FPGA 기말 프로젝트 보고서

과목: FPGA

담당교수: 김선희 교수님

학번: 201921302, 202221381, 202221340,

이름: 유지훈, 표승현, 김지윤

목차

1. 서론
   1. 주제의 배경 및 목적…………………………………………………………………………………………………………1
   2. 연구 방법………………..…………………………………………………………………………………………………………1

2 본론

2.1 state machine 및 동작 원리 설명………………………………………………………………………………………4

2.3 Controller, Byte2ascii, Top 코드 주석…………………………………………………………………………5-36

2.2 시뮬레이션 결과…………………………………………………………………………………………………………36-37

3 결론

3.1 고찰 및 기대효과……………………………………………………………………………………………………………..37

3.2 참고문헌……………………………………………………………………………………………………………………37-38

1. 서론
   1. 주제의 배경 및 목적

지난해 유엔 기후변화 정부 간 협의체인 IPCC에서 발표한 IPCC 제 6차 보고서에 따르면 지구의 표현 평균기온의 상승이 산업화 이전과 비교했을 때 1.5도 이하가 되기는 매우 어려울 것으로 전망하며, 현재와 비슷한 양의 온실가스 배출이 지속될 경우 기온이 1.5도 상승하는 시점이 앞선 2018년 IPCC가 예측했던 시기보다 10년 앞당겨진 2021~2040년이 될 것으로 예측되며 이에 각 분야의 기후변화 대응방안 적용이 시급한 상황이다. 이와 같은 기후변화 위기 상황에서 농업 분야는 큰 영향을 받을 것으로 예상된다. 기온 상승, 강수 패턴 변화, 극한 기상 현상의 증가 등은 농작물 생산에 직접적인 영향을 미치며, 이는 식량 안보와 경제적 안정성에 심각한 위협이 되기도 한다. 기후변화와 같은 농업에 큰 영향을 끼치면서 변화가 큰 현상에 대한 적극적은 대응이 필요하며 이는 지속 가능한 농업 방법을 개발하고 적용하는 데 중점을 둬야 한다. 스마트팜(smart Farm)은 이러한 기후변화 대응의 중요한 방안 중 하나로 주목 받고 있으며 정보통신 기술을 활용하여 농업 생산성을 높이고 자원 사용의 효율성을 극대화하며 농업 환경 영향을 최소화하는 농업 시스템을 의미한다. IoT, AI, 빅데이터 분석, 다양한 센서를 이용한 자동화 시스템 등 실시간으로 변화하는 환경에 맞게 최석화할 수 있다. 이를 통해 농업 생산의 불확실성을 줄이고 보다 안정적이고 지속 가능한 식량 생산을 가능하게 한다. 본 프로젝트는 스마트팜의 이점을 조사하고 FPGA를 기반으로 한 스마트팜을 제작하였으며 프z로젝트 산출물을 통해 다양한 농업 생산 효율성을 높이고 미래 농업의 방향성을 담아 진행하였다.

* 1. 연구 방법

본 프로젝트는 FPGA 보드를 사용하여 verilog 언어를 기반으로한 다양한 기능을 담은 스마트팜을 제작하였다. 연구 방법으로는 크게 시뮬레이션과 코드 작성으로 다음과 같이 진행하였다. 첫째, 모듈을 기능에 따라 나누어 각 모듈의 상관관계를 표현한 로직도를 시각적으로 작성하였다. 둘째, 각 모듈마다 state machine을 그려 코드 작성을 하였고 테스트벤치를 각 모듈마다 만들어 simulation한 후 wave를 통해 output을 확인하였다. 세 번째로 top모듈로 각 모듈에서 필요한 변수들을 인스턴스해주었으며 전체에 대한 simulation을 통해 프로젝트의 의도와 맞게 동작하는지 확인하였다. 마지막으로 quartus를 이용하여 모듈을 합성한 후 FPGA 보드에 올려 동작을 확인하였다.

1. 본론
   1. state machine 및 동작 원리 설명

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

시스템이 가동되면 초기 설정값을 로드하고 모든 센서와 장치를 초기화한다. 이는 시스템이 정상적으로 작동할 수 있도록 준비하는 단계이다. 다음으로 사용자가 버튼을 클릭하면 시스템은 광센서의 현재 값을 읽어온다. 광센서는 주변의 조도(밝기)를 측정하여 raw data를 생성한다. 이 데이터를 기반으로 LED를 켜서 농작물에 적절한 조명을 유지하려고 한다. 두 번째 push\_button을 누르게 되면 FND에 A-p를 display한다. 아침부터 오후시간 사이에 온풍기와 스프링쿨러가 작동되고 있음을 알려주는 장치이다. 그 후 온풍기 설정을 우선으로 한다. 이는 특정 온도 범위 내에서 농작물의 성장 조건을 유지시키기 위함이다. 온풍기 작동 여부는 LED를 통해 확인할 수 있다. 다음으로 스프링쿨러 세팅이다. 사용자가 스플링쿨러 세팅 버튼을 누르게 되면 시스템은 5초 동안 모터가 작동하게 된다. 이는 농작물에 필요한 수분을 자동으로 분사하는 과정이다. 그 후 오전-오후 사이에 동작이 끝나게 되면 P-A가 FND에 display 된다. 그리고 똑같은 과정을 반복한 후 오후-오전 사이에 동작이 끝나게 되면 done 신호를 전송하게 된다. 이는 오후-오전 시간에 스프링쿨러 작동이 끝났음을 알려준다. 시스템은 done 신호를 받으면 작동을 멈추고 state가 IDLE로 전환되어 다음 명령을 대기하게 된다.

