เอกสารการออกแบบ

(BotBot)

จัดทำโดย

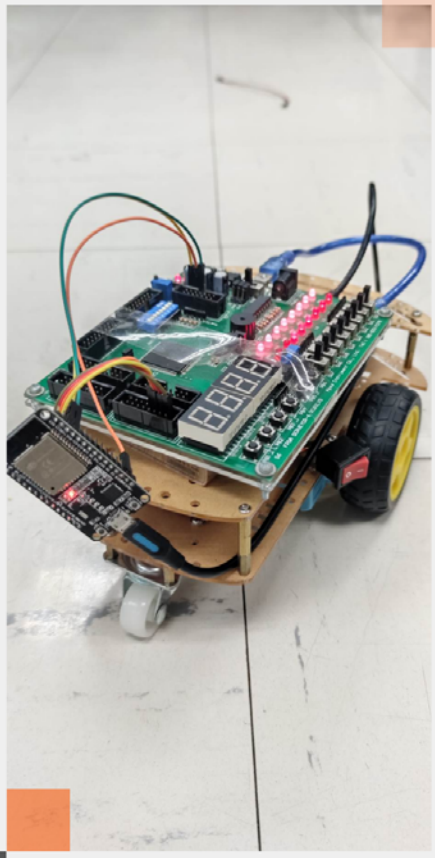
นายพชรพล	โชคคุณ	63010631	Sec 1
นางสาวภัฑราณิษฐ์	เทศเจริญ	63010727	Sec 1
นายมาเหนือเมฆ	ประดิษฐพงษ์	63010789	Sec 1

เสนอ

รศ.ดร เจริญ วงษ์ชุ่มเย็น

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ภาคเรียนที่ 2 ปีการศึกษา 2565



BotBot

หุ่นยนต์บังคับไร้สายด้วยบอร์ด FPGA และ ESP32

ไอเดีย

อุปกรณ์

- FPGA
- NodeMCU ESP32
- Motor และล้อ
- แหล่งจ่ายพลังงาน
- โครงสร้างอื่น ๆ เช่น อะคริลิค

ที่มาละความสำคัญ

ปฏิเสธไม่ได้เลยว่าเทคโนโลยีการสื่อสารและเครือข่ายไร้สายในปัจจุบันค่อนข้างก้าวหน้าไปไกล ซึ่งเทคโนโลยีดังกล่าวนี้ก็ได้ทำให้เกิดหุ่นยนต์ควบคุมแบบไร้สายขึ้น โดยที่หุ่นยนต์ควบคุมแบบไร้สายนั้นช่วยเพิ่มประสิทธิภาพในหลาย ๆ ด้าน ไม่ว่าจะเป็นด้านการผลิต หรือแม้แต่ด้านการดูแลสุขภาพ โดยหุ่นยนต์เหล่านี้ยังสามารถนำไปทดลองร่วมกับอัลกอริทึมใหม่ได้ง่าย และปรับแต่งระบบต่าง ๆ ได้โดยที่ไม่ต้องทำหุ่นยนต์ซ้ำ

ประโยชน์

1. หุ่นยนต์ที่พัฒนาขึ้นสามารถต้นแบบให้กับการประยุกต์ใช้งานในรูปแบบอื่น ๆ ในอนาคต

2. หุ่นยนต์ที่พัฒนาขึ้นสามารถลดความเสี่ยงที่อาจจะเกิดขึ้นจากพื้นที่ที่เป็นอันตรายได้

วิธีใช้งาน

บังคับทิศทางของหุ่นยนต์ผ่านรีโมทไร้สาย

คำนำ

เอกสารการออกแบบเล่มนี้จัดเป็นส่วนหนึ่งในรายวิชา 01076023 COMPUTER HARDWARE DESIGN การออกแบบฮาร์ดแวร์คอมพิวเตอร์ โดยจัดทำขึ้นเพื่ออธิบายถึงที่มาที่ไป หลักการและเหตุผล วัตถุประสงค์ ประโยชน์ หลักการใช้งาน หลักการออกแบบ และรายละเอียดต่าง ๆ ของ BotBot ที่ถูกออกแบบควบคู่ไปกับการประยุกต์ใช้ภาษา VHDL

หวังเป็นอย่างยิ่งว่าเอกสารการออกแบบเล่มนี้จะเป็นประโยชน์แก่ผู้สนใจศึกษาเกี่ยวก่อนการออกแบบฮาร์ดแวร์ฮาร์ดแวร์คอมพิวเตอร์ ร่วมกับภาษา VHDL และหากมีข้อผิดพลาดประการใดทางคณะผู้จัดทำก็ขออภัยมา ณ ที่นี่ด้วย

คณะผู้จัดทำ

30 เมษายน พ.ศ. 2566

สารบัญ

สารบัญ (ต่อ)

บทที่ 1
ที่มาและความสำคัญ

ที่มาและความสำคัญ

ปฏิเสธไม่ได้เลยว่าเทคโนโลยีการสื่อสารและเครือข่ายไร้สายในปัจจุบันค่อนข้างก้าวหน้าไปไกล ซึ่งเทคโนโลยีดังกล่าวนี้ก็ได้ทำให้เกิดหุ่นยนต์ควบคุมแบบไร้สายขึ้น โดยที่หุ่นยนต์ควบคุมแบบไร้สายนั้นมีข้อได้เปรียบมากมายและมีความสำคัญอย่างมากในด้านต่าง ๆ เนื่องจากสามารถทำงานจากระยะไกลผ่านเทคโนโลยีการสื่อสารไร้สาย เช่น Wi-Fi หรือคลื่นความถี่วิทยุ อีกทั้งการควบคุมหุ่นยนต์แบบไร้สายก็ยังช่วยเพิ่มความปลอดภัยและประสิทธิภาพอย่างมากในสภาพแวดล้อมที่เป็นอันตราย หุ่นยนต์เหล่านี้จะช่วยลดความเสี่ยงของการบาดเจ็บหรือเสียชีวิตได้ จากการลดความจำเป็นในการแสดงตัวของมนุษย์ในสถานการณ์อันตราย เช่น การจัดการสารพิษหรือการทำงานในพื้นที่ที่มีความเสี่ยง นอกจากนี้หุ่นยนต์ที่ควบคุมแบบไร้สายยังช่วยเพิ่มประสิทธิภาพในภาคอุตสาหกรรมต่าง ๆ ด้วย ไม่ว่าจะเป็นด้านการผลิต หรือแม้แต่ด้านการดูแลสุขภาพ โดยหุ่นยนต์เหล่านี้จะสามารถตั้งโปรแกรมให้ทำงานซ้ำๆ หรืองานที่ใช้แรงงานมากได้อย่างแม่นยำและสม่ำเสมอ ซึ่งนำไปสู่อัตราการผลิตที่เพิ่มขึ้นและต้นทุนที่ลดลง ด้วยการทำให้กระบวนการเป็นอัตโนมัติ อีกทั้งหุ่นยนต์ควบคุมแบบไร้สายยังส่งเสริมความก้าวหน้าในการวิจัยและพัฒนาอีกด้วย เนื่องจากสามารถนำไปใช้ซ้ำหรือต่อยอดใหม่ได้ นำไปทดลองร่วมกับอัลกอริทึมใหม่ได้ง่าย และปรับแต่งระบบต่าง ๆ ได้โดยที่ไม่ต้องทำหุ่นยนต์ซ้ำ

วัตถุประสงค์

1. เพื่อเป็นต้นแบบให้กับการประยุกต์ใช้งานในรูปแบบอื่น ๆ ในอนาคต
2. เพื่อลดความเสี่ยงที่อาจจะเกิดขึ้นจากพื้นที่ที่เป็นอันตราย

ขอบเขตขอบของการศึกษา

เป็นเพียงการศึกษาการควบคุมไร้สายด้วย FPGA ร่วมกับ NodeMCU ESP32 เท่านั้น

ประโยชน์

1. หุ่นยนต์ที่พัฒนาขึ้นสามารถต้นแบบให้กับการประยุกต์ใช้งานในรูปแบบอื่น ๆ ในอนาคต
2. หุ่นยนต์ที่พัฒนาขึ้นสามารถลดความเสี่ยงที่อาจจะเกิดขึ้นจากพื้นที่ที่เป็นอันตรายได้

บทที่ 2

หลักการใช้งาน

อุปกรณ์ที่ใช้

1. FPGA 1 บอร์ด

รูปที่ 1 FPGA

2. NodeMCU ESP32 2 บอร์ด

รูปที่ 2 NodeMCU ESP32

3. Motor และล้อ 2 ชุด

รูปที่ 3 motor และล้อ

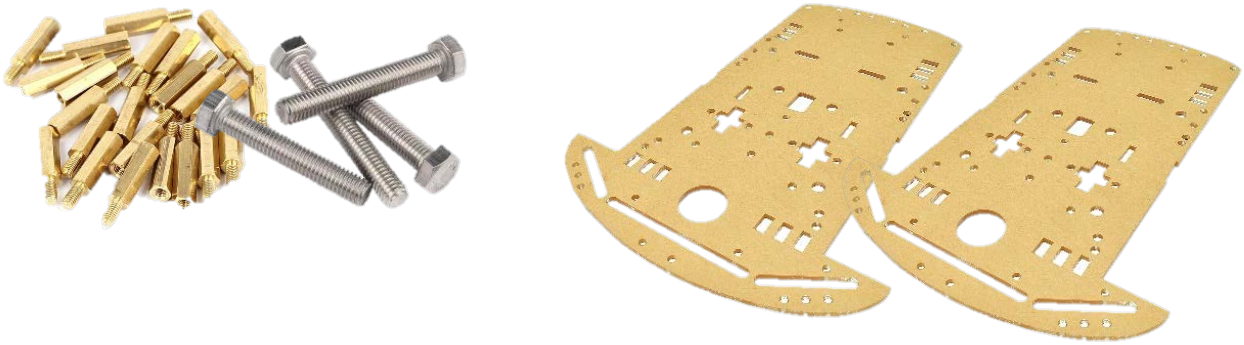
4. joystick shield

รูปที่ 4 joystick shield

5. แหล่งจ่ายพลังงาน

รูปที่ 5 แหล่งจ่ายพลังงาน

6. โครงสร้างอื่น ๆ เช่น อะคริลิค

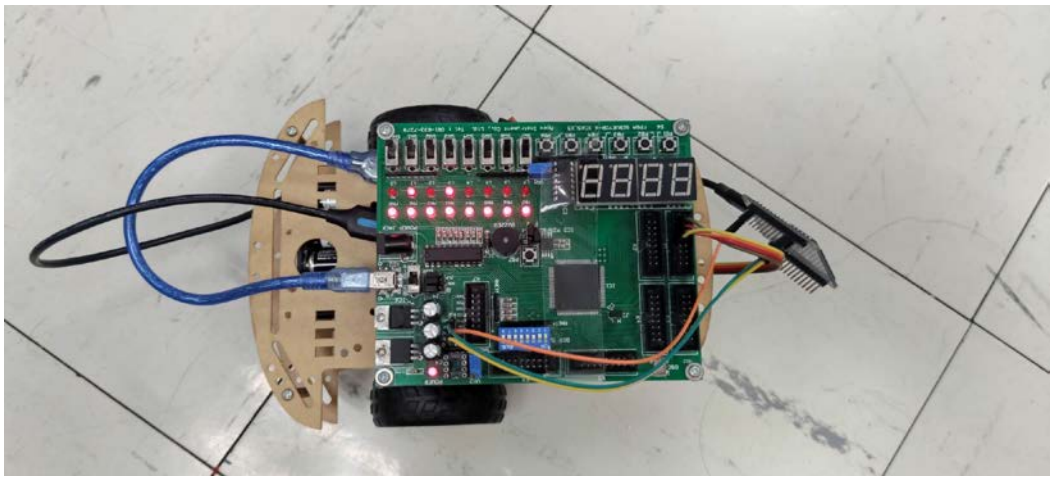


รูปที่ 6 โครงสร้าง

รายละเอียดและความสามารถของอุปกรณ์แต่ละชิ้น

1. FPGA

เป็นวงจรรวมที่มีฟังก์ชันลอจิกที่ตั้งโปรแกรมได้ ช่วยให้นักออกแบบสร้างวงจรและระบบดิจิทัลแบบกำหนดเองได้ FPGA เป็นที่รู้จักในด้านความสามารถในการตั้งโปรแกรมซ้ำ ซึ่งช่วยให้สามารถแก้ไขการกำหนดค่าลอจิกต่างภายในได้โดยไม่จำเป็นต้องเปลี่ยนฮาร์ดแวร์ มีโครงสร้างหลัก เป็น Configurable Logic Blocks (CLBs) ซึ่งมี look-up table (LUT) และ flip-flop CLB ซึ่งสามารถกำหนดค่าเพื่อใช้ฟังก์ชันลอจิกต่างๆ และสามารถตั้งโปรแกรมการเชื่อมต่อโครงข่ายระหว่าง CLB เพื่อสร้างวงจรดิจิทัลที่ซับซ้อนได้ FPGA ช่วยให้สามารถประมวลผลประสิทธิภาพสูงและประมวลผลตามเวลาจริง ทำให้เหมาะสำหรับการใช้งานที่ต้องการพลังในการคำนวณสูง FPGA สามารถตั้งโปรแกรมโดยใช้ภาษาคำอธิบายฮาร์ดแวร์ (HDL) เช่น VHDL หรือ Verilog ช่วยให้นักออกแบบสามารถอธิบายลักษณะการทำงานที่ต้องการของวงจรในระดับนามธรรมที่สูงขึ้น ระดับนามธรรมนี้ช่วยเพิ่มประสิทธิภาพของกระบวนการออกแบบ นอกจากนี้ยังสามารถใช้ FPGA เพื่อเร่งความเร็วงานหรืออัลกอริทึมเฉพาะได้โดยใช้ตัวเร่งฮาร์ดแวร์ ซึ่งให้การปรับปรุงประสิทธิภาพเมื่อเทียบกับการใช้งานบนซอฟต์แวร์ FPGA มีเวลาแฝงต่ำและปริมาณงานสูงเนื่องจากความสามารถในการประมวลผลแบบขนานและการออกแบบฮาร์ดแวร์ที่ปรับให้เหมาะสมกับการประมวลผลข้อมูลจำนวนมากแบบเรียลไทม์ได้อย่างมีประสิทธิภาพ ทำให้เหมาะสำหรับการใช้งานต่าง ๆ เช่น การสตรีมข้อมูล และการประมวลผลสัญญาณตามเวลาจริง อีกทั้งยังสามารถใช้เครื่องมือตรวจแก้จุดบกพร่องในระบบเพื่อตรวจสอบและวิเคราะห์พฤติกรรมของวงจรแบบเรียลไทม์ เพื่อช่วยอำนวยความสะดวกในกระบวนการตรวจแก้จุดบกพร่องและการตรวจสอบระหว่างการพัฒนาได้อีก

