



รายงานวิชา 2100301
การฝึกงานวิศวกรรม (ENGINEERING PRACTICE)

จดทำโดย

นาย บรรณวิชญ์ ทีปสว่าง
รหัสประจำตัว 6430216821
ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์
จุฬาลงกรณ์มหาวิทยาลัย

หน่วยงานที่ฝึกงาน

บริษัท เอ็นดีอาร์ โซลูชั่น (ประเทศไทย) จำกัด
เลขที่ 128/409 อาคารพญาไทพลาซ่า ชั้นที่ 37 ถนนพญาไท
แขวงทุ่งพญาไท เขตราชเทวี กรุงเทพมหานคร 10400

วิศวกรผู้ดูแล

นาย ศักดิ์รพี เต็มศิริพันธุ์
Hardware Developer

ช่วงระยะเวลาการฝึกงาน

ตั้งแต่วันที่ 27 พฤษภาคม 2567
ถึงวันที่ 31 กรกฎาคม 2567

รวมระยะเวลาการฝึกงาน

10 สัปดาห์ 44 วัน 350 ชั่วโมง

ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์
จุฬาลงกรณ์มหาวิทยาลัย

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของรายวิชา 2100301 การฝึกงานวิศวกรรม (ENGINEERING PRACTICE) ปีการศึกษา 2567 คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย จัดทำขึ้นเพื่อรวบรวมความรู้ที่ได้รับ งานที่ได้รับมอบหมาย และประสบการณ์จากการฝึกงานที่ บริษัท เอ็นดีอาร์ โซลูชัน (ประเทศไทย) จำกัด ตั้งแต่วันที่ 27 พฤษภาคม พ.ศ.2567 ถึง วันที่ 31 กรกฎาคม พ.ศ.2567 ระหว่างการฝึกงานได้ศึกษาและ ใช้งานบอร์ด Arrow DECA MAX10 และ Terasic SoCKit สำหรับการออกแบบวงจรดิจิทัลด้วย FPGA โดยใช้ ภาษา VHDL รวมถึงการออกแบบการทำงานร่วมกับ Soft core & Hard core Processor โดยรายงานฉบับนี้ แบ่งออกเป็น 4 บท ได้แก่ 1. บทนำ 2. รายละเอียดของบริษัทที่เข้าฝึกงาน 3. รายละเอียดงาน 4. สรุป

ทั้งนี้ทางผู้จัดทำต้องขอขอบคุณทางบริษัท เอ็นดีอาร์ โซลูชัน (ประเทศไทย) จำกัด ที่ได้ให้โอกาส สำหรับการเข้าฝึกงานครั้งนี้ ขอขอบคุณผู้ดูแลการฝึกงาน และ บุคลากรทุกท่าน ที่มอบความรู้ ให้ความ ช่วยเหลือและคำแนะนำตลอดระยะเวลาการฝึกงาน

บรรณวิชญ์ ทีปสว่าง

นิสิตฝึกงาน

สารบัญ

| | หน้า |
|---|-----------|
| คำนำ | ก |
| สารบัญ | ข |
| บทที่ 1 บทนำ | 1 |
| 1.1 วัตถุประสงค์ของการฝึกงาน | 1 |
| 1.2 ช่วงเวลาการฝึกงาน | 1 |
| 1.3 สภาพการทำงานระหว่างฝึกงาน | 1 |
| 1.4 ผู้ควบคุมดูแลการฝึกงาน | 1 |
| บทที่ 2 รายละเอียดของบริษัทที่เข้าฝึกงาน | 2 |
| 2.1 ที่ตั้งและข้อมูลการติดต่อ | 2 |
| 2.2 ประวัติโดยย่อ | 2 |
| 2.3 ลักษณะงานโดยรวม | 2 |
| 2.4 ระบบบริหาร | 2 |
| บทที่ 3 รายละเอียดงาน | 3 |
| 3.1 การใช้งาน ARROW DECA (MAX10) | 3 |
| 3.1.1 เรียนรู้การสร้างโปรเจค VHDL เพื่อออกแบบวงจรดิจิทัลบน MAX10 FPGA | 3 |
| 3.1.2 ศึกษาและใช้งาน NIOS II SOFT CPU | 10 |
| 3.1.3 การสร้าง GPIO IP และ สัญญาณ PWM | 15 |
| 3.2 การใช้งาน TERASIC SoC KIT (CYCLONE V SOC) | 18 |
| 3.2.1 RUNNING LINUX | 18 |
| 3.2.2 ใช้ U-BOOT DEBUG GPIO IP | 20 |
| 3.2.3 SINE WAVE GENERATOR | 21 |
| 3.3 เรียนรู้การใช้งาน OSCILLOSCOPE วัดสัญญาณแรงดันไฟฟ้าบนบอร์ด | 24 |
| บทที่ 4 สรุป | 25 |

| | |
|-----------------------------------|-----------|
| 4.1 ประโยชน์ที่ได้รับจากการฝึกงาน | 25 |
| 4.2 ปัญหา อุปสรรค และข้อเสนอแนะ | 25 |
| เอกสารอ้างอิง | 26 |
| ภาคผนวก | |

บทที่ 1 บทนำ

1.1 วัตถุประสงค์ของการฝึกงาน

การฝึกงานวิศวกรรมเมืองตุ่นประสงค์เพื่อนำความรู้และทักษะที่ได้เรียนรู้มาไปประยุกต์ใช้ในการทำงานจริง และช่วยให้เข้าใจกระบวนการทำงานในองค์กร ทั้งการออกแบบ การพัฒนา และการแก้ไขปัญหาต่างๆ นอกจากนี้ยังเป็นโอกาสที่ดีในการพัฒนาทักษะด้านการทำงานร่วมกับผู้อื่น การสื่อสาร และการจัดการเวลาซึ่งจะทำให้ได้รับประสบการณ์อันเป็นประโยชน์ต่อการทำงานในอนาคต

1.2 ช่วงเวลาการฝึกงาน

ช่วงเวลาฝึกงานคือ วันที่ 27 พฤษภาคม พ.ศ.2567 ถึง วันที่ 31 กรกฎาคม พ.ศ.2567 โดยเวลาทำงานคือ วันจันทร์ถึงวันศุกร์ 8:00น. – 17:00น. พัก 1 ชั่วโมง ซึ่งนับเป็นระยะเวลาฝึกงานรวมทั้งหมด 10 สัปดาห์ 44 วัน 350 ชั่วโมง

1.3 สภาพการทำงานระหว่างฝึกงาน

ฝึกงานแบบ Work From Home ดังนั้นจึงฝึกงานอยู่ที่บ้านของตนเองเป็นหลัก โดยจะมีประชุมเพื่อ update ความคืบหน้าและสอบถามเพิ่มเติมทุกวันอังคารและพฤหัสบดี เวลา 8:00น. – 9:00น. นอกเหนือจากเวลาดังกล่าวสามารถติดต่อผู้ควบคุมดูแลได้ผ่านทาง Google Chat และ Discord

มีโอกาสเข้าฝึกงานที่บริษัทสองครั้ง ครั้งแรกได้เข้าไปเรียนรู้การใช้งาน Oscilloscope และครั้งที่สอง เป็นการตรวจเยี่ยมการฝึกงานจากอาจารย์

1.4 ผู้ควบคุมดูแลการฝึกงาน

ชื่อ : นาย ศักดิ์รพี เต็มศิริพันธุ์

ตำแหน่ง : Hardware Developer

อีเมล : sakrappee@ndrsolution.com

บทที่ 2 รายละเอียดของหน่วยงานที่เข้ามิถุน

2.1 สถานที่ตั้งและข้อมูลการติดต่อ

สำนักงานใหญ่

เลขที่ 128/409 อาคารพญาไทพลาซ่า ชั้นที่ 37 ถนนพญาไท

แขวงหุ่งพญาไท เขตราชเทวี กรุงเทพมหานคร 10400

2.2 ประวัติโดยย่อ

บริษัท เอ็นดีอาร์ โซลูชัน (ประเทศไทย) จำกัด ได้ก่อตั้งขึ้นเมื่อวันพุธที่ 17 มิถุนายน พ.ศ. 2553 โดยเป็นบริษัทร่วมทุนกับทาง NDR Co., Ltd. (Japan) ทุนจดทะเบียน 10,000,000 บาท ประกอบกิจการด้านวิศวกรรมอิเล็กทรอนิกส์และคอมพิวเตอร์ หมวดธุรกิจ : กิจกรรมงานวิศวกรรมและการให้คำปรึกษาทางด้านเทคนิค ที่เกี่ยวข้อง

2.3 ลักษณะงานโดยรวม

- ให้บริการทางด้านรับจ้างออกแบบพัฒนา ด้านวิศวกรรมอิเล็กทรอนิกส์และคอมพิวเตอร์
- รับจ้างออกแบบพัฒนา ชาร์ดแวร์และระบบสมองกลฝังตัว (Embedded System) ให้กับลูกค้าที่อยู่ทั่วไปและต่างประเทศ
- บริการทางด้านระบบเทคโนโลยีสารสนเทศให้กับกลุ่มธุรกิจทางด้านพลังงาน
- สร้างความร่วมมือกับสถานศึกษาในการส่งเสริมพัฒนาระบบสมองกลฝังตัวในประเทศไทย

2.4 ระบบบริหาร

แบ่งออกเป็น 3 Business Unit ได้แก่

Design & Development Center (DDC)

Energy Management Solution (EMS)

Factory Automation Solution (FA)

บทที่ 3 รายละเอียดงาน

3.1 การใช้งาน ARROW DECA (MAX10)

DECA คือ บอร์ดสำหรับการพัฒนาและการทดลองทางวิศวกรรมที่ต้องการออกแบบจรดิจิทัลและระบบสมองกลฝังตัว ซึ่งประกอบด้วยองค์ประกอบต่างๆ เช่น

1. FPGA Device: ใช้ชิป MAX10 FPGA จาก Altera
2. On-board Memory: มีหน่วยความจำบนบอร์ดทั้ง SDRAM และ Flash
3. I/O Pins, Analog-to-Digital Converter (ADC)
4. Communication Interfaces: รองรับการเชื่อมต่อแบบ UART, SPI, I2C และอื่น ๆ
5. Development Tools: รองรับการพัฒนาด้วยเครื่องมือจาก Intel เช่น Quartus Prime และ Nios II EDS

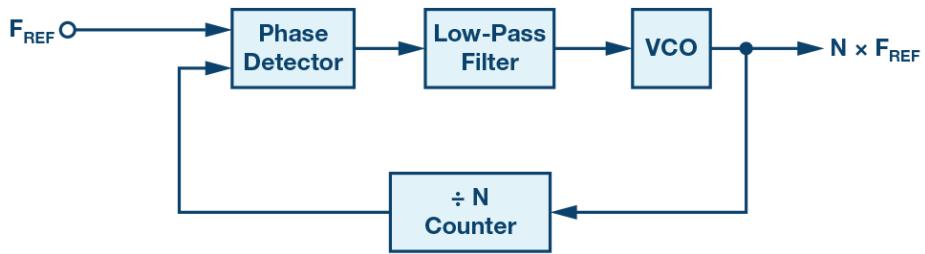
3.1.1 เรียนรู้การสร้างโปรเจค VHDL เพื่อออกแบบจรดิจิทัลบน MAX10 FPGA

รายละเอียดของงานที่ทำ : เรียนรู้หลักการเขียน VHDL ด้วยการออกแบบทั้งหมด 4 โปรเจค ได้แก่

1. การใช้ Slide switch บนบอร์ดในการควบคุมการเปิด-ปิดของ LED
2. การควบคุมให้ LED เปิด-ปิดทุกๆ 1 วินาที
3. การควบคุมให้ LED เปิด-ปิด โดยความถี่สามารถเปลี่ยนได้จาก Tact switch
4. การสร้าง Counter 8 bits โดยใช้ Tact switch ในการนับเพิ่มหรือลด และแสดงค่าผ่าน LED (ระยะเวลา 2 สัปดาห์)

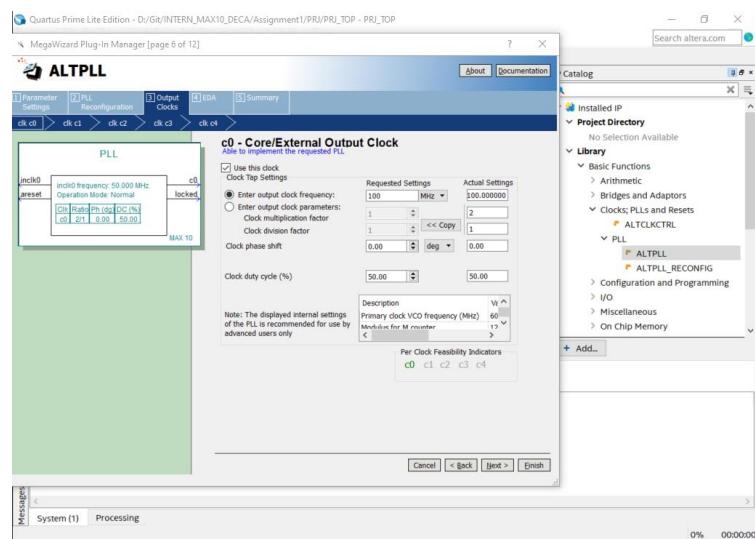
ความรู้พื้นฐานที่ใช้ : FPGA คือ ชิปที่ภายในประกอบด้วย Logic Gate จำนวนมาก ซึ่งสามารถโปรแกรมลงไปเพื่อให้ Logic Gate ดังกล่าวเข้ามาร่วมกันตามที่ต้องการได้ หมายความว่า ก็สามารถสร้างวงจรดิจิทัลขึ้นมาได้ตามที่โปรแกรมลงไว้และทุกวงจรจะทำงานพร้อมกันหมด นอกจากนี้ยังสามารถโปรแกรมใหม่ได้เรื่อยๆ จึงเป็นอุปกรณ์ที่มักนำมาใช้ในการออกแบบและทดสอบเทคโนโลยีใหม่ๆ

