคำนวณหาค่าพารามิเตอร์ต่างๆข้างต้น จะเห็นว่า ระบบการทำงานของ PCA Timer ในโหมดของการสร้างสัญญาณ PWM นั้น การทำงานของวงจรนับจะเป็นแบบ 8 บิต โดยให้รีจิสเตอร์ CL ทำการนับค่า ซึ่งค่าการนับของ CL จะวนรอบการนับอยู่ระหว่าง 00H-FFH หรือ 256 ค่า และเมื่อการนับเกิดการ Overflow หรือเกินจาก FFH ก็จะกลับไปเริ่มต้นนับใหม่จาก 00H อีก โดยก่อนที่จะเริ่มต้นนับใหม่จาก FFH เป็น 00H นั้น จะมีการ Reload ค่าจาก CCAPnH มาให้กับ CCAPnL ก่อนเสมอ โดยค่าของ CCAPnH จะเป็นค่าสำหรับกำหนดคาบเวลาของสัญญาณซีกลบและซีกบวก (Duty Cycle ของ PWM) ส่วนค่าของ 20mS / (CL x 256) จะเป็นค่า Period ของสัญญาณ PWM ที่จะทำการ Reload ซ้ำใหม่

ซึ่งสัญญาณ PWM ที่เราต้องการในการควบคุมการหมุนของ DC SERVO MOTOR นั้น เราต้องการสัญญาณ PWM ที่มีขนาดของคาบเวลา (PERIOD) เท่ากับ 20mS ดังนั้นเพื่อให้การทำงานของโปรแกรมไม่เกิดความซ้ำซ้อน เราจึงไม่ต้องการเข้าไปเปลี่ยนแปลงค่าการนับของรีจิสเตอร์ CL กล่าวคือ จะปล่อยให้รีจิสเตอร์ CL ทำการนับแบบอิสระ แบบวนรอบจาก 00H-FFH ต่อเนื่องกันไปไม่รู้จบ จนกว่าจะมีการสั่งหยุดการนับ ดังนั้นในอันดับแรกเราจึงต้องทำการคำนวณหาค่าของสัญญาณนาฬิกาที่จะป้อนเป็น Input ของการนับของ PCA Timer หรือ CL เพื่อจะได้ทราบว่าจะต้องป้อนค่าสัญญาณนาฬิกาด้วยค่าความถี่เท่าใดให้กับ PCA Timer ทำการนับจำนวน 256ครั้ง (00H-FFH) จึงจะได้คาบเวลา 20mS ซึ่งจะได้เท่ากับ 78.125ไมโครวินาที (78.125uS)

ซึ่งระบบการนับของ PCA Timer นั้นสามารถเลือกแหล่งกำเนิดสัญญาณนาฬิกาได้ 4 แหล่ง แต่แหล่งที่สามารถจะนำมาใช้ได้กับกรณีนี้ คือ จากการ Overflow ของ Timer0 ดังนั้นจะต้องทำการกำหนดโหมดการทำงานของระบบ Timer0 เพื่อให้สร้างสัญญาณนาฬิกาที่มีคาบเวลา 78.125ไมโครวินาที (78.125uS) ซึ่งค่าความถี่ของ XTAL ที่ป้อนให้กับ CPU และ Timer0 นั้นจะเป็น 18.432MHz ซึ่งจะได้คาบเวลาของสัญญาณนาฬิกา 18.432MHz เป็น 1/18.432MHz หรือ 54.253นาโนวินาที(54.253nS) โดยการทำงานของ CPU และ Timer0 จะทำงานแบบ X2 โหมด หรือ 6 Clock / 1Machine Cycle ดังนั้น ซึ่งจะมีค่าเป็น 54.253nS x6 หรือ 325.520 นาโนวินาที (325.520nS)

โดยเราต้องการให้ระบบ Timer0 ทำการนับและเกิดการ Overflow ด้วยคาบเวลา 78.125uS ซึ่ง 1 Cycle ของ Timer0 ที่ทำการนับจะมีค่าเป็น 325.520nS ดังนั้นจะต้องให้ Timer0 ทำการนับจำนวน 240 ครั้ง (78.125uS / 325.520nS) ซึ่งจะเห็นว่าการนับของ Timer มีค่าไม่เกิน 256 ครั้ง ดังนั้นเราจึงทำการกำหนดโหมดการทำงานของ Timer0 ให้ทำงานแบบ 8 Bit Auto-Reload เพื่อโปรแกรมจะได้ไม่เสียเวลาไปทำการ Reload ค่าการนับให้กับ Timer0 ใหม่ เมื่อเกิดการ Overflow ในแต่ละครั้ง แต่เนื่องจากระบบการนับของ Timer0 เป็นแบบนับขึ้น ดังนั้นในการกำหนดค่าการนับให้กับ Timer0 จะต้องนำค่าสูงสุดของการนับ (FFH หรือ 256) มาทำการหักลบออกจากค่าที่ต้องการให้นับ ซึ่งในที่นี้เราต้องการให้ Timer0 ทำการนับ 240 ครั้ง ดังนั้นจึงต้องกำหนดค่าการ Reload ของ Timer0 เป็น 16 (256-240) สำหรับการคำนวณหาค่า Duty Cycle ของสัญญาณ PWM จะคำนวณเพียง 2 ค่า คือ 1mS และ 2mS ซึ่งการนับของ PCA Timer นั้นจะนับจากสัญญาณนาฬิกาค่า 78.125uS ดังนั้นจึงได้ค่าสำหรับกำหนดค่า Duty Cycle ของ PWM เป็น 243(256-(1mS/78.125uS)) และ 230(/256-(2mS/78.125uS)) ตามลำดับ