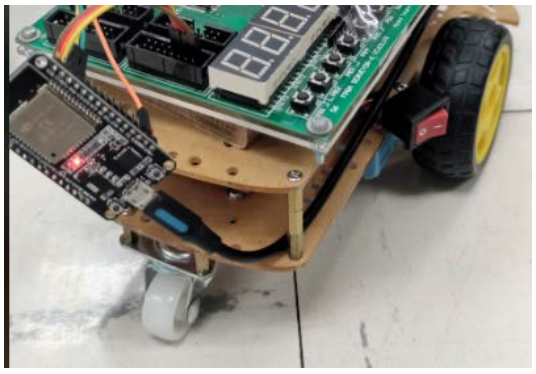


รูปที่ 7 FPGA on BotBot

2. NodeMCU ESP32

NodeMCU ESP32 เป็นบอร์ดเอนกประสงค์นี้มีความสามารถที่หลากหลาย เช่น การรองรับการเชื่อมต่อ Wi-Fi และ Bluetooth ช่วยให้ผสานรวมกับเครือข่ายไร้สายและการสื่อสารกับอุปกรณ์อื่น ๆ ได้อย่างราบรื่น นอกจากนี้ยังมีตัวเลือกอินพุตและเอาต์พุตที่หลากหลาย รองรับภาษาโปรแกรมและเฟรมเวิร์กที่หลากหลาย สามารถเขียนโปรแกรมได้ผ่าน Arduino IDE, MicroPython หรือภาษาสคริปต์ NodeMCU Lua ยิ่งไปกว่านั้น NodeMCU ESP32 ยังมีคุณสมบัติในตัวมากมาย เช่น หน่วยความจำแฟลชออนบอร์ด พอร์ต micro USB สำหรับจ่ายไฟและตั้งโปรแกรม มีขนาดกะทัดรัดและใช้งานง่าย หรือแม้แต่ตัวควบคุมแรงดันไฟฟ้าเพื่อการทำงานที่เสถียร โดยสรุปแล้ว NodeMCU ESP32 มีความหลากหลายที่ดีทั้งในด้านตัวอุปกรณ์ที่มีความสามารถที่ค่อนข้างครอบคลุม หรือแม้แต่การรองรับหลายภาษา หลายเฟรมเวิร์ม

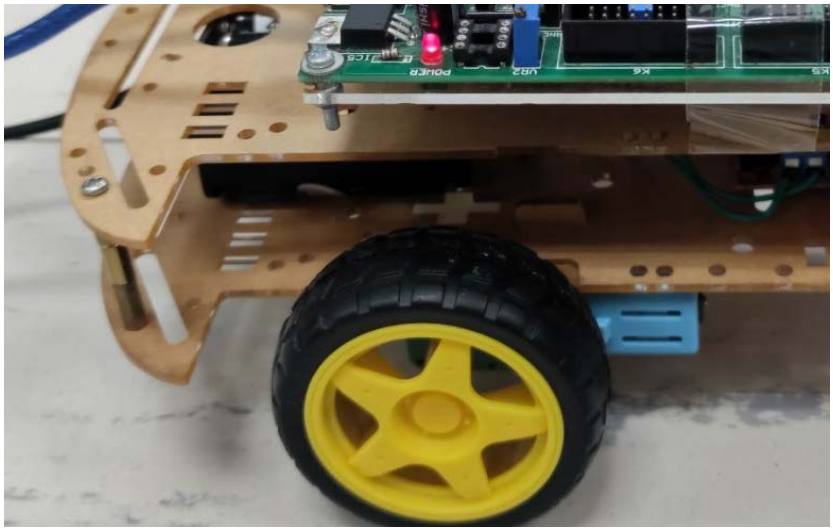
รูปที่ 8 ESP32 on joystick



รูปที่ 9 ESP32 on BotBot

3. Motor และล้อ

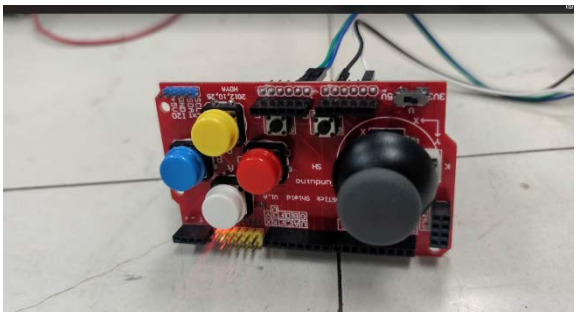
มอเตอร์เป็นอุปกรณ์เครื่องกลไฟฟ้าที่เปลี่ยนพลังงานไฟฟ้าเป็นพลังงานกล ทำให้สามารถทำงานได้หลากหลายขึ้นอยู่กับประเภทการใช้งานและการออกแบบ ความสามารถหลักประการหนึ่งของมอเตอร์คือความสามารถในการสร้างการเคลื่อนที่แบบหมุน ด้วยการใช้กระแสไฟฟ้ากับขดลวดของมอเตอร์สนามแม่เหล็กจะถูกสร้างขึ้น ทำให้เกิดแรงบิดที่ทำให้เพลาของมอเตอร์หมุน ซึ่งในที่นี้เมื่อนำล้อเข้ามาต่อกับมอเตอร์ก็จะทำหน้าที่ไม่ต่างอะไรกับล้อรถยนต์ที่สามารถเคลื่อนที่ได้ด้วยการจ่ายไฟฟ้าเข้า



รูปที่ 10 Motor และล้อ on BotBot

4. joystick shield

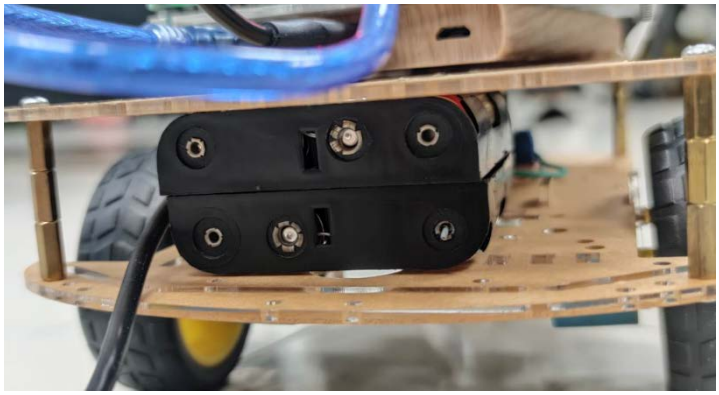
แผงจอยสติ๊กเป็นโมดูลเสริมที่ช่วยลดขีดจำกัดของไมโครคอนโทรลเลอร์ให้สามารถควบคุมและปรับเปลี่ยนตำแหน่งตามแกนต่าง ๆ ได้ง่ายขึ้น เช่น ขึ้น/ลง และซ้าย/ เหมาะสำหรับการทำหุ่นยนต์ เกม จอยสติ๊กกับไมโครคอนโทรลเลอร์ด้วย เพราะมักจะใช้ SPI หรือ I2C เพื่อทำการสื่อสาร ซึ่งสิ่งนี้ช่วยให้สามารถควบคุมจอยสติ๊กได้ง่ายโดยที่ไม่ต้องใช้การเดินสายไฟให้ซับซ้อน โดยสรุปแล้วแผงจอยสติ๊กจะมีความสำคัญหรือจำเป็นก็ต่อเมื่อมีความต้องการที่จะเพิ่มการควบคุมให้กับอุปกรณ์อื่น ๆ



รูปที่ 11 joystick in BotBot

5. แหล่งจ่ายพลังงาน

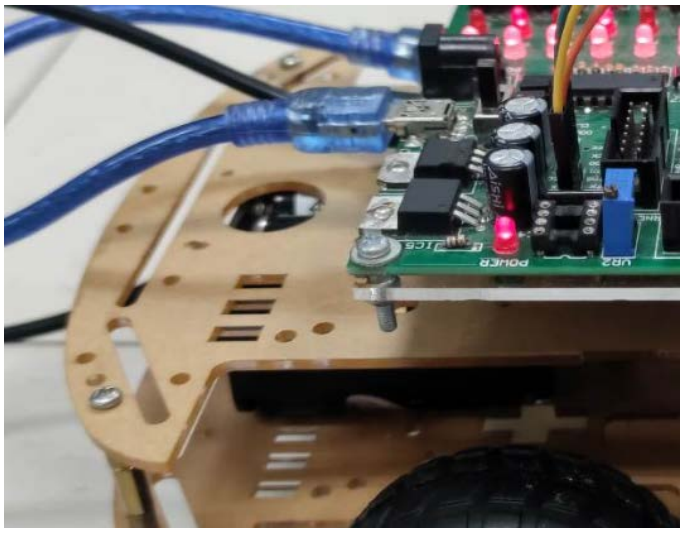
เป็นแหล่งจ่ายพลังงานเพื่อให้ชิ้นงานสารมารถทำงานได้โดยไม่ต้องต้องไฟจากอาคาร



รูปที่ 12 แหล่งจ่ายพลังงานของ BotBot

6. โครงสร้างอื่น ๆ เช่น อะคริลิค

ใช้สำหรับเป็นประกอบทำเป็นชิ้นงาน



รูปที่ 13 โครงสร้างของ BotBot

ภาพรวมของชิ้นงาน

ภาพรวมของชิ้นงานก็คือเป็นหุ่นยนต์บังคับไร้สายสามารถควบคุมให้เดินหน้า ถอยหลัง เลี้ยวซ้าย หรือเลี้ยวขวาได้อิสระตามที่ต้องการ

หลักการใช้งาน

หลักการใช้งานของชิ้นงานคือการบังคับตัวหุ่นยนต์จากห้องควบคุมโดยจะดูผ่านกล้องวงจรปิด หรืออาจจะเป็นการควบคุมจากระยะไกลที่สายตายังมองเห็นเพื่อสำรวจพื้นที่ที่อาจจะเป็นอันตรายหรือมีความเสี่ยง อีกทั้งยังสามารถนำไปประยุกต์ใช้งานต่อได้อยากหลากหลายและต้องเสียเวลาจัดทำตัวหุ่นยนต์ขึ้นใหม่

หลักการทำงาน

หลักการทำงานคือจะเป็นการควบคุมการหมุนของล้องผ่านการควบคุมจาก joystick shield โดยตัว FPGA ก็จะประมวลผลว่าจอยคันโยกใดถูกกด ถ้าหากจอยคันโยกที่ใช้สำหรับเดินหน้าถูกกดมอเตอร์ก็จะหมุนไปด้านหน้า ถ้าหากจอยคันโยกที่ใช้สำหรับเดินถอยหลังถูกกดมอเตอร์ก็จะหมุนกลับมาอีกด้าน หรือถ้าจอยคันโยกที่ใช้สำหรับควบคุมการเลี้ยวซ้ายมอเตอร์ด้านขวาก็จะหมุนเดินหน้าและมอเตอร์ซ้ายก็จะหมุนถอยหลัง ในทางกลับกันถ้ากดจอยคันโยกที่ใช้สำหรับควบคุมการเลี้ยวขวามอเตอร์ด้านซ้ายก็จะหมุนเดินหน้าแทนและมอเตอร์ขวาก็จะหมุนกลับด้าน

บทที่ 3

หลักการออกแบบ

หลักการออกแบบ

หลักการออกแบบของ BotBot ไม่มีอะไรมาก โดยเริ่มแรกก่อนที่จะออกแบบได้ก็ต้องมาแจกแจงออกมาก่อนว่าต้องการให้ BotBot มีฟังก์ชันอะไรบ้าง อยากให้ทำอะไรบ้าง ทำได้แค่ไหน มีเงื่อนไขอะไรบ้าง อยากจะใช้วิธีการสื่อสารแบบไหน ถ้าใช้แบบนี้ควรจะเลือกใช้อะไร เช่น หากต้องการให้ตัว BotBot มีการติดต่อสื่อสารกันด้วย WiFi เพื่อให้สามารถนำไปประยุกต์ใช้กับการทำ IoT ได้ง่ายขึ้นการใช้การสื่อสารด้วย nRF ก็อาจจะไม่เหมาะสม เป็นต้น

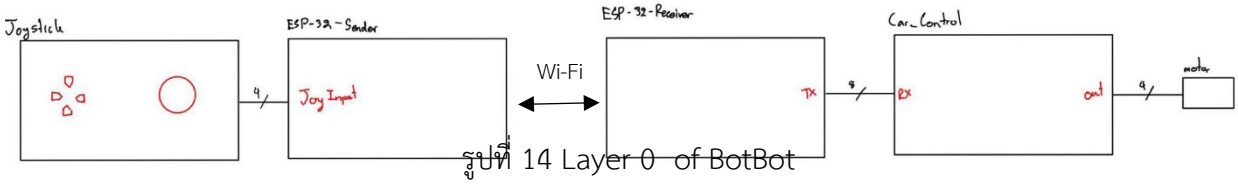
ภาพรวมการออกแบบเบื้องต้น

BotBot เป็นหุ่นยนต์ที่ขับเคลื่อน 2 ล้อ จากการควบคุมด้วย joystick โครงสร้างทำมาจากอะคริลิค ตัวประมวลผลประกอบด้วย NodeMCU ESP32 2 บอร์ด โดยที่บอร์ดแรกจะเอาไว้ใช้สำหรับรับค่าจาก joystick แล้วส่งค่าไปให้บอร์ดตัวที่สองด้วยการสื่อสารแบบ WiFi หลังจากนั้นบอร์ด ESP32 ตัวที่สองก็จะทำการแมพค่าที่ได้จาก ESP32 ตัวแรกไปให้ FPGA ผ่านการสื่อสารด้วย UART จากนั้นตัว FPGA ก็จะทำการประมวลผลและควบคุมว่ามอเตอร์ไหนที่ควรจะทำงาน