* 1. 코드 주석

|  |
| --- |
| //Controller 코드 주석  module Controller(  input clk,  input n\_rst,  input bt\_start,  input bt\_setting,    input [7:0] rx\_data,  input rx\_done,  input [7:0] led\_data,  input spi\_done,  output reg motor\_signal,  output reg heat\_signal,  output reg [7:0] led\_out,  output reg morning\_signal,  output reg after\_signal,  output reg Day\_done);  `ifdef SIM  parameter ONE\_SECOND = 10;  `else  // 50 MHz clock -> (1/(d5208)) -> 9,600 rate  parameter ONE\_SECOND = 50\_000\_000;  `endif  parameter IDLE = 4'h0;  parameter LIGHT\_READ = 4'h1;  parameter FND1 = 4'h2;  parameter SETTING = 4'h3;  parameter RUNNING = 4'h4;  parameter LIGHT\_READ\_2 = 4'h5;  parameter FND2 = 4'h6;  parameter SETTING\_2 = 4'h7;  parameter RUNNING\_2 = 4'h8;  parameter DONE = 4'h9;  reg [3:0] c\_state, n\_state;  // RUNNING ⑤객臾띄솒됱굚蹂뜟  reg [25:0] sec\_counter;  reg [3:0] runnig\_time;  // 1Day 타이밍 로직  // RUNNING 상태 타이머 및 리셋 로직  always @(posedge clk or negedge n\_rst) begin  if (!n\_rst) begin  sec\_counter <= 26'h0000;  runnig\_time <= 3'h0;  end else begin  // c\_state가 RUNNING 이거나 RUNNING\_2일 때 counter로 초 계산  if (c\_state == RUNNING || c\_state == RUNNING\_2) begin  if (sec\_counter < ONE\_SECOND - 1) begin  sec\_counter <= sec\_counter + 26'h0001;  end else begin  sec\_counter <= 26'h0000;  if (runnig\_time < 3'h5)  runnig\_time <= runnig\_time + 3'h1;  end  end else begin  sec\_counter <= 26'h0000; // RUNNING 상태가 아닐 때 sec\_counter 리셋  runnig\_time <= 3'h0;  end  end  end  // c\_state 초기화  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  c\_state <= IDLE;  end else begin  c\_state <= n\_state;  end  // 다음 state로 넘어가는 로직  always @(\*)  case(c\_state)  IDLE : begin  n\_state = (bt\_start == 1'b1) ? LIGHT\_READ : c\_state;  end  LIGHT\_READ : begin  n\_state = (spi\_done == 1'b1) ? FND1 : c\_state;  end  FND1 : begin  n\_state = SETTING;  end  SETTING : begin  n\_state = (bt\_setting == 1'b1) ? RUNNING : c\_state;  end  RUNNING : begin  n\_state = (runnig\_time == 3'h5) ? LIGHT\_READ\_2 : c\_state;  end  LIGHT\_READ\_2 : begin  n\_state = (spi\_done == 1'b1) ? FND2 : c\_state;  end  FND2 : begin  n\_state = SETTING\_2;  end  SETTING\_2 : begin  n\_state = (bt\_setting == 1'b1) ? RUNNING\_2 : c\_state;  end  RUNNING\_2 : begin  n\_state = (runnig\_time == 3'h5) ? DONE : c\_state;  end  DONE : begin  n\_state = IDLE;  end  default : begin  n\_state = IDLE;  end  endcase  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // SPI 완료 신호를 받고 LED 데이터를 반전하여 led\_out으로 출력  always @(posedge clk or negedge n\_rst)  if(!n\_rst)  led\_out <= 8'h00;  else begin  if(c\_state == LIGHT\_READ || c\_state == LIGHT\_READ\_2) begin  if(spi\_done == 1'b1) begin  led\_out <= ~led\_data;  end  end  end  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // (SETTING & MOTOR)  // motor\_signal과 heat\_signal을 state(SETTING거나 SETTING\_2일 때)에 따라 bt\_setting이 1일 때 running\_time이 5보다 작을 때 1이 할당  always @(posedge clk or negedge n\_rst) begin  if (!n\_rst) begin  motor\_signal <= 1'b0;  heat\_signal <= 1'b0;  end else begin  if (c\_state == SETTING || c\_state == SETTING\_2) begin  if (bt\_setting == 1'b1) begin  motor\_signal <= (runnig\_time < 3'h5) ? 1'b1 : 1'b0;  heat\_signal <= (runnig\_time < 3'h5) ? 1'b1 : 1'b0;  end else begin  motor\_signal <= 1'b0;  heat\_signal <= 1'b0;  end  // running\_time이 5가 되면(5초가 지나면) motor\_signal 과 heat\_signal에 0으로 초기화  end else if (runnig\_time == 3'h5) begin  motor\_signal <= 1'b0;  heat\_signal <= 1'b0;  end else begin  motor\_signal <= motor\_signal;  heat\_signal <= heat\_signal;  end  end  end  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // FND로 낮인지 밤인지 구별해주기  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  morning\_signal <= 1'b0;  after\_signal <= 1'b0;  end  // morning\_signal이 1일 때 FND에 A-P display  // after\_signal이 1일 때 FND에 P-A display  else begin  if(c\_state == FND1) begin  morning\_signal <= 1'b1;  after\_signal <= 1'b0;  end  else if(c\_state == FND2) begin  after\_signal <= 1'b1;  morning\_signal <= 1'b0;  end  else if(runnig\_time == 3'h5) begin  after\_signal <= 1'b0;  morning\_signal <= 1'b0;  end  else begin  after\_signal <= after\_signal;  morning\_signal <= morning\_signal;  end  end  // Day\_done  // Day\_done 신호를 플립플롭으로 설계  always @(posedge clk or negedge n\_rst)  if(!n\_rst)  Day\_done <= 1'b0;  else begin  // c\_state가 DONE 일때 Day\_done 신호를 내보낸다.  if(c\_state == DONE) begin  Day\_done <= 1'b1;  End  // c\_state가 DONE이 아닐 때는 Day\_done 신호를 0으로 둔다.  else  Day\_done <= 1'b0;  end  endmodule |