ซึ่งจากวิธีการกำหนดค่าพารามิเตอร์ต่างๆข้างต้น จะเป็นการควบคุมให้วงจร PWM1(P1.4) และ PWM2(P1.5) ทำการสร้างสัญญาณ PWM ซึ่งมีคาบเวลา หรือ Period ขนาด 20mS คงที่เท่ากันทั้ง2 ช่อง ส่วนค่า Duty Cycle นั้น PWM แต่ละช่อง สามารถกำหนดได้อย่างอิสระตามต้องการระหว่าง 78.125uS ไปจนถึง 20mS โดยค่า Duty Cycle จะกำหนดให้กับรีจิสเตอร์ CCAPnH (PWM1 = CCAP1H ส่วน PWM2 = CCAP2H) โดยค่าที่กำหนดให้กับรีจิสเตอร์ CCAPnH นั้นจะเป็นจุดแบ่งของสัญญาณ PWM ระหว่าง ”0” และ “1” เช่น ถ้ากำหนดค่าของ CCAPnH เป็น 243 จะหมายถึงว่า สัญญาณ PWM จะมีสภาวะเป็น “0” จำนวน 243 Cycle และจะมีสภาวะเป็น “1” จำนวน 13 Cycle ดังนั้นการที่กำหนดให้ค่า CCAPnH มีค่า 243 จึงเป็นการกำหนดให้สัญญาณ PWM มีสภาวะเป็น “0” เป็นเวลา 19mS และมีสภาวะเป็น “1” เป็นเวลา 1mS โดยประมาณนั่นเอง

```
Ccap1h = 256 - 26       'กำหนดค่า Pwm1 Duty Cycle = 2.0ms = ล้อซ้ายเดินหน้า
Ccap2h = 256 - 13       'กำหนดค่า Pwm2 Duty Cycle = 1.0ms = ล้อขวาเดินหน้า
```

แสดง ตัวอย่างการกำหนด Duty Cycle ให้กับ PWM ด้วย BASCOM-8051

การเขียนโปรแกรมควบคุมการเคลื่อนที่ของ SERVO MOTOR ด้วย BASCOM-8051

สำหรับในกรณีที่ผู้ใช้ เลือกภาษาเบสิก BASCOM-8051 เป็นภาษาในการพัฒนาโปรแกรมนั้น ก็นับว่ามีความสะดวกอยู่ไม่น้อยเลยทีเดียว เนื่องจาก BASCOM-8051 เองได้สร้างฟังก์ชั่นคำสั่งสำหรับใช้ควบคุมการทำงานของ SERVO MOTOR จัดเตรียมไว้ให้ใช้งานเป็นที่เรียบร้อยแล้ว โดยใช้คำสั่ง CONFIG SERVOS ซึ่งคำสั่งนี้มีรายละเอียดการใช้งานดังนี้คือ

คำสั่ง CONFIG SERVOS

หน้าที่ คำสั่งนี้ใช้สำหรับกำหนดหน้าที่การทำงานของ Pin Port สำหรับทำหน้าที่สร้างสัญญาณ PWM เพื่อใช้ในการควบคุมการทำงานของ DC SERVO MOTOR

รูปแบบคำสั่ง CONFIG SERVOS = number , SERVOx = Px.y , RELOAD = value

- **number** หมายถึง จำนวนของ SERVO MOTOR ที่ต้องการกำหนดใช้งาน ซึ่งสามารถกำหนดได้สูงสุด 16 ค่าคือ SERVO1 ถึง SERVO16 ตามลำดับ โดยเมื่อกำหนดค่าของ number เป็นจำนวนเท่าใด ก็จะต้องตามด้วยคำสั่งสำหรับกำหนด Port Pin ของ SERVO ตามจำนวนที่กำหนดไว้ด้วย โดยใช้คำสั่ง SERVOx = Px.y
- **SERVOx** หมายถึง หมายเลขของ SERVO โดยจะมีค่าเริ่มต้นจาก 1 ถึง 16
- **Px.y** คือค่า Port Pin ที่จะใช้ควบคุมการทำงานของ SERVO แต่ละตัว เช่น ถ้ากำหนดให้ SERVO1 = P1.4 จะเป็นการกำหนดให้ P1.4 ทำหน้าที่สร้างสัญญาณ Pulse สำหรับควบคุม SERVO ตัวที่1
- **RELOAD** หมายถึง ค่าที่กำหนดให้โปรแกรม BASCOM-8051 ใช้ในการ Reload ค่าให้กับวงจร Timer0 ในการสร้างสัญญาณ Pulse Width Modulation ซึ่งถ้าไม่มีการกำหนดจะมีค่าเป็น 100uS โดยหน่วยเวลานี้จะใช้อ้างอิงจากค่าความถี่ XTAL ค่า12MHz

คำสั่งนี้เป็นคำสั่งแบบ Compiler Directive กล่าวคือ เมื่อ BASCOM-8051 พบคำสั่งนี้แล้วจะมีการตรวจสอบค่าพารามิเตอร์ต่างๆของคำสั่ง และสร้างโปรแกรมและกำหนดองค์ประกอบต่างๆที่จำเป็นให้กับโปรแกรมโดยอัตโนมัติ โดยคำสั่งนี้ BASCOM-8051 จะสงวน Timer0 ไว้ใช้งานในคำสั่งด้วย ดังนั้น เมื่อเรียกใช้คำสั่งนี้แล้วโปรแกรมของผู้ใช้จะไม่สามารถนำ Timer0,การ Interrupt ของ Timer0 และรีจิสเตอร์ต่างๆที่ใช้สำหรับควบคุมการทำงานของ Timer0 ไปใช้งานเพื่อจุดประสงค์อื่นๆได้อีก

โดยคำสั่งนี้ BASCOM-8051 จะใช้ Timer0 ในการนับเวลาเพื่อสร้างสัญญาณ Pulse ให้กับ Port Pin ต่างๆ ซึ่งเมื่อใช้คำสั่งนี้ BASCOM-8051 จะทำการกำหนดหน้าที่การทำงานของวงจร Timer0 ของ CPU ในตระกูล MCS-51 ให้ทำงานในโหมด 8-Bit Auto Reload สำหรับนับเวลาเพื่อสร้างสัญญาณ Pulse ให้กับแต่ละ Port Pin ตามที่กำหนดไว้ในคำสั่ง โดย BASCOM-8051 จะใช้วิธีการ Interrupt ของ Timer0 ตามค่า RELOAD ที่กำหนดไว้ใน Option ของคำสั่ง ซึ่งถ้าไม่มีการกำหนดค่า RELOAD โปรแกรม BASCOM-8051 จะทำการกำหนดค่า RELOAD ให้เองด้วยค่า 100uS (อ้างอิงความถี่ 12MHz) โดยทุกๆครั้งที่เกิดการ Overflow จาก Timer0 จะมีการตรวจสอบและเปลี่ยนแปลงค่าความกว้าง Pulse ของแต่ละ Port Pin ดังนั้นจะเห็นได้ว่าความละเอียดหรือจำนวน Step ในการสร้างสัญญาณ Pulse ของ BASCOM-8051 โดยการใช้คำสั่ง CONFIG SERVOS นี้จะขึ้นอยู่กับค่า RELOAD ที่กำหนดด้วย เช่น ถ้ากำหนดค่า RELOAD ไว้ 100uS ก็จะหมายถึงว่า เราสามารถสั่งให้เปลี่ยนแปลงขนาดความกว้างของ Pulse ได้ครั้งละ 100uS