PLL คือ วงจรที่ใช้ในการสร้างสัญญาณที่มีความถี่ออกต่างไปจากความถี่ที่รับเข้ามาแต่ยังคงรักษาเฟสให้ตรงกัน ซึ่งประกอบด้วย Phase Detector เพื่อเปรียบเทียบสัญญาณออกกับสัญญาณเข้า Low-Pass Filter (LPF) สำหรับกรอง noise ส่วน VCO ทำหน้าที่สร้างสัญญาณที่ความถี่สามารถเปลี่ยนแปลงได้ตามแรงดันไฟฟ้าที่ได้รับจาก LPF ซึ่งมาจากสัญญาณความแตกต่างของเฟสที่ตรวจจับได้จาก Phase Detector



รูปที่ 1 Block Diagram ของวงจร PLL

โดยบน Quartus II ซึ่งเป็นซอฟต์แวร์ที่พัฒนาโดย Altera เพื่อใช้ในการออกแบบและจำลองวงจรดิจิทัล สำหรับ FPGA สามารถใช้งาน PLL ได้จาก IP Catalog ด้วยการกำหนดค่าพารามิเตอร์บน Quartus II และโปรแกรมลงบน FPGA



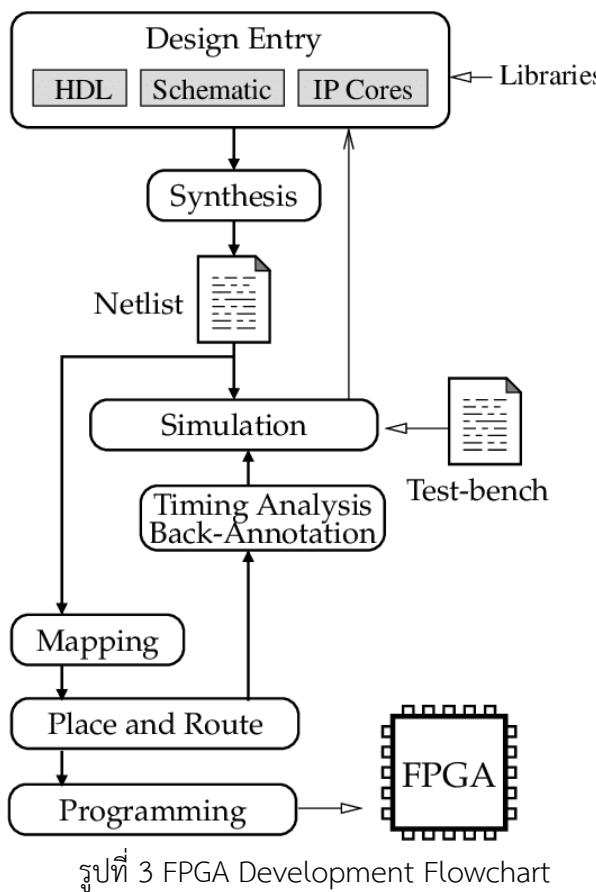
รูปที่ 2 หน้าต่างที่ใช้ในการกำหนดค่าพารามิเตอร์ของ PLL บน Quartus II

IP (Intellectual Property) คือ โมดูลที่มีการพัฒนาและทดสอบมาแล้ว ซึ่งสามารถนำไปใช้ในการออกแบบ วงจรหรือระบบได้โดยไม่ต้องออกแบบใหม่ สำหรับ Quartus II จะมี IP Cores ซึ่งคือการออกแบบ硬件ที่สามารถใช้ใน FPGA ไว้ให้ใช้งานอยู่หลายรูปแบบ

FPGA Development Flow คือ กระบวนการในการออกแบบ พัฒนา ทดสอบ และโปรแกรม FPGA เพื่อให้ตรงตามความต้องการของผู้ใช้งาน ประกอบด้วยหลายขั้นตอน ดังนี้

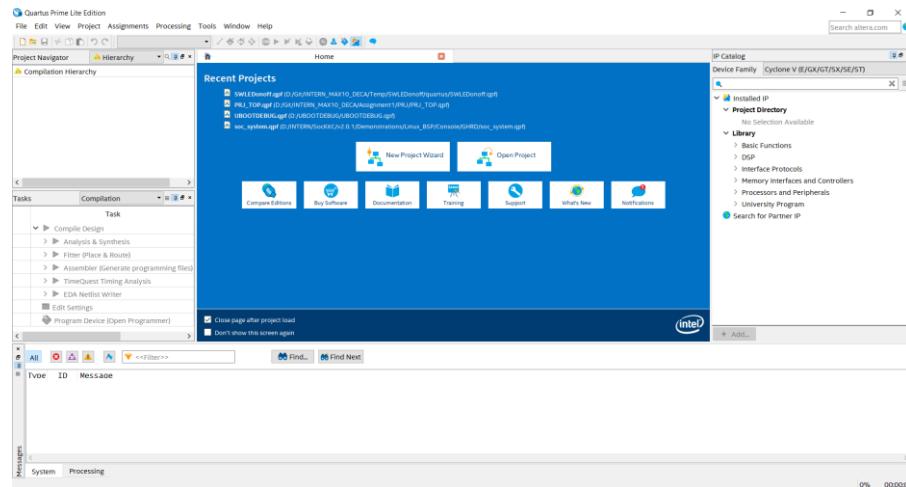
1. Design เป็นการกำหนดรายละเอียดของวงจรที่ต้องการ ซึ่งจะบ่งบอกถึงส่วนประกอบต่างๆ ของวงจร
2. Synthesis ขั้นตอนการสังเคราะห์วงจร hardware ที่เกิดจากโค้ด HDL ประกอบด้วยการแปลงโค้ดให้เป็นวงจรโลジกิค แล้วทำการเทียบวงจรเข้ากับ FPGA (mapping) สำหรับใช้ในการทำ Place & Route ในการสังเคราะห์จะมีการเลือก FPGA เป้าหมายที่ใช้สำหรับการสังเคราะห์วงจร
3. Simulation เป็นขั้นตอนการจำลองการทำงานของโค้ด VHDL หรือ Verilog ในระดับ RTL (Register Transfer Level) ในการจำลองการทำงาน อาจจะใช้การเขียน testbench เพื่อสร้างสัญญาณป้อนเข้าไปยังวงจรที่ต้องการทดสอบ

4. Place & Route ซึ่งเป็นการนำเอาวงจรโลจิกมาจัดวางบน FPGA จากนั้นซอฟต์แวร์จะทำการคำนวณค่าความหน่วงของส่วนโลจิก (Logic Delay) พร้อมทั้งค่าความหน่วงของสายสัญญาณ (Interconnect Delay) จริงออกมา ในรูปแบบของไฟล์ SDF (Standard Delay Format) ซึ่งจะนำไปใช้สำหรับการจำลองการทำงานทางเวลา (Timing Simulation)
5. Timing Analysis เป็นการจำลองการทำงานของวงจร โดยมีการคิดค่าความหน่วงของเกต (Gate Delay) และความหน่วงการเชื่อมต่อที่เกิดขึ้นจริง (Interconnect Delay) ในการกำหนดการวางแผนโลจิกและการเชื่อมต่อสัญญาณใน FPGA
6. Programming ขั้นตอนการโปรแกรม Configuration File ลงบน FPGA ใช้วิธีโปรแกรมไฟล์บิตลงบนหน่วยเก็บความจำแบบแฟลช (Flash Memory) หรือ EEPROM หรือ EEPROM



รูปที่ 3 FPGA Development Flowchart

ขั้นตอนการทำงาน : การใช้ Slide switch บันบอร์ดในการควบคุมการเปิด-ปิดของ LED
เริ่มจากการสร้างโปรเจกบน Quartus II และสร้างไฟล์ VHDL ภายในโปรเจก



รูปที่ 4 โปรแกรม Quartus II

แก้ไขไฟล์ดังกล่าวบนโปรแกรม Notepad++ โดยออกแบบให้รับ Input จาก Slide switch, Tact switch และ สัญญาณ Clock 50 MHz และส่ง Output ออกยัง LED โดยออกแบบให้ LED เป็น HIGH หรือ LOW ตาม Slide switch และ หยุดการทำงานถ้ารีเซ็ต Tact switch

```

D:\Git\INTERN_MAX10_DECA\Temp\SWLEDonoff\hdl\SWLEDonoff.vhd - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
SWLEDonoff.vhd

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4  use IEEE.STD.TEXT.all;
5
6  Entity SWLEDonoff Is
7    Port (
8      -- Input from Switch & Reset Button
9      SW0      : in std_logic;
10     Rst      : in std_logic; -- Active-Low reset
11     Clk50   : in std_logic;
12
13     -- LED Output
14     LED0    : out std_logic
15   );
16 End Entity SWLEDonoff;
17
18 Architecture rtl Of SWLEDonoff Is
19
20   -- Signal declaration
21   signal LED0_State : std_logic := '0'; -- Internal signal to maintain the LED state
22   signal Stop        : std_logic := '0'; -- Internal signal to indicate reset was pressed
23
24 Begin
25   -- Output assignment
26   LED0 <= not LED0_State; -- Assign the internal LED state to the output
27
28   -- Process to control the LED based on the switch input and reset
29   u_LED0 : Process(Clk50) Is
30     Begin
31       if rising_edge(Clk50) then
32         if Rst = '0' then
33           Stop <= '1'; -- Latch the stop condition when reset is pressed
34         elsif Stop = '0' then
35           LED0_State <= SW0; -- Control LED based on the switch when reset is not pressed
36         else
37           LED0_State <= '0'; -- Turn off the LED if stop condition is latched
38         end if;
39       End process;
40
41   End Architecture rtl;
42

```

รูปที่ 5 VHDL code ในโปรเจกที่ 1 ซึ่งแก้ไขบนโปรแกรม Notepad++

จากนั้นสามารถ Compile ไฟล์ทั้งหมดในโปรเจคบน Quartus II แล้วโปรแกรมลงบน FPGA ได้
สำหรับโปรเจคที่เหลือจะทำการสร้างโปรเจคและโปรแกรมลงบน FPGA ในลักษณะเดียวกัน จึงจะแสดงรายละเอียดเฉพาะการออกแบบ แหล่ง ลักษณะของโค้ด VHDL ที่เขียน

การควบคุมให้ LED เปิด-ปิดทุกๆ 1 วินาที

ออกแบบให้ LED เปิด-ปิดทุกๆ 1 วินาที โดยอาศัยการนับสัญญาณ Clock ซึ่งมีความถี่ 50 MHz จึงสร้าง Counter ขนาด 26 bit ในการนับ 50 ล้านครั้ง เมื่อนับครบก็จะ Toggle สัญญาณ LED สำหรับการ Reset (Rst) จะทำให้ LED ดับขณะที่กำลัง Reset อุ่น

```

6      Entity LEDonoff1s Is
7      Port (
8          -- Clk 50 MHz
9          Clk50    : in  std_logic;
10         Rst      : in  std_logic;
11         -- LED Output
12         LED0     : out std_logic
13     );
14 End Entity LEDonoff1s;
```

รูปที่ 6 ส่วน Entity ของโค้ด VHDL ในโปรเจคที่ 2 ซึ่งใช้ในการกำหนดสัญญาณ Input/Output

การควบคุมให้ LED เปิด-ปิด โดยความถี่สามารถเปลี่ยนได้จาก Tact switch

ออกแบบให้การกด Tact Switch (TSW) ครั้งแรก LED จะเปิด-ปิดทุกๆ 1 วินาที โดยอาศัยการนับสัญญาณ Clock ซึ่งมีความถี่ 50 MHz จึงสร้าง Counter ขนาด 26 bit ในการนับ 50 ล้านครั้ง เมื่อนับครบก็จะ Toggle สัญญาณ LED แต่หลังจากการครั้งที่สอง จะเปลี่ยนเป็น เปิด-ปิดทุกๆ 0.5 วินาทีแทน โดยให้ Counter นับแค่ 25 ล้านครั้ง แล้ว Toggle LED การกดครั้งที่สาม LED จะดับทั้งหมดและการกดครั้งถัดไป LED จะเปิด-ปิดทุกๆ 1 วินาที วนไปเรื่อยๆ สำหรับการ Reset (Rst) จะทำให้ LED ดับขณะที่กำลัง Reset อุ่น

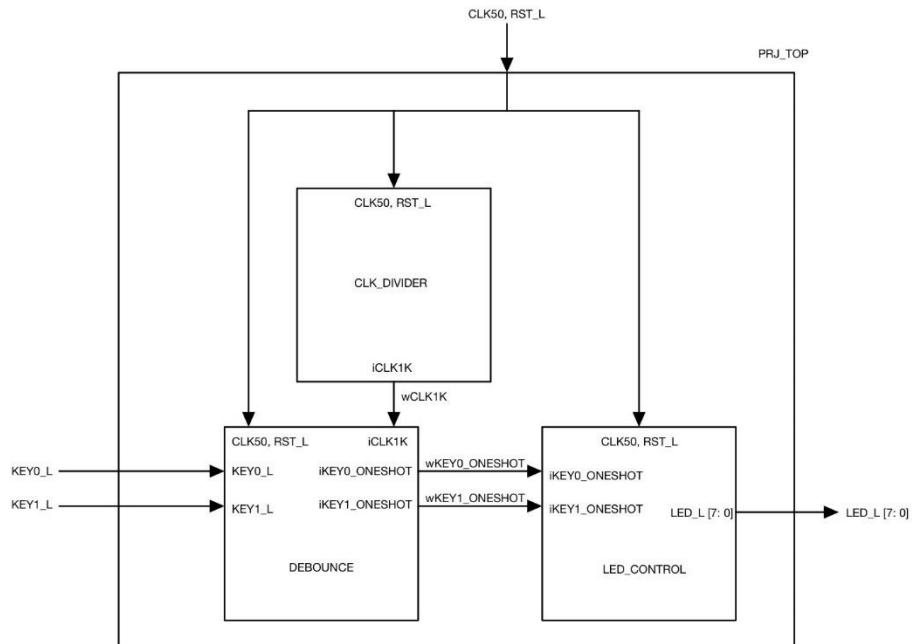
```

6      Entity LEDonoffXXs Is
7      Port (
8          -- Clk 50 MHz
9          Clk50    : in  std_logic;
10         Rst      : in  std_logic;
11         -- Input KEY[0] & KEY[1]
12         TSW      : in  std_logic;
13         -- LED Output
14         LED0     : out std_logic
15     );
16 End Entity LEDonoffXXs;
```