| //Byte2ascii 코드 주석  module byte2ascill(  input clk,  input n\_rst,  input day\_done, // controlller 하루 완료 신호 입력  input tx\_done, // 전송 완료 신호 입력  output done, // 완료 신호 출력  output reg[7:0] data\_out // 출력 데이터, 8비트  );  // 상태 매크로 정의  parameter IDLE = 4'h0; // 초기 상태  parameter S1 = 4'h1; // 첫 번째 문자 'D' 전송 상태  parameter W1 = 4'h2; // 첫 번째 문자 전송 대기 상태  parameter S2 = 4'h3; // 두 번째 문자 'o' 전송 상태  parameter W2 = 4'h4; // 두 번째 문자 전송 대기 상태  parameter S3 = 4'h5; // 세 번째 문자 'n' 전송 상태  parameter W3 = 4'h6; // 세 번째 문자 전송 대기 상태  parameter S4 = 4'h7; // 네 번째 문자 'e' 전송 상태  parameter W4 = 4'h8; // 네 번째 문자 전송 대기 상태  // 현재 상태와 다음 상태 레지스터 선언  reg [3:0] c\_state, n\_state;  reg uart\_start;  // 상태 레지스터의 업데이트: 비동기 리셋 처리 포함  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  c\_state <= IDLE; // 리셋 시 초기 상태로 전환  end  else begin  c\_state <= n\_state; // 현재 상태를 다음 상태로 업데이트  end  // 데이터 출력 레지스터의 업데이트  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  data\_out <= 8'h00; // 리셋 시 데이터 출력 초기화  end  else begin  if(c\_state == S1)  data\_out <= 8'h44; // 상태 S1에서 문자 'D' 출력  else if(c\_state == S2)  data\_out <= 8'h6F; // 상태 S2에서 문자 'o' 출력  else if(c\_state == S3)  data\_out <= 8'h6E; // 상태 S3에서 문자 'n' 출력  else if(c\_state == S4)  data\_out <= 8'h65; // 상태 S4에서 문자 'e' 출력  else  data\_out <= data\_out; // 다른 경우에는 이전 값 유지  end  // 상태 천이 로직 정의  always @(\*)  case(c\_state)  IDLE : begin  n\_state = (day\_done == 1'b1) ? S1 : c\_state; // day\_done==1 되면 상태 S1로 전환  end  S1 : begin  n\_state = (uart\_start == 1'b1) ? W1 : c\_state; // uart\_start==1 상태 W1로 전환  end  W1 : begin  n\_state = (tx\_done == 1'b1) ? S2 : c\_state; // tx\_done==1 되면 상태 S2로 전환  end  S2 : begin  n\_state = (uart\_start == 1'b1) ? W2 : c\_state; // uart\_start==1 되면 상태 W2로 전환  end  W2 : begin  n\_state = (tx\_done == 1'b1) ? S3 : c\_state; // tx\_done==1 되면상태 S3로 전환  end  S3 : begin  n\_state = (uart\_start == 1'b1) ? W3 : c\_state; // uart\_start==1 되면 상태 W3로 전환  end  W3 : begin  n\_state = (tx\_done == 1'b1) ? S4 : c\_state; // tx\_done==1 되면 상태 S4로 전환  end  S4 : begin  n\_state = (uart\_start == 1'b1) ? W4 : c\_state; // uart\_start==1 되면 상태 W4로 전환  end  W4: begin  n\_state = (tx\_done == 1'b1) ? IDLE : c\_state; // tx\_done==1 되면 상태 IDLE로 전환  end  default : begin  n\_state = IDLE; // 기본 상태는 IDLE  end  endcase  // UART 시작 신호 제어  always @(\*) begin  if(c\_state == S1 || c\_state == S2 || c\_state == S3 || c\_state == S4)  uart\_start = 1'b1; // 상태 S1, S2, S3, S4에서는 uart\_start 신호 활성화  else  uart\_start = 1'b0; // 다른 상태에서는 uart\_start 신호 비활성화  end  // 완료 신호 출력  assign done = uart\_start; // 완료 신호는 uart\_start 신호와 동일  endmodule |
| //TotalFND 코드 주석  module Total\_FND (  input wire morning\_signal, // 아침 신호 입력  input wire after\_signal, // 오후 신호 입력  output reg [6:0] fnd1, // 첫 번째 FND 출력 (7세그먼트)  output reg [6:0] fnd2, // 두 번째 FND 출력 (7세그먼트)  output reg [6:0] fnd3 // 세 번째 FND 출력 (7세그먼트)  );  // 7세그먼트 디스플레이에 표시할 문자들의 매크로 정의  parameter [6:0] CHAR\_A = 7'b000\_1000; // 문자 'A'에 해당하는 7세그먼트 디스플레이 값  parameter [6:0] CHAR\_P = 7'b000\_1100; // 문자 'P'에 해당하는 7세그먼트 디스플레이 값  parameter [6:0] CHAR\_MINUS = 7'b011\_1111; // 문자 '-'에 해당하는 7세그먼트 디스플레이 값  // 조합 논리 블록: FND 출력 결정  always @(\*) begin  if (morning\_signal) begin  // 아침 신호가 활성화된 경우  fnd1 <= CHAR\_A; // 첫 번째 FND에 'A' 표시  fnd2 <= CHAR\_MINUS; // 두 번째 FND에 '-' 표시  fnd3 <= CHAR\_P; // 세 번째 FND에 'P' 표시  end else if (after\_signal) begin  // 오후 신호가 활성화된 경우  fnd1 <= CHAR\_P; // 첫 번째 FND에 'P' 표시  fnd2 <= CHAR\_MINUS; // 두 번째 FND에 '-' 표시  fnd3 <= CHAR\_A; // 세 번째 FND에 'A' 표시  end else begin  // 어느 신호도 활성화되지 않은 경우  fnd1 <= 7'h7f; // 첫 번째 FND에 아무것도 표시하지 않음 (비활성화)  fnd2 <= 7'h7f; // 두 번째 FND에 아무것도 표시하지 않음 (비활성화)  fnd3 <= 7'h7f; // 세 번째 FND에 아무것도 표시하지 않음 (비활성화)  end  end  endmodule |
| //Controller 코드 주석  module Controller(  input clk,  input n\_rst,  input bt\_start,  input bt\_setting,    input [7:0] rx\_data,  input rx\_done,  input [7:0] led\_data,  input spi\_done,  output reg motor\_signal,  output reg heat\_signal,  output reg [7:0] led\_out,  output reg morning\_signal,  output reg after\_signal,  output reg Day\_done);  `ifdef SIM  parameter ONE\_SECOND = 10;  `else  // 50 MHz clock -> (1/(d5208)) -> 9,600 rate  parameter ONE\_SECOND = 50\_000\_000;  `endif  parameter IDLE = 4'h0;  parameter LIGHT\_READ = 4'h1;  parameter FND1 = 4'h2;  parameter SETTING = 4'h3;  parameter RUNNING = 4'h4;  parameter LIGHT\_READ\_2 = 4'h5;  parameter FND2 = 4'h6;  parameter SETTING\_2 = 4'h7;  parameter RUNNING\_2 = 4'h8;  parameter DONE = 4'h9;  reg [3:0] c\_state, n\_state;  // RUNNING ⑤객臾띄솒됱굚蹂뜟  reg [25:0] sec\_counter;  reg [3:0] runnig\_time;  // 1Day 타이밍 로직  // RUNNING 상태 타이머 및 리셋 로직  always @(posedge clk or negedge n\_rst) begin  if (!n\_rst) begin  sec\_counter <= 26'h0000;  runnig\_time <= 3'h0;  end else begin  // c\_state가 RUNNING 이거나 RUNNING\_2일 때 counter로 초 계산  if (c\_state == RUNNING || c\_state == RUNNING\_2) begin  if (sec\_counter < ONE\_SECOND - 1) begin  sec\_counter <= sec\_counter + 26'h0001;  end else begin  sec\_counter <= 26'h0000;  if (runnig\_time < 3'h5)  runnig\_time <= runnig\_time + 3'h1;  end  end else