ตัวอย่างการใช้คำสั่ง

```
CONFIG SERVOS = 2 , SERVO1 = P1.4 , SERVO2 = P1.5 , RELOAD = 100
SERVO1 = 10 'กำหนดให้ P1.4 (Servo1) = 1mS Pulse (10x100uS=1000uS=1mS)
SERVO2 = 20 'กำหนดให้ P1.5 (Servo2) = 2mS Pulse (20x100uS=2000uS=2mS)
```

จากตัวอย่างการใช้คำสั่งข้างต้น จะเป็นการกำหนดให้ BASCOM-8051 สร้างสัญญาณในการควบคุม SERVO จำนวน 2 ตัว โดย SERVO1 จะใช้สัญญาณ P1.4 และ SERVO2 จะใช้สัญญาณ P1.5 ในการควบคุม โดยความละเอียดในการสร้างสัญญาณ Pulse ในแต่ละ Step จะมีค่าเป็น 100uS ซึ่งจากตัวอย่างจะเห็นได้ว่ามีการกำหนดให้โปรแกรมสร้างสัญญาณ Pulse ขนาด 1mS ให้กับ SERVO1 ส่วน SERVO2 จะกำหนดให้สร้างสัญญาณ Pulse ขนาดความกว้างเป็น 2mS โดยหลักการคำนวณค่าความกว้างของ Pulse นั้นเราจะใช้คาบเวลาที่ต้องการเป็นตัวตั้งแล้วหารด้วยค่า RELOAD ซึ่งจะได้ผลลัพธ์เป็นค่าที่จะต้องกำหนดให้กับ SERVO แต่ละตัว ตัวอย่างเช่น ต้องการกำหนดให้สร้างสัญญาณ Pulse ขนาด 1mS ให้กับ SERVO1 จะได้ว่า

ค่าที่จะต้องกำหนดให้กับ SERVO = ค่าความกว้าง Pulse ที่ต้องการ / ค่า Reload

= 1mS / 100uS = $1x10^{-3}$ / $1x10^{-6}$

= 10

ดังนั้นจะต้องกำหนดค่าให้กับ SERVO1 ด้วย 10 เพื่อสร้าง Pulse ที่มีความกว้างเป็น 1mS

ตัวอย่างการคำนวณ การกำหนดค่าให้กับ SERVO กรณีใช้งานกับความถี่ 12MHz

เนื่องจากคำสั่ง CONFIG SERVOS นี้ BASCOM-8051 สร้างคำสั่งโดยใช้อ้างอิงการทำงานของ CPU ที่ทำงานด้วยความเร็วจากค่าความถี่ของ XTAL ที่ 12MHz ซึ่งถ้าระบบฮาร์ดแวร์ของบอร์ดที่ใช้ทำงานที่ความถี่ XTAL = 12MHz ค่าต่างๆจะถูกต้องตามหลักการคำนวณที่แสดงให้เห็นในข้างทั้งหมด แต่ในกรณีที่การทำงานของ CPU มีความเร็วที่แตกต่างจาก 12MHz มากๆแล้ว การทำงานของคำสั่งนี้จะมีความผิดพลาดเกิดขึ้นเนื่องจากพบว่า BASCOM-8051 จะกำหนดโหมดการทำงานของ Timer0 ให้ทำงานในโหมดการนับแบบ 8-Bit Auto Reload ดังนั้นเมื่อการทำงานของ CPU เร็วมากๆค่าที่จะกำหนดให้ Timer0 ทำการนับนั้น ในบางครั้งจะมีค่ามากกว่า 256 ค่า ทำให้ค่าหน่วยเวลาต่างๆไม่เป็นไปตามที่คำนวณได้

ตัวอย่างเช่น ในกรณีของบอร์ด ET-ROBOT RD2 นั้น CPU จะทำงานด้วยค่าความถี่ XTAL = 18.432MHz ด้วยความเร็วที่เร็วกว่า CPU ในตระกูล MCS51 ปรกติถึง 2 เท่าตัว (X2 Mode) ดังนั้นเพื่อให้ง่ายต่อการทำความเข้าใจจึงขอเทียบค่าความเร็วในการทำงานของ CPU เป็น 18.432MHz คูณด้วย 2 ซึ่งจะมีค่าเท่ากับ 36.864MHz ซึ่งเมื่อใช้คำสั่ง CONFIG SERVOS โดยมีการกำหนดค่า RELOAD เป็น 100uS จะได้ว่า

คาบเวลาของ XTAL 36.864MHz	= 1 / 36.864MHz
	= 27.127nS (27.127×10^{-9} Sec)
1 Machine Cycle ของ CPU	= 12 x 27.127nS
	= 325.5nS
ต้องการคาบเวลา 100uS	= 100uS / 325.5ns
	= 307.134 หรือ 0133H

ซึ่งจะเห็นได้ว่าค่าเวลาที่คำนวณได้นั้นมีค่ามากกว่า 255(FFH) ซึ่งเกินค่าการ Overflow ของ Timer0 ซึ่งถูกกำหนดให้ทำงานในโหมดการนับแบบ 8 บิต ซึ่งในกรณี เช่นนี้ พบว่าโปรแกรม BASCOM-8051 เอง จะมีข้อผิดพลาดจากการแปลคำสั่ง โดยพบว่าโปรแกรม BASCOM-8051 จะนำค่าใน 8บิตล่าง (จากตัวอย่างคือ 33H หรือ 51) ไปกำหนดให้กับ Timer0 ทำการนับ ซึ่งแทนที่ Timer0 จะนับได้ค่าเวลาเป็น 100uS ตามที่คำนวณไว้ จะกลายเป็นว่า Timer0 นับได้เพียง16.6uS เท่านั้น ดังนั้นในกรณีของการเขียนโปรแกรมควบคุมการทำงานของ SERVO ด้วยโปรแกรม BASCOM-8051 โดยใช้กับบอร์ด ET-ROBOT RD2 นั้นจะต้องทำการคำนวณค่าเป็นการเฉพาะเพื่อแก้ไขปัญหาดังกล่าวแทน โดยใช้ค่าเวลาในการ RELOAD ที่ไม่เกิน 255 (FFH) ค่า ซึ่งในที่นี้เราจะใช้วิธีการกำหนดให้ค่า RELOAD มีค่าเป็นศูนย์ ซึ่งจะทำให้การนับของ Timer0 มีค่าเท่ากับ 256 ครั้ง เนื่องจาก Timer0 จะทำการนับก่อนแล้วจึงตรวจสอบค่าว่าเกิดการ Overflow หรือยัง ซึ่งเมื่อกำหนดค่า RELOAD เป็นศูนย์ ก็จะเป็นการกำหนดให้การนับของ