รายละเอียดของการออกแบบ

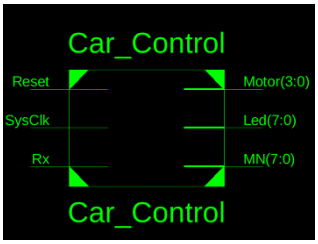
Top-Down design

Layer 0



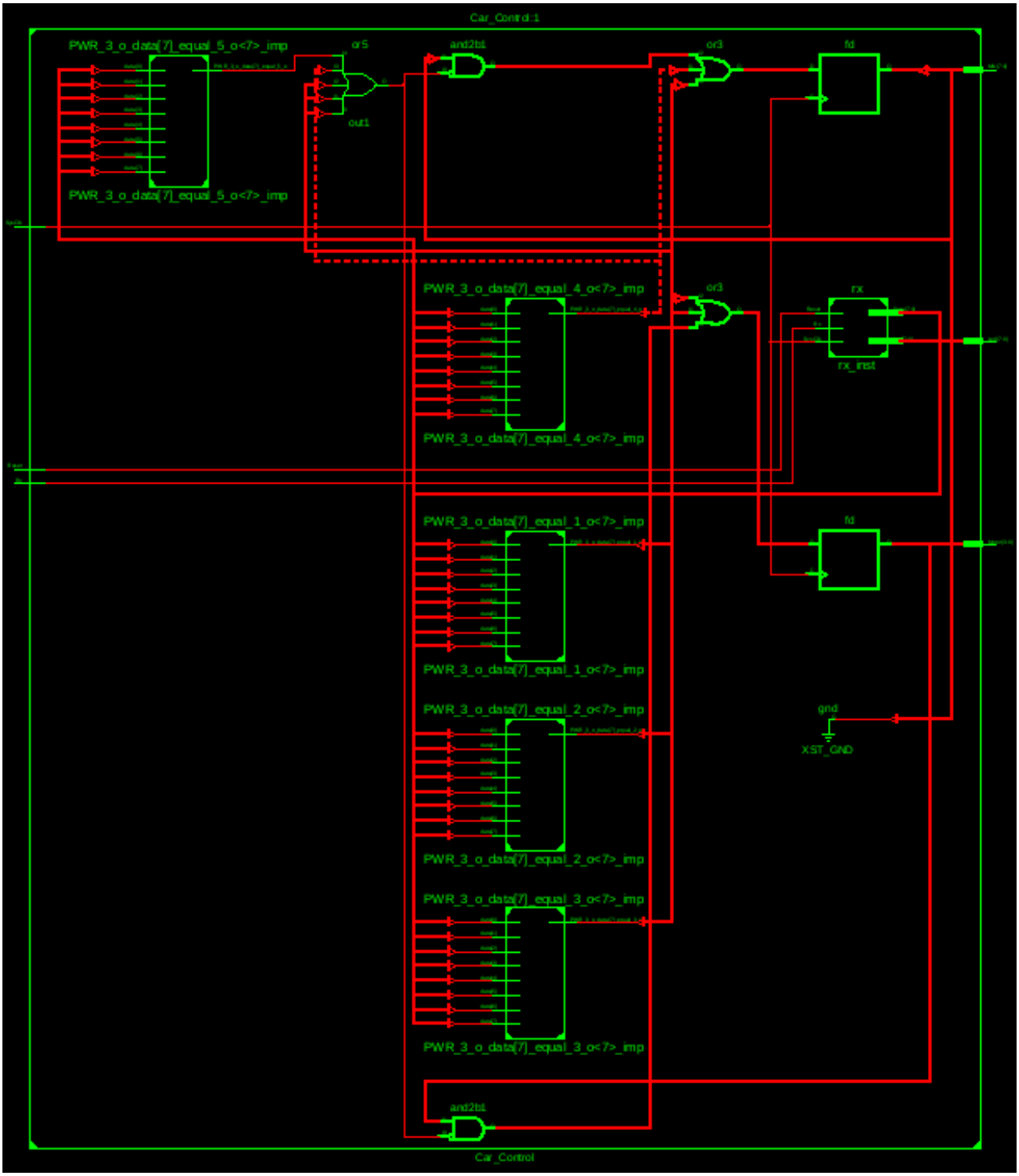
รูปที่ 14 Layer 0 of BotBot

Layer 1 Car_Control



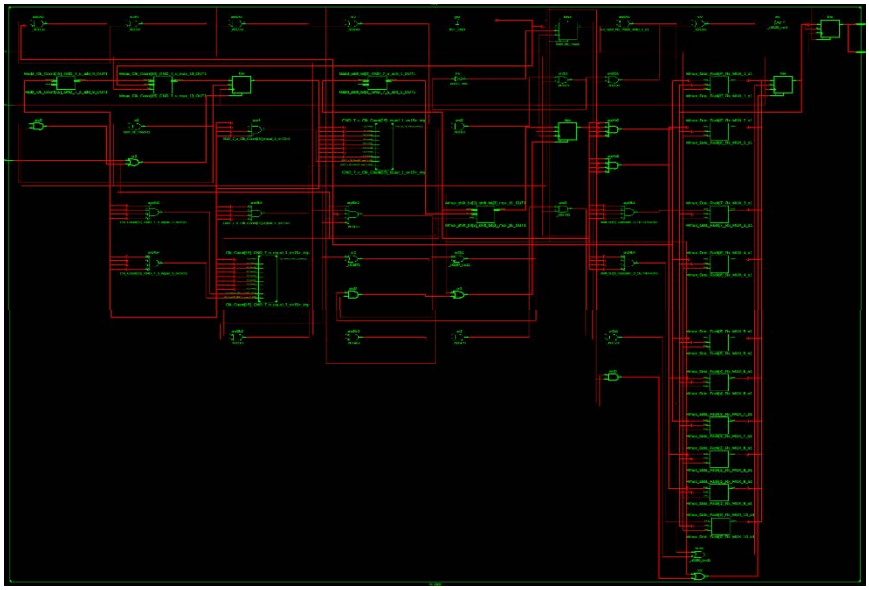
รูปที่ 15 Layer 1 Car_Control of BotBot

Layer 2 Car_control



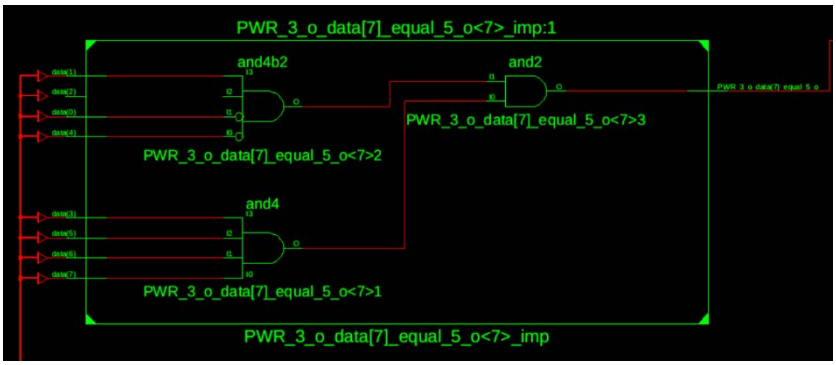
รูปที่ 16 Layer 2 Car_Control of BotBot

Layer 3 rx_inst



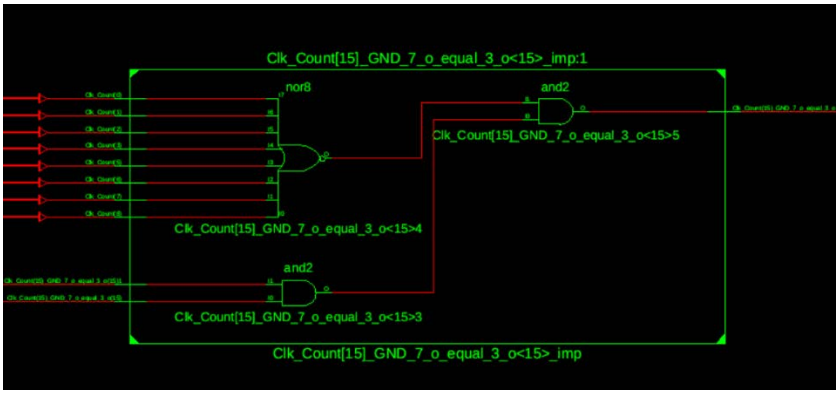
รูปที่ 17 Layer 3 rx_inst of BotBot

Layer 3 PWR_3_o_data[7]_equal_5_o<7>_imp:1



รูปที่ 18 Layer 3 PWR_3_o_data[7]_equal_5_o<7>_imp:1of BotBot

Layer 4 Clk_Count[15]_GND7_equal_3_o<15>_imp:1



รูปที่ 19 Layer 4 Clk_Count[15]_GND7_equal_3_o<15>_imp:1 of BotBot

เทคนิค และกระบวนการออกแบบ

การติดต่อสื่อสารระหว่าง FPGA และ NodeMCU ESP32 ด้วย UART

การเชื่อมต่อ UART ในไมโครคอนโทรลเลอร์ ESP32 กับ FPGA นั้นเกี่ยวข้องกับการสร้างการเชื่อมต่อทางกายภาพและการตรวจสอบการกำหนดค่าที่เหมาะสมสำหรับการสื่อสารระหว่างอุปกรณ์ทั้งสอง ขั้นแรก ระบุพิน UART ทั้งบน ESP32 และ FPGA ซึ่งโดยปกติจะระบุว่าเป็น TX (ส่ง) และ RX (รับ) ในทำนองเดียวกัน FPGA ควรมีพิน UART ที่สอดคล้องกันสำหรับการส่งและรับข้อมูลเช่นกัน จากนั้นให้เชื่อมต่อพิน TX ของ ESP32 เข้ากับพิน RX ของ FPGA สุดท้ายเพื่อให้แน่ใจว่ามีการซิงโครไนซ์และส่งข้อมูลที่ถูกต้อง จำเป็นต้องเขียนโปรแกรมเฟิร์มแวร์ ESP32 เพื่อจัดการกับการสื่อสาร UART และทดสอบการเชื่อมต่อ UART ว่าสามารถทำงานได้หรือไม่

ติดต่อสื่อสารระหว่าง NodeMCU ESP32 ด้วย WiFi

การเชื่อมต่อไมโครคอนโทรลเลอร์ ESP32 สองตัวผ่าน Wi-Fi ช่วยให้สามารถสื่อสารไร้สายระหว่างอุปกรณ์ได้ ช่วยอำนวยความสะดวกในการแลกเปลี่ยนข้อมูลและฟังก์ชันการทำงานร่วมกัน โดยในการสร้างการเชื่อมต่อ Wi-Fi จำเป็นต้องตั้งค่าอุปกรณ์ ESP32 หนึ่งเครื่องเป็นจุดเชื่อมต่อ (AP) ในขณะที่อุปกรณ์ ESP32 อีกเครื่องหนึ่งทำหน้าที่เป็นสถานีที่เชื่อมต่อกับจุดเชื่อมต่อ จุดเชื่อมต่อ ESP32 สร้างเครือข่าย Wi-Fi ด้วย SSID (ชื่อเครือข่าย) และรหัสผ่านเฉพาะ สถานี ESP32 จะสแกนหาเครือข่ายที่มีอยู่และเชื่อมต่อกับจุดเชื่อมต่อโดยใช้ข้อมูลประจำตัวที่ให้มา เมื่ออุปกรณ์ ESP32 ทั้งสองเชื่อมต่อกับเครือข่าย Wi-Fi เดียวกัน ก็จะเกิดการเชื่อมต่อ TCP/IP ระหว่างอุปกรณ์ทั้งสองขึ้น โดยที่ ESP32 ตัวหนึ่งทำหน้าที่เป็นเซิร์ฟเวอร์ ในขณะที่อีกตัวหนึ่งทำหน้าที่เป็นไคลเอนต์ และเมื่อสร้างการเชื่อมต่อ TCP/IP แล้ว อุปกรณ์ ESP32 จะสามารถแลกเปลี่ยนข้อมูลและสื่อสารกันได้

Code และคำอธิบาย

Code ส่วน nodeMCU ESP32

Code ส่วนของตัวหุ่นยนต์ (สามารถดูเพิ่มเติมได้จาก google drive ที่แนบ)

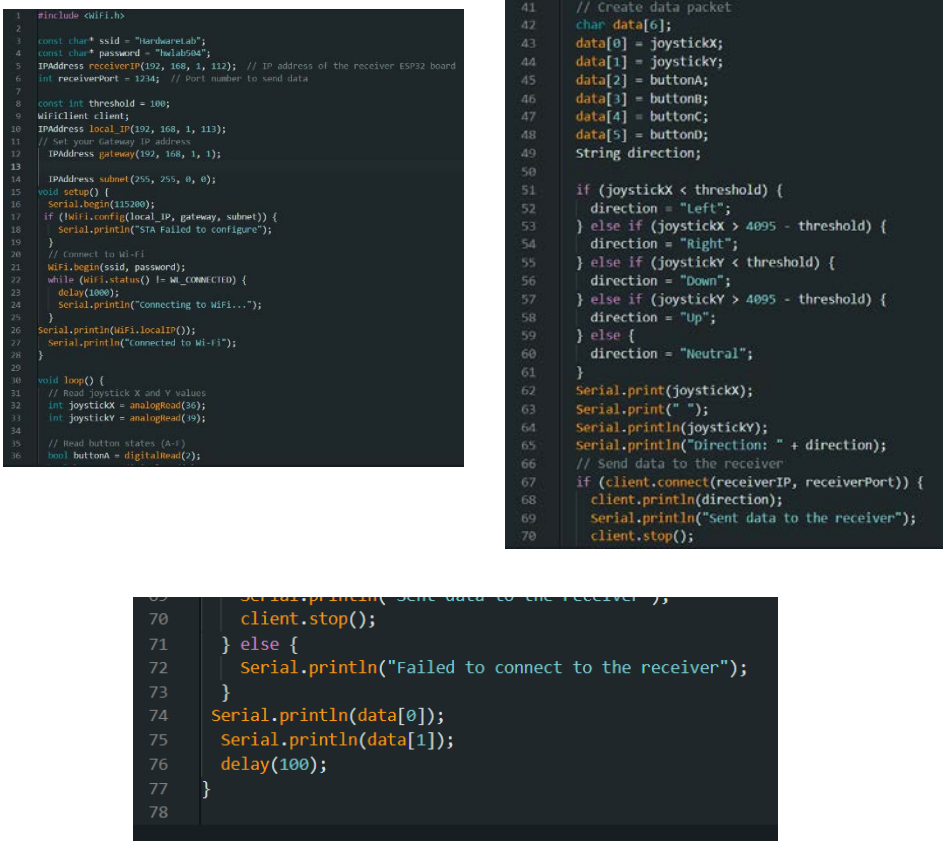
```
#include <WiFi.h>
#include <WiFiServer.h>

const char* ssid = "HardwareLab";
const char* password = "hwlab904";
int receiverPort = 1234;  // Port number to receive data

WiFiServer server(receiverPort);
WiFiClient client;

  IPAddress local_IP(192, 168, 1, 112);
// Set your Gateway IP address
  IPAddress gateway(192, 168, 1, 1);

  IPAddress subnet(255, 255, 0, 0);
void setup() {
  Serial.begin(2400);
  pinMode(2,OUTPUT);
  digitalWrite(2,LOW);
  if (!WiFi.config(local_IP, gateway, subnet)) {
    Serial.println("STA Failed to configure");
  }
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }

  Serial.println("Connected to Wi-Fi");

  Serial.println(WiFi.localIP());
  // Start the server
  server.begin();
  Serial.println("Server started");

}

void loop() {
  char data[6];
  byte sendData = 0x00;
  // Check if a client has connected
  client = server.available();
  if (client) {
    while (client.connected()) {
      // Read data from the client
      if (client.available()) {
        String receivedData = client.readStringUntil('\n');
        Serial.println("Received data from the sender:");
        digitalWrite(2,HIGH);
          if  (receivedData.indexOf("Up") != -1) {
        sendData = 0xAA;
        Serial.println(receivedData);
      }
      else if (receivedData.indexOf("Left") != -1){
        sendData = 0xBB;
        Serial.println(receivedData);
      }
      else if (receivedData.indexOf("Right") != -1){
        sendData = 0xCC;
        Serial.println(receivedData);
      }
      else if (receivedData.indexOf("Down") != -1){
        sendData = 0xDD;
        Serial.println(receivedData);
      }
      else if (receivedData.indexOf("Neutral") != -1){
        sendData = 0xEE;
        Serial.println(receivedData);
      }
      Serial.println(receivedData);
      }
      Serial.println(receivedData);
      Serial.write(sendData);
      Serial.println(sendData,BIN);
      Serial.println(sendData);
        }
      }
    client.stop();
    digitalWrite(2,LOW);
  }
}
```