begin  sec\_counter <= 26'h0000; // RUNNING 상태가 아닐 때 sec\_counter 리셋  runnig\_time <= 3'h0;  end  end  end  // c\_state 초기화  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  c\_state <= IDLE;  end else begin  c\_state <= n\_state;  end  // 다음 state로 넘어가는 로직  always @(\*)  case(c\_state)  IDLE : begin  n\_state = (bt\_start == 1'b1) ? LIGHT\_READ : c\_state;  end  LIGHT\_READ : begin  n\_state = (spi\_done == 1'b1) ? FND1 : c\_state;  end  FND1 : begin  n\_state = SETTING;  end  SETTING : begin  n\_state = (bt\_setting == 1'b1) ? RUNNING : c\_state;  end  RUNNING : begin  n\_state = (runnig\_time == 3'h5) ? LIGHT\_READ\_2 : c\_state;  end  LIGHT\_READ\_2 : begin  n\_state = (spi\_done == 1'b1) ? FND2 : c\_state;  end  FND2 : begin  n\_state = SETTING\_2;  end  SETTING\_2 : begin  n\_state = (bt\_setting == 1'b1) ? RUNNING\_2 : c\_state;  end  RUNNING\_2 : begin  n\_state = (runnig\_time == 3'h5) ? DONE : c\_state;  end  DONE : begin  n\_state = IDLE;  end  default : begin  n\_state = IDLE;  end  endcase  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // SPI 완료 신호를 받고 LED 데이터를 반전하여 led\_out으로 출력  always @(posedge clk or negedge n\_rst)  if(!n\_rst)  led\_out <= 8'h00;  else begin  if(c\_state == LIGHT\_READ || c\_state == LIGHT\_READ\_2) begin  if(spi\_done == 1'b1) begin  led\_out <= ~led\_data;  end  end  end  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // (SETTING & MOTOR)  // motor\_signal과 heat\_signal을 state(SETTING거나 SETTING\_2일 때)에 따라 bt\_setting이 1일 때 running\_time이 5보다 작을 때 1이 할당  always @(posedge clk or negedge n\_rst) begin  if (!n\_rst) begin  motor\_signal <= 1'b0;  heat\_signal <= 1'b0;  end else begin  if (c\_state == SETTING || c\_state == SETTING\_2) begin  if (bt\_setting == 1'b1) begin  motor\_signal <= (runnig\_time < 3'h5) ? 1'b1 : 1'b0;  heat\_signal <= (runnig\_time < 3'h5) ? 1'b1 : 1'b0;  end else begin  motor\_signal <= 1'b0;  heat\_signal <= 1'b0;  end  // running\_time이 5가 되면(5초가 지나면) motor\_signal 과 heat\_signal에 0으로 초기화  end else if (runnig\_time == 3'h5) begin  motor\_signal <= 1'b0;  heat\_signal <= 1'b0;  end else begin  motor\_signal <= motor\_signal;  heat\_signal <= heat\_signal;  end  end  end  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // FND로 낮인지 밤인지 구별해주기  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  morning\_signal <= 1'b0;  after\_signal <= 1'b0;  end  // morning\_signal이 1일 때 FND에 A-P display  // after\_signal이 1일 때 FND에 P-A display  else begin  if(c\_state == FND1) begin  morning\_signal <= 1'b1;  after\_signal <= 1'b0;  end  else if(c\_state == FND2) begin  after\_signal <= 1'b1;  morning\_signal <= 1'b0;  end  else if(runnig\_time == 3'h5) begin  after\_signal <= 1'b0;  morning\_signal <= 1'b0;  end  else begin  after\_signal <= after\_signal;  morning\_signal <= morning\_signal;  end  end  // Day\_done  // Day\_done 신호를 플립플롭으로 설계  always @(posedge clk or negedge n\_rst)  if(!n\_rst)  Day\_done <= 1'b0;  else begin  // c\_state가 DONE 일때 Day\_done 신호를 내보낸다.  if(c\_state == DONE) begin  Day\_done <= 1'b1;  End  // c\_state가 DONE이 아닐 때는 Day\_done 신호를 0으로 둔다.  else  Day\_done <= 1'b0;  end  endmodule |
| //Byte2ascii 코드 주석  module byte2ascill(  input clk,  input n\_rst,  input day\_done, // controlller 하루 완료 신호 입력  input tx\_done, // 전송 완료 신호 입력  output done, // 완료 신호 출력  output reg[7:0] data\_out // 출력 데이터, 8비트  );  // 상태 매크로 정의  parameter IDLE = 4'h0; // 초기 상태  parameter S1 = 4'h1; // 첫 번째 문자 'D' 전송 상태  parameter W1 = 4'h2; // 첫 번째 문자 전송 대기 상태  parameter S2 = 4'h3; // 두 번째 문자 'o' 전송 상태  parameter W2 = 4'h4; // 두 번째 문자 전송 대기 상태  parameter S3 = 4'h5; // 세 번째 문자 'n' 전송 상태  parameter W3 = 4'h6; // 세 번째 문자 전송 대기 상태  parameter S4 = 4'h7; // 네 번째 문자 'e' 전송 상태  parameter W4 = 4'h8; // 네 번째 문자 전송 대기 상태  // 현재 상태와 다음 상태 레지스터 선언  reg [3:0] c\_state, n\_state;  reg uart\_start;  // 상태 레지스터의 업데이트: 비동기 리셋 처리 포함  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  c\_state <= IDLE; // 리셋 시 초기 상태로 전환  end  else begin  c\_state <= n\_state; // 현재 상태를 다음 상태로 업데이트  end  // 데이터 출력 레지스터의 업데이트  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  data\_out <= 8'h00; // 리셋 시 데이터 출력 초기화  end  else begin  if(c\_state == S1)  data\_out <= 8'h44; // 상태 S1에서 문자 'D' 출력  else if(c\_state == S2)  data\_out <= 8'h6F; // 상태 S2에서 문자 'o' 출력  else if(c\_state == S3)  data\_out <= 8'h6E; // 상태 S3에서 문자 'n' 출력  else if(c\_state == S4)  data\_out <= 8'h65; // 상태 S4에서 문자 'e' 출력  else  data\_out <= data\_out; // 다른 경우에는 이전 값 유지  end  // 상태 천이 로직 정의  always @(\*)  case(c\_state)  IDLE : begin  n\_state = (day\_done == 1'b1) ? S1 : c\_state; // day\_done==1 되면 상태 S1로 전환  end  S1 : begin  n\_state = (uart\_start == 1'b1) ? W1 : c\_state; // uart\_start==1 상태 W1로 전환  end  W1 : begin  n\_state = (tx\_done == 1'b1) ? S2 : c\_state; // tx\_done==1 되면 상태 S2로 전환  end  S2 : begin  