Timer0 ต้องนับจนกว่าค่าการนับจะย้อนกลับมาเป็นศูนย์อีกครั้งหนึ่ง ซึ่งก็คือ 256 ครั้งพอดี (0,1,2,3…255,0) ซึ่งในกรณีนี้ค่าเวลาที่ได้จากการนับในแต่ละครั้งจะสามารถคำนวณได้ ดังตัวอย่าง

1 Machine Cycle ของ CPU	= 12 x 27.127nS
	= 325.5nS
ค่าการนับของ Timer0 256ครั้ง	= 256 x 325.5nS
	= 83.35uS
ต้องการคาบเวลา 1mS	= 1mS / 83.35uS
	= 11.99
ต้องกำหนดค่าให้กับ SERVO	= 12
ต้องการคาบเวลา 2mS	= 2mS / 83.35uS
	= 23.99
ต้องกำหนดค่าให้กับ SERVO	= 24

ตัวอย่างการคำนวณ การกำหนดค่าให้กับ SERVO กรณีใช้งานกับความถี่ 36.864MHz

ดังนั้น ในกรณีที่ใช้งานคำสั่ง CONFIG SERVOS กับรถหุ่นยนต์ ET-ROBOT RD2 นั้น การกำหนดค่าให้กับ SERVO จึงต้องใช้วิธีการคำนวณตามรูปแบบข้างต้น ซึ่งสามารถแสดงให้เห็นส่วนของโปรแกรมสำหรับควบคุม SERVO ดังนี้

```
CONFIG SERVOS = 2 , SERVO1 = P1.4 , SERVO2 = P1.5 , RELOAD = 0
SERVO1 = 12          'กำหนดให้ P1.4 (Servo1) = 1mS Pulse
SERVO2 = 24          'กำหนดให้ P1.5 (Servo2) = 2mS Pulse
```