รูปที่ 20 Code ส่วนของตัวหุ่นยนต์

คำอธิบาย

เป็น code ในส่วนเชื่อมต่อ WiFi และ Host ตัวเองเป็น Server ในวง Subnet นั้น คอยรับข้อมูลทิศทางจากตัวส่งสัญญาณ แล้วจึงส่ง Flag ผ่านทางช่อง Serial เข้าบอร์ด FPGA

Code ส่วนของรีโมท (สามารถดูเพิ่มเติมได้จาก google drive ที่แนบ)



```
#include <WiFi.h>

const char* ssid = "HardwareLab";
const char* password = "hwlab504";
IPAddress receiverIP(192, 168, 1, 112);  // IP address of the receiver ESP32 board
int receiverPort = 1234;  // Port number to send data

const int threshold = 100;
WiFiClient client;
IPAddress local_IP(192, 168, 1, 113);
// Set your Gateway IP address
  IPAddress gateway(192, 168, 1, 1);

  IPAddress subnet(255, 255, 0, 0);
void setup() {
  Serial.begin(115200);
 if (!WiFi.config(local_IP, gateway, subnet)) {
    Serial.println("STA Failed to configure");
  }
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
Serial.println(WiFi.localIP());
  Serial.println("Connected to Wi-Fi");
}

void loop() {
  // Read joystick X and Y values
  int joystickX = analogRead(36);
  int joystickY = analogRead(39);

  // Read button states (A-F)
  bool buttonA = digitalRead(2);
  bool buttonB = digitalRead(3);
  bool buttonC = digitalRead(4);
  bool buttonD = digitalRead(5);

  // Create data packet
  char data[6];
  data[0] = joystickX;
  data[1] = joystickY;
  data[2] = buttonA;
  data[3] = buttonB;
  data[4] = buttonC;
  data[5] = buttonD;
  String direction;

  if (joystickX < threshold) {
    direction = "Left";
  } else if (joystickX > 4095 - threshold) {
    direction = "Right";
  } else if (joystickY < threshold) {
    direction = "Down";
  } else if (joystickY > 4095 - threshold) {
    direction = "Up";
  } else {
    direction = "Neutral";
  }
  Serial.print(joystickX);
  Serial.print(" ");
  Serial.println(joystickY);
  Serial.println("Direction: " + direction);
  // Send data to the receiver
  if (client.connect(receiverIP, receiverPort)) {
    client.println(direction);
    Serial.println("Sent data to the receiver");
    client.stop();
  } else {
    Serial.println("Failed to connect to the receiver");
  }
 Serial.println(data[0]);
  Serial.println(data[1]);
  delay(100);
}
```

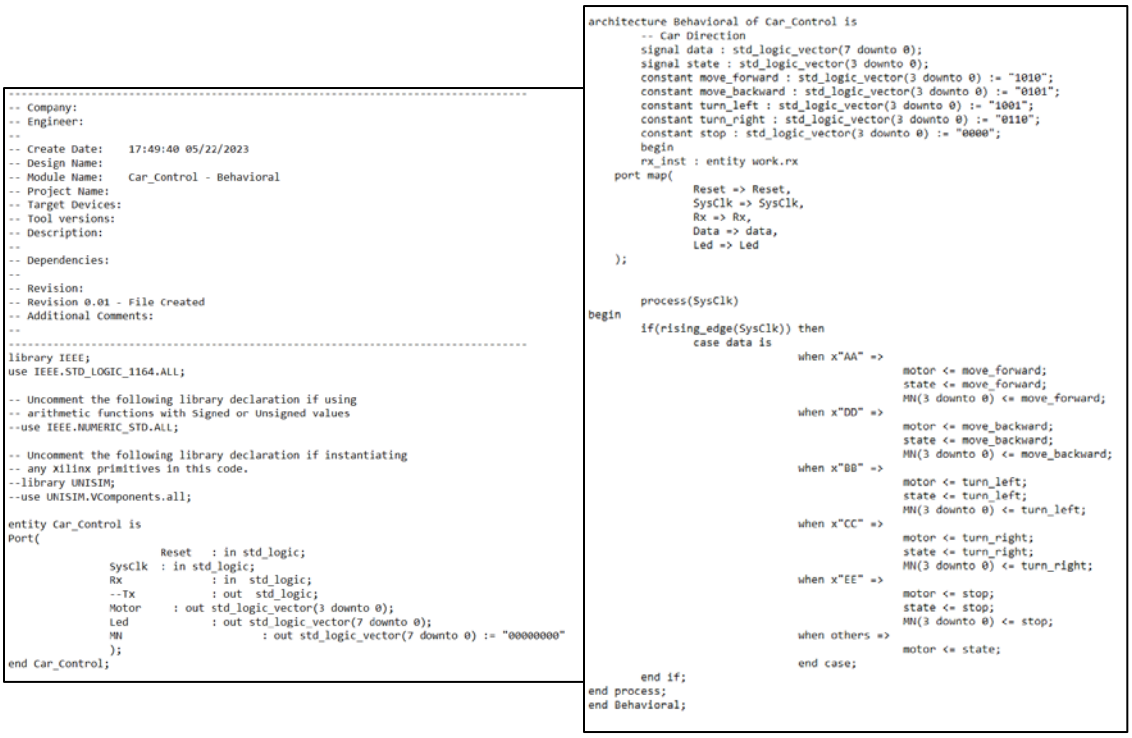
รูปที่ 21 Code ส่วนของรีโมท

คำอธิบาย

เป็น code ในส่วนเชื่อมต่อ WiFi แล้วเชื่อมต่อ NodeMCU ESP32 ที่เป็นส่วนของตัวรับข้อมูลที่อยู่บนรถ โดยที่ตัว NodeMCU ESP32 ที่ทำหน้าที่เป็น Remote หรือตัวส่งสัญญาณจะคอยรับสัญญาณ Analog จาก Joystick และแปลงเป็นทิศทางส่งไปยัง ตัวรับสัญญาณ

Code ส่วน FPGA

Code ส่วนการควบคุมมอเตอร์ (สามารถดูเพิ่มเติมได้จาก google drive ที่แนบ)



```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    17:49:40 05/22/2023
-- Design Name:
-- Module Name:    Car_Control - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Car_Control is
Port(                   Reset   : in std_logic;
                SysClk  : in std_logic;
                Rx              : in  std_logic;
                --Tx            : out  std_logic;
                Motor     : out std_logic_vector(3 downto 0);
                Led             : out std_logic_vector(7 downto 0);
                MN                      : out std_logic_vector(7 downto 0) := "00000000"
                );
end Car_Control;
```

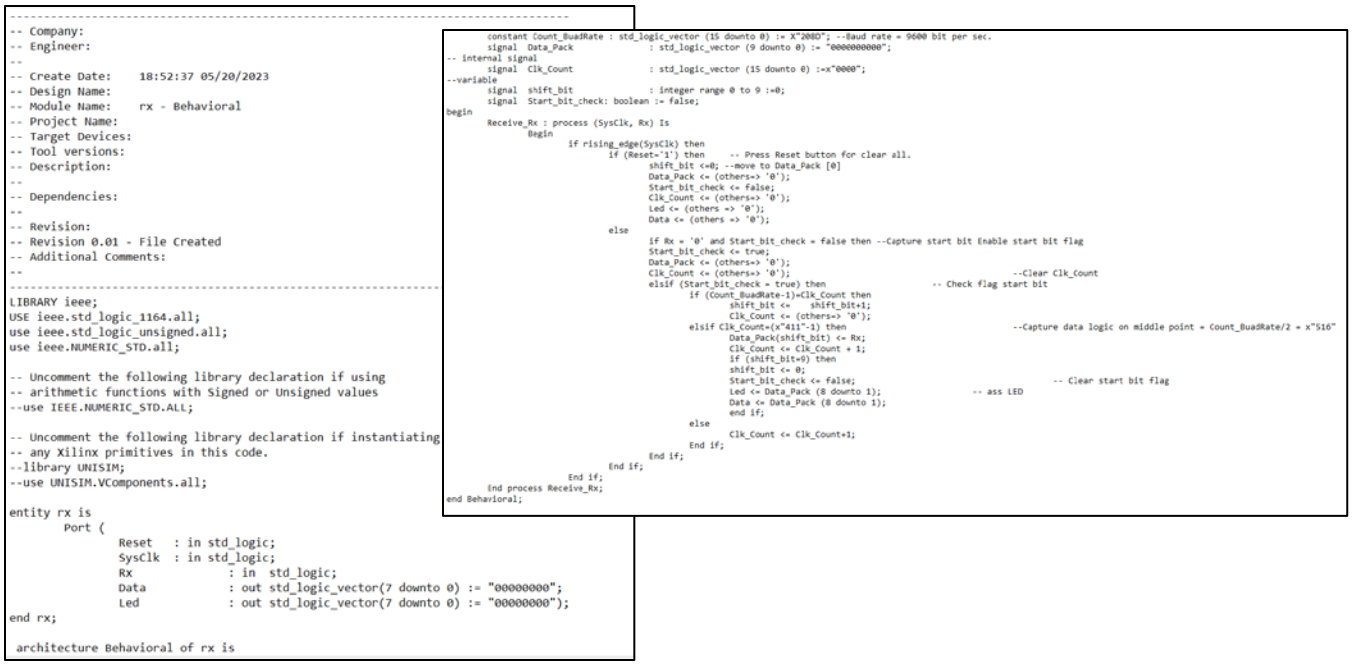
```
architecture Behavioral of Car_Control is
        -- Car Direction
        signal data : std_logic_vector(7 downto 0);
        signal state : std_logic_vector(3 downto 0);
        constant move_forward : std_logic_vector(3 downto 0) := "1010";
        constant move_backward : std_logic_vector(3 downto 0) := "0101";
        constant turn_left : std_logic_vector(3 downto 0) := "1001";
        constant turn_right : std_logic_vector(3 downto 0) := "0110";
        constant stop : std_logic_vector(3 downto 0) := "0000";
        begin
        rx_inst : entity work.rx
    port map(
                Reset => Reset,
                SysClk => SysClk,
                Rx => Rx,
                Data => data,
                Led => Led
    );


        process(SysClk)
begin
        if(rising_edge(SysClk)) then
                case data is
                                        when x"AA" =>
                                                        motor <= move_forward;
                                                        state <= move_forward;
                                                        MN(3 downto 0) <= move_forward;
                                        when x"DD" =>
                                                        motor <= move_backward;
                                                        state <= move_backward;
                                                        MN(3 downto 0) <= move_backward;
                                        when x"BB" =>
                                                        motor <= turn_left;
                                                        state <= turn_left;
                                                        MN(3 downto 0) <= turn_left;
                                        when x"CC" =>
                                                        motor <= turn_right;
                                                        state <= turn_right;
                                                        MN(3 downto 0) <= turn_right;
                                        when x"EE" =>
                                                        motor <= stop;
                                                        state <= stop;
                                                        MN(3 downto 0) <= stop;
                                        when others =>
                                                        motor <= state;
                                        end case;
        end if;
end process;
end Behavioral;
```

รูปที่ 22 Code ส่วนการควบคุมมอเตอร์

คำอธิบาย

เป็น code ในส่วนการรับข้อมูลมาจาก RX แล้วนำ flag มาเช็คว่าที่ส่งเข้ามาเป็น Flag ประเภทไหน แล้วจึงส่งสัญญาณ Digital ไปที่ Motor driver โดยที่ Flag 0xAA คือเคลื่อนที่ไปข้างหน้า, Flag 0xBB คือเลี้ยวซ้าย, Flag 0xCC คือเลี้ยวขวา, Flag 0xDD คือถอยหลัง และ Flag 0xEE คือหยุดนิ่ง ซึ่งสัญญาณ Digital ที่ส่งไปยัง Motor Driver จะส่งไปยัง Logic Monitor เพื่อตรวจสอบสัญญาณที่ถูกต้องอีกด้วย