n\_state = (uart\_start == 1'b1) ? W2 : c\_state; // uart\_start==1 되면 상태 W2로 전환  end  W2 : begin  n\_state = (tx\_done == 1'b1) ? S3 : c\_state; // tx\_done==1 되면상태 S3로 전환  end  S3 : begin  n\_state = (uart\_start == 1'b1) ? W3 : c\_state; // uart\_start==1 되면 상태 W3로 전환  end  W3 : begin  n\_state = (tx\_done == 1'b1) ? S4 : c\_state; // tx\_done==1 되면 상태 S4로 전환  end  S4 : begin  n\_state = (uart\_start == 1'b1) ? W4 : c\_state; // uart\_start==1 되면 상태 W4로 전환  end  W4: begin  n\_state = (tx\_done == 1'b1) ? IDLE : c\_state; // tx\_done==1 되면 상태 IDLE로 전환  end  default : begin  n\_state = IDLE; // 기본 상태는 IDLE  end  endcase  // UART 시작 신호 제어  always @(\*) begin  if(c\_state == S1 || c\_state == S2 || c\_state == S3 || c\_state == S4)  uart\_start = 1'b1; // 상태 S1, S2, S3, S4에서는 uart\_start 신호 활성화  else  uart\_start = 1'b0; // 다른 상태에서는 uart\_start 신호 비활성화  end  // 완료 신호 출력  assign done = uart\_start; // 완료 신호는 uart\_start 신호와 동일  endmodule |
| //TotalFND 코드 주석  module Total\_FND (  input wire morning\_signal, // 아침 신호 입력  input wire after\_signal, // 오후 신호 입력  output reg [6:0] fnd1, // 첫 번째 FND 출력 (7세그먼트)  output reg [6:0] fnd2, // 두 번째 FND 출력 (7세그먼트)  output reg [6:0] fnd3 // 세 번째 FND 출력 (7세그먼트)  );  // 7세그먼트 디스플레이에 표시할 문자들의 매크로 정의  parameter [6:0] CHAR\_A = 7'b000\_1000; // 문자 'A'에 해당하는 7세그먼트 디스플레이 값  parameter [6:0] CHAR\_P = 7'b000\_1100; // 문자 'P'에 해당하는 7세그먼트 디스플레이 값  parameter [6:0] CHAR\_MINUS = 7'b011\_1111; // 문자 '-'에 해당하는 7세그먼트 디스플레이 값  // 조합 논리 블록: FND 출력 결정  always @(\*) begin  if (morning\_signal) begin  // 아침 신호가 활성화된 경우  fnd1 <= CHAR\_A; // 첫 번째 FND에 'A' 표시  fnd2 <= CHAR\_MINUS; // 두 번째 FND에 '-' 표시  fnd3 <= CHAR\_P; // 세 번째 FND에 'P' 표시  end else if (after\_signal) begin  // 오후 신호가 활성화된 경우  fnd1 <= CHAR\_P; // 첫 번째 FND에 'P' 표시  fnd2 <= CHAR\_MINUS; // 두 번째 FND에 '-' 표시  fnd3 <= CHAR\_A; // 세 번째 FND에 'A' 표시  end else begin  // 어느 신호도 활성화되지 않은 경우  fnd1 <= 7'h7f; // 첫 번째 FND에 아무것도 표시하지 않음 (비활성화)  fnd2 <= 7'h7f; // 두 번째 FND에 아무것도 표시하지 않음 (비활성화)  fnd3 <= 7'h7f; // 세 번째 FND에 아무것 |
| // uart\_tx 주석  // clock : 50 MHz  // (50M/9.6k) = 5208.3333 -> 5208 => h1458  //`define SIM  //`define CP  module uart\_tx(  clk,  n\_rst, // active low push button  baudrate, // 0 : 9600, 1 : 19200...  start, // active 1  din, // switch  done,  uart\_txd // UART TX DATA  );  `ifdef SIM  parameter T\_DIV\_BIT = 4; // 2-bit  parameter T\_DIV\_0 = 4'd15; // 0-15 : 16 // 50 MHz clock -> 9,600 rate  parameter T\_DIV\_HALF\_0 = 4'd7; // 0- 7 : 8  parameter T\_DIV\_1 = 4'd7; // 0- 7 : 8 // 50 MHz clock -> 9,600 rate  parameter T\_DIV\_HALF\_1 = 4'd3; // 0- 3 : 4  `else  // 50 MHz clock -> (1/(d5208)) -> 9,600 rate  parameter T\_DIV\_BIT = 13; // 5207 : 13-bit  parameter T\_DIV\_0 = 13'd5207; // 0-5207 : 5208 // 50 MHz clock -> 9,600 rate  parameter T\_DIV\_HALF\_0 = 13'd2603; // 5208/2 = 2604 // 50 MHz clock -> 9,600 rate  parameter T\_DIV\_1 = 13'd5207; // 0-2603 : 2604 // 50 MHz clock -> 19,200 rate  parameter T\_DIV\_HALF\_1 = 13'd1301; // 2604/2 = 1302 // 50 MHz clock -> 19,200 rate  `endif  input clk;  input n\_rst;  input baudrate; // 0 : 9600, 1 : 19200...  input start;  input [7:0] din;  output done;  output uart\_txd;  // ---------------------------------------------------  // UART - TX  // ---------------------------------------------------  reg tx\_en;  reg [9:0] tx\_data;  reg [T\_DIV\_BIT-1:0] cnt\_tx\_div;  wire [T\_DIV\_BIT-1:0] t\_div; // (50M/9.6k)  wire clk\_tx\_en;  assign t\_div = (baudrate == 1'b1)? T\_DIV\_1 : T\_DIV\_0;  assign clk\_tx\_en = (cnt\_tx\_div == t\_div)? 1'b1 : 1'b0;  reg [3:0] cnt\_bit; // cnt 11 : 0-10  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  tx\_en <= 1'b0;  end  else begin  tx\_en <= (start == 1'b1)? 1'b1 :  (clk\_tx\_en == 1'b1 && cnt\_bit == 4'hb)? 1'b0 : tx\_en;  end  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  cnt\_tx\_div <= {(T\_DIV\_BIT){1'b0}};  end  else begin  if (tx\_en == 1'b1) begin  cnt\_tx\_div <= (clk\_tx\_en == 1'b1)? {(T\_DIV\_BIT){1'b0}} :  cnt\_tx\_div + {{(T\_DIV\_BIT-1){1'b0}},1'b1};  end  else begin  cnt\_tx\_div <= {(T\_DIV\_BIT){1'b0}};  end  end  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  cnt\_bit <= 4'd0;  end  else begin  if (tx\_en == 1'b1) begin  if (clk\_tx\_en == 1'b1)  cnt\_bit <= cnt\_bit + 4'd1;  end  else begin  cnt\_bit <= 4'd0;  end  end  // tx register shifting out  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  tx\_data <= 10'h3FF;  end  else begin  if (tx\_en == 1'b1) begin  if (clk\_tx\_en == 1'b1) begin // tx\_en == 1'b1  if (cnt\_bit == 4'h0)  tx\_data <= {1'b0, din, 1'b0}; // 1-bit parity, 8-bit data, 1-bit start  else  tx\_data <= {1'b1, tx\_data[9:1]}; // stop bit = 1  end  end  else begin  tx\_data <= 10'h3FF;  end  end  assign uart\_txd = tx\_data[0];  assign done = (cnt\_bit == 4'hc)? 1'b1 : 1'b0;  endmodule |