แสดง ตัวอย่างการควบคุม SERVO ด้วย BASCOM-8051

```
'/*************************************/;
'/* Program Demo For ET-ROBOT RD2 V1.0 */;
'/* Controller  : P89C51RD2(Philips)   */;
'/* Run X-TAL   : 18.432 Mhz           */;
'/*             : X2 Mode(6 Cycle Run) */;
'/* Complier    : BASCOM-51(V2.0.11.0) */;
'/* Generate Pulse Control Servo Motor */;
'/*************************************/;

$regfile = "89c51rd.dat"        'กำหนดให้ใช้งานกับ CPU เบอร์ P89C51RD2(Philips)
$ramstart = 0
$ramsize = 256
$crystal = 36864000             'กำหนดค่า XTAL = 18.432MHz แบบ X2 Mode
Config Servos = 2 , Servo1 = P1.4 , Servo2 = P1.5 , Reload = 0
Declare Sub Robot_forward       'โปรแกรมย่อยสำหรับบังคับให้รถหุ่นยนต์เดินหน้า
Declare Sub Robot_backward      'โปรแกรมย่อยสำหรับบังคับให้รถหุ่นยนต์ถอยหลัง
Declare Sub Robot_fast_left     'โปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวซ้ายอย่างเร็ว
Declare Sub Robot_slow_left     'โปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวซ้ายแบบปรกติ
Declare Sub Robot_fast_right    'โปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวขวาอย่างเร็ว
Declare Sub Robot_slow_right    'โปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวขวาแบบปรกติ
Declare Sub Robot_stop          'โปรแกรมย่อยสำหรับบังคับให้รถหุ่นยนต์หยุด
Robot_stop                      'สั่งให้รถหยุดอยู่กับที่ก่อน เพื่อรอการกดสวิตช์ Start
Bitwait P0.2 , Reset            'รอการกดสวิตช์ Start ก่อนจึงเริ่มทำงาน (P0.2="0")

Do                              'จุดเริ่มต้นของคำสั่งวนรอบ (DO..LOOP)
  Robot_forward                 'สั่งให้รถหุ่นยนต์เคลื่อนที่ไปข้างหน้า
  Wait 1                        'หน่วงเวลาสำหรับการเคลื่อนที่ไปข้างหน้า
  Robot_stop                    'สั่งให้รถหุ่นยนต์หยุด
  Wait 1                        'หน่วงเวลาสำหรับหยุด

  Robot_backward                'สั่งให้รถหุ่นยนต์ถอยหลัง
  Wait 1                        'หน่วงเวลาสำหรับการถอยหลัง
  Robot_stop                    'สั่งให้รถหุ่นยนต์หยุด
  Wait 1                        'หน่วงเวลาสำหรับหยุด

  Robot_fast_right              'สั่งให้รถหุ่นยนต์เลี้ยวขวาอย่างเร็ว
```

```
  Wait 1                          'หน่วงเวลาสำหรับการเลี้ยว
  Robot_stop                      'สั่งให้รถหุ่นยนต์หยุด
  Wait 1                          'หน่วงเวลาสำหรับหยุด

  Robot_fast_left                 'สั่งให้รถหุ่นยนต์เลี้ยวซ้ายอย่างเร็ว
  Wait 1                          'หน่วงเวลาสำหรับเลี้ยว
  Robot_stop                      'สั่งให้รถหุ่นยนต์หยุด
  Wait 1                          'หน่วงเวลาสำหรับหยุด

  Robot_slow_right                'สั่งให้รถหุ่นยนต์เลี้ยวขวาแบบปรกติ
  Wait 4                          'หน่วงเวลาสำหรับเลี้ยว
  Robot_stop                      'สั่งให้รถหุ่นยนต์หยุด
  Wait 1                          'หน่วงเวลาสำหรับหยุด

  Robot_slow_left                 'สั่งให้รถหุ่นยนต์เลี้ยวซ้ายแบบปรกติ
  Wait 4                          'หน่วงเวลาสำหรับเลี้ยว
  Robot_stop                      'สั่งให้รถหุ่นยนต์หยุด
  Wait 1                          'หน่วงเวลาสำหรับหยุด
Loop                              'ย้อนกลับไปวนรอบจากจุดเริ่มต้น(เดินหน้า)ใหม่


'/*****************/;
'/* ROBOT Forward */;
'/*****************/;
'
Sub Robot_forward                 'จุดเริ่มต้นโปรแกรมสำหรับให้รถหุ่นยนต์เคลื่อนที่ไปข้างหน้า
  Servo1 = 24                     'กำหนดให้ SERVO1(ซ้าย) = 2mS = เคลื่อนที่ไปข้างหน้า
  Servo2 = 12                     'กำหนดให้ SERVO2(ขวา) = 1mS = เคลื่อนที่ไปข้างหน้า
End Sub                           'จุดสิ้นสุดโปรแกรมสำหรับให้รถหุ่นยนต์เคลื่อนที่ไปข้างหน้า


'/******************/;
'/* ROBOT Backward */;
'/******************/;
'
Sub Robot_backward                'จุดเริ่มต้นโปรแกรมย่อยสำหรับบังคับให้รถหุ่นยนต์ถอยหลัง
  Servo1 = 12                     'กำหนดให้ SERVO1(ซ้าย) = 1mS = ถอยหลัง
```

```
  Servo2 = 24              'กำหนดให้ SERVO2(ขวา) = 2mS = ถอยหลัง
End Sub                    'จุดสิ้นสุดโปรแกรมย่อยสำหรับบังคับให้รถหุ่นยนต์ถอยหลัง


'/*************************/;
'/* ROBOT Turn Left(Fast) */;
'/*************************/;
'
Sub Robot_fast_left        'จุดเริ่มต้นโปรแกรมย่อยสำหรับให้รถหุ่นยนต์เลี้ยวซ้ายอย่างเร็ว
  Servo1 = 12              'กำหนดให้ SERVO1(ซ้าย) = 1mS = ถอยหลัง
  Servo2 = 12              'กำหนดให้ SERVO2(ขวา) = 1mS = เคลื่อนที่ไปข้างหน้า
End Sub                    'จุดสิ้นสุดโปรแกรมย่อยสำหรับให้รถหุ่นยนต์เลี้ยวซ้ายอย่างเร็ว


'/*************************/;
'/* ROBOT Turn Left(Slow) */;
'/*************************/;
'
Sub Robot_slow_left        'จุดเริ่มต้นโปรแกรมสำหรับให้รถหุ่นยนต์เลี้ยวซ้ายแบบปรกติ
  Servo1 = -1              'กำหนดให้ SERVO1 (ซ้าย) = "1" = หยุดการเคลื่อนที่
  Servo2 = 12              'กำหนดให้ SERVO2(ขวา) = 1mS = เคลื่อนที่ไปข้างหน้า
End Sub                    'จุดสิ้นสุดโปรแกรมสำหรับให้รถหุ่นยนต์เลี้ยวซ้ายแบบปรกติ


'/*************************/;
'/* ROBOT Turn Right(Fast) */;
'/*************************/;
'
Sub Robot_fast_right       'จุดเริ่มต้นโปรแกรมย่อยสำหรับให้รถหุ่นยนต์เลี้ยวขวาอย่างเร็ว
  Servo1 = 24              'กำหนดให้ SERVO1(ซ้าย) = 2mS = เคลื่อนที่ไปข้างหน้า
  Servo2 = 24              'กำหนดให้ SERVO2(ขวา) = 2mS = ถอยหลัง
End Sub                    'จุดสิ้นสุดโปรแกรมย่อยสำหรับให้รถหุ่นยนต์เลี้ยวขวาอย่างเร็ว


'/*************************/;
'/* ROBOT Turn Right(Slow) */;
'/*************************/;
'
Sub Robot_slow_right       'จุดเริ่มต้นโปรแกรมสำหรับให้รถหุ่นยนต์เลี้ยวขวาแบบปรกติ
  Servo1 = 24              'กำหนดให้ SERVO1(ซ้าย) = 2mS = เคลื่อนที่ไปข้างหน้า
```

```
  Servo2 = -1                ‘กำหนดให้ SERVO2 (ขวา) = "1" = หยุดการเคลื่อนที่
End Sub                      ‘จุดสิ้นสุดโปรแกรมสำหรับให้รถหุ่นยนต์เลี้ยวขวาแบบปรกติ


'/**************;
'/* ROBOT Stop *;
'/**************;
'
Sub Robot_stop               ‘จุดเริ่มต้นโปรแกรมย่อยสำหรับบังคับให้รถหุ่นยนต์หยุด
  Servo1 = -1                ‘กำหนดให้ SERVO1 (ซ้าย) = "1" = หยุดการเคลื่อนที่
  Servo2 = -1                ‘กำหนดให้ SERVO2 (ขวา) = "1" = หยุดการเคลื่อนที่
End Sub                      ‘จุดสิ้นสุดโปรแกรมย่อยสำหรับบังคับให้รถหุ่นยนต์หยุด

End
```