รูปที่ 7 ส่วน Entity ของโค้ด VHDL ในโปรเจคที่ 3

การสร้าง Counter 8 bits โดยใช้ Tact switch ในการนับเพิ่มหรือลด และแสดงค่าผ่าน LED
 ออกแบบให้การกด Tact switch ปุ่มแรก (KEY0) จะทำให้ Counter 8 bit นับขึ้นครั้งละ 1 และปุ่มที่สอง (KEY1) จะทำให้ Counter 8 bit นับลงครั้งละ 1 โดยแสดงค่าอ้อมมาผ่าน LED ในรูปแบบ Binary กล่าวคือให้ LED ดวงแรก (LEDO) แทน LSB เรียงลำดับไปจนถึง LED ดวงสุดท้าย (LED7) แทน MSB โดยถ้า LED ติด แทน bit 1 ตั้งแต่ bit 0 การ Reset จะทำให้ Counter กลับไปมีค่าเป็น 0 สำหรับการเขียน VHDL จะประกอบไปด้วย 4 ไฟล์ ได้แก่

1. CLK_DIVIDER ใช้สร้างสัญญาณนาฬิกาที่ความถี่ 1 kHz โดยการใช้ Counter นับสัญญาณนาฬิกาที่ความถี่ 50 MHz 25000 ครั้ง แล้ว Toggle
2. DEBOUNCE ทำการ Debounce บุ่ม KEY0 และ KEY1 โดยการใช้ Counter นับสัญญาณบุ่มตังกล่าวตามขอบขาขึ้นของสัญญาณนาฬิกาความถี่ 1 kHz (1 ms) โดยเปรียบเทียบกับสัญญาณก่อนหน้าที่เก็บใน Register โดย Counter นับขึ้นเมื่อสัญญาณเหมือนเดิม และเมื่อนับได้ถึง 10 รอบ จึงถือว่าสัญญาณนั้นเกิดขึ้นจริง เพื่อแก้ปัญหาการแก่วงของสัญญาณที่รับมาจากปุ่มกดที่ผิวสัมผัสยังไม่แนบสนิทในช่วงแรกที่กด
3. LED_CONTROL ควบคุมการแสดงผลของ LED โดยแสดงตาม Counter ที่นับขึ้นเมื่อ KEY1 ถูกกด และนับลงเมื่อ KEY0 ถูกกด
4. PRJ_TOP เป็น Entity ระดับบนสุด ที่จะเชื่อมต่อโดยตรงกับ Pins บนบอร์ด และรวมการออกแบบข้างต้นไว้และกำหนดการเชื่อมต่อกันภายใน



รูปที่ 8 Block Diagram ของการออกแบบโปรเจคที่ 4

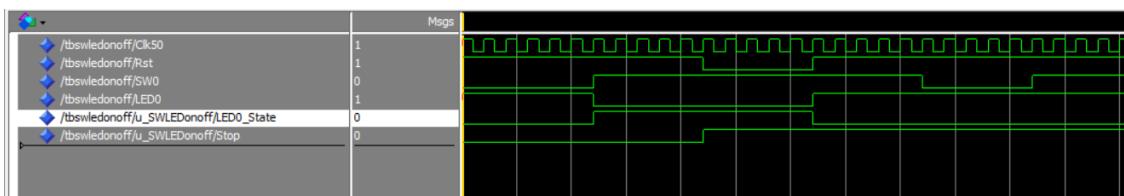
ในโปรเจคนี้ได้เรียนรู้การตั้งชื่อสัญญาณตามข้อกำหนดภายในบริษัท โดยมีรายละเอียดดังนี้

Signal name start with

- i : Module to Module Signal
- w : Module Internal Signal & Register
- k : Constant
- s : State
- v : variable

ผลลัพธ์ : การใช้ Slide switch บนบอร์ดในการควบคุมการเปิด-ปิดของ LED

จากผลการ Simulation จะพบว่า LED แสดงค่าตรงข้ามกับ Slide switch (SW0) เมื่อจาก LED เป็น Active Low และหลังจากระบบถูก Reset (Rst = '0') LED จะดับตลอดเวลา



รูปที่ 9 ผลจากการ Simulation โปรเจคที่ 1

การควบคุมให้ LED เปิด-ปิดทุกๆ 1 วินาที

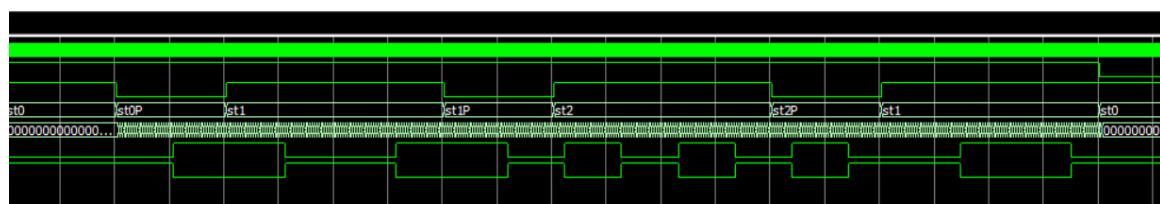
จากผลการ Simulation จะพบว่า LED มีการเปลี่ยนค่าระหว่าง '0' กับ '1' ทุกๆ 1 วินาที



รูปที่ 10 ผลจากการ Simulation โปรเจคที่ 2

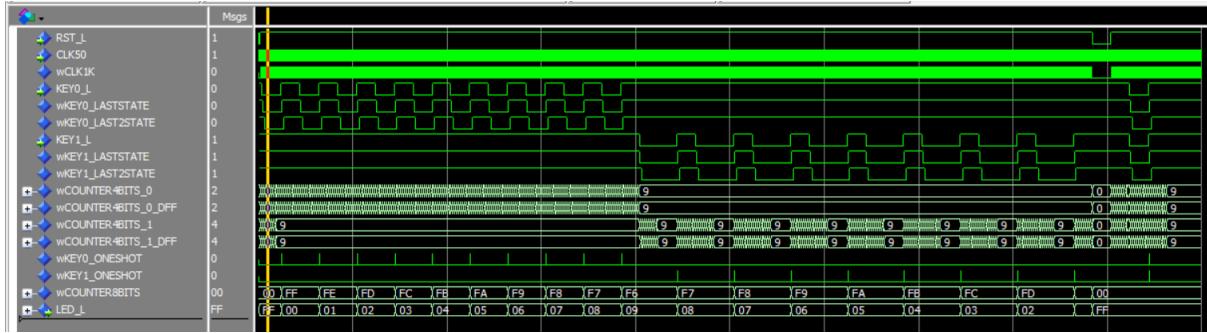
การควบคุมให้ LED เปิด-ปิด โดยความถี่สามารถเปลี่ยนได้จาก Tact switch

จากผลการ Simulation จะพบว่า LED (สัญญาณในบรรทัดสุดท้าย) มีการเปลี่ยนความถี่ในการเปิด-ปิด หลังจากกด Tact switch (สัญญาณในบรรทัดที่ 3) ตามที่ออกแบบไว้



รูปที่ 11 ผลจากการ Simulation โปรเจคที่ 3

การสร้าง Counter 8 bits โดยใช้ Tact switch ในการนับเพิ่มหรือลด และ แสดงค่าผ่าน LED
 จากผลการ Simulation จะพบว่า LED แสดงค่าตาม Counter 8 bit (แสดงค่าตรงข้ามเนื่องจาก LED Active Low) และนับขึ้นเมื่อ KEY0 ถูกกด นับลงเมื่อ KEY1 ถูกกด



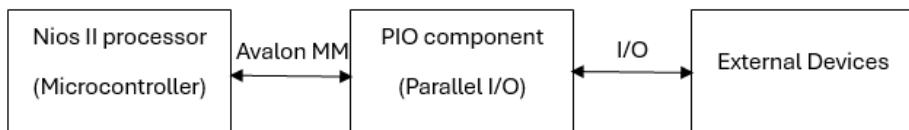
รูปที่ 12 ผลจากการ Simulation โปรเจกต์ที่ 4

สำหรับวิธีการแสดงผลลัพธ์ที่ได้หลังจากโปรแกรมลงบนบอร์ดสามารถดูได้ในภาคผนวก

3.1.2 คึกษาและใช้งาน NIOS II SOFT CPU

รายละเอียดของงานที่ทำ : เรียนรู้การใช้ NIOS II ที่สร้างภายใน FPGA ในการรับและส่งข้อมูลไปยัง Console และ Pins บนบอร์ด (ระยะเวลา 1 สัปดาห์)

ความรู้พื้นฐานที่ใช้ : Nios II คือ Soft Processor Core ที่ออกแบบโดย Intel โดยสามารถสร้างจาก logics ภายใน FPGA ซึ่งหากต้องการใช้ NIOS II ในการส่งหรือรับสัญญาณจากอุปกรณ์ภายนอก เช่น LED, Switch จะมีแผนภาพการเชื่อมต่ออย่างง่าย ดังนี้



รูปที่ 13 Block Diagram การควบคุมอุปกรณ์ภายนอกด้วย NIOS II

โดยที่ Avalon Memory-Mapped Interface (Avalon-MM) ใช้ในการสื่อสารระหว่าง master และ slave ผ่านการ mapping memory คือ การสื่อสารระหว่าง master และ slave ผ่านที่อยู่หน่วยความจำ (memory address) สามารถใช้ในการเข้าถึงข้อมูล register-mapped peripherals และอื่นๆ มีสัญญาณควบคุมเช่น read, write, address, และ data โดย NIOS II ใช้ Avalon-MM Interface ดังนี้
 Data Master : ช่องทางการเข้าถึงข้อมูล (Data Path) ของ Nios II Processor ซึ่งใช้ในการอ่านและเขียน ข้อมูลไปยังอุปกรณ์หรือหน่วยความจำอื่น ๆ บน FPGA เช่น การอ่านข้อมูลจากหน่วยความจำหรือการส่ง ข้อมูลไปยังพอร์ต GPIO

Instruction Master : ช่องทางการเข้าถึงคำสั่ง (Instruction Path) ของ Nios II Processor ซึ่งใช้ในการดึงคำสั่งจากหน่วยความจำเพื่อประมวลผล เช่น การดึงคำสั่งโปรแกรมจาก Flash Memory เพื่อให้ CPU ประมวลผล

จากนั้นจึงศึกษาโครงสร้างภายในพอร์ต I/O ของ Processor โดยศึกษาจาก ATmega328p

1. โครงสร้างภายในของพอร์ต I/O บน ATmega328p

แต่ละพอร์ต I/O บน ATmega328p มีโครงสร้างภายในที่ประกอบด้วยวงจรควบคุมหลายส่วน เช่น:

Data Direction Register (DDR): ควบคุมพอร์ตว่าเป็น input หรือ output

PORT Register: ใช้ในการตั้งค่าเอาต์พุต ถ้าพอร์ตถูกตั้งเป็น output

PIN Register: ใช้ในการอ่านค่าของพอร์ต ถ้าพอร์ตถูกตั้งเป็น input

2. การตั้งค่าพอร์ตเป็น output

พอร์ต PBO ตั้งค่าให้เป็นเอาต์พุต โดยการตั้งค่า DDRB (Data Direction Register for Port B) ค่า Register

DDRB ในตำแหน่งบิต 0 (PBO) จะถูกตั้งค่าเป็น 1

3. การส่งสัญญาณจากพอร์ตเอาต์พุต

หลังจากที่พอร์ต PBO ถูกตั้งค่าเป็นเอาต์พุตแล้ว การส่งสัญญาณไป PBO จะเกี่ยวข้องกับการตั้งค่า PORTB

Register ในตำแหน่งบิต 0:

ถ้าเราตั้งค่า PORTB Register บิต 0 เป็น 1 วงจรภายในจะเชื่อมต่อขา PBO กับ Vcc

ถ้าเราตั้งค่า PORTB Register บิต 0 เป็น 0 วงจรภายในจะเชื่อมต่อขา PBO กับ GND

4. การกำหนดให้ output ที่ PBO เป็น HIGH/LOW

DDRB : Address 0x04 (0x24)

- 0x04 : ที่อยู่ของ DDRB ใน Data Space
- 0x24 : ที่อยู่ของ DDRB ใน I/O Memory Space

PORTB : Address 0x05 (0x25)

- 0x05 : ที่อยู่ของ PORTB ใน Data Space
- 0x25 : ที่อยู่ของ PORTB ใน I/O Memory Space

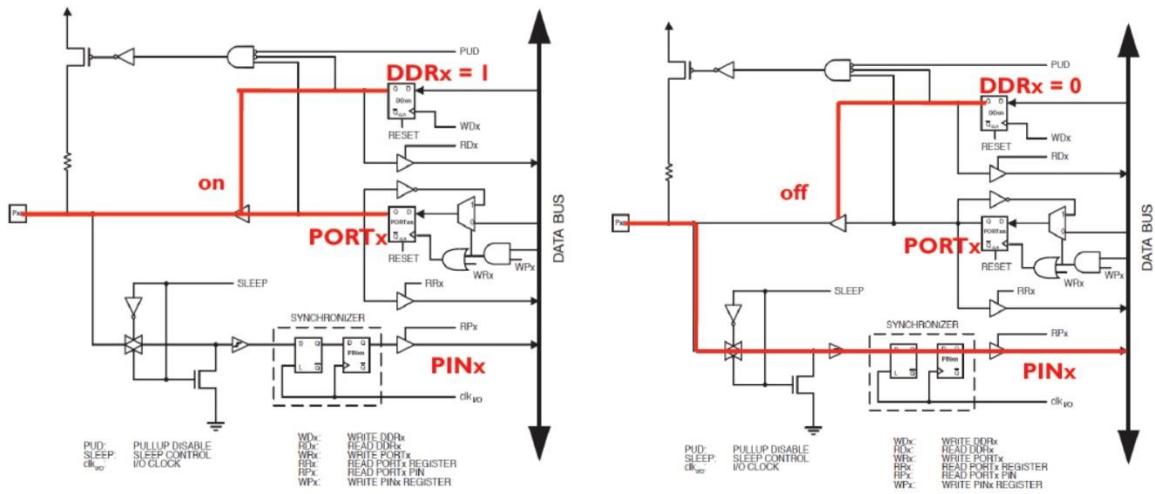
Data Space : ใช้สำหรับเก็บข้อมูลที่โปรแกรมต้องการในการประมวลผล เช่น ตัวแปร, ข้อมูลชั่วคราว เป็นต้น