|  |
| --- |
| // spi\_master\_adc 주석  module spi\_master\_adc(  input clk,  input n\_rst,  input n\_start,  input sdata,  output reg sclk,  output reg cs\_n,  output done,  output [7:0] led  );  reg [7:0] data\_buffer;  reg sclk\_rise;  reg [4:0] cnt;  reg [4:0] cnt\_sclk;  reg n\_start\_0d1;  reg n\_start\_0d2;  reg w\_n\_start;  // 아래 두 플립플롭은 start 신호에 대한 edge detection을 수행하는 코드이다.  // 신호를 지연시켜주어 두 신호의 값이 (n\_start\_0d1 == 1'b0) && (n\_start\_0d2 == 1'b1) 에 해당하면 w\_n\_start를 0으로 두어  // cs\_n을 작동할 signal을 만들어준다.  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  n\_start\_0d1 <= 1'b1;  n\_start\_0d2 <= 1'b1;  end  else begin  n\_start\_0d1 <= n\_start;  n\_start\_0d2 <= n\_start\_0d1;  end  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  w\_n\_start <= 1'b1;  end  else begin  w\_n\_start <= ((n\_start\_0d1 == 1'b0) && (n\_start\_0d2 == 1'b1)) ? 1'b0 : 1'b1;  end    // cs\_n 신호는 start 신호가 들어올 때 나타나도록 설정한다. 이때 들어오는 start 신호는 edge detection이 된 신호여야 한다.  // start 신호가 들어오면 cs\_n 신호를 0으로 둔다 만일 cnt\_sclk의 값이 16이라면 data 전달을 마친 상태이기 때문에 다시 cs\_n을 1로 설정해준다.  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  cs\_n <= 1'b1;  end  else begin  cs\_n <= (w\_n\_start == 1'b0) ? 1'b0 : (cnt\_sclk == 5'h10) ? 1'b1 : cs\_n;  end  // cnt 만들기  // cs\_n이 low인 상황에서 작동하도록 설계했다.  // 25보다 작으면 카운트를 계속 해주고 26이 되는 순간 cnt를 0으로 만들어준다. 이외의 상황은 cnt 값을 유지해준다  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  cnt <= 5'h0;  end  else begin  cnt <= (cs\_n == 1'b1) ? 5'h00 : (cnt <= 5'h19) ? cnt + 5'h01 : (cnt == 5'h1a) ? 5'h00 : cnt;  end  // sclk의 high low 만들기  // 앞서 만든 cnt를 이용하여 sclk를 만들어준다. 0부터 12까지는 1을 유지해주고, 13부터 25까지는 0을 유지해준다.  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  sclk <= 1'b1;  end  else begin  if(cnt >= 5'h00 && cnt <= 5'h0b) begin  sclk <= 1'b1;  end  else if(cnt >= 5'h0c && cnt <= 5'h19) begin  sclk <= 1'b0;  end  else  sclk <= sclk;  end  // sclk\_rise 만들기  // cnt가 25인 지점에서 sclk가 posedge임을 나타내 주기 위한 신호를 만들었다.  // 25이면 1의 값을 갖고 1이 아닌 상황에서는 0을 주었다.  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  sclk\_rise <= 1'b0;  end  else begin  if(cnt == 5'h19) begin  sclk\_rise <= 1'b1;  end  else  sclk\_rise <= 1'b0;  end  // cnt\_sclk counter  // sclk\_rise가 1인 순간에만 15까지(0 3개 + data 8개 + 0 4개) 값을 1 증가시키도록 한다.  // 만일 값이 16이 되면 cnt\_sclk 값을 0으로 둔다.  always @(posedge clk or negedge n\_rst)  if(!n\_rst) begin  cnt\_sclk <= 5'h00;  end  else begin  if(sclk\_rise == 1'b1 && cnt\_sclk <= 5'hf) begin  cnt\_sclk <= cnt\_sclk + 5'h01;  end  else if(cnt\_sclk == 5'h10) begin  cnt\_sclk <= 5'h00;  end  else  cnt\_sclk <= cnt\_sclk;  end  // data\_buffer에 값 넣기  // sclk가 posedge에서 동작하는 것처럼 설계하기 위해서 sclk\_rise 신호를 이용했다.  // cnt\_sclk는 data의 전송 상태를 정해주기 위해 만든 카운터 이므로 해당 값을 이용하여 0부터 10 사이에서는 값을 시프트 해주면서 넣어주고 그 이후의 값이 오면 값을 유지해준다.  always @(posedge clk or negedge n\_rst)  if(!n\_rst)begin  data\_buffer <= 8'h00;  end  else begin  data\_buffer <= (sclk\_rise == 1'b1) ? (cnt\_sclk >= 5'h03 && cnt\_sclk <= 5'h0a) ? {data\_buffer[6:0], sdata} : data\_buffer : data\_buffer;  end  // data\_buffer에 led 값을 넣어준다.  assign led = data\_buffer;  assign done = (cnt\_sclk == 5'h10) ? 1'b1 : 1'b0;  endmodule |