จากตัวอย่างโปรแกรมข้างต้น เป็นตัวอย่างโปรแกรมสำหรับควบคุมการเคลื่อนที่ของตัวรถหุ่นยนต์ ET-ROBOT RD2 ให้เคลื่อนที่ไปในทิศทางต่างๆ เช่น การเดินหน้า การถอยหลัง การเลี้ยวซ้ายอย่างเร็ว การเลี้ยวขวาอย่างเร็ว การเลี้ยวซ้ายแบบปรกติ และ การเลี้ยวขวาแบบปรกติ โดยจากตัวอย่างนั้น ส่วนของโปรแกรมสำหรับควบคุมการเคลื่อนที่ในลักษณะต่างๆดังกล่าวข้างต้น นั้นจะสร้างเป็นโปรแกรมย่อยไว้ ซึ่งในบทต่อๆไป เราสามารถนำเอาส่วนของโปรแกรมย่อยต่างๆเหล่านี้ไปใช้งานได้ทันที โดยการทำงานของโปรแกรมหลักจะอยู่ในส่วนของกลุ่มคำสั่งต่างๆที่อยู่ในวงรอบ DO..LOOP ซึ่งเป็นการสั่งให้รถเคลื่อนที่ไปในทิศทางต่างๆสลับกับการหยุดนิ่ง โดยจะเริ่มต้นด้วยการสั่งให้รถหุ่นยนต์เคลื่อนที่ไปข้างหน้า แล้วจึงสั่งหน่วงเวลาไว้ระยะหนึ่งเพื่อให้รถเคลื่อนที่ไปข้างหน้า(คำสั่ง WAIT) จากนั้นจึงสั่งให้รถหยุดพร้อมกับหน่วงเวลาการหยุดไว้ระยะหนึ่ง แล้ว ตามด้วยการสั่งเลี้ยวขวาอย่างเร็ว,หยุดเคลื่อนที่,เลี้ยวซ้ายอย่างเร็ว,หยุดเคลื่อนที่,เลี้ยวขวาแบบปรกติ,หยุดการเคลื่อนที่,เลี้ยวซ้ายแบบปรกติ,หยุดการเคลื่อนที่ โดยโปรแกรมจะวนทำงานซ้ำอย่างนี้ไปเรื่อยๆไม่รู้จบ โดยสาระ สำคัญของโปรแกรมมีส่วนที่ควรทำความเข้าใจดังนี้

```
Bitwait P0.2 , Reset        'รอการกดสวิตช์ Start ก่อนจึงเริ่มทำงาน (P0.2="0")
```

คำสั่งนี้ใช้สำหรับสั่งให้โปรแกรมรอจนกว่าขาสัญญาณ P0.2 จะมีค่าเป็น “0” ซึ่ง P0.2 จะต่อไว้กับ Switch-Start ดังนั้นคำสั่งนี้จึงใช้สำหรับรอให้มีการกด Switch-Start เสียก่อน โปรแกรมจึงจะเริ่มต้นทำงาน ซึ่งถ้าไม่มีการใช้คำสั่งนี้ในโปรแกรม อาจเกิดปัญหาขึ้นได้ เช่น หลังจากการ Download โปรแกรมให้กับบอร์ดเสร็จแล้วอาจทำให้รถเคลื่อนที่ไปข้างหน้าทันที

```
$CRYSTAL = 36864000
```

คำสั่งนี้เป็นการประกาศให้ BASCOM-8051 ทราบว่าระบบฮาร์ดแวร์ของบอร์ดทำงานด้วยความเร็วของสัญญาณ Crystal ความถี่ 36.864MHz (ในบอร์ดใช้ 18.432MHz แต่ CPU ทำงานด้วยความเร็วเป็น 2 เท่า)

```
Config Servos = 2 , Servo1 = P1.4 , Servo2 = P1.5 , Reload = 0
```

คำสั่งนี้ใช้กำหนดให้ BASCOM-8051 สร้างสัญญาณ Pulse สำหรับควบคุมการทำงานของ SERVO จำนวน 2 ชุด โดย SERVO1 ใช้สัญญาณ P1.4 ส่วน SERVO2 ใช้สัญญาณ P1.5 ในการควบคุมการทำงานของ SERVO โดยค่าการ Reload จะกำหนดให้มีค่าเป็นศูนย์ ดังได้อธิบายเหตุผลไปแล้วในข้างต้นที่ผ่านมา

ซึ่งจะเห็นได้ว่าในส่วนของโปรแกรมย่อยต่างๆนั้น จะประกอบขึ้นจากชุดคำสั่งสำหรับควบคุมการเคลื่อนที่ของ SERVO ที่ใช้ในการควบคุมการหมุนของ ล้อซ้าย และ ล้อขวา เพื่อบังคับให้ล้อ ซ้ายและขวาหยุดหมุน หรือ หมุนแบบตามเข็ม หรือ ทวนเข็ม เพื่อให้ได้ทิศทางการเคลื่อนที่ของตัวรถหุ่นยนต์ตามที่เราต้องการ ซึ่งทิศทางการเคลื่อนที่ของตัวรถที่มีความจำเป็นต้องใช้งานนั้น จะมีอยู่ 7 ลักษณะด้วยกันคือ การหยุดการเคลื่อนที่หรืออยู่กับที่ การเคลื่อนที่ไปข้างหน้า การเคลื่อนที่ถอยหลัง การเลี้ยวซ้ายแบบรวดเร็ว การเลี้ยวขวาแบบรวดเร็ว การเลี้ยวซ้ายแบบปรกติ และ การเลี้ยวขวาแบบปรกติ

ซึ่งเราสามารถนำเอาลักษณะการเคลื่อนที่ต่างๆเหล่านี้มาเรียบเรียงเพื่อควบคุมการเคลื่อนที่ของตัวรถให้ไปในตำแหน่งและทิศทางตามที่เราต้องการได้ ซึ่งอาจต้องพิจารณาเงื่อนไขและสิ่งแวดล้อมต่างๆเป็นเงื่อนไขประกอบด้วย โดยการสั่งงานในรถเคลื่อนที่แบบมีเงื่อนไขนั้นจะได้ศึกษากันในบทต่อไป

การควบคุม SERVO MOTOR ด้วย PWM โดยใช้ภาษาเบสิก BASCOM-8051

ดังได้ทราบมาแล้วว่า CPU เบอร์ P89C51RD2 นั้นจะมีวงจร PWM บรรจุไว้ภายในตัวอยู่แล้ว ซึ่งการนำเอาสัญญาณ PWM มาใช้ในการควบคุมการทำงานของ SERVO MOTOR นั้น นับว่าเป็นวิธีการที่มีประสิทธิภาพมากที่สุด เนื่องจากกระบวนการสร้างสัญญาณ Pulse สำหรับนำมาใช้ควบคุมการทำงานของวงจร PWM นั้น จะไม่รบกวนการทำงานของ CPU เลย ส่วนในกรณีที่ใช้คำสั่ง CONFIG SERVO ของ BASCOM-8051 ในการสร้างสัญญาณ Pulse นั้น CPU จะต้องวนรอบทำการตรวจสอบการสร้างสัญญาณอยู่ตลอดเวลา ทำให้ประสิทธิภาพการทำงานของ CPU ลดต่ำลงกว่าที่ควรจะเป็น ซึ่งถึงแม้ว่า BASCOM-8051 เอง จะไม่มีคำสั่งสำหรับควบคุมการทำงานของ PWM โดยตรง แต่เราก็สามารถทำการเขียนโปรแกรมสั่งงานให้วงจร PWM ทำหน้าที่สร้างสัญญาณ Pulse ออกไปใช้ควบคุมการทำงานของ SERVO MOTOR ได้เอง ดังได้กล่าวอธิบายไปแล้วในข้างต้น