Code ส่วนการสื่อสารแบบ UART (สามารถดูเพิ่มเติมได้จาก google drive ที่แนบ)



```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    18:52:37 05/20/2023
-- Design Name:
-- Module Name:    rx - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.NUMERIC_STD.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity rx is
        Port (
                Reset   : in std_logic;
                SysClk  : in std_logic;
                Rx              : in  std_logic;
                Data            : out std_logic_vector(7 downto 0) := "00000000";
                Led             : out std_logic_vector(7 downto 0) := "00000000");
end rx;

 architecture Behavioral of rx is
        constant Count_BuadRate : std_logic_vector (15 downto 0) := X"208D"; --Baud rate = 9600 bit per sec.
        signal  Data_Pack               : std_logic_vector (9 downto 0) := "0000000000";
-- internal signal
        signal  Clk_Count               : std_logic_vector (15 downto 0) :=x"0000";
--variable
        signal  shift_bit               : integer range 0 to 9 :=0;
        signal  Start_bit_check: boolean := false;
begin
        Receive_Rx : process (SysClk, Rx) Is
                Begin
                        if rising_edge(SysClk) then
                                if (Reset='1') then     -- Press Reset button for clear all.
                                        shift_bit <=0; --move to Data_Pack [0]
                                        Data_Pack <= (others=> '0');
                                        Start_bit_check <= false;
                                        Clk_Count <= (others=> '0');
                                        Led <= (others => '0');
                                        Data <= (others => '0');
                                else
                                        if Rx = '0' and Start_bit_check = false then --Capture start bit Enable start bit flag
                                        Start_bit_check <= true;
                                        Data_Pack <= (others=> '0');
                                        Clk_Count <= (others=> '0');                                            --Clear Clk_Count
                                        elsif (Start_bit_check = true) then                     -- Check flag start bit
                                                if (Count_BuadRate-1)=Clk_Count then
                                                        shift_bit <=    shift_bit+1;
                                                        Clk_Count <= (others=> '0');
                                                elsif Clk_Count=(x"411"-1) then                                 --Capture data logic on middle point = Count_BuadRate/2 = x"516"
                                                        Data_Pack(shift_bit) <= Rx;
                                                        Clk_Count <= Clk_Count + 1;
                                                        if (shift_bit=9) then
                                                        shift_bit <= 0;
                                                        Start_bit_check <= false;                                               -- Clear start bit flag
                                                        Led <= Data_Pack (8 downto 1);                  -- ass LED
                                                        Data <= Data_Pack (8 downto 1);
                                                        end if;
                                                else
                                                        Clk_Count <= Clk_Count+1;
                                                End if;
                                        End if;
                                End if;
                        End if;
        End process Receive_Rx;
end Behavioral;
```

รูปที่ 23 Code ส่วนการสื่อสารแบบ UART

คำอธิบาย

เป็น code ในส่วนการรับข้อมูลจากช่อง rx บนบอร์ด โดยจะเช็ค Start และ Stop bit ของข้อมูลที่ส่งเข้ามา ตามความเร็วของการส่ง Serial ของ NodeMCU ESP32 จากนั้นจึงนำข้อมูลเข้าสู่ ส่วน Car_control และแสดงผลบน Led เพื่อไว้ตรวจสอบข้อมูลที่ถูกส่งเข้ามา

บทที่ 4

ผลการทดสอบ

วิธีการทดสอบ

เนื่องจากชิ้นงานคือหุ่นยนต์ที่สามารถควบคุมแบบไร้สายได้ ทางคณะผู้จัดทำจึงมีแนวคิดที่จะทดสอบประสิทธิภาพและความสามารถของชิ้นงาน 2 วิธี ได้แก่ ทดสอบว่าสามารถควบคุมได้ไกลแค่ไหน และทดสอบว่าสามารถทำงานได้ถูกต้องตามความคาดหวังไหม

ขั้นตอนการทดสอบ

วิธีการทดสอบแบ่งออกเป็น 2 วิธี ได้แก่ ทดสอบว่าสามารถควบคุมได้ไกลแค่ไหน และทดสอบว่าสามารถทำงานได้ถูกต้องตามความคาดหวังไหม แต่ละวิธีมีขั้นตอนดังต่อไปนี้ ทดสอบว่าสามารถควบคุมได้ไกลแค่ไหน จะเริ่มจากการควบคุมหุ่นยนต์ในระยะ 5 เมตร และอาจจะเพิ่มขึ้นไปเรื่อย ๆ ทีละ 5 เมตร ในแนวแกน x และ แกน Y เริ่มจากการทดสอบภายในชั้นเดียวกันของอาคารแล้วเพิ่มขึ้นทีละชั้น ส่วนวิธีที่ 2 คือการทดสอบว่าสามารถทำงานได้ถูกต้องตามความคาดหวังไหม โดยมีขั้นตอนคือการกดควบคุมปุ่มในลักษณะของคันโยกแล้วเช็คว่าตรงตามที่คาดหวังไว้ไหม โดยหาค่าความแม่นยำเฉลี่ยใน 10 ครั้ง

ผลการทดสอบ

ตารางผลการทดสอบระยะที่สามารถควบคุมได้

ระยะทาง แกน x	ผลการทดสอบ	ผลที่คาดหวัง
5 เมตร	สามารถควบคุมได้ปกติ	สามารถควบคุมได้ปกติ
10 เมตร	สามารถควบคุมได้ปกติ	สามารถควบคุมได้ปกติ
15 เมตร	สามารถควบคุมได้ปกติ	สามารถควบคุมได้ปกติ
20 เมตร	สามารถควบคุมได้ปกติ	สามารถควบคุมได้ปกติ
25 เมตร	สามารถควบคุมได้ปกติ	สามารถควบคุมได้ปกติ
30 เมตร	สามารถควบคุมได้ปกติ แต่ค่อนข้างช้า	สามารถควบคุมได้ปกติ
35 เมตร	สามารถควบคุมได้ปกติ แต่ค่อนข้างช้า	สามารถควบคุมได้ปกติ

ระยะทาง แกน Y	ผลการทดสอบ	ผลที่คาดหวัง
ภายในชั้นเดียวกัน	สามารถควบคุมได้ปกติ	สามารถควบคุมได้ปกติ
ห่าง 1 ชั้น	สามารถควบคุมได้ปกติ	สามารถควบคุมได้ปกติ
ห่าง 2 ชั้น	สามารถควบคุมได้ปกติ	สามารถควบคุมได้ปกติ
ห่าง 3 ชั้น	สามารถควบคุมได้ปกติ	สามารถควบคุมได้ปกติ
ห่าง 4 ชั้น	สามารถควบคุมได้ปกติ แต่ค่อนข้างช้า	สามารถควบคุมได้ปกติ
ห่าง 5 ชั้น	สามารถควบคุมได้ปกติ แต่ค่อนข้างช้า	สามารถควบคุมได้ปกติ

ตารางผลการทดสอบความแม่นยำของการควบคุม

ปุ่มที่กด	ผลครั้งที่ 1	ผลครั้งที่ 2	ผลครั้งที่ 3	ผลครั้งที่ 4	ผลครั้งที่ 5	ผลครั้งที่ 6	ผลครั้งที่ 7	ผลครั้งที่ 8	ผลครั้งที่ 9	ผลครั้งที่ 10	ผลที่คาดหวัง
บังคับขึ้น	เดินหน้า แต่ช้า และไม่ตรง 100%	เดินหน้า แต่ช้า และไม่ตรง 100%	เดินหน้า แต่ไม่ตรง 100%	เดินหน้า ค้าง แต่ไม่ตรง 100%	เดินหน้า แต่ไม่ตรง 100%	เดินหน้า แต่ไม่ตรง 100%	เดินหน้า แต่ไม่ตรง 100%	เดินหน้า แต่ไม่ตรง 100%	เดินหน้า แต่ไม่ตรง 100%	เดินหน้า แต่ไม่ตรง 100%	เดินหน้า
บังคับไปทางซ้าย	เลี้ยวซ้าย	เลี้ยวซ้าย	เลี้ยวซ้าย	หมุนรอบตัวเอง 1 ที	เลี้ยวซ้าย เกิน	เลี้ยวซ้าย	เลี้ยวซ้าย	เลี้ยวซ้าย เกิน	เลี้ยวซ้าย เกิน	หมุนรอบตัวเอง 1 ที	เลี้ยวซ้าย
บังคับลง	ถอยหลัง ค้างและ เคลื่อนที่ไม่ตรง100%	ถอยหลัง ค้างและ เคลื่อนที่ ไม่ตรง 100%	ถอยหลัง และ เคลื่อนที่ ไม่ตรง 100%	ถอยหลัง ช้า ค้าง และ เคลื่อนที่ ไม่ตรง 100%	ถอยหลัง ค้างและ เคลื่อนที่ ไม่ตรง 100%	ถอยหลัง และ เคลื่อนที่ ไม่ตรง 100%	ถอยหลัง และ เคลื่อนที่ ไม่ตรง 100%	ถอยหลัง ค้างและ เคลื่อนที่ ไม่ตรง 100%	ถอยหลัง ค้างและ เคลื่อนที่ ไม่ตรง 100%	ถอยหลัง ช้า ค้าง และ เคลื่อนที่ ไม่ตรง 100%	ถอยหลัง
บังคับไปทางขวา	เลี้ยวขวา	เลี้ยวขวา เกิน	เลี้ยวขวา เกิน	เลี้ยวขวา เกิน	เลี้ยวขวา เกิน	เลี้ยวขวา เกิน	เลี้ยวขวา	เลี้ยวขวา	เลี้ยวขวา	เลี้ยวขวา	เลี้ยวขวา

กำหนดให้

ถ้าเป็นไปตามที่คาดหวัง มีค่าเป็น 1

ถ้าเป็นไปตามที่คาดหวังไม่ 100% มีค่าเป็น 0.5

ถ้าไม่เป็นไปตามที่คาดหวัง มีค่าเป็น 0

ดังนั้น จะได้

ค่าความแม่นยำของการบังคับขึ้น เป็น 0.5 บังคับไปทางซ้าย เป็น 0.65 บังคับลง เป็น 0.5 และ บังคับไปทางขวา เป็น 0.75

บทที่ 5
อภิปรายและข้อเสนอแนะ

สรุปผลการทดสอบ

จากการสร้างหุ่นยนต์ควบคุมและขับเคลื่อนแบบไร้สายจะเห็นได้ว่าผลการทดสอบค่อนข้างเป็นไปตามที่ต้องการ แม้อาจจะมีข้อผิดพลาดบ้างเล็กน้อยโดยจะเห็นได้ว่าค่าความแม่นยำของการบังคับขึ้น เป็น 0.5 แม้จะเดินหน้าได้แต่ก็ยังช้าและเดินหน้าไม่ตรงอย่างที่ควรจะเป็น ส่วนการบังคับไปทางซ้าย มีค่าความแม่นยำเป็น 0.65 โดยจาก 10 ครั้งอาจจะยังมีการเลี้ยวเกิน หรือทำงานผิดพลาดไปเลยจนหมุนรอบตัวเอง ส่วนการบังคับลงหรือการบังคับถอยหลัง ขออนุมานว่ามีค่าความแม่นยำเป็น 0.5 เพราะถึงแม้จะถอยหลังได้จริงแต่ก็อาจจะมีดีเลย์และการเคลื่อนที่ที่ไม่เป็นเส้นตกอยู่ดี และการบังคับไปทางขวา มีค่าความแม่นยำเป็น 0.75 โดยบางครั้งอาจจะมีการเลี้ยวเกินอยู่บ้าง

ปัญหาที่พบ

1. การเชื่อมต่อด้วย WiFiClient ทำให้หุ่นยนต์มีดีเลย์เกิดขึ้น ทำให้ไม่สามารถกำหนดการเคลื่อนที่ให้ถูกต้องได้ เช่น หากต้องการเลี้ยวก็จะไม่สามารถรู้ได้ว่าสั่งให้ตัวหุ่นยนต์หันไปทางด้านนั้นมากแค่ไหนแล้ว เวลาเลี้ยวซ้ายหรือเลี้ยวขวาจึงมักจะหมุนเกินที่ต้องการเสมอ

2. การเชื่อมต่อด้วย WiFiClient ไม่เสถียร ทำให้บางครั้ง NodeMCU-32 ที่ทำหน้าที่เป็นตัวส่งไม่สามารถส่งหาตัวรับได้ อาจจึงต้องมีการ Reset การทำงาน NodeMCU-32 ใหม่

3. การส่งข้อมูลจาก NodeMCU-32 เข้าช่อง Serial ของ FPGA มีรูปแบบสัญญาณ Digital ที่เกินความต้องการเป็นจำนวนมาก เช่น หากต้องการให้หุ่นยนต์เคลื่อนที่ไปข้างหน้าจะส่ง สัญญาณ 8 bit ซึ่งเป็น 0xAA เข้า FPGA แต่ในช่วงที่มีการรับส่งระหว่าง NodeMCU-32 กันเอง ตัว NodeMCU-32 ที่เป็นตัว Reciever ก็จะส่งค่าอื่น ๆ เข้าไปในบอร์ด FPGA เหมือนกัน

4. มอเตอร์ของทั้งสองล้อหมุนด้วยความเร็วที่ไม่เท่ากัน ทำให้การเคลื่อนที่ไปในทิศทางเดียว ตัวหุ่นยนต์จะเคลื่อนที่เอียงไปทางใดทางหนึ่ง

5. แหล่งจ่ายไฟหรือแหล่งให้พลังงานไม่เพียงพอกับบอร์ดต่าง ๆ ซึ่ง Power Supply ขนาด 12V ที่เชื่อมต่อกับตัวขับเคลื่อนมอเตอร์ ไม่สามารถจ่ายไฟเข้าทั้ง FPGA และ NodeMCU-32 ได้ เนื่องมาจากทั้งลักษณะช่องทางในการจ่ายไฟเข้าบอร์ดไม่สอดคล้องกับช่องทางที่ Power Supply สามารถจ่ายได้ และทั้งการต่อบอร์ดทั้งหมดเข้ากับ Power Supply ทำให้แหล่งจ่ายไฟไม่เพียงพอ

วิธีการแก้ไขปัญหา

1. ในด้านของการดีเลย์และการเชื่อมต่อที่ไม่เสถียร สามารถแก้ไขได้โดยการเปลี่ยนโปรโตคอลในการรับและส่งข้อมูล เป็นรูปแบบอื่น เช่น ส่งคลื่นจากโมดูล nRF24L01 MQTT TCP Boardcast โดยจำเป็นต้องทดสอบเพื่อเปรียบเทียบหาความเร็วและเสถียรภาพในการรับส่งของแต่ละโปรโตคอล

2. สัญญาณ Digital อื่น ๆ ที่รบกวนการทำงานของหุ่นยนต์สามารถแก้ได้จากการสร้างสัญญาญ Digital ที่ใช้ควบคุมตัวหุ่นยนต์ ให้เป็นเอกลักษณ์ ซึ่งมีโอกาสซ้ำกับสัญญาณรบกวนอื่น ๆ น้อย

3.ในส่วนของความเร็วในการหมุนของมอเตอร์ สามารถแก้ไขได้จากการเปลี่ยนมอเตอร์รวมไปถึงออกแบบวงจรและแรงดันไฟฟ้าที่ใช้ควบคุมมอเตอร์ใหม่