I/O Memory Space : ใช้สำหรับเก็บข้อมูลที่ใช้ในการสื่อสารกับอุปกรณ์ภายนอก เช่น การควบคุมพอร์ต I/O, การสื่อสารผ่าน UART, SPI, I2C เป็นต้น

เมื่อเขียนโปรแกรมสำหรับ ATmega328P จะใช้ทั้งสองที่อยู่เพื่อเข้าถึง โดยไม่มีผลต่อการทำงานของโปรแกรม

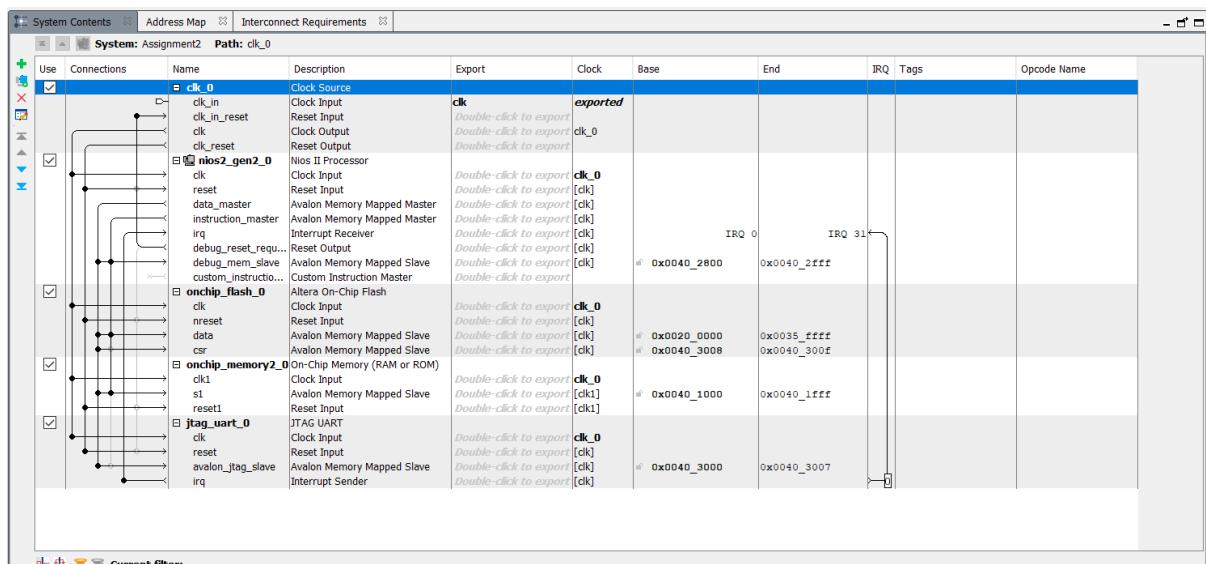
DDRB Register Bit0 → ‘1’ (0x04 / 0x24 Bit0 → ‘1’)

PORTB Register Bit0 → ‘1’ (0x05 / 0x25 Bit0 → ‘1’)



รูปที่ 14 โครงสร้างภายใน ATmega328p เมื่อ PORT เป็น output (ซ้าย) และ input (ขวา)

ขั้นตอนการทำงาน : การสร้างโปรเจคที่รัน Software บน NIOS II เพื่อส่งข้อมูลไปแสดงบน Console ผ่าน USB Blaster JTAG เริ่มจากการออกแบบระบบที่ประกอบด้วย NIOS II บน Platform Designer ของ Quartus (Qsys system) ซึ่งประกอบด้วยองค์ประกอบดังนี้ NIOS II, Onchip memory, JTAG UART



รูปที่ 15 การออกแบบระบบบน Platform Designer สำหรับรัน Software บน NIOS II

จากนั้นสร้าง software ที่จะรันบน Eclipse ของ Quartus เป็นไฟล์ .HEX

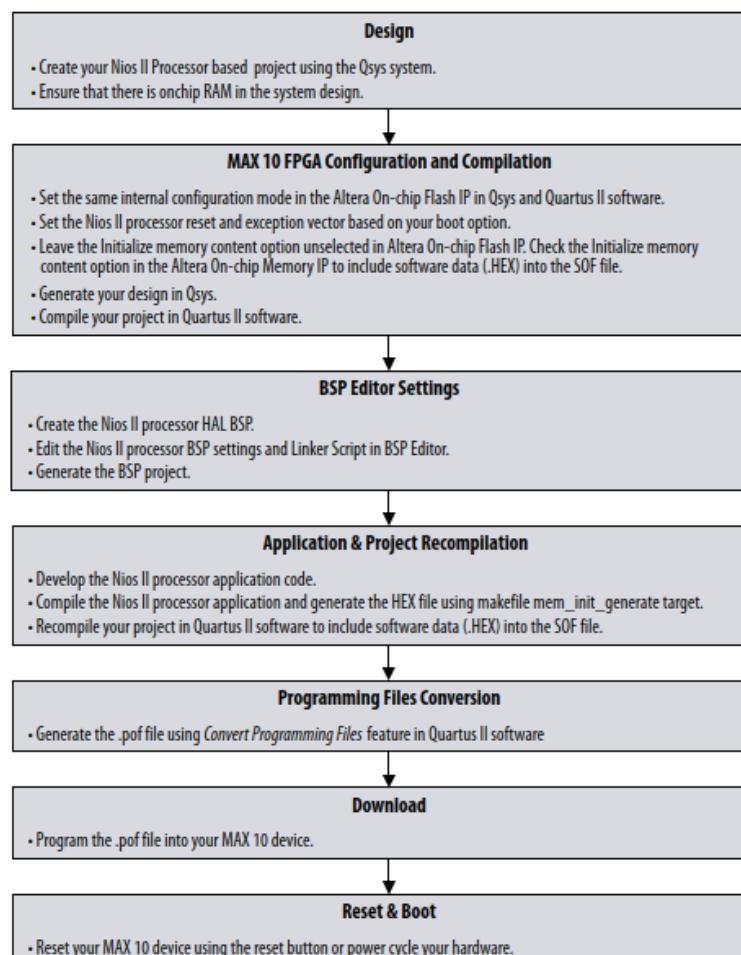
```

2* * "Small Hello World" example.
80
81 #include "sys/alt_stdio.h"
82
83 int main()
84 {
85 alt_putstr("Hello World\n");
86
87 /* Event loop never exits. */
88 while (1);
89
90 return 0;
91 }
92

```

รูปที่ 16 การสร้าง Software บน Eclipse

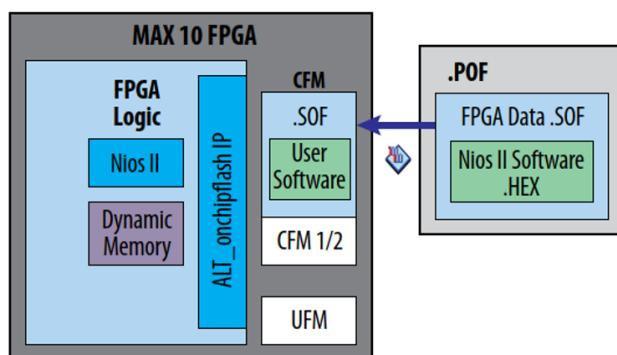
แล้ว compile รวมกับระบบที่สร้างมาก่อนหน้านี้ เพื่อให้ได้ไฟล์ .POF สำหรับโปรแกรมลงบน FPGA วิธีการอย่างละเอียดในแต่ละขั้นตอนแสดงดังนี้



รูปที่ 17 Flow chart แสดงขั้นตอนการรัน software บน NIOS II

ในโปรเจคนี้จะใช้ Boot Option 3 ซึ่ง Boot Option สำหรับ NIOS II หมายถึงวิธีการที่ระบบ NIOS II จะเริ่มต้นการทำงาน (boot) จากแหล่งข้อมูลหรือหน่วยความจำที่กำหนดไว้ Boot Option 3 คือ การที่ CFM จะทำการ Initialized ในขณะที่ FPGA กำลัง Config อญญาติให้หลังจาก reset แล้ว Nios II อยู่ในสภาพพร้อมทำงาน ซึ่ง Boot Option นี้ จะไม่รองรับการทำงานแบบ Dual Boot (มี 2 Image ใน FPGA 1 ตัว) Software ทั้งหมดจะถูกรันบน OCRAM เท่านั้น ในขั้นตอนการอุปกรณ์จะต้อง check Initialize flash content ใน RAM หรือ หากมีการแก้ไข Software ใหม่ จะต้องทำการ Compile FPGA ทุกครั้ง ไม่เหมาะสำหรับการ Debug เพราะใช้เวลา Compile ค่อนข้างนาน

"image" หมายถึงไฟล์หรือโค้ดที่ประกอบไปด้วยโปรแกรมหรือ Software ที่ต้องการรันบน Nios II



รูปที่ 18 แผนภาพการโปรแกรม .POF ลงบน FPGA

ผลที่ได้คือสามารถรันโปรแกรมบน NIOS II ได้และส่งข้อความ “Hello World” มาแสดงบน Console ได้

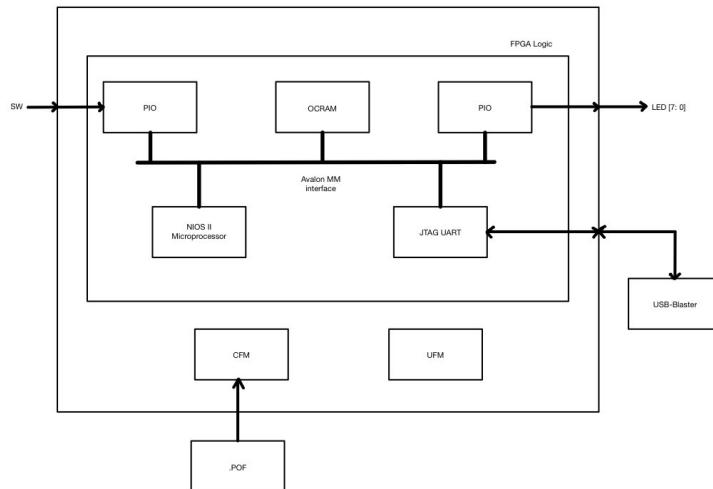
```
-----
Altera Nios2 Command Shell [GCC 4]
Version 17.1, Build 590
-----

bannawich@MSI /cygdrive/c/intelFPGA_lite/17.1
$ nios2-terminal
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "Arrow MAX 10 DECA [USB-1]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Hello World
-----
```

รูปที่ 19 NIOS II Console

ถัดมาจะทำการสร้าง Software ที่รับค่าจาก Switch เพื่อนำไปควบคุม LED บนบอร์ด จึงต้องอุปกรณ์ที่ต้องการจะทำงาน Platform Designer ใหม่ โดยเพิ่ม PIO ซึ่งเป็น component ในการอุปกรณ์ที่ต้องการจะทำงาน Parallel I/O Interfaces มักใช้ใน Nios II เพื่อควบคุมอุปกรณ์ดิจิทัลภายนอก เช่น LED และ Switch



รูปที่ 20 Block Diagram แสดงการเชื่อมต่อระบบ ที่ใช้ NIOS II ควบคุม Switch และ LED

ผลที่ได้คือสามารถเขียน Software ใหม่ โดยถ้าค่าจาก Switch เป็น ‘1’ LED จะกะพริบ ถ้าเป็น ‘0’ LED จะดับ สามารถดูผลลัพธ์ได้จากวิดีโอในภาคผนวก

3.1.3 การสร้าง GPIO IP และ สัญญาณ PWM

รายละเอียดของงานที่ทำ : การสร้าง GPIO IP ขึ้นมาด้วยตัวเอง และ การเพิ่ม GPIO IP ที่สร้างขึ้นนี้เป็น IP ใน Platform Designer บน Quartus จากนั้นสร้าง PWM IP ซึ่งมีลักษณะเหมือน GPIO IP แบบ output ที่สามารถสร้างสัญญาณแบบ PWM ได้ (ระยะเวลา 2 สปเดาท์)

ความรู้พื้นฐานที่ใช้ : GPIO ย่อมาจาก General Purpose Input/Output คือ ขา Input และ Output ใน Microcontroller หรือ Microprocessor ที่สามารถใช้ควบคุมหรือรับข้อมูลจากอุปกรณ์ภายนอกได้ โดย GPIO สามารถกำหนดให้เป็น Input/Output/BiDirection ก็ได้

ใน Nios II ระบบที่ใช้ Avalon Bus จะมีโครงสร้างที่ทุกอุปกรณ์ (เช่น หน่วยความจำ, พอร์ต I/O) จะเชื่อมต่อเข้ากับ Bus เดียวกัน ซึ่ง Nios II จะเป็น Master ที่สามารถเข้าถึง components ต่าง ๆ ที่เป็น Slave ผ่าน Bus นี้ได้ ซึ่ง Avalon Bus จะถูกสร้างขึ้นโดยอัตโนมัติภายใน Platform Designer

สัญญาณที่ใช้ใน Avalon Bus

address: ที่อยู่ของข้อมูลที่ต้องการอ่านหรือเขียน

read: สัญญาณที่บ่งบอกว่า Master ต้องการอ่านข้อมูลจาก Slave

write: สัญญาณที่บ่งบอกว่า Master ต้องการเขียนข้อมูลไปยัง Slave

writedata: ข้อมูลที่ต้องการเขียนไปยัง Slave

readdata: ข้อมูลที่ Slave ส่งกลับไปยัง Master เมื่อมีการอ่าน

waitrequest: สัญญาณจาก Slave ที่บอก Master ให้รอเนื่องจากไม่พร้อมที่จะรับหรือส่งข้อมูล