ซึ่งในที่นี้จะขอแสดงให้เห็นถึงวิธีการเขียนโปรแกรมด้วย BASCOM-8051 ในการสั่งให้วงจร PWM ทำการสร้างสัญญาณ Pulse เพื่อส่งออกไปควบคุมการทำงานของ SERVO MOTOR เอง โดยลักษณะการทำงานของโปรแกรมในตัวอย่างนี้จะเหมือนกับตัวอย่างของการใช้คำสั่ง CONFIG SERVO ที่ผ่านมาทุกประการ โดยมีการดัดแปลงโปรแกรมเพียง 2 ส่วน คือ

- เปลี่ยนจากการใช้คำสั่ง CONFIG SERVO เป็นกลุ่มคำสั่งสำหรับ Initial ให้ Timer0 และ PCA สำหรับสร้าง PWM
- เปลี่ยนจากการใช้คำสั่ง SERVO1 เป็นการกำหนดค่าให้กับรีจิสเตอร์ CCAP1H
- เปลี่ยนจากการใช้คำสั่ง SERVO2 เป็นการกำหนดค่าให้กับรีจิสเตอร์ CCAP2H

โดยตัวอย่างโปรแกรมที่จะแสดงให้เห็นต่อไปนี้ สามารถนำไปเพื่อใช้ในการสร้างสัญญาณ Pulse สำหรับควบคุมการเคลื่อนที่ของตัวรถหุ่นยนต์ ET-ROBOT RD2 ได้ทันที โดยลักษณะของโปรแกรมสามารถแสดงให้เห็นได้ดังต่อไปนี้

```
'/*************************************/;
'/* Program Demo For ET-ROBOT RD2 V1.0 */;
'/* Controller  : P89C51RD2(Philips)   */;
'/* Run X-TAL   : 18.432 Mhz           */;
'/*             : X2 Mode(6 Cycle Run) */;
'/* Complier    : BASCOM-51(V2.0.11.0) */;
'/* Generate PWM -> Control Servo Motor*/;
'/*************************************/;

$regfile = "89c51rd.dat"    'กำหนดให้ใช้งานกับ CPU เบอร์ P89C51RD2(Philips)
$ramstart = 0
$ramsize = 256
$crystal = 36864000         'กำหนดค่าความเร็ว XTAL = 18.432MHz แบบ X2 Mode

Config Timer0 = Timer , Gate = Internal , Mode = 2 ' Timer0 =
8Bit Auto Reload
Load Timer0 , 240           'กำหนดค่า Reload  = 240x325.52ns = 78.125uS
Disable Timer0              'ห้ามไม่ให้เกิดการ Interrupt จากการทำงานของ Timer0
Start Timer0                'เริ่มการทำงานของ Timer0 เพื่อ Trig PCA Counter
Cmod = &B00000100           'ให้บิต CPS(1:0) = 10 (นับจาก Overflow ของ Timer0)
Ccapm1 = &B01000010         'Set Bit ECOM1,PWM1 (Enable PWM1)
Ccapm2 = &B01000010         'Set Bit ECOM2,PWM2 (Enable PWM2)
Ccap1h = 0                  'กำหนดค่า PWM1 Duty Cycle = 20mS
Ccap2h = 0                  'กำหนดค่า PWM2 Duty Cycle = 20mS
Ccon = &B01000000           'Set Bit CR (เริ่มการทำงานของระบบ PCA Timer)

Declare Sub Robot_forward      'โปรแกรมย่อยสำหรับบังคับให้รถเดินหน้า
Declare Sub Robot_backward     'โปรแกรมย่อยสำหรับบังคับให้รถถอยหลัง
Declare Sub Robot_fast_left    'โปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวซ้ายอย่างเร็ว
Declare Sub Robot_slow_left    'โปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวซ้ายแบบปรกติ
Declare Sub Robot_fast_right   'โปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวขวาอย่างเร็ว
Declare Sub Robot_slow_right   'โปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวขวาแบบปรกติ
Declare Sub Robot_stop         'โปรแกรมย่อยสำหรับบังคับให้รถหยุด

Robot_stop                     'สั่งให้รถหยุดอยู่กับที่ก่อน เพื่อรอการกดสวิตช์ Start
Bitwait P0.2 , Reset           'รอการกดสวิตช์ Start ก่อนจึงเริ่มทำงาน (P0.2="0")
```

```
Do                              'จุดเริ่มต้นของคำสั่งวนรอบ (DO..LOOP)

  Robot_forward                 'สั่งให้รถหุ่นยนต์เคลื่อนที่ไปข้างหน้า
  Wait 1                        'หน่วงเวลาสำหรับการเคลื่อนที่ไปข้างหน้า
  Robot_stop                    'สั่งให้รถหุ่นยนต์หยุด
  Wait 1                        'หน่วงเวลาสำหรับหยุด

  Robot_backward                'สั่งให้รถหุ่นยนต์ถอยหลัง
  Wait 1                        'หน่วงเวลาสำหรับการถอยหลัง
  Robot_stop                    'สั่งให้รถหุ่นยนต์หยุด
  Wait 1                        'หน่วงเวลาสำหรับหยุด

  Robot_fast_right              'สั่งให้รถหุ่นยนต์เลี้ยวขวาอย่างเร็ว
  Wait 1                        'หน่วงเวลาสำหรับการเลี้ยว
  Robot_stop                    'สั่งให้รถหุ่นยนต์หยุด
  Wait 1                        'หน่วงเวลาสำหรับหยุด

  Robot_fast_left               'สั่งให้รถหุ่นยนต์เลี้ยวซ้ายอย่างเร็ว
  Wait 1                        'หน่วงเวลาสำหรับเลี้ยว
  Robot_stop                    'สั่งให้รถหุ่นยนต์หยุด
  Wait 1                        'หน่วงเวลาสำหรับหยุด

  Robot_slow_right              'สั่งให้รถหุ่นยนต์เลี้ยวขวาแบบปรกติ
