ECE4144 – Test 1

Name: Frank Li | **ID:** jl13581

The code for the midterm 1 is pasted below. Also, the main.cpp is uploaded:

```
#include <Arduino.h>

void setup() {
  // Set up GPIO
  DDRD |= (1<<PD0); //  Bay Lock Valve    - PD0 - Output
  DDRD |= (1<<PD1); //  Ocean Lock Valve  - PD1 - Output
  DDRD |= (1<<PD2); //  Bay Lock          - PD2 - Output
  DDRD |= (1<<PD3); //  Ocean Lock        - PD3 - Output
  DDRD &= ~(1<<PD4); // Bay Lock Status   - PD4 - Input
  DDRD &= ~(1<<PD5); // Ocean Lock Status - PD5 - Input
  DDRD &= ~(1<<PD6); // Transition Status - PD6 - Input

  // Make sure all locks and valves are closed at the beginning
  PORTD &= ~(1<<PORTD0); // Bay Lock Valve
  PORTD &= ~(1<<PORTD1); // Ocean Lock Valve
  PORTD &= ~(1<<PORTD2); // Bay Lock
  PORTD &= ~(1<<PORTD3); // Ocean Lock

  // Set up the ADC
  ADCSRA = (1<<ADEN) | (1<<ADATE) |
          (1<<ADPS2) | (1<<ADPS1) |
          (1<<ADPS0);// ADC Enable; Auto Trigger Enable; Prescaler 128
                     // ADCSRB at Free Running Mode, thus 0000
}

/*
  Check if the levels of the bay and the transition waterway are the same.
  OUTPUT: 1 - same; 0 - not same.
*/
bool checkBayTransitionLevel(){
  const uint16_t difference = 5; // Reasonable differences between water
                                    levels to consider as the same.
  const uint16_t timeout = 10000; // 10s to Timeout
  uint32_t start = millis();

  while(true){
    // Set up ADC0 - Bay Level
    uint16_t bay_lv = readADC(0); // Bay water level

    // Set up ADC1 - Transition Level.
    uint16_t transition_lv = readADC(1); // Transition waterway water lavel

    // Check if bay water level equals to transition waterway level.
       If same, return true; else, continue the loop to check.
    if(abs((bay_lv - transition_lv)) <= difference){
      return true;
    }

    // Timeout Checking
    if((millis() - start) >= timeout){
      return false;
    }
    delay(100);
  }
}

/*
  Check if the levels of the the transition waterway and ocean are the same.
  OUTPUT: 1 - same; 0 - not same.
*/
bool checkTransitionOceanLevel(){
  const uint16_t difference = 5; // Reasonable differences between water levels
                                    to consider as the same.
  const uint16_t timeout = 10000; // 10s to Timeout
  uint16_t start = millis();

  while(true){
    // Set up ADC1 - Transition Level
    uint16_t transition_lv = readADC(1); // Transition waterway water lavel

    // Set up ADC2 - Ocean Level
    uint16_t ocean_lv = readADC(2); // Ocean water lavel

    // Check if ocean water level equals to transition waterway level.
       If same, return true; else, continue the loop to check.
    if (abs((ocean_lv - transition_lv)) <= difference){
        return true;
      }

    // Timeout Checking
    if((millis() - start) >= timeout){
      return false;
    }

    delay(100);
  }
}

/*
  Checking if the transition waterway is occupied or not.
  OUTPUT: 1 - occupied; 0 - not occupied.
*/
bool checkIsTransitionOccupied(){
  // If the transition is occupied, return true; else, continue the
     loop to check.
    if(PIND & (1<<PIND6)){
      return true;
    }else{
      return false;
    }
}

/*
  Switch between ADC channels and read data.
  OUTPUT: ADCW data from ADC, 16-bit.