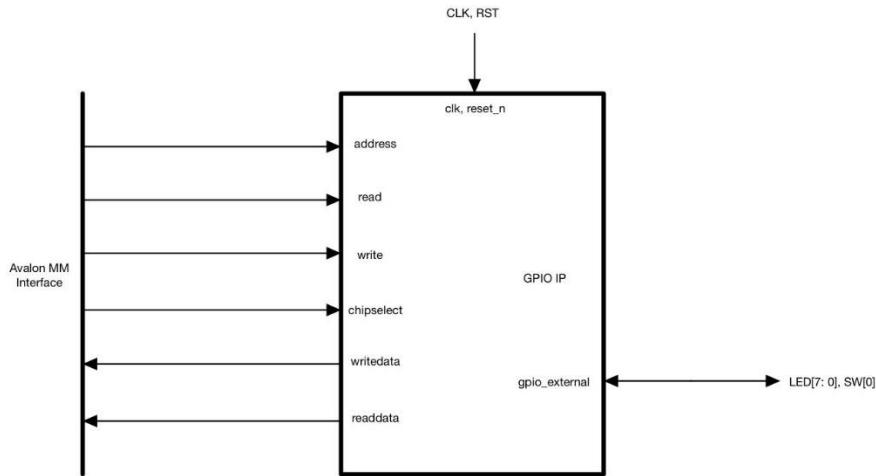
byteenable: สัญญาณที่ระบุว่า byte ใดในบล็อกข้อมูลที่ถูกเปิดใช้งาน (valid) ซึ่งช่วยในการจัดการการถ่ายโอนข้อมูลที่มีขนาดต่าง ๆ

burstcount: ระบุจำนวนของการถ่ายโอนที่ต้องการทำต่อเนื่อง (burst mode)

response: สัญญาณที่บ่งบอกสถานะการตอบสนองจาก Slave เช่น ตอบสนองว่าเป็นการตอบสนองปกติหรือมีข้อผิดพลาด

chipselect: สัญญาณที่ใช้เลือก Slave ตัวใดตัวหนึ่งในระบบ

ขั้นตอนการทำงาน : ออกรูปแบบ GPIO IP ที่จะสร้างเป็น Avalon MM Slave ในระบบ โดยกำหนดสัญญาณ Input/Output ทั้งหมด พร้อมทั้งวาด Block Diagram คร่าวๆ และ ออกรูปแบบการทำงานด้วย Register Map



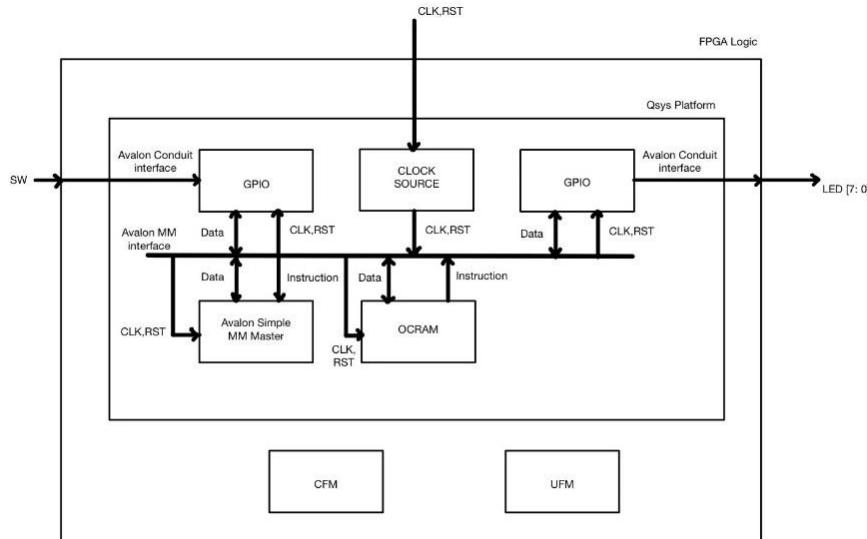
รูปที่ 21 Block Diagram ของ GPIO IP

| Register Map for the GPIO IP | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|------------------|--|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
| Register Name | DATA 0x00 [31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| External Connections | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Bits | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | | | | | | | | | | | | | | | |
| I/O Pins | ● | ● | R/W | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | | | | | | | | | | | | | | | |
| Default values | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | |
| Bits | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | | | | | | | | | | | | | | | |
| I/O Pins | ● | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | | | | | | | | | | | | | | | |
| Default values | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | |
| Bits | Descriptions | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| [31:0] | R | Read the Data value that will store from GPIO External I/O Port when the same bit on DIRECTION Register sets to '0' | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | W | Write the Data value to drive on GPIO External I/O Port when the same bit on DIRECTION Register sets to '1' | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ● * Available for used as an External I/O Port | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Register Name | DIRECTION 0x04 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| External Connections | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Bits | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | | | | | | | | | | | | | | | |
| I/O Pins | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | | | | | | | | | | | | | | | |
| Default values | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | |
| Bits | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | | | | | | | | | | | | | | | |
| I/O Pins | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | | | | | | | | | | | | | | | |
| Default values | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | |
| Bits | Descriptions | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| [31:0] | R | Read the Direction of each GPIO External I/O Port on this Register | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | W | Write the Direction control for each GPIO External I/O Port which a value of '0' sets to be an input and '1' to be an output | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

รูปที่ 22 Register Map ของ GPIO IP

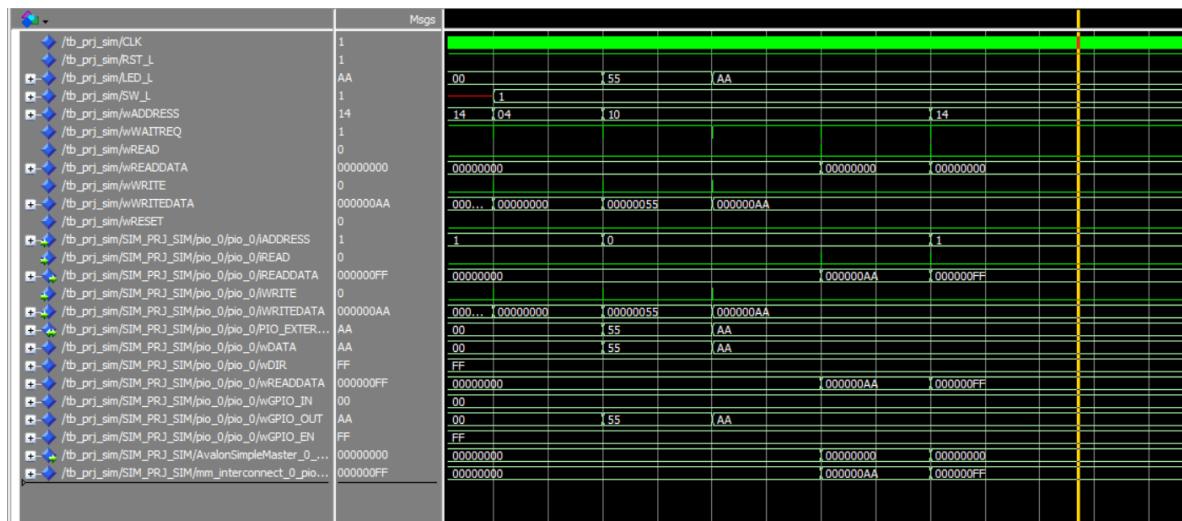
การทำงานของ GPIO IP จะเป็นแบบ BiDirection กล่าวคือสามารถกำหนดให้เป็น Port Input หรือ Output ได้จากค่าที่เขียนลงในช่อง DIRECTION Register ดังนั้น Process การทำงานก็จะแบ่งเป็น Write กับ Read โดยจะมีการพิจารณา Address เพื่อเลือก Write/Read ระหว่าง DATA/DIRECTION ตามที่ได้ระบุไว้ใน Register Map

ผลลัพธ์ : แทนที่ PIO ของระบบในข้อ 3.1.2 ด้วย GPIO IP ที่สร้างขึ้นมา จะได้ระบบที่มีการเชื่อมต่อ ดังนี้



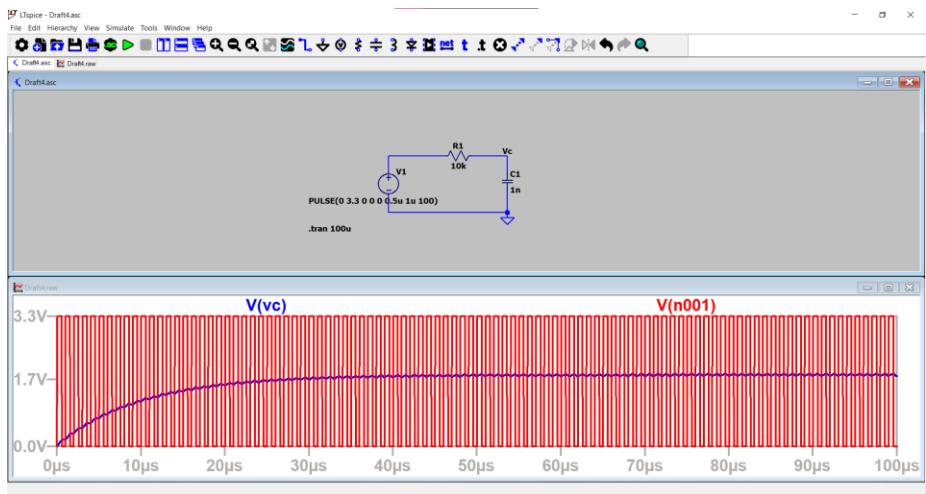
รูปที่ 23 Block Diagram ของระบบที่ใช้งาน GPIO IP

ผลจากการ Simulation พบว่า GPIO IP ที่นี้ในผู้ที่รับค่าจาก Switch และผู้ที่ส่งค่าไปยัง LED สามารถเขียน และอ่านข้อมูลลงบนทั้ง Data และ Direction Register ของแต่ละ Port ได้



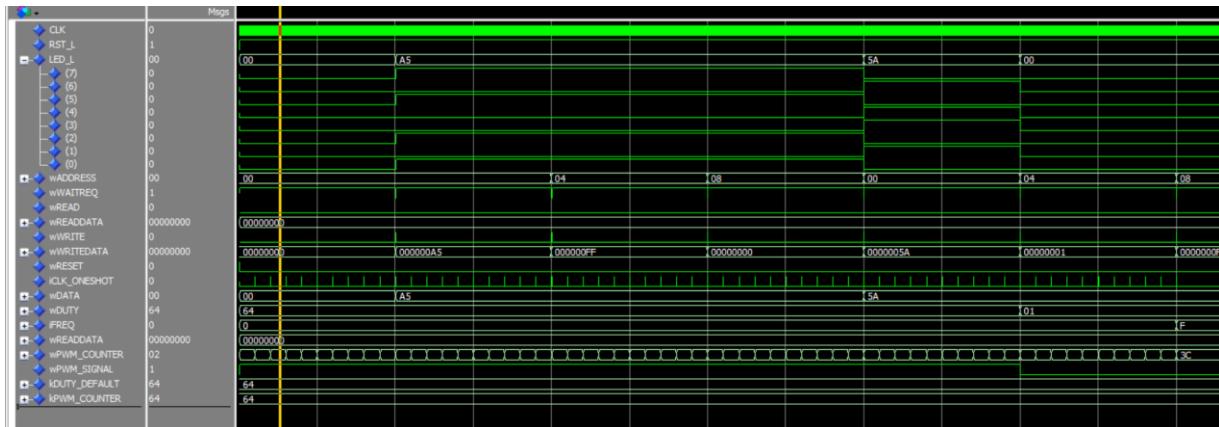
รูปที่ 24 Simulation ของระบบที่ใช้งาน GPIO IP

ต่อมาจะแทนที่ GPIO IP ในผู้ที่ส่งสัญญาณออกไปยัง LED ในรูปแบบ PWM ที่กำหนด Duty cycle และ Frequency ได้ โดย PWM (Pulse Width Modulation) เป็นเทคนิคที่ใช้ในการควบคุมกำลังหรือแรงดันในระบบดิจิทัล โดยการเปลี่ยนแปลงความกว้างของพัลส์สัญญาณดิจิตอล Frequency ของสัญญาณ PWM คือจำนวนครั้งที่สัญญาณ High และ Low ในหนึ่งวินาที Duty cycle ของ PWM คือสัดส่วนเวลาที่สัญญาณอยู่ในสถานะ High ต่อเวลาทั้งหมดของหนึ่งรอบสัญญาณ คิดเป็นเปอร์เซ็นต์ (%)



รูปที่ 25 จำลองการสร้าง Analog signal จาก PWM ใน LTSpice โดยการใช้ Low-pass Filter

ผลจากการ Simulation พบว่า PWM IP สามารถส่งสัญญาณ PWM ไปยัง LED ได้ กล่าวคือสามารถกำหนด Duty Cycle และ Frequency ของสัญญาณได้



รูปที่ 26 Simulation ของระบบที่ใช้งาน PWM IP

3.2 การใช้งาน TERASIC SoCKIT (CYCLONE V SoC)

SoCKit เป็นบอร์ดสำหรับการออกแบบชาร์ตแวร์ที่ประกอบด้วย Altera Cyclone V System-on-Chip (SoC) FPGA ซึ่งเป็นชิปที่รวม FPGA และ ARM Cortex-A9 (HPS) ไว้ในชิปเดียว

3.2.1 RUNNING LINUX

รายละเอียดของงานที่ทำ : การทำให้ HPS รันระบบปฏิบัติการ Linux ชี้บุตขึ้นมาจาก SD card ได้ (ระยะเวลา 1 สัปดาห์)

ความรู้พื้นฐานที่ใช้ : U-Boot (Universal Bootloader) เป็นโปรแกรมที่ใช้เพื่อโหลดและเริ่มต้นระบบปฏิบัติการหลักของอุปกรณ์นั้นๆ โดย U-Boot มักถูกติดตั้งลงใน flash memory

Config mode หรือโหมดการตั้งค่าใน CycloneV SoCKit สามารถแบ่งออกเป็นหลายโหมดตามวิธีการที่ใช้ในการโปรแกรม FPGA และ SoC แบ่งออกเป็น