4. ปัญหาแหล่งจ่ายไฟ สามารถแก้ไขได้จากการเพิ่มแหล่งจ่ายไฟอื่น ๆ เช่น Power Bank เนื่องจากช่อง VIN ของ NodeMCU ไม่สามารถรับแรงดันไฟฟ้า 12 V และแรงดันไฟฟ้า 3.3 V จาก FPGA ไม่เพียงพอ และตัวบอร์ด FPGA เอง สามารถต่อแหล่งจ่ายไฟได้แค่ 2 แบบ Power Supply เดิม 12 V ที่มีขา VCC และ GND ไม่สามารถเชื่อมต่อโดยตรงกับ FPGA

ข้อเสนอแนะ

1. สามารถใช้ Module หรือโปรโตคอลอื่นในการรับส่งข้อมูลเพื่อเพิ่มเสถียรภาพและความรวดเร็วได้
2. สามารถใช้ FPGA ส่งสัญญาณแทนได้ แต่ต้องมี Module หรือช่องทาง Wireless อื่น ๆ ช่วย
3. สามารถเพิ่มจำนวนมอเตอร์เพื่อเพิ่มความเร็วและลดภาระของ Power Supply และตัวมอเตอร์ได้
4. สามารถใช้ปุ่มกดแทน Joy Stick เพื่อลดความคลาดเคลื่อนในการแมพและส่งสัญญาณ Analog

สิ่งที่จะพัฒนาต่อในอนาคต

1. ใช้ Module nRF24L01 ในการรับและส่งสัญญาณ เพื่อแก้ไขปัญหาการใช้ WiFi ที่อาจจะทำให้เกิดดีเลย์ขึ้น

(Code ตัวอย่างที่ใช้เซ็ต Register และการทำงานของโมดูล nRF24L01)

```
entity nrf is
   port(
        CLK      : in  std_logic; -- system clock
        RST      : in  std_logic; -- high active synchronous reset
        -- SPI SLAVE INTERFACE
        SCLK     : out  std_logic; -- SPI clock
        CS_N     : out  std_logic; -- SPI chip select, active in low
        MOSI     : out  std_logic; -- SPI serial data from master to slave
        MISO     : in std_logic; -- SPI serial data from slave to master
        CE       : out std_logic := '0';
        -- SPI Data Send
        SPI_IN   : in std_logic_vector(7 downto 0);
        MN  : out std_logic_vector(7 downto 0)
   );
end nrf;

architecture Behavioral of nrf is
   signal data_in : std_logic_vector(7 downto 0) := (others => '0');
   signal data_last : std_logic := '0';
   signal data_in_vld : std_logic := '0';
   signal data_in_rdy : std_logic;
   signal data_out : std_logic_vector(7 downto 0);
   signal data_out_vld : std_logic := '0';
   signal next_state,setup_state : integer range 0 to 30 := 0;
   signal payload_rdy : std_logic;
   signal payload_count : integer := 0;
   signal rst_btn      : std_logic;
```

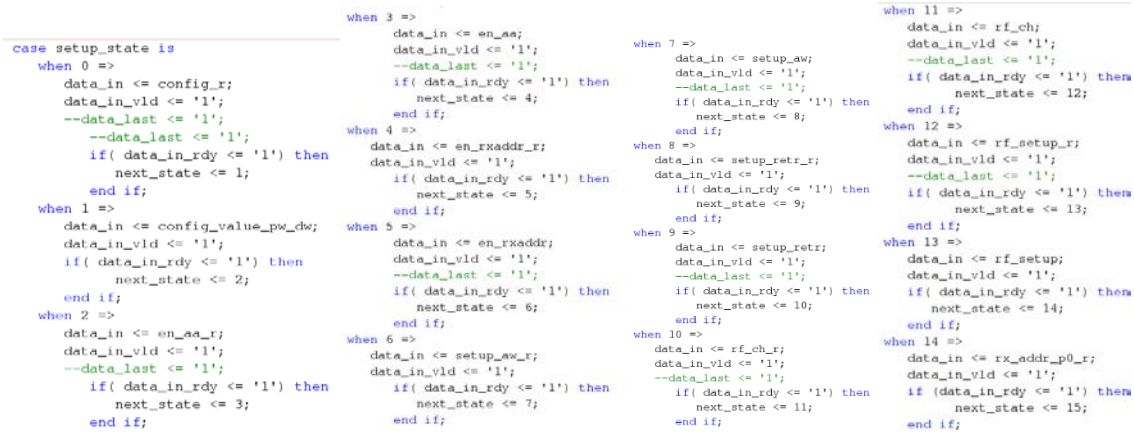
```
-- Config NRF Register Read
constant config_r       : std_logic_vector(7 downto 0) := "00100000";
constant en_aa_r        : std_logic_vector(7 downto 0) := "00100001";
constant en_rxaddr_r    : std_logic_vector(7 downto 0) := "00100010";
constant setup_aw_r     : std_logic_vector(7 downto 0) := "00100011";
constant setup_retr_r   : std_logic_vector(7 downto 0) := "00100100";
constant rf_ch_r        : std_logic_vector(7 downto 0) := "00100101";
constant rf_setup_r     : std_logic_vector(7 downto 0) := "00100110";
constant status_r       : std_logic_vector(7 downto 0) := "00100111";
--constant observe_tx_r    : std_logic_vector(7 downto 0) := "00001000";
--constant cd_r            : std_logic_vector(7 downto 0) := "00001001";
constant rx_addr_p0_r   : std_logic_vector(7 downto 0) := "00101010";
--constant rx_addr_p1_r    : std_logic_vector(7 downto 0) := "00001011";
--constant rx_addr_p2_r    : std_logic_vector(7 downto 0) := "00001100";
--constant rx_addr_p3_r    : std_logic_vector(7 downto 0) := "00001101";
--constant rx_addr_p4_r    : std_logic_vector(7 downto 0) := "00001110";
--constant rx_addr_p5_r    : std_logic_vector(7 downto 0) := "00001111";
constant tx_addr_r      : std_logic_vector(7 downto 0) := "00110000";
constant rx_pw_p0_r     : std_logic_vector(7 downto 0) := "00110001";
--constant rx_pw_p1_r      : std_logic_vector(7 downto 0) := "00010010";
--constant rx_pw_p2_r      : std_logic_vector(7 downto 0) := "00010011";
--constant rx_pw_p3_r      : std_logic_vector(7 downto 0) := "00010100";
--constant rx_pw_p4_r      : std_logic_vector(7 downto 0) := "00010101";
--constant rx_pw_p5_r      : std_logic_vector(7 downto 0) := "00010110";
--constant fifo_status_r   : std_logic_vector(7 downto 0) := "00010111";
```

```
-- Config NRF Value
constant config_value_pw_up : std_logic_vector(7 downto 0) := "00001110"; -- Enable CRC 2 Byte,PWR_UP = 1,PRX=1
constant config_value_pw_dw : std_logic_vector(7 downto 0) := "00001100"; -- Enable CRC 2 Byte,PWR_UP = 0,PRX=1
constant en_aa : std_logic_vector(7 downto 0) := "00000000"; -- Enable All auto ack
constant en_rxaddr : std_logic_vector(7 downto 0) := "00000001"; -- Enable data pipe 0
constant setup_aw : std_logic_vector(7 downto 0) := "00000001"; -- Address Width 3 byte
constant setup_retr : std_logic_vector(7 downto 0) := "00000000"; -- No Re-Transmit
constant rf_ch : std_logic_vector(7 downto 0) := "00000001"; --RF channel same RX TX
constant rf_setup : std_logic_vector(7 downto 0) := "00000111"; -- 1Mbps 0dBm
constant status : std_logic_vector(7 downto 0) := "01110000"; -- Reset IRQ to listen "01000000"
--constant observe_tx : std_logic_vector(7 downto 0) := "00000000";

constant rx_addr_p0_1 : std_logic_vector(7 downto 0) := "00000001"; --01
--constant rx_addr_p0_2 : std_logic_vector(7 downto 0) := "11100111"; --E7 Reset Value
--constant rx_addr_p0_3 : std_logic_vector(7 downto 0) := "11100111"; --E7 Reset Value

--constant rx_addr_p1 : std_logic_vector(7 downto 0) := "11000010"; --C2 Reset Value For 3 Bytes
--constant rx_addr_p3 : std_logic_vector(7 downto 0) := "11000011"; --C3 Reset Value
--constant rx_addr_p4 : std_logic_vector(7 downto 0) := "11000100"; --C4 Reset Value
--constant rx_addr_p5 : std_logic_vector(7 downto 0) := "11000101"; --C5 Reset Value

constant tx_addr_1 : std_logic_vector(7 downto 0) := "00000001"; --E7 Reset Value
--constant tx_addr_2 : std_logic_vector(7 downto 0) := "11100111"; --E7 Reset Value
--constant tx_addr_3 : std_logic_vector(7 downto 0) := "11100111"; --E7 Reset Value

constant rx_pw_p0 : std_logic_vector(7 downto 0) := "00000001"; -- Payload 1 Byte
```

```
constant zero_val : std_logic_vector(7 downto 0) := (others => '0'); -- For rx_pw_p1-p5
constant reset_val : std_logic_vector(7 downto 0) := "11100111"; --E7 For rx_addr_p0_2-3, rx_addr_p3-p5, tx_addr_1-3
--SPI Instruction
constant FLUSH_TX     : std_logic_vector(7 downto 0) := "11100001";
constant FLUSH_RX     : std_logic_vector(7 downto 0) := "11100010";
constant R_RX_PAYLOAD : std_logic_vector(7 downto 0) := "01100001";
constant W_TX_PAYLOAD : std_logic_vector(7 downto 0) := "10100000";
constant NOP          : std_logic_vector(7 downto 0) := "11111111";
```

```
begin
    rst_btn <= not RST;
spi_master_inst : entity work.spi_master
        generic map(
        CLK_FREQ    => 12e6,
        SCLK_FREQ   => 1e6,
        SLAVE_COUNT => 1
        )
        port map (
            CLK => CLK,
            RST => RST,
        -- SPI SLAVE INTERFACE
        SCLK => SCLK,
        CS_N(0) => CS_N,
        MOSI => MOSI,
        MISO => MISO,
        -- INPUT USER INTERFACE
        DIN => data_in,
        DIN_ADDR => (others => '0'),
        DIN_LAST => '1',
        DIN_VLD => data_in_vld,
        DIN_RDY => data_in_rdy,
        -- OUTPUT USER INTERFACE
        DOUT => data_out,
        DOUT_VLD => data_out_vld
        );
```

```
process (CLK)
begin
   if (rising_edge(CLK)) then
      if (reset = '1') then
         setup_state <= 0;
      else
         setup_state <= next_state;
      end if;
   end if;
end process;

process(setup_state,data_in_rdy,SPI_IN)
   variable counter : natural range 0 to (130000 / 20e6);
   variable counter2 : natural range 0 to (2000000 / 20e6);
begin
   next_state <= setup_state;
   data_last <= '0';
   data_in_vld <= '0';
   if (setup_state = 0 or setup_state = 1) then
      CE <= '0';
   else
      CE <= '1';
   end if;
```



```
case setup_state is
   when 0 =>
      data_in <= config_r;
      data_in_vld <= '1';
      --data_last <= '1';
         --data_last <= '1';
         if( data_in_rdy <= '1') then
            next_state <= 1;
         end if;
   when 1 =>
      data_in <= config_value_pw_dw;
      data_in_vld <= '1';
      if( data_in_rdy <= '1') then
            next_state <= 2;
      end if;
   when 2 =>
      data_in <= en_aa_r;
      data_in_vld <= '1';
      --data_last <= '1';
         if( data_in_rdy <= '1') then
            next_state <= 3;
         end if;
```

```
when 3 =>
      data_in <= en_aa;
      data_in_vld <= '1';
      --data_last <= '1';
      if( data_in_rdy <= '1') then
         next_state <= 4;
      end if;
when 4 =>
   data_in <= en_rxaddr_r;
   data_in_vld <= '1';
      if( data_in_rdy <= '1') then
         next_state <= 5;
      end if;
when 5 =>
      data_in <= en_rxaddr;
      data_in_vld <= '1';
      --data_last <= '1';
      if( data_in_rdy <= '1') then
         next_state <= 6;
      end if;
when 6 =>
   data_in <= setup_aw_r;
   data_in_vld <= '1';
      if( data_in_rdy <= '1') then
         next_state <= 7;
      end if;
```

```
when 7 =>
      data_in <= setup_aw;
      data_in_vld <= '1';
      --data_last <= '1';
      if( data_in_rdy <= '1') then
         next_state <= 8;
      end if;
when 8 =>
   data_in <= setup_retr_r;
   data_in_vld <= '1';
      if( data_in_rdy <= '1') then
         next_state <= 9;
      end if;
when 9 =>
      data_in <= setup_retr;
      data_in_vld <= '1';
      --data_last <= '1';
      if( data_in_rdy <= '1') then
         next_state <= 10;
      end if;
when 10 =>
   data_in <= rf_ch_r;
   data_in_vld <= '1';
   --data_last <= '1';
      if( data_in_rdy <= '1') then
         next_state <= 11;
      end if;
```

```
when 11 =>
   data_in <= rf_ch;
   data_in_vld <= '1';
   --data_last <= '1';
   if( data_in_rdy <= '1') then
         next_state <= 12;
   end if;
when 12 =>
   data_in <= rf_setup_r;
   data_in_vld <= '1';
   --data_last <= '1';
   if( data_in_rdy <= '1') then
         next_state <= 13;
   end if;
when 13 =>
   data_in <= rf_setup;
   data_in_vld <= '1';
   if( data_in_rdy <= '1') then
      next_state <= 14;
   end if;
when 14 =>
   data_in <= rx_addr_p0_r;
   data_in_vld <= '1';
   if (data_in_rdy <= '1') then
         next_state <= 15;
   end if;
```

```
when 15 =>
   data_in <= rx_addr_p0_l;
   data_in_vld <= '1';
   if (data_in_rdy <= '1') then
      next_state <= 16;
   end if;
when 16 =>
   data_in <= rx_addr_p0_l;
   data_in_vld <= '1';
   if (data_in_rdy <= '1') then
      next_state <= 17;
   end if;
when 17 =>
   data_in <= rx_addr_p0_l;
   data_in_vld <= '1';
   if (data_in_rdy <= '1') then
      next_state <= 18;
   end if;
when 18 =>
   data_in <= tx_addr_r;
   data_in_vld <= '1';
   if (data_in_rdy <= '1') then
      next_state <= 19;
   end if;
```

```
when 19 =>
   data_in <= tx_addr_l;
   data_in_vld <= '1';
   if (data_in_rdy <= '1') then
      next_state <= 20;
   end if;
when 20 =>
   data_in <= tx_addr_l;
   data_in_vld <= '1';
   if (data_in_rdy <= '1') then
      next_state <= 21;
   end if;
when 21 =>
   data_in <= tx_addr_l;
   data_in_vld <= '1';
   if (data_in_rdy <= '1') then
      next_state <= 22;
   end if;
when 22 =>
   data_in <= rx_pw_p0_r;
   data_in_vld <= '1';
   if( data_in_rdy <= '1') then
      next_state <= 23;
   end if;
```

```
when 23 =>
   data_in <= rx_pw_p0;
   data_in_vld <= '1';
   if( data_in_rdy <= '1') then
      next_state <= 24;
   end if;
when 24 =>
   data_in <= FLUSH_TX;
   data_in_vld <= '1';
   if( data_in_rdy <= '1') then
      next_state <= 25;
   end if;
when 25 =>
   data_in <= status_r;
   data_in_vld <= '1';
   if( data_in_rdy <= '1') then
      next_state <= 26;
   end if;
when 26 =>
   data_in <= status;
   data_in_vld <= '1';
   if( data_in_rdy <= '1') then
      next_state <= 27;
   end if;
```

```
when 27 =>
   data_in <= W_TX_PAYLOAD;
   data_in_vld <= '1';
   if (data_in_rdy = '1') then
      next_state <= 28;
   end if;
when 28 =>
   data_in <= SPI_IN;
   data_in_vld <= '1';
   if (data_in_rdy = '1') then
      if (payload_count < 3) then
         payload_count <= payload_count + 1;
         next_state <= 29; -- to W_TX_PAYLOAD but change to config_r
      else
         payload_count <= 0;
         next_state <= 26; -- to Flush TX
      end if;
end if;
when 29 =>
   data_in <= config_r;
   data_in_vld <= '1';
      if( data_in_rdy <= '1') then
         next_state <= 30;
      end if;
```

```
when 30 =>
   data_in <= config_value_pw_up;
   data_in_vld <= '1';
   if( data_in_rdy <= '1') then
      next_state <= 0;
   end if;
end case;
MN <= data_in;
end process;

end Behavioral;
```