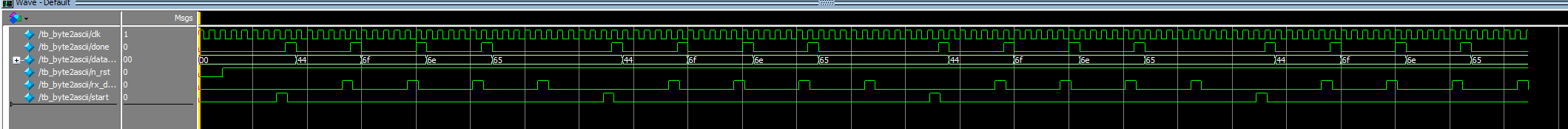
|  |
| --- |
| module top(  input clk,  input n\_rst,  input bt\_start,  input push\_spi\_start,  input bt\_setting,  output sclk,  output cs\_n,  input sdata,  output [7:0] led\_out, // LED 출력  output heat\_signal, // 난방 소자 제어 신호  output motor\_signal, // 모터 제어 신호  output txd, // UART 전송 데이터 출력  output [6:0] fnd1, // FND1 출력  output [6:0] fnd2, // FND2 출력  output [6:0] fnd3 // FND3 출력  );  `ifdef SIM  parameter ONE\_SECOND = 10; // 시뮬레이션을 위한 1초 파라미터  parameter T\_DIV\_BIT = 4; // UART 디바이더 비트  parameter T\_DIV\_0 = 4'd15; // UART 디바이더 값 (9600 baud rate)  parameter T\_DIV\_HALF\_0 = 4'd7; // UART 디바이더 반값  parameter T\_DIV\_1 = 4'd7; // UART 디바이더 값 (19200 baud rate)  parameter T\_DIV\_HALF\_1 = 4'd3; // UART 디바이더 반값  `else  parameter ONE\_SECOND = 50\_000\_000; // 실제 하드웨어를 위한 1초 파라미터  parameter T\_DIV\_BIT = 13; // UART 디바이더 비트  parameter T\_DIV\_0 = 13'd5207; // UART 디바이더 값 (9600 baud rate)  parameter T\_DIV\_HALF\_0 = 13'd2603; // UART 디바이더 반값  parameter T\_DIV\_1 = 13'd5207; // UART 디바이더 값 (19200 baud rate)  parameter T\_DIV\_HALF\_1 = 13'd1301; // UART 디바이더 반값  `endif  // 내부 신호 정의  wire done; // SPI 완료 신호  wire tx\_done; // UART 전송 완료 신호  wire [7:0] tx\_data; // UART 전송 데이터  wire inv\_txd; // 인버트된 UART 전송 데이터  wire [7:0] rx\_data; // UART 수신 데이터  wire rx\_done; // UART 수신 완료 신호  wire baudrate; // UART baudrate 선택 신호  assign baudrate = 1'b0; // 기본 baudrate 설정  // FND 제어 신호  wire morning\_signal; // 오전 신호  wire after\_signal; // 오후 신호  wire Day\_done; // 하루 완료 신호  // LED 데이터  wire [7:0] led;  // 시작 버튼 에지 디텍터  wire w\_start;  reg d\_start;  reg d2\_start;  always @(posedge clk or negedge n\_rst)  if (!n\_rst) begin  d\_start <= 1'b0;  d2\_start <= 1'b0;  end else begin  d\_start <= ~bt\_start;  d2\_start <= d\_start;  end  assign w\_start = ((d\_start == 1'b1) && (d2\_start == 1'b0)) ? 1'b1 : 1'b0;  // 설정 버튼 에지 디텍터  wire w\_bt\_setting;  reg d\_setting;  reg d2\_setting;  always @(posedge clk or negedge n\_rst)  if (!n\_rst) begin  d\_setting <= 1'b0;  d2\_setting <= 1'b0;  end else begin  d\_setting <= ~bt\_setting;  d2\_setting <= d\_setting;  end  assign w\_bt\_setting = ((d\_setting == 1'b1) && (d2\_setting == 1'b0)) ? 1'b1 : 1'b0;  // Controller 모듈 인스턴스  Controller u\_Controller(  .clk(clk),  .n\_rst(n\_rst),  .bt\_start(w\_start),  .bt\_setting(w\_bt\_setting),  .rx\_data(rx\_data),  .rx\_done(rx\_done),  .led\_data(led),  .spi\_done(done),  .led\_out(led\_out),  .heat\_signal(heat\_signal),  .motor\_signal(motor\_signal),  .morning\_signal(morning\_signal),  .after\_signal(after\_signal),  .Day\_done(Day\_done)  );  // SPI 마스터 모듈 인스턴스  spi\_master\_adc u\_spi\_master\_adc(  .clk(clk),  .n\_rst(n\_rst),  .n\_start(push\_spi\_start),  .sdata(sdata),  .sclk(sclk),  .cs\_n(cs\_n),  .done(done),  .led(led)  );  // byte2ascii 모듈 인스턴스  wire b2a\_start;  byte2ascill u\_byte2ascill(  .clk(clk),  .n\_rst(n\_rst),  .day\_done(Day\_done),  .tx\_done(tx\_done),  .done(b2a\_start),  .data\_out(tx\_data)  );  // UART 전송 신호 연결  wire rxd;  assign txd = ~inv\_txd;  // UART 전송 모듈 인스턴스  uart\_tx #(  .T\_DIV\_BIT(T\_DIV\_BIT),  .T\_DIV\_0(T\_DIV\_0),  .T\_DIV\_HALF\_0(T\_DIV\_HALF\_0),  .T\_DIV\_1(T\_DIV\_1),  .T\_DIV\_HALF\_1(T\_DIV\_HALF\_1)  ) u\_uart\_tx(  .clk(clk),  .n\_rst(n\_rst),  .baudrate(baudrate),  .start(b2a\_start),  .din(tx\_data),  .done(tx\_done),  .uart\_txd(inv\_txd)  );  // FND 출력 모듈 인스턴스  Total\_FND u\_Total\_FND(  .morning\_signal(morning\_signal),  .after\_signal(after\_signal),  .fnd1(fnd1),  .fnd2(fnd2),  .fnd3(fnd3)  );  endmodule |