JTAG Mode : ใช้สำหรับการ debug และการเขียนโปรแกรมโดยตรงผ่านพอร์ต JTAG มักใช้ในการพัฒนาระบบและการทดสอบเนื่องจากสามารถเขียนโปรแกรมได้รวดเร็ว

AS Mode (Active Serial) : ใช้ serial flash memory สำหรับการบูต FPGA จะโหลดการตั้งค่าจาก Flash เมื่อต้องการบูต มักใช้ในการผลิตจริงเนื่องจากการตั้งค่าจะถูกเก็บอยู่ในหน่วยความจำภายใน

PS Mode (Passive Serial) : โปรแกรมจากอุปกรณ์ภายนอก เช่น คอมพิวเตอร์หรือไมโครคอนโทรลเลอร์ จะส่งข้อมูลการตั้งค่าไปยัง FPGA

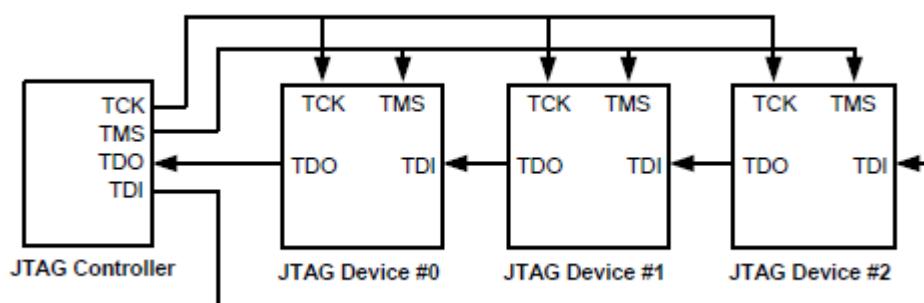
FPP (Fast Passive Parallel) : ใช้การเชื่อมต่อแบบขนาน (parallel) เพื่อเพิ่มความเร็วในการโหลด bitstream ลงใน FPGA

FPGA Configuration through HPS : ใน SoC ที่มีทั้ง FPGA และ HPS (Hard Processor System) การตั้งค่า FPGA สามารถทำผ่าน HPS ได้ HPS สามารถเขียนโปรแกรม FPGA ได้โดยตรงหลังจากบูตขึ้นมา

SD Card Boot : ใช้ SD card ที่มีไฟล์การตั้งค่าเพื่อบูตระบบ แนะนำสำหรับการอัพเดตเฟิร์มแวร์

ใน Cyclone V SoC FPGA และ HPS จะมี JTAG TAP (Test Access Port) ที่แยกออกจากกัน โดยมี TAP controller ของตัวเองที่ใช้สำหรับการโปรแกรมโดยตรงผ่าน JTAG ที่ใช้ TDI (Test Data In), TDO (Test Data Out), TCK (Test Clock), และ TMS (Test Mode Select)

การเชื่อมต่อแบบ Daisy Chain: เพื่อให้สามารถใช้ JTAG ร่วมกันได้ในกรณีที่ต้องการโปรแกรมหรือ debug ทั้ง FPGA และ HPS ผ่านเพียงพอร์ตเดียว สามารถใช้เทคนิค daisy chain โดย TDI ของ FPGA จะต่อเข้ากับ TDI ของ HPS, และ TDO ของ HPS จะต่อกลับมายัง TDO ของ FPGA แบบนี้จะทำให้สามารถโปรแกรมทั้งคู่ในระบบได้ด้วยพอร์ตเดียว



รูปที่ 27 JTAG Chain Connection

ขั้นตอนการทำงาน : ใช้โปรแกรม Win32 Disk Imager ดาวน์โหลด Linux Image ลงบน SD card

จากนั้นตั้งค่า Pins BOOTSELECT[2:0] บนบอร์ดให้เป็น “101” เมื่อ HPS เริ่มต้นการทำงานจะบูตจาก SD card ที่มี U-Boot และ Linux Kernel อยู่ภายใน

ผลลัพธ์ : ใช้โปรแกรม Tera Term เชื่อมต่อผ่าน Serial Port กับบอร์ดเพื่อแสดงข้อมูลที่ส่งออกมายัง HPS ซึ่งจะเห็นได้ว่าเมื่อเปิดไฟบนบอร์ด U-Boot ก็จะบูต Linux ขึ้นมาบน HPS ได้

รูปที่ 28 โปรแกรม Tera Term แสดงข้อมูลการรัน Linux Kernel

3.2.2 ใช้ U-BOOT DEBUG GPIO IP

รายละเอียดของงานที่ทำ : การทดสอบการทำงาน (Debug) ของ GPIO IP ด้วย U-Boot
 (ระยะเวลา 1 สัปดาห์)

ความรู้พื้นฐานที่ใช้ : U-Boot สามารถใช้ในการเข้าถึง memory ต่างๆ ได้ ซึ่งหากต้องการเข้าถึง memory ของ FPGA จะต้องเป็น Lightweight FPGA Slaves จึงจะเข้าถึงได้ สำหรับ address ของ Lightweight FPGA Slave ใน Cyclone V base address จะเริ่มที่ 0xFF200000 และ offset ไปตาม address ของ Slave ตัวนั้นใน Platform Designer

ขั้นตอนการทำงาน : สร้างระบบใน Platform Designer ซึ่งประกอบด้วย HPS, JTAG to Avalon Master Bridge, GPIO IP, JTAG UART และโปรแกรมลงบนบอร์ด จากนั้น เชื่อมต่อบอร์ดกับ computer ผ่าน UART และเปิด Tera Term ขึ้นมาโดยตั้งค่า port และ speed ให้ถูกต้อง และ Reset HPS เพื่อให้โหลด U-boot ขึ้นมา และ เปิดใช้งาน HPS-FPGA bridges สุดท้าย ใช้ command ‘mw’ ในการเขียนข้อมูลลงบน Registers และ ‘md’ ในการอ่าน

ผลลัพธ์ : สามารถใช้ U-Boot ในการเข้าถึง GPIO IP ได้ และ Debug โดยการเขียนและอ่านข้อมูลได้ ดังนี้

```
SOCFPGA_CYCLONE5 # md.l 0xFF200004 1
ff200004: 00000000
SOCFPGA_CYCLONE5 # mw.i 0xFF200004 0xF
SOCFPGA_CYCLONE5 # md.l 0xFF200004 1
ff200004: 0000000f
SOCFPGA_CYCLONE5 # mw.i 0xFF200000 0xA
```

รูปที่ 29 การ Debug GPIO IP ด้วย U-Boot

md.l 0xFF200004 1 : อ่าน data ที่ address 0xFF200004 จำนวน 1 ชุด

mw.l 0xFF200004 0xF : เขียน data 0xF เข้าไปที่ address 0xFF200004

md.l 0xFF200004 1 : อ่าน data ที่ address 0xFF200004 จำนวน 1 ชุด

mw.l 0xFF200000 0xA : เขียน data 0xA เข้าไปที่ address 0xFF200000

3.2.3 SINE WAVE GENERATOR

รายละเอียดของงานที่ทำ : สร้างสัญญาณ sine wave ในรูปแบบ 8 bit ที่หากนำไปผ่าน Digital to Analog Converter จะได้สัญญาณ sine wave ออกมานา โดยการเขียนข้อมูลลงไปด้วย U-Boot และเก็บลง FIFO จากนั้นค่อยอ่านออกมาทีเดียว จากนั้นเปลี่ยน FIFO เป็น Onchip RAM (ระยะเวลา 2 สัปดาห์)

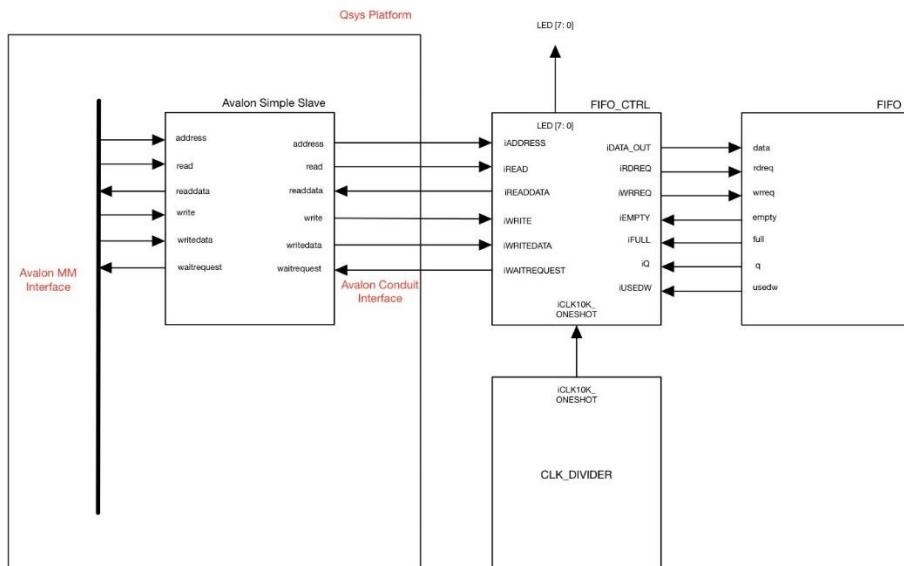
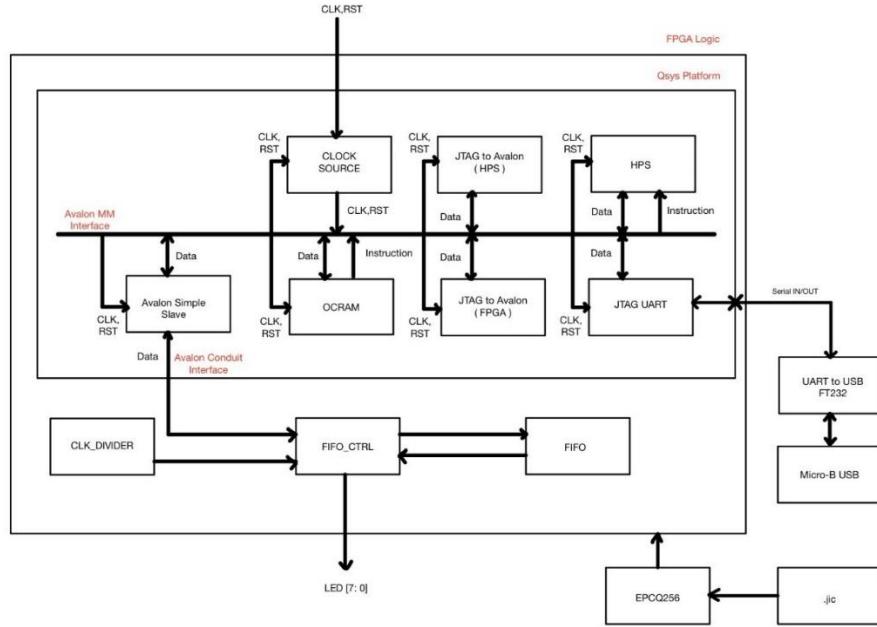
ความรู้พื้นฐานที่ใช้ : Digital to Analog Convert

Binary Weighted Resistor Method: วิธีนี้ใช้ตัวต้านทานที่มีค่าตั้งไว้ในลักษณะ Binary หาก LSB มีความต้านทาน R บิตต่อมาก็จะมีตัวต้านทานที่มีค่า $R/2$

R-2R Ladder Method: วิธีนี้ใช้ค่าตัวต้านทานเพียง 2 ค่า คือ R และ $2R$ ซึ่งต่อในรูปแบบ Ladder โดยแต่ละบิตจะควบคุมสวิตซ์ระหว่างกราวด์และระดับบน Ladder ซึ่งวิธีนี้ก็ให้ความแม่นยำกว่าวิธี Binary Weighted เนื่องจากใช้ตัวต้านทานที่หลากหลายน้อยกว่า

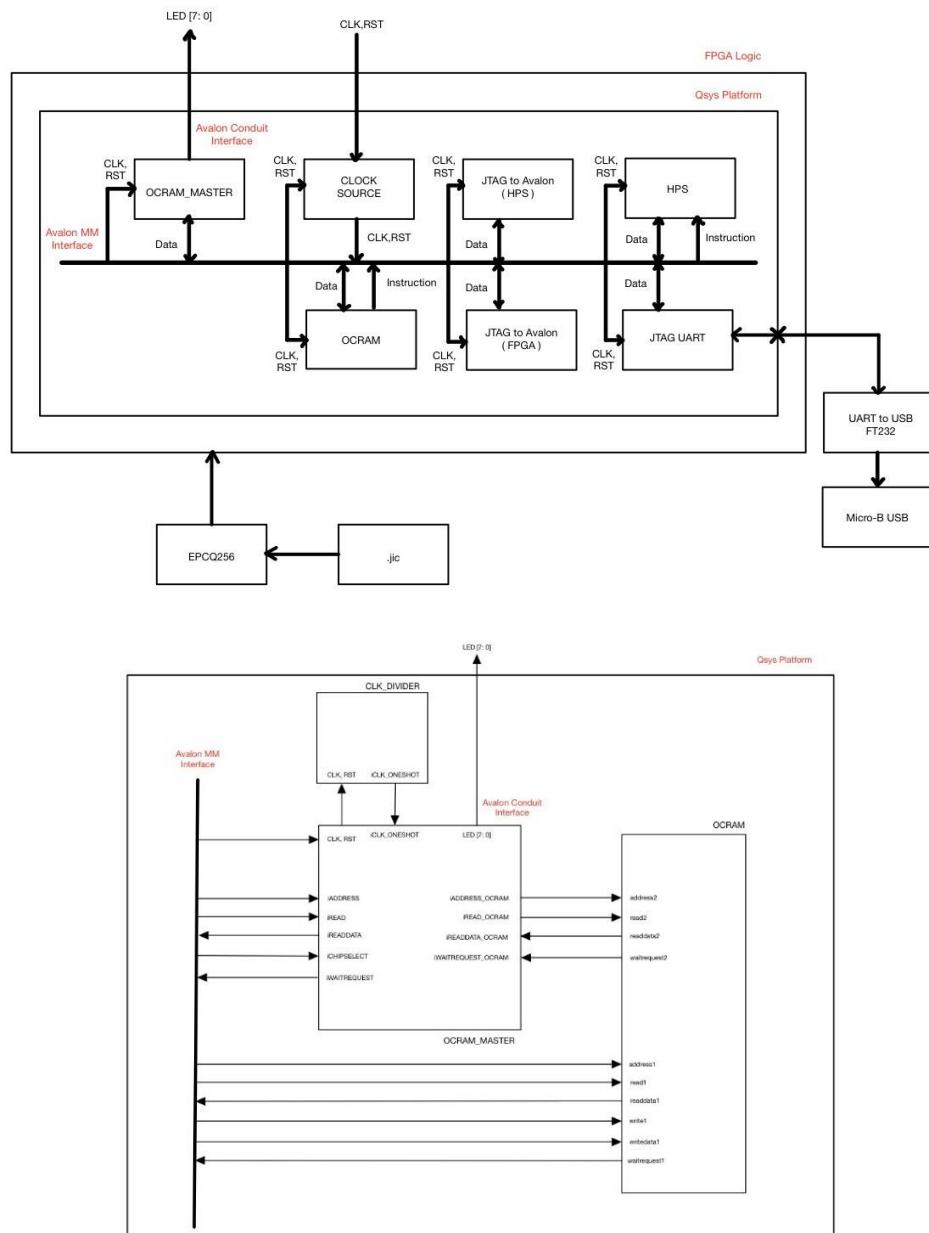
FIFO (First-In-First-Out) คือหน่วยความจำที่ข้อมูลที่ถูกเขียนเข้าไปก่อนจะถูกอ่านออกก่อน และเมื่ออ่านออกแล้วข้อมูลก็จะถูกลบออกไป