(ตัวอย่าง Code SPI Interface ใน FPGA)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.MATH_REAL.ALL;

-- THE SPI MASTER MODULE SUPPORT ONLY SPI MODE 0 (CPOL=0, CPHA=0)!!!

entity SPI_MASTER is
    Generic (
        CLK_FREQ    : natural := 20e6; -- set system clock frequency in Hz
        SCLK_FREQ   : natural := 2e6;  -- set SPI clock frequency in Hz (condition: SCLK_FREQ <= CLK_FREQ/10)
        WORD_SIZE   : natural := 8;    -- size of transfer word in bits, must be power of two
        SLAVE_COUNT : natural := 1     -- count of SPI slaves
    );
    Port (
        CLK      : in  std_logic; -- system clock
        RST      : in  std_logic; -- high active synchronous reset
        -- SPI MASTER INTERFACE
        SCLK     : out std_logic; -- SPI clock
        CS_N     : out std_logic_vector(SLAVE_COUNT-1 downto 0); -- SPI chip select, active in low
        MOSI     : out std_logic; -- SPI serial data from master to slave
        MISO     : in  std_logic; -- SPI serial data from slave to master
        -- INPUT USER INTERFACE
        DIN      : in  std_logic_vector(WORD_SIZE-1 downto 0); -- data for transmission to SPI slave
        DIN_ADDR : in  std_logic_vector(natural(ceil(log2(real(SLAVE_COUNT))))-1 downto 0); -- SPI slave address
        DIN_LAST : in  std_logic; -- when DIN_LAST = 1, last data word, after transmit will be asserted CS_N
        DIN_VLD  : in  std_logic; -- when DIN_VLD = 1, data for transmission are valid
        DIN_RDY  : out std_logic; -- when DIN_RDY = 1, SPI master is ready to accept valid data for transmission
        -- OUTPUT USER INTERFACE
        DOUT     : out std_logic_vector(WORD_SIZE-1 downto 0); -- received data from SPI slave
        DOUT_VLD : out std_logic  -- when DOUT_VLD = 1, received data are valid
    );
end entity;
```

```
architecture RTL of SPI_MASTER is

    constant DIVIDER_VALUE : natural := (CLK_FREQ/SCLK_FREQ)/2;
    constant WIDTH_CLK_CNT : natural := natural(ceil(log2(real(DIVIDER_VALUE))));
    constant WIDTH_ADDR    : natural := natural(ceil(log2(real(SLAVE_COUNT))));
    constant BIT_CNT_WIDTH : natural := natural(ceil(log2(real(WORD_SIZE))));

    type state_t is (idle, first_edge, second_edge, transmit_end, transmit_gap);

    signal addr_reg        : unsigned(WIDTH_ADDR-1 downto 0);
    signal sys_clk_cnt     : unsigned(WIDTH_CLK_CNT-1 downto 0);
    signal sys_clk_cnt_max : std_logic;
    signal spi_clk         : std_logic;
    signal spi_clk_rst     : std_logic;
    signal din_last_reg_n  : std_logic;
    signal first_edge_en   : std_logic;
    signal second_edge_en  : std_logic;
    signal chip_select_n   : std_logic;
    signal load_data       : std_logic;
    signal miso_reg        : std_logic;
    signal shreg           : std_logic_vector(WORD_SIZE-1 downto 0);
    signal bit_cnt         : unsigned(BIT_CNT_WIDTH-1 downto 0);
    signal bit_cnt_max     : std_logic;
    signal rx_data_vld     : std_logic;
    signal master_ready    : std_logic;
    signal present_state   : state_t;
    signal next_state      : state_t;

begin

    ASSERT (DIVIDER_VALUE >= 5) REPORT "condition: SCLK_FREQ <= CLK_FREQ/10" SEVERITY ERROR;

    load_data <= master_ready and DIN_VLD;
    DIN_RDY   <= master_ready;

    -- ------------------------------------------------------------------------
```

```
begin

    ASSERT (DIVIDER_VALUE >= 5) REPORT "condition: SCLK_FREQ <= CLK_FREQ/10" SEVERITY ERROR;

    load_data <= master_ready and DIN_VLD;
    DIN_RDY   <= master_ready;

    -- ------------------------------------------------------------------------
    --  SYSTEM CLOCK COUNTER
    -- ------------------------------------------------------------------------

    sys_clk_cnt_max <= '1' when (to_integer(sys_clk_cnt) = DIVIDER_VALUE-1) else '0';

    sys_clk_cnt_reg_p : process (CLK)
    begin
        if (rising_edge(CLK)) then
            if (RST = '1' or sys_clk_cnt_max = '1') then
                sys_clk_cnt <= (others => '0');
            else
                sys_clk_cnt <= sys_clk_cnt + 1;
            end if;
        end if;
    end process;
```

```
    -- ------------------------------------------------------------------------
    --  SPI CLOCK GENERATOR AND REGISTER
    -- ------------------------------------------------------------------------

    spi_clk_gen_p : process (CLK)
    begin
        if (rising_edge(CLK)) then
            if (RST = '1' or spi_clk_rst = '1') then
                spi_clk <= '0';
            elsif (sys_clk_cnt_max = '1') then
                spi_clk <= not spi_clk;
            end if;
        end if;
    end process;

    SCLK <= spi_clk;

    -- ------------------------------------------------------------------------
    --  BIT COUNTER
    -- ------------------------------------------------------------------------

    bit_cnt_max <= '1' when (bit_cnt = WORD_SIZE-1) else '0';

    bit_cnt_p : process (CLK)
    begin
        if (rising_edge(CLK)) then
            if (RST = '1' or spi_clk_rst = '1') then
                bit_cnt <= (others => '0');
            elsif (second_edge_en = '1') then
                bit_cnt <= bit_cnt + 1;
            end if;
        end if;
    end process;
```

```
-- ------------------------------------------------------------------------
--  SPI MASTER ADDRESSING
-- ------------------------------------------------------------------------

addr_reg_p : process (CLK)
begin
    if (rising_edge(CLK)) then
        if (RST = '1') then
            addr_reg <= (others => '0');
        elsif (load_data = '1') then
            addr_reg <= unsigned(DIN_ADDR);
        end if;
    end if;
end process;

one_slave_g: if (SLAVE_COUNT = 1) generate
    CS_N(0) <= chip_select_n;
end generate;

more_slaves_g: if (SLAVE_COUNT > 1) generate
    cs_n_g : for i in 0 to SLAVE_COUNT-1 generate
        cs_n_p : process (addr_reg, chip_select_n)
        begin
            if (addr_reg = i) then
                CS_N(i) <= chip_select_n;
            else
                CS_N(i) <= '1';
            end if;
        end process;
    end generate;
end generate;
```

```
-- ------------------------------------------------------------------------
--  DIN LAST RESISTER
-- ------------------------------------------------------------------------

din_last_reg_n_p : process (CLK)
begin
    if (rising_edge(CLK)) then
        if (RST = '1') then
            din_last_reg_n <= '0';
        elsif (load_data = '1') then
            din_last_reg_n <= not DIN_LAST;
        end if;
    end if;
end process;

-- ------------------------------------------------------------------------
--  MISO SAMPLE REGISTER
-- ------------------------------------------------------------------------

miso_reg_p : process (CLK)
begin
    if (rising_edge(CLK)) then
        if (first_edge_en = '1') then
            miso_reg <= MISO;
        end if;
    end if;
end process;
```

```
-- ------------------------------------------------------------------------
--  DATA SHIFT REGISTER
-- ------------------------------------------------------------------------

shreg_p : process (CLK)
begin
    if (rising_edge(CLK)) then
        if (load_data = '1') then
            shreg <= DIN;
        elsif (second_edge_en = '1') then
            shreg <= shreg(WORD_SIZE-2 downto 0) & miso_reg;
        end if;
    end if;
end process;

DOUT <= shreg;
MOSI <= shreg(WORD_SIZE-1);

-- ------------------------------------------------------------------------
--  DATA OUT VALID RESISTER
-- ------------------------------------------------------------------------

dout_vld_reg_p : process (CLK)
begin
    if (rising_edge(CLK)) then
        if (RST = '1') then
            DOUT_VLD <= '0';
        else
            DOUT_VLD <= rx_data_vld;
        end if;
    end if;
end process;
```

```
    -- ------------------------------------------------------------------------
    --  SPI MASTER FSM
    -- ------------------------------------------------------------------------

    -- PRESENT STATE REGISTER
    fsm_present_state_p : process (CLK)
    begin
        if (rising_edge(CLK)) then
            if (RST = '1') then
                present_state <= idle;
            else
                present_state <= next_state;
            end if;
        end if;
    end process;

        -- NEXT STATE LOGIC
        fsm_next_state_p : process (present_state, DIN_VLD, sys_clk_cnt_max,
                                    bit_cnt_max)
        begin
            case present_state is
                when idle =>
                    if (DIN_VLD = '1') then
                        next_state <= first_edge;
                    else
                        next_state <= idle;
                    end if;

                when first_edge =>
                    if (sys_clk_cnt_max = '1') then
                        next_state <= second_edge;
                    else
                        next_state <= first_edge;
                    end if;

                when second_edge =>
                    if (sys_clk_cnt_max = '1') then
                        if (bit_cnt_max = '1') then
                            next_state <= transmit_end;
                        else
                            next_state <= first_edge;
                        end if;
                    else
                        next_state <= second_edge;
                    end if;

                when transmit_end =>
                    if (sys_clk_cnt_max = '1') then
                        next_state <= transmit_gap;
                    else
                        next_state <= transmit_end;
                    end if;
```

```
                when transmit_gap =>
                    if (sys_clk_cnt_max = '1') then
                        next_state <= idle;
                    else
                        next_state <= transmit_gap;
                    end if;

                when others =>
                    next_state <= idle;
            end case;
        end process;
```

```
-- OUTPUTS LOGIC
fsm_outputs_p : process (present_state, din_last_reg_n, sys_clk_cnt_max)
begin
    case present_state is
        when idle =>
            master_ready   <= '1';
            chip_select_n  <= not din_last_reg_n;
            spi_clk_rst    <= '1';
            first_edge_en  <= '0';
            second_edge_en <= '0';
            rx_data_vld    <= '0';

        when first_edge =>
            master_ready   <= '0';
            chip_select_n  <= '0';
            spi_clk_rst    <= '0';
            first_edge_en  <= sys_clk_cnt_max;
            second_edge_en <= '0';
            rx_data_vld    <= '0';

        when second_edge =>
            master_ready   <= '0';
            chip_select_n  <= '0';
            spi_clk_rst    <= '0';
            first_edge_en  <= '0';
            second_edge_en <= sys_clk_cnt_max;
            rx_data_vld    <= '0';

        when transmit_end =>
            master_ready   <= '0';
            chip_select_n  <= '0';
            spi_clk_rst    <= '1';
            first_edge_en  <= '0';
            second_edge_en <= '0';
            rx_data_vld    <= sys_clk_cnt_max;
```

```
            when transmit_gap =>
                master_ready   <= '0';
                chip_select_n  <= not din_last_reg_n;
                spi_clk_rst    <= '1';
                first_edge_en  <= '0';
                second_edge_en <= '0';
                rx_data_vld    <= '0';

            when others =>
                master_ready   <= '0';
                chip_select_n  <= not din_last_reg_n;
                spi_clk_rst    <= '1';
                first_edge_en  <= '0';
                second_edge_en <= '0';
                rx_data_vld    <= '0';
        end case;
    end process;

end architecture;
```

2. ติดตั้งเซนเซอร์ที่สามารถตรวจจับอันตรายอื่น ๆ เพิ่มเติม เพื่อให้หุ่นยนต์สามารถตรวจจับความเสี่ยงต่าง ๆ ในพื้นที่บริเวณนั้นได้

(ตัวอย่าง Code สำหรับการใช้ Module MQ135 เพื่อวัดแก๊ส CO2 และส่งข้อมูลผ่าน nRF24L01)

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#define MQ135_PIN A0

// RL ดูจากหลังบอร์ด
int MQ_RESISTOR = 1000;

// RO ที่คำนวณได้
long RO = 3212.58627;

//min และ max Rs/Ro
float MINRSRO = 0.358;
float MAXRSRO = 2.428;

// a และ b คำนวณได้จากกราฟ
float a = 116.602;
float b = -2.769;
RF24 radio(9, 8);  // CE, CSN
char text[32];
//address through which two modules communicate.
const byte address[6] = "00001";
void setup() {
  Serial.begin(9600);
  // Pre heat 20 วินาที
  Serial.println("Pre-heat sensor 20 seconds");
  delay(20000);
  Serial.println("Sensor ready start reading");
  radio.begin();
  //set the address
  radio.openWritingPipe(address);
  //Set module as transmitter
  radio.stopListening();
}
```

```
void loop() {
// อ่านค่า ADC และแปลงเป็นค่า Rs พร้อมแสดงค่าทาง Serial Monitor
  int ADCRAW = analogRead(MQ135_PIN);
  float RS = ((1024.0*MQ_RESISTOR)/ADCRAW) - MQ_RESISTOR;
  Serial.print("Rs: ");
  Serial.println(RS);

// คำนวณ Rs/Ro พร้อมแสดงค่าทาง Serial Monitor
  float RSRO = RS/RO;
  Serial.print("Rs/Ro: ");
  Serial.println(RSRO);

// ถ้า Rs/Ro อยู่ในช่วงที่วัดได้ ให้คำนวณและแสดงค่า ppm
// ถ้าค่าที่ได้น้อยหรือมากเกินไปให้แสดงข้อความว่าเกินช่วงที่เซ็นเซอร์จะวัดได้
  if(RSRO < MAXRSRO && RSRO > MINRSRO) {
   float ppm = a*pow(RS/RO, b);
   Serial.print("CO2 : ");
   Serial.print(ppm);
   Serial.println(" ppm");

  sprintf(text, "CO2 : %d ppm", ppm);
  Serial.println(text);
  const char text1[] = "Hello World";
  //radio.write(&text, sizeof(text));
  radio.write(&text1, sizeof(text1));
  }
  else {
   Serial.println("Out of range.");
  }

 delay(5000);
}
```