2.3 시뮬레이션 결과

(1) controlloer

스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

(2) byte2ascii

(3) uart\_tx

스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

(4) spi\_master\_adc

3. 결론

3.1 고찰 및 기대효과

본 프로젝트는 환경 변화에 따라 농작물의 생산량 차이를 줄이고자 사용자가 설정한 조건에 따라 농장이 유지되도록하는 작은 스마트팜을 verilog 기반의 FPGA를 제작하였다. 이를 통해 기대할 수 있는 효과로 첫 번째 노동력 절감이다. 우리가 제작한 스마트팜 또한 다양한 작업을 자동으로 수행하므로 농작업에 필요한 인력과 시간을 대폭 줄일 수 있다. 이는 노동력이 부족한 농촌 지역의 발전에도 기여함을 기대한다. 두 번째로 기후 변화에 따라 농작물의 질이 달라지는 것이 아니라 사용자가 설정한 값이 기후에 따라 변경할 수 있기 때문에 농작물의 품질을 향상시킬 수 있다. 이는 시작 경쟁력 향상과 함께 소비자 만족도를 높이는 효과가 있다. 다음으로 지속 가능한 농업 실현이다. 환경 변화에 유연하게 대응하면서도 자원을 효율적으로 사용하여 지속 가능한 농업을 실현할 수 있다. 이는 장기적으로 농업 환경의 보전과 지속 가능성에 기여한다. 본 프로젝트는 스마트팜이 우리 사회에 주는 이점을 활용하여 FPGA 보드를 통해 구현하였다. 뿐만 아니라 우리 사회는 빅데이터를 활용하여 수집된 데이터를 분석하고 작물 생육과 환경 조건의 상관 관계를 파악하여 이를 바탕으로 예측 모델을 만드는 것에 대한 추후 연구가 필요한 상황이다.

3.2 참고문헌

Choi, Joon Sung, & Park, Cheon-bo, (2019). A Study of the Influences of the Korean Green Standard for Energy and Environmental Design for Apartment Housing on Climate Change Preparation. The Regional Association of Architectural Institute of Korea, 21(2), 117 – 124.

Jeong, Chang Seok, (2019). Measures to Activate Plant Factories for the Improvement of Citizens’ life. International School of Urban Sciences University of Seoul, MA.

Kim, Junsung, Kim, Yongsung & Lee, Namgyu, (2021). A Study on the Three-Dimensional Combination Plan for the Activation of Smart Farms for Apartment Households. Journal of the Korea Institute of Spatial Design, 16(8), 49-60.