ขั้นตอนการทำงาน : ออกแบบระบบซึ่งประกอบด้วยระบบที่สร้างจาก Platform Designer และ FPGA logic ที่ประกอบด้วย FIFO_CTRL, CLK_DIVIDER, FIFO ซึ่งภายในระบบที่สร้างจาก Platform Designer จะมี Avalon Simple Slave ที่ใช้ส่งออกข้อมูลใน Avalon MM Interface สู่ FPGA logic เพื่อตีงเอาข้อมูลที่เขียนลงมาจาก U-Boot ไปเก็บใน FIFO ด้วยการควบคุมจาก FIFO_CTRL



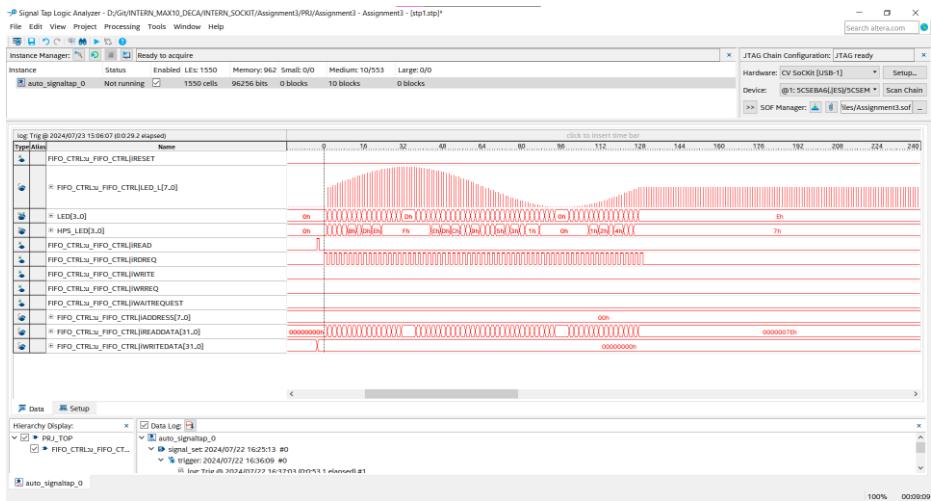
รูปที่ 30 Block Diagram ของระบบที่ใช้สร้าง sine wave เก็บใน FIFO

ต่อมาเปลี่ยนจากการเก็บข้อมูลบน FIFO เป็น Onchip RAM เนื่องจากเมื่ออ่านข้อมูลแล้วข้อมูลนั้นจะไม่ถูกลบไปทำให้สามารถวนกลับไปเริ่มอ่านใหม่ได้ ซึ่งจะทำให้ได้ sinewave แบบต่อเนื่อง โดยการออกแบบจะสร้าง OCRAM_MASTER ใน Platform Designer เพื่อใช้ควบคุมการอ่านข้อมูลใน Onchip RAM ให้อ่านวนไปเรื่อยๆ ส่วนการเขียนนั้นสามารถเขียนลงบน Onchip RAM ได้โดยตรงผ่าน Avalon MM Interface

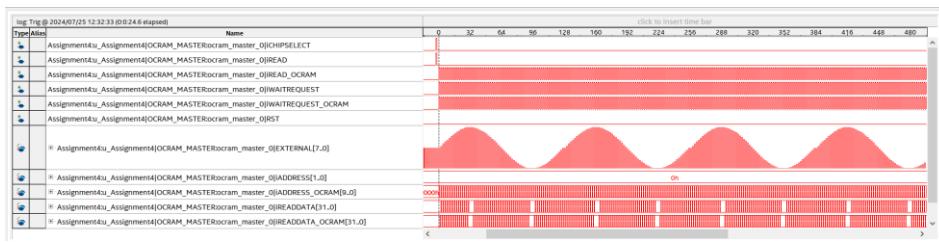


รูปที่ 31 Block Diagram ของระบบที่ใช้สร้าง sine wave เก็บใน Onchip RAM

ผลลัพธ์ : สามารถใช้ Signal Tap จับสัญญาณที่อ่านออกมายจาก FIFO ที่เขียนลงไปผ่าน U-Boot ได้ ออกแบบเป็น sine wave ในคาบตามที่เขียนลงไป



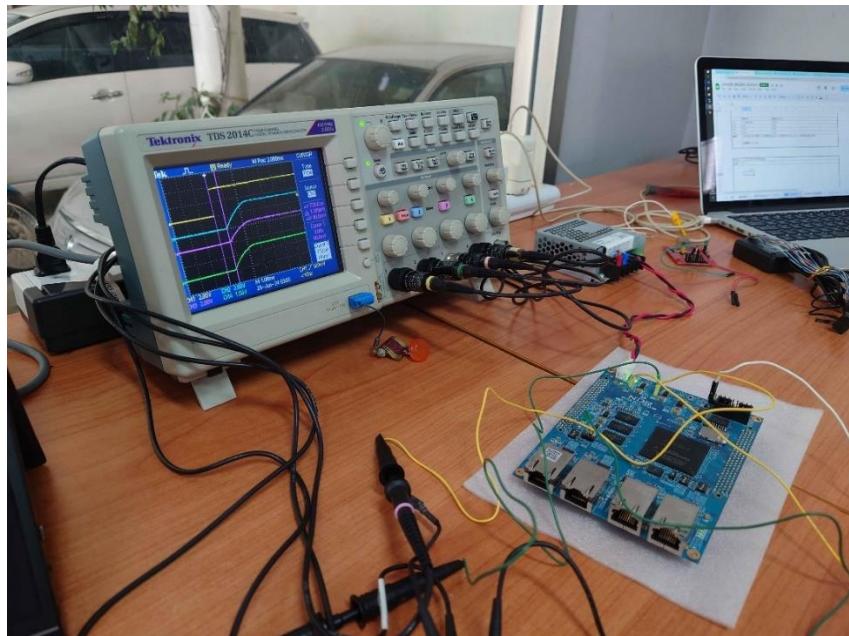
รูปที่ 32 ใช้ Signal Tap จับสัญญาณ sine wave ที่เก็บใน FIFO และอ่านออกมาย่างต่อเนื่องได้เมื่อเก็บข้อมูลใน Onchip RAM



รูปที่ 33 ใช้ Signal Tap จับสัญญาณ sine wave ที่เก็บใน Onchip RAM

3.3 เรียนรู้การใช้งาน Oscilloscope วัดสัญญาณแรงดันไฟฟ้าบนบอร์ด

โดยเป็นการวัดสัญญาณแรงดันไฟในส่วนต่างๆของบอร์ด เพื่อตรวจสอบว่าระบบทำงานได้อย่างถูกต้องและเสถียรหรือไม่ รวมถึงวัดแรงดันไฟเลี้ยงและช่วงเวลาในการเปลี่ยนแปลงภายใต้เงื่อนไขการทำงานที่ต้องการ



รูปที่ 34 ภาพการใช้ Oscilloscope วัดสัญญาณแรงดันไฟฟ้าบนบอร์ด

บทที่ 4 สรุป

4.1 ประโยชน์ที่ได้รับจากการฝึกงาน

จากการฝึกงานครั้งนี้ ผู้จัดทำได้ความรู้เกี่ยวกับ FPGA จำนวนมาก เช่น การใช้งานซอฟต์แวร์สำหรับการออกแบบ FPGA ทักษะในการเขียน VHDL และ การใช้เครื่องมือต่างๆ และการออกแบบระบบตั้งแต่พื้นฐานจนถึงมีการเชื่อมต่อและการทำงานร่วมกับระบบอื่นๆ เช่น การเชื่อมต่อกับ Microprocessor, หน่วยความจำ, และอุปกรณ์ I/O ได้รับประสบการณ์ในการทำงานจริง ทั้งการแก้ไขปัญหา การจัดการเวลา รวมถึงทักษะการสื่อสารและนำเสนอ รวมถึงได้รับคำปรึกษาและคำแนะนำจากผู้ดูแลและบุคลากรทุกท่าน

4.2 ปัญหา อุปสรรค และข้อเสนอแนะ

ในการทำงานออกแบบมีประสิทธิภาพจำเป็นต้องมีการออกแบบอย่างละเอียดและครบถ้วนก่อนลงมือทำ แต่ในช่วงแรกของการทำงานผู้จัดทำมักจะลงมือทำพร้อมๆ กับการออกแบบไปด้วย เนื่องจากขาดความรู้ความเข้าใจในการออกแบบ硬件 ซึ่งทำให้งานที่ออกแบบนั้นไม่ถูกต้องและไม่ถูกหลักการ นอกจากนี้ยังมี Tools บางอย่างบนโปรแกรม เช่น Platform Designer และ Eclipse ที่ไม่เคยใช้มาก่อน จึงใช้เวลาค่อนข้างนานในการศึกษา

เอกสารอ้างอิง

- [1] <https://ndrsolution.com>
- [2] <https://www.nectec.or.th/news/news-public-document/fpga-design-iot-html.html>
- [3] http://www.palettesoft.co.jp/technology/pic/pic_sample_sw0.htm
- [4] https://www.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
- [5] <https://ndrsolution.com/2021/09/24/nios-ii-processor-booting-methods-in-max10-fpga/>
- [6] <https://www.macnica.co.jp/en/business/semiconductor/articles/intel/113961/>
- [7] https://www.cin.ufpe.br/~tsmcf/mnl_avalon_bus.pdf
- [8] <https://ndrsolution.com/2021/11/12/fpga-avalon-bus-simulation/>
- [9] <https://saixiii.com/unix-linux-command/>
- [10] <https://www.intel.com/content/www/us/en/docs/programmable/683360/18-0/lightweight-hps-to-fpga-bridge.html>
- [11] <https://www.intel.com/content/www/us/en/programmable/hps/cyclone-v/hps.html#sfo1410069906123.html>
- [12] <https://www.electronics-tutorials.ws/combination/r-2r-dac.html>

ภาคผนวก

รายงานการฝึกงานทุกสองสัปดาห์ ฉบับที่.....1.....

ชื่อ-สกุล..... บวรณิชัย ทีปส่อง เลขประจำตัว..... 6430216821
 ชื่อบริษัท/หน่วยงานที่ฝึกงาน..... บริษัท ทีมีเดอร์ โซลูชั่น (ประเทศไทย) จำกัด

| วัน/เดือน/ปี | จำนวนชั่วโมง | งานที่ปฏิบัติโดยย่อ | ลงชื่อนิสิต |
|---|--------------|--|--|
| 24/05/2567 | 8* | 下げทวนค่าลูป DECA MAX10 Datasheet | บวรณิชัย |
| 27/05/2567 | 8* | เขียนรูปกราฟฟังก์ชัน Bit และ ตัวกราฟ FPGA เซ็ตอัตโนมัติ | บวรณิชัย |
| 28/05/2567 | 8* | ตัวกราฟ circuit ของ DECA MAX10 และ เขียน VHDL on/off LED ด้วย switch | บวรณิชัย |
| 29/05/2567 | 8* | เขียน VHDL on/off LED ด้วย switch และ มีปุ่ม RESET ใช้งานจริงๆ และ simulation | บวรณิชัย |
| 30/05/2567 | 8* | เขียน VHDL on/off LED ทุกๆ 1s และ กดปุ่มเพื่อเปลี่ยนเป็น 0.5 s และ simulation | บวรณิชัย |
| 31/05/2567 | 8* | เขียน VHDL กดปุ่มเพื่อให้ LED บนชิ้น/ควบคุมการต่อสัญญาณ LED แบบ Binary และ simulation | บวรณิชัย |
| 04/06/2567 | 8* | ตัวกราฟลักษณะการ debounce และ ออกแบบการควบคุมการต่อสัญญาณ LED แบบ Binary | บวรณิชัย |
| 05/06/2567 | 8* | เขียน VHDL กดปุ่มเพื่อควบคุม LED บนชิ้น/ควบคุม Binary และ simulation | บวรณิชัย |
| 06/06/2567 | 8* | สรุปปุ่ม/แก้ไขการอภิบาย Block Diagram, Timing Diagram, VHDL | บวรณิชัย |
| 07/06/2567 | 8* | ตัวกราฟ UART และ NIOS II | บวรณิชัย |
| | | | |
| | | | |
| จำนวนชั่วโมงฝึกงานรวมในรายงานฉบับนี้ | 80* | ขอรับรองว่ารายงานฉบับนี้เป็นความจริงทุกประการ |  ลงชื่อ..... บวรณิชัย ผู้ดูแลการฝึกงาน (..... นายศักดิ์พี เต็มศิริพันธ์) |
| จำนวนชั่วโมงฝึกงานรวมในรายงานฉบับก่อนหน้า | - | | |
| จำนวนชั่วโมงฝึกงานรวมทั้งหมด | 80* | ตำแหน่ง..... Hardware Developer วันที่..... 11...../..... มิถุนายน...../..... 2567..... | |

ข้อคิดเห็นของผู้ดูแลการฝึกงาน (ถ้ามี)

มีความตั้งใจ รับผิดชอบต่องานที่ได้รับมอบหมาย

หมายเหตุ 1. ให้นิสิตทราบรายงานทุกสองสัปดาห์ทุกฉบับไว้ในภาคผนวกของรายงานฉบับสมบูรณ์ด้วย

2. ให้นิสิตใส่เครื่องหมาย * หลังตัวเลขจำนวนชั่วโมงในวันที่นิสิตฝึกงานในลักษณะ Work from Home ด้วย

รายงานการฝึกงานทุกสองสัปดาห์ ฉบับที่.....2.....