3. เพิ่มเซ็นเซอร์อื่น ๆ เพื่อให้สามารถทำงานได้เฉพาะทางมากขึ้น

บรรณานุกรม

- Interfacing Dual-Axis Joystick Shield with Arduino

- esp32 communication with esp32

- UART

- เพิ่มเติม

ภาคผนวก

Code & Comment

Code ส่วน nodeMCU ESP32

Code ส่วนของตัวหุ่นยนต์

```
#include <WiFi.h>
#include <WiFiServer.h>

const char* ssid = "HardwareLab";
const char* password = "hwlab504";
int receiverPort = 1234;  // Port number to receive data

WiFiServer server(receiverPort);
WiFiClient client;

  IPAddress local_IP(192, 168, 1, 112);
// Set your Gateway IP address
  IPAddress gateway(192, 168, 1, 1);

  IPAddress subnet(255, 255, 0, 0);
void setup() {
  Serial.begin(2400);
  pinMode(2,OUTPUT);
  digitalWrite(2,LOW);
  if (!WiFi.config(local_IP, gateway, subnet)) {
    Serial.println("STA Failed to configure");
  }
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }

  Serial.println("Connected to Wi-Fi");

  Serial.println(WiFi.localIP());
  // Start the server
  server.begin();
  Serial.println("Server started");

}

void loop() {
  char data[6];
  byte sendData = 0x00;
  // Check if a client has connected
  client = server.available();
  if (client) {
    while (client.connected()) {
      // Read data from the client
      if (client.available()) {
        String receivedData = client.readStringUntil('\n');
        Serial.println("Received data from the sender:");
        digitalWrite(2,HIGH);
          if  (receivedData.indexOf("Up") != -1) {
      sendData = 0xAA;
      Serial.println(receivedData);
    }
    else if (receivedData.indexOf("Left") != -1){
      sendData = 0xBB;
      Serial.println(receivedData);
    }
    else if (receivedData.indexOf("Right") != -1){
      sendData = 0xCC;
      Serial.println(receivedData);
    }
    else if (receivedData.indexOf("Down") != -1){
      sendData = 0xDD;
      Serial.println(receivedData);
    }
    else if (receivedData.indexOf("Neutral") != -1){
      sendData = 0xEE;
      Serial.println(receivedData);
    }
    Serial.println(receivedData);
      }
      Serial.println(receivedData);
      Serial.write(sendData);
      Serial.println(sendData,BIN);
      Serial.println(sendData);
        }
      }
    client.stop();
    digitalWrite(2,LOW);
   }
}
```

Code ส่วนของรีโมท

```ino
#include <WiFi.h>

const char* ssid = "HardwareLab";
const char* password = "hwlab504";
IPAddress receiverIP(192, 168, 1, 112);  // IP address of the receiver ESP32 board
int receiverPort = 1234;  // Port number to send data

const int threshold = 100;
WiFiClient client;
IPAddress local_IP(192, 168, 1, 113);
// Set your Gateway IP address
  IPAddress gateway(192, 168, 1, 1);

  IPAddress subnet(255, 255, 0, 0);
void setup() {
  Serial.begin(115200);
 if (!WiFi.config(local_IP, gateway, subnet)) {
    Serial.println("STA Failed to configure");
  }
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
Serial.println(WiFi.localIP());
  Serial.println("Connected to Wi-Fi");
}

void loop() {
  // Read joystick X and Y values
  int joystickX = analogRead(36);
  int joystickY = analogRead(39);

  // Read button states (A-F)
  bool buttonA = digitalRead(2);
  bool buttonB = digitalRead(3);
  bool buttonC = digitalRead(4);
  bool buttonD = digitalRead(5);

  // Create data packet
  char data[6];
  data[0] = joystickX;
  data[1] = joystickY;
  data[2] = buttonA;
  data[3] = buttonB;
  data[4] = buttonC;
  data[5] = buttonD;
  String direction;

  if (joystickX < threshold) {
    direction = "Left";
  } else if (joystickX > 4095 - threshold) {
    direction = "Right";
  } else if (joystickY < threshold) {
    direction = "Down";
  } else if (joystickY > 4095 - threshold) {
    direction = "Up";
  } else {
    direction = "Neutral";
  }
  Serial.print(joystickX);
  Serial.print(" ");
  Serial.println(joystickY);
  Serial.println("Direction: " + direction);
  // Send data to the receiver
  if (client.connect(receiverIP, receiverPort)) {
    client.println(direction);
    Serial.println("Sent data to the receiver");
    client.stop();
  } else {
    Serial.println("Failed to connect to the receiver");
  }
Serial.println(data[0]);
 Serial.println(data[1]);
  delay(100);
}
```

Code ส่วน FPGA

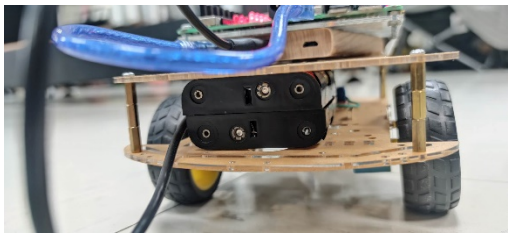
Code ส่วนการควบคุมมอเตอร์

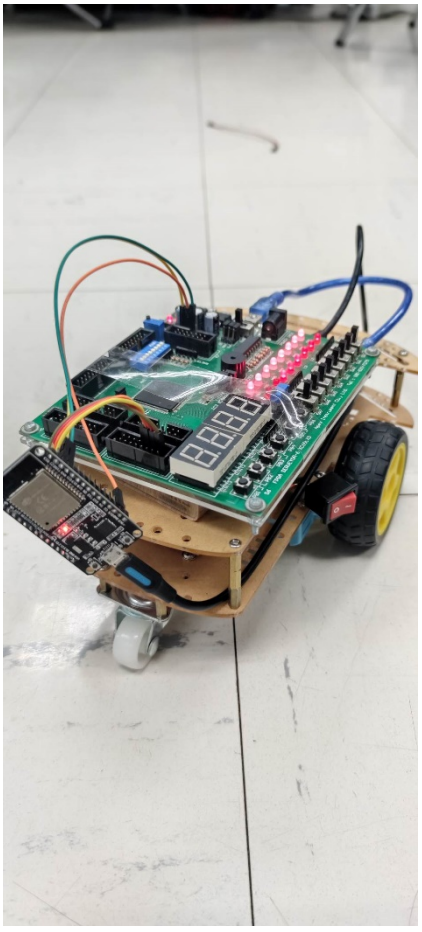
```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    17:49:40 05/22/2023
-- Design Name:
-- Module Name:    Car_Control - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Car_Control is
Port(
                        Reset   : in std_logic;
                SysClk  : in std_logic;
                Rx              : in  std_logic;
                --Tx            : out  std_logic;
                Motor     : out std_logic_vector(3 downto 0);
                Led             : out std_logic_vector(7 downto 0);
                MN                      : out std_logic_vector(7 downto 0) := "00000000"
                );
end Car_Control;

architecture Behavioral of Car_Control is
        -- Car Direction
        signal data : std_logic_vector(7 downto 0);
        signal state : std_logic_vector(3 downto 0);
        constant move_forward : std_logic_vector(3 downto 0) := "1010";
        constant move_backward : std_logic_vector(3 downto 0) := "0101";
        constant turn_left : std_logic_vector(3 downto 0) := "1001";
        constant turn_right : std_logic_vector(3 downto 0) := "0110";
        constant stop : std_logic_vector(3 downto 0) := "0000";
        begin
        rx_inst : entity work.rx
    port map(
                Reset => Reset,
                SysClk => SysClk,
                Rx => Rx,
                Data => data,
                Led => Led
    );


        process(SysClk)
begin
        if(rising_edge(SysClk)) then
                case data is
                                when x"AA" =>
                                                motor <= move_forward;
                                                state <= move_forward;
                                                MN(3 downto 0) <= move_forward;
                                when x"DD" =>
                                                motor <= move_backward;
                                                state <= move_backward;
                                                MN(3 downto 0) <= move_backward;
                                when x"BB" =>
                                                motor <= turn_left;
                                                state <= turn_left;
                                                MN(3 downto 0) <= turn_left;
                                when x"CC" =>
                                                motor <= turn_right;
                                                state <= turn_right;
                                                MN(3 downto 0) <= turn_right;
                                when x"EE" =>
                                                motor <= stop;
                                                state <= stop;
                                                MN(3 downto 0) <= stop;
                                when others =>
                                                motor <= state;
                                end case;
        end if;
end process;
end Behavioral;
```

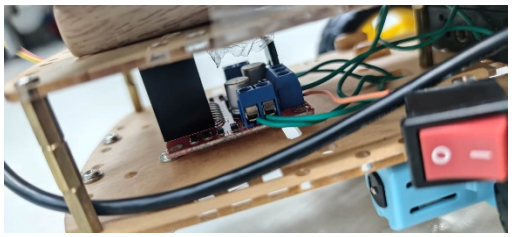
Code ส่วนการสื่อสารแบบ UART

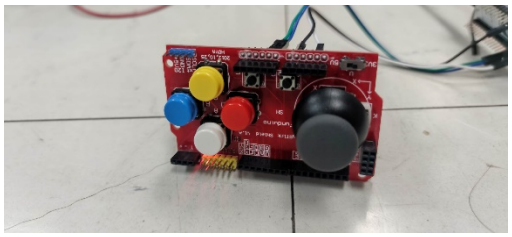
```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    18:52:37 05/20/2023
-- Design Name:
-- Module Name:    rx - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.NUMERIC_STD.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity rx is
        Port (
                Reset   : in std_logic;
                SysClk  : in std_logic;
                Rx              : in  std_logic;
                Data            : out std_logic_vector(7 downto 0) := "00000000";
                Led             : out std_logic_vector(7 downto 0) := "00000000");
end rx;

 architecture Behavioral of rx is

        constant Count_BuadRate : std_logic_vector (15 downto 0) := X"208D"; --Baud rate = 9600 bit per sec.
        signal  Data_Pack               : std_logic_vector (9 downto 0) := "0000000000";
-- internal signal
        signal  Clk_Count               : std_logic_vector (15 downto 0) :=x"0000";
--variable
        signal  shift_bit               : integer range 0 to 9 :=0;
        signal  Start_bit_check: boolean := false;
begin
        Receive_Rx : process (SysClk, Rx) Is
                Begin
                        if rising_edge(SysClk) then
                                if (Reset='1') then     -- Press Reset button for clear all.
                                        shift_bit <=0; --move to Data_Pack [0]
                                        Data_Pack <= (others=> '0');
                                        Start_bit_check <= false;
                                        Clk_Count <= (others=> '0');
                                        Led <= (others => '0');
                                        Data <= (others => '0');
                                else
                                        if Rx = '0' and Start_bit_check = false then --Capture start bit Enable start bit flag
                                        Start_bit_check <= true;
                                        Data_Pack <= (others=> '0');
                                        Clk_Count <= (others=> '0');                                                    --Clear Clk_Count
                                        elsif (Start_bit_check = true) then                     -- Check flag start bit
                                                if (Count_BuadRate-1)=Clk_Count then
                                                        shift_bit <=    shift_bit+1;
                                                        Clk_Count <= (others=> '0');
                                                elsif Clk_Count=(x"411"-1) then                                 --Capture data logic on middle point = Count_BuadRate/2 = x"516"
                                                        Data_Pack(shift_bit) <= Rx;
                                                        Clk_Count <= Clk_Count + 1;
                                                        if (shift_bit=9) then
                                                        shift_bit <= 0;
                                                        Start_bit_check <= false;                                               -- Clear start bit flag
                                                        Led <= Data_Pack (8 downto 1);                  -- ass LED
                                                        Data <= Data_Pack (8 downto 1);
                                                        end if;
                                                else
                                                        Clk_Count <= Clk_Count+1;
                                                End if;
                                        End if;
                                End if;
                        End if;
        End process Receive_Rx;
end Behavioral;
```

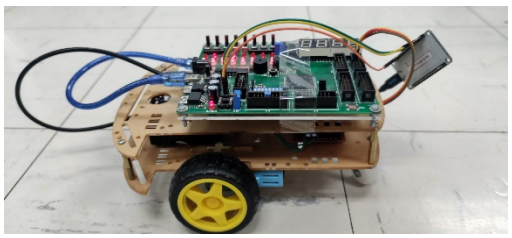
รูปชิ้นงาน แบบ ถ่ายโชว์

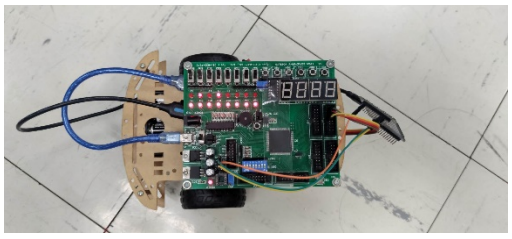












รูปทีมงานทั้งหมด

นายพชรพล โชคคุณ 63010631

นางสาวภัฑราณิษฐ์ เทศเจริญ 63010727

นายมาเหนือเมฆ ประดิษฐพงษ์ 63010789