  Wait 4                        'หน่วงเวลาสำหรับเลี้ยว
  Robot_stop                    'สั่งให้รถหุ่นยนต์หยุด
  Wait 1                        'หน่วงเวลาสำหรับหยุด

  Robot_slow_left               'สั่งให้รถหุ่นยนต์เลี้ยวซ้ายแบบปรกติ
  Wait 4                        'หน่วงเวลาสำหรับเลี้ยว
  Robot_stop                    'สั่งให้รถหุ่นยนต์หยุด
  Wait 1                        'หน่วงเวลาสำหรับหยุด

Loop                            'ย้อนกลับไปวนรอบจากจุดเริ่มต้น(เดินหน้า)ใหม่
```

```
'/*****************/;
'/* ROBOT Forward */;
'/*****************/;
'
Sub Robot_forward          'จุดเริ่มต้นโปรแกรมย่อยสำหรับบังคับให้รถเคลื่อนที่ไปข้างหน้า
  Ccap1h = 256 - 26        'กำหนดให้ SERVO1(ซ้าย) = 2mS = เคลื่อนที่ไปข้างหน้า
  Ccap2h = 256 - 13        'กำหนดให้ SERVO2(ขวา) = 1mS = เคลื่อนที่ไปข้างหน้า
End Sub                    'จุดสิ้นสุดโปรแกรมย่อยสำหรับบังคับให้รถเคลื่อนที่ไปข้างหน้า


'/******************/;
'/* ROBOT Backward */;
'/******************/;
'
Sub Robot_backward         'จุดเริ่มต้นโปรแกรมย่อยสำหรับบังคับให้รถหุ่นยนต์ถอยหลัง
  Ccap1h = 256 - 13        'กำหนดให้ SERVO1(ซ้าย) = 1mS = ถอยหลัง
  Ccap2h = 256 - 26        'กำหนดให้ SERVO2(ขวา) = 2mS = ถอยหลัง
End Sub                    'จุดสิ้นสุดโปรแกรมย่อยสำหรับบังคับให้รถหุ่นยนต์ถอยหลัง


'/*************************/;
'/* ROBOT Turn Left(Fast) */;
'/*************************/;
'
Sub Robot_fast_left        'จุดเริ่มต้นโปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวซ้ายอย่างเร็ว
  Ccap1h = 256 - 13        'กำหนดให้ SERVO1(ซ้าย) = 1mS = ถอยหลัง
  Ccap2h = 256 - 13        'กำหนดให้ SERVO2(ขวา) = 1mS = เคลื่อนที่ไปข้างหน้า
End Sub                    'จุดสิ้นสุดโปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวซ้ายอย่างเร็ว


'/*************************/;
'/* ROBOT Turn Left(Slow) */;
'/*************************/;
'
Sub Robot_slow_left        'จุดเริ่มต้นโปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวซ้ายแบบปรกติ
  Ccap1h = 0               'กำหนดให้ SERVO1 (ซ้าย) = "1" = หยุดการเคลื่อนที่
  Ccap2h = 256 - 13        'กำหนดให้ SERVO2(ขวา) = 1mS = เคลื่อนที่ไปข้างหน้า
End Sub                    'จุดสิ้นสุดโปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวซ้ายแบบปรกติ
```

```
'/*************************/;
'/* ROBOT Turn Right(Fast) */;
'/*************************/;
'
Sub Robot_fast_right      'จุดเริ่มต้นโปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวขวาอย่างเร็ว
  Ccap1h = 256 - 26       'กำหนดให้ SERVO1(ซ้าย) = 2mS = เคลื่อนที่ไปข้างหน้า
  Ccap2h = 256 - 26       'กำหนดให้ SERVO2(ขวา) = 2mS = ถอยหลัง
End Sub                   'จุดสิ้นสุดโปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวขวาอย่างเร็ว


'/*************************/;
'/* ROBOT Turn Right(Slow) */;
'/*************************/;
'
Sub Robot_slow_right      'จุดเริ่มต้นโปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวขวาแบบปรกติ
  Ccap1h = 256 - 26       'กำหนดให้ SERVO1(ซ้าย) = 2mS = เคลื่อนที่ไปข้างหน้า
  Ccap2h = 0              'กำหนดให้ SERVO2 (ขวา) = "1" = หยุดการเคลื่อนที่
End Sub                   'จุดสิ้นสุดโปรแกรมย่อยสำหรับบังคับให้รถเลี้ยวขวาแบบปรกติ


'/**************/;
'/* ROBOT Stop */;
'/**************/;
'
Sub Robot_stop            'จุดเริ่มต้นโปรแกรมย่อยสำหรับบังคับให้รถหุ่นยนต์หยุด
  Ccap1h = 0              'กำหนดให้ SERVO1 (ซ้าย) = "1" = หยุดการเคลื่อนที่
  Ccap2h = 0              'กำหนดให้ SERVO2 (ขวา) = "1" = หยุดการเคลื่อนที่
End Sub                   'จุดสิ้นสุดโปรแกรมย่อยสำหรับบังคับให้รถหุ่นยนต์หยุด

End
```

จากตัวอย่างโปรแกรมที่แสดงให้เห็นนี้ สามารถนำไปใช้งานสำหรับควบคุมการเคลื่อนที่ของตัวรถหุ่นยนต์ได้ทันที ซึ่งสำหรับบอร์ด ET-ROBOT RD2 นั้น ขอแนะนำให้ใช้วิธีการควบคุม SERVO MOTOR ด้วยระบบฮาร์ดแวร์ของ PWM ตามแนวทางที่แสดงไว้ในตัวอย่างนี้จะดีที่สุด เนื่องจากวิธีการนี้จะช่วยให้การทำงานของ CPU เกิดประสิทธิภาพสูงสุดเพราะไม่ต้องเสียเวลามา สร้างสัญญาณ Pulse เอง และเป็นการใช้ความสามารถของระบบฮาร์ดแวร์ต่างๆได้อย่างคุ้มค่าและมีประสิทธิภาพมากที่สุดด้วย