ชื่อ-สกุล.....นรันดร์ ทีฆสวาง..... เลขประจำตัว.....6430216821
ชื่อบริษัท/หน่วยงานที่ฝึกงาน.....บริษัท เทคโนโลจี จำกัด (ประเทศไทย) ตำแหน่ง.....ค้าขาย

| วัน/เดือน/ปี | จำนวนชั่วโมง | งานที่ปฏิบัติโดยย่อ | ลงชื่อนิสิต |
|---|--------------|---|-------------|
| 10/06/2024 | 8* | ศึกษา NIOS II และโครงสร้าง MCU สำหรับงาน | นรันดร์ |
| 11/06/2024 | 8* | ทดลองใช้ NIOS II ผ่านช่องทาง串行 Console | นรันดร์ |
| 12/06/2024 | 8* | ทดลองใช้ NIOS II ผ่านช่องทาง串行串行 LED | นรันดร์ |
| 13/06/2024 | 8* | เขียนโปรแกรม C code บน NIOS II รับค่า SW จาก串行串行 LED แล้ว | นรันดร์ |
| 14/06/2024 | 0 | ค้าขาย | นรันดร์ |
| 17/06/2024 | 8* | ออกแบบ Avalon Bus และต่อสาย GPIO IP กับ | นรันดร์ |
| 18/06/2024 | 8* | Simulation GPIO IP กับซิมเมดี้ | นรันดร์ |
| 19/06/2024 | 8* | เขียนบอร์ด GPIO IP ที่สามารถรับ input และ ส่ง output ได้ | นรันดร์ |
| 20/06/2024 | 8* | เขียนบอร์ด GPIO IP ที่มีขาแบบ BiDirection Port | นรันดร์ |
| 21/06/2024 | 8* | ทดลองใช้ GPIO IP กับ Quartus และ BiDirection เพื่อ Debug | นรันดร์ |
| | | GPIO IP กับซิมเมดี้ | |
| | | | |
| จำนวนชั่วโมงฝึกงานรวมในรายงานฉบับนี้ | 72 | ขอรับรองว่ารายงานฉบับนี้เป็นความจริงทุกประการ | |
| จำนวนชั่วโมงฝึกงานรวมในรายงานฉบับก่อนหน้า | 80 | ลงชื่อ..... <u>นรันดร์</u> ผู้ดูแลการฝึกงาน (นาย ศักดิ์ ธรรมรงค์ ตำแหน่ง หัวหน้า.....) | |
| จำนวนชั่วโมงฝึกงานรวมทั้งหมด | 152 | ตำแหน่ง..... <u>Hardware developer</u> วันที่..... <u>25</u>/..... <u>มิถุนายน</u>/..... <u>2024</u> | |

ข้อคิดเห็นของผู้ดูแลการฝึกงาน (ถ้ามี)

หมายเหตุ 1. ให้นิสิตทราบรายงานทุกสองสัปดาห์ทุกฉบับไว้ในภาคผนวกของรายงานฉบับสมบูรณ์ด้วย

2. ให้นิสิตใส่เครื่องหมาย * หลังตัวเลขจำนวนชั่วโมงในวันที่นิสิตฝึกงานในลักษณะ Work from Home ด้วย

รายงานการฝึกงานทุกสองสัปดาห์ ฉบับที่.....3.....

ชื่อ-สกุล.....บริรักษ์ ทีปสกุล..... เลขประจำตัว.....6430216821
 ชื่อบริษัท/หน่วยงานที่ฝึกงาน.....บริษัท เอ็นจีอาร์ โซลูชัน (ประเทศไทย) จำกัด..... ตำแหน่ง.....ศึกษา

| วัน/เดือน/ปี | จำนวนชั่วโมง | งานที่ปฏิบัติโดยย่อ | ลงชื่อนิสิต |
|---|--------------|--|--|
| 24/06/2024 | 8* | Debug GPIO IP ผ่านเว็บไซต์ และทำ Simulation | บริรักษ์ |
| 25/06/2024 | 7 | เรียนรู้การใช้ Oscilloscope เพื่อวัดคุณภาพของงาน | บริรักษ์ |
| 26/06/2024 | 8* | ศึกษา PWM และการอ่านเข้าสู่บอร์ด Cyclone V SoC ผ่านพอร์ต LED | บริรักษ์ |
| 27/06/2024 | 8* | ออกแบบวงจรด้วย PWM IP และเพิ่ม Register Map | บริรักษ์ |
| 28/06/2024 | 8* | สร้าง PWM IP สำหรับอ่านค่า Duty Cycle และ Frequency ไป | บริรักษ์ |
| 01/07/2024 | 8* | ปรับปรุง ของ IP และทำ Simulation | บริรักษ์ |
| 02/07/2024 | 8* | ศึกษาขนาด Cyclone V SoC | บริรักษ์ |
| 03/07/2024 | 8* | ทดลองรัน Linux บนบอร์ด โดย boot จาก SD card | บริรักษ์ |
| 04/07/2024 | 8* | ทดสอบการทำงานของ software บน HPS และภาษา command ใน Linux | บริรักษ์ |
| 05/07/2024 | 8* | สร้าง FPGA project ไฟฟ้าแบบ Cyclone V โดยboot จาก ECPQ256 | บริรักษ์ |
| | | | |
| | | | |
| จำนวนชั่วโมงฝึกงานรวมในรายงานฉบับนี้ | 79 | ขอรับรองว่ารายงานฉบับนี้เป็นความจริงทุกประการ | ลงชื่อ..... <u>บริรักษ์</u>ผู้ดูแลการฝึกงาน (... นายศักดิรพี เต็มศิริพันธ์) ตำแหน่ง..... Hardware Developer วันที่..... 11/..... กรกฏาคม/..... 2567 |
| จำนวนชั่วโมงฝึกงานรวมในรายงานฉบับก่อนหน้า | 152 | | |
| จำนวนชั่วโมงฝึกงานรวมทั้งหมด | 231 | | |

ข้อคิดเห็นของผู้ดูแลการฝึกงาน (ถ้ามี)

หมายเหตุ 1. ให้นิสิตทราบรายงานทุกสองสัปดาห์ทุกฉบับไว้ในภาคผนวกของรายงานฉบับสมบูรณ์ด้วย

2. ให้นิสิตใส่เครื่องหมาย * หลังตัวเลขจำนวนชั่วโมงในวันที่นิสิตฝึกงานในลักษณะ Work from Home ด้วย

รายงานการฝึกงานทุกสองสัปดาห์ ฉบับที่.....๔

ชื่อ-สกุล..... บร.ณิชญ์ ทันปะร่วง เลขประจำตัว..... ๖๔๓๐๒๑๖๘๒๑
ชื่อบริษัท/หน่วยงานที่ฝึกงาน..... บริษัท เอ็นดีที โซลูชัน (ประเทศไทย) จำกัด

| วัน/เดือน/ปี | จำนวนชั่วโมง | งานที่ปฏิบัติโดยย่อ | ลงชื่อนิสิต |
|---|--------------|--|--|
| 08/07/2024 | 8* | ทดลองใช้งานทดลอง project MIPI Camera และ DECA MAX10 | บรรณาธิการ |
| 09/07/2024 | 8* | ใช้ uboot debug PIO IP ของ Altera (ผ่าน Qsys จำลองช่อง) | บรรณาธิการ |
| 10/07/2024 | 8* | ใช้ uboot debug GPIO IP ที่ส่งซึ่งกันๆ ของ project ใน CD ROM | บรรณาธิการ |
| 11/07/2024 | 8* | ใช้ uboot debug โครงการ project ในมือถือและสร้าง Qsys บน | บรรณาธิการ |
| 12/07/2024 | 8* | ออกแบบ FIFO และ DAC (R-2R circuit) | บรรณาธิการ |
| 13/07/2024 | 0 | ภาคี | บรรณาธิการ |
| 16/07/2024 | 8* | ออกแบบนากลีฟรุ่ง 8-bit sine wave และเขียนโค้ดภาษา C บน FPGA ด้วย uboot | บรรณาธิการ |
| 17/07/2024 | 8* | ปรับปรุงการขออาณบนาและสั่นสะเทือน Qsys สำหรับ project | บรรณาธิการ |
| 18/07/2024 | 8* | สร้าง FPGA ตามที่ขออาณบนา | บรรณาธิการ |
| 19/07/2024 | 8* | ทำ Simulation และ Signal Tap เพื่อกลับไป project ที่สร้างขึ้น | บรรณาธิการ |
| | | | |
| | | | |
| จำนวนชั่วโมงฝึกงานรวมในรายงานฉบับนี้ | 72 | ขอรับรองว่ารายงานฉบับนี้เป็นความจริงทุกประการ |  ลงชื่อ..... นายศักดิ์พี เต็มศิริพันธ์ |
| จำนวนชั่วโมงฝึกงานรวมในรายงานฉบับก่อนหน้า | 231 | ผู้ดูแลการฝึกงาน (.....) | |
| จำนวนชั่วโมงฝึกงานรวมทั้งหมด | 303 | ตำแหน่ง..... Hardware Developer | |
| | | วันที่..... 25/ กรกฏาคม / 2567 | |

ข้อคิดเห็นของผู้ดูแลการฝึกงาน (ถ้ามี)

หมายเหตุ 1. ให้นิสิตทราบรายงานทุกสองสัปดาห์ทุกฉบับไว้ในภาคผนวกของรายงานฉบับสมบูรณ์ด้วย

2. ให้นิสิตใส่เครื่องหมาย * หลังตัวเลขจำนวนชั่วโมงในวันที่นิสิตฝึกงานในลักษณะ Work from Home ด้วย

รายงานการฝึกงานทุกสองสัปดาห์ ฉบับที่.....5

ชื่อ-สกุล..... ปรรภ.นิธิวนิช จันทร์กุวงศ์ เลขประจำตัว..... 6430216821.....
ชื่อบริษัท/หน่วยงานที่ฝึกงาน..... บริษัท เทคโนวอร์ จำกัด (ประเทศไทย) จำกัด.....

| วัน/เดือน/ปี | จำนวนชั่วโมง | งานที่ปฏิบัติโดยย่อ | ลงชื่อนิสิต |
|---|--------------|--|-------------|
| 23/07/2024 | 8* | ออกแบบการสร้าง 8-bit sine wave และพัฒนาลงใน Onchip RAM | นรธนิสฐ์ |
| | | ติดตั้ง U-Boot และสร้าง Project ตามที่อาจารย์เปิด | |
| 24/07/2024 | 8* | ทำการ Simulation & Signal Tap เพื่อทดสอบ และประมวลผล Project ที่สร้าง | บรรณวิทย์ |
| 25/07/2024 | 8* | ติดตั้ง TTL MACRO files และสร้างสัญญาณเสียงจาก sine wave | บรรณวิทย์ |
| 26/07/2024 | 7 | เขียนนักความเรื่องการสร้าง Custom GPIO IP | นรธนิสฐ์ |
| 30/07/2024 | 8* | เขียนนักความเรื่องการใช้ U-Boot Debug GPIO IP | บรรณวิทย์ |
| 31/07/2024 | 8* | ปั๊บปูรุษความ และเขียนรายงาน | นรธนิสฐ์ |
| | | | |
| | | | |
| | | | |
| | | | |
| จำนวนชั่วโมงฝึกงานรวมในรายงานฉบับนี้ | 47 | ขอรับรองว่ารายงานฉบับนี้เป็นความจริงทุกประการ | |
| จำนวนชั่วโมงฝึกงานรวมในรายงานฉบับก่อนหน้า | 303 | ลงชื่อ.....  ผู้ดูแลการฝึกงาน (.....นายศักดิรพี เต็มศิริพันธ์.....) | |
| จำนวนชั่วโมงฝึกงานรวมทั้งหมด | 350 | ตำแหน่ง..... Hardware Developer | |
| | | วันที่..... 2/ สิงหาคม / 2567 | |

ข้อคิดเห็นของผู้ดูแลการฝึกงาน (ถ้ามี)

.....

.....

หมายเหตุ 1. ให้นิสิตทราบรายงานทุกสองสัปดาห์ทุกฉบับไว้ในภาคผนวกของรายงานฉบับสมบูรณ์ด้วย

2. ให้นิสิตใส่เครื่องหมาย * หลังตัวเลขจำนวนชั่วโมงในวันที่นิสิตฝึกงานในลักษณะ Work from Home ด้วย

Github สำหรับเก็บโค้ดและไฟล์ต่างๆตลอดการฝึกงาน

<https://github.com/bannawich46/INTERN.git>

Video แสดงการทำงานมีอุปกรณ์ลงบนบอร์ดของโปรเจคในหัวข้อ 3.1 – 3.3

https://drive.google.com/drive/folders/1elh_khG_ztxlUKeEZsm2rcfSXpjPB1ri?usp=sharing

Datasheet ของ DECA MAX10

<https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=&No=944&PartNo=1>

Datasheet ของ SoCKit

<https://www.terasic.com.tw/cgi-bin/page/archive.pl?CategoryNo=167&No=816>