  @param cs chip/channel select
*/
```

```
uint16_t readADC(uint8_t cs){
  if(cs == 0){
    ADMUX = (1<<REFS0);// Set reference and channel
    DIDR0 = (1<<ADC0D);
    ADCSRA |= (1 << ADSC); // Start conversion
    return ADCW; // Return the ADC value
  }else{
    ADMUX = (1<<REFS0) | (1<<cs); // Set reference and channel
    DIDR0 = (1<<cs);
    ADCSRA |= (1 << ADSC); // Start conversion
    return ADCW; // Return the ADC value
  }
}

/*
  Open the Bay Lock Valve
*/
bool openBayValve(){
  PORTD |= (1<<PORTD0); // Open the bay lock valve
  delay(100);

  // Check if the Bay Valve is opened successfully
  if(PORTD & (1<<PD0)){
    return true;
  }
  return false;
}

/*
  Close the Bay Lock Valve
*/
bool closeBayValve(){
  PORTD &= ~(1<<PORTD0); // Close the bay lock valve
  delay(100);

  // Check if the Bay Valve is closed successfully
  if(!(PORTD & (1<<PD0))){
    return true;
  }
  return false;
}

/*
  Open the Ocean Lock Valve
*/
bool openOceanValve(){
  PORTD |= (1<<PORTD1); // Open ocean valve
  delay(100);

  // Check if the Ocean Valve is opened successfully
  if(PORTD & (1<<PD1)){
    return true;
  }
  return false;
}
```

```
/*
  Close the Ocean Lock Valve
*/
bool closeOceanValve(){
  PORTD &= ~(1<<PORTD1); // Close ocean valve
  delay(100);

  // Check if the Ocean Valve is closed successfully
  if(!(PORTD & (1<<PD1))){
    return true;
  }
  return false;
}

/*
  Open the Bay Lock
*/
bool openBayLock(){
  PORTD |= (1<<PORTD2); // Open the Bay lock
  delay(100);

  // Close the Bay Valve
  if(closeBayValve() && (PIND & (1<<PIND4))){
    return true;
  }
  return false;
}

/*
  Close the Bay Lock
*/
bool closeBayLock(){
  PORTD &= ~(1<<PORTD2); // Close the Bay lock
  delay(100);

  // Check if the Bay Lock is closed successfully
  if(!(PIND & (1<<PIND4))){
    return true;
  }
  return false;
}

/*
  Open the Ocean Lock
*/
bool openOceanLock(){
  PORTD |= (1<<PORTD3); // Open the ocean lock
  delay(100);

  // Check if the Ocean Lock is opened successfully
  if(closeOceanValve() && (PIND & (1<<PIND5))){
    return true;
  }
  return false;
}
```

```
/*
  Close the Ocean Lock
*/
bool closeOceanLock(){
  PORTD &= ~(1<<PORTD3); // Close the ocean lock
  delay(100);

  // Check if the Ocean Lock is closed successfully
  if(!(PIND & (1<<PIND5))){
    return true;
  }
  return false;
}

/*
  Moving water from the Bay area, through the transition waterway,
  and finally reach the ocean area.
*/
uint8_t beginTransitBayToOcean(){
  // Either lock is open or a boat is already in transition
  if((PIND & (1<<PIND4)) || (PIND & (1<<PIND5)) || (PIND & (1<<PIND6))){
    return 0;
  }

  uint8_t step = 1; // Step counter

  /*
    Steps:  1. open bay valve
            2. check bay and transition water level
            2. open bay lock
            4. check transition occupied
            5. close bay lock
            6. open ocean valve
            7. check transition ocean water level
            8. open ocean lock
            9. check transition occupied
            10. close ocean lock
  */
  while(true){
    switch (step){
      case 1:
        if(openBayValve()){
          step = 2;
          break;
        }else{
          return 2;
        }
      case 2:
        if(checkBayTransitionLevel()){
          step = 3;
          break;
        }else{
          step = 2;
          break;
        }
      case 3:
```

```
  if(openBayLock()){
    step = 4;
    break;
  }else{
    return 2;
  }
case 4:
  if(checkIsTransitionOccupied()){
    step = 5;
    break;
  }else{
    step = 4;
    break;
  }
case 5:
  if(closeBayLock()){
    step = 6;
    break;
  }else{
    return 2;
  }
case 6:
  if(openOceanValve()){
    step = 7;
    break;
  }else{
    return 2;
  }
case 7:
  if(checkTransitionOceanLevel()){
    step = 8;
    break;
  }else{
    step = 7;
    break;
  }
case 8:
  if(openOceanLock()){
    step = 9;
    break;
  }else{
    return 2;
  }
case 9:
  if(!checkIsTransitionOccupied()){
    step = 10;
    break;
  }else{
    step = 9;
    break;
  }
case 10:
  if(closeOceanLock()){
    return 1;
  }else{
    return 2;
```

```
        }

      default:
        return 2;
    }

    delay(100);
  }
}

uint8_t beginTransitOceanToBay(){
  // Either lock is open or a boat is already in transition
  if ((PIND & (1<<PIND4)) || (PIND & (1<<PIND5)) || (PIND & (1<<PIND6))) {
    return 0;
  }

  uint8_t step = 1; // Step counter

  /*
    Steps:  1. open ocean valve
            2. check transition ocean water level
            2. open ocean lock
            4. check transition occupied
            5. close ocean lock
            6. open bay valve
            7. check transition bay water level
            8. open bay lock
            9. check transition occupied
            10. close bay lock
  */
  while(true){
    switch (step){
      case 1:
        if(openOceanValve()){
          step = 2;
          break;
        }else{
          return 2;
        }
      case 2:
        if(checkTransitionOceanLevel()){
          step = 3;
          break;
        }else{
          step = 2;
          break;
        }
      case 3:
        if(openOceanLock()){
          step = 4;
          break;
        }else{
          return 2;
        }
      case 4:
        if(checkIsTransitionOccupied()){
```

```
        step = 5;
        break;
      }else{
        step = 4;
        break;
      }
    case 5:
      if(closeOceanLock()){
        step = 6;
        break;
      }else{
        return 2;
      }
    case 6:
      if(openBayValve()){
        step = 7;
        break;
      }else{
        return 2;
      }
    case 7:
      if(checkTransitionOceanLevel()){
        step = 8;
        break;
      }else{
        step = 7;
        break;
      }
    case 8:
      if(openBayLock()){
        step = 9;
        break;
      }else{
        return 2;
      }
    case 9:
      if(!checkIsTransitionOccupied()){
        step = 10;
        break;
      }else{
        step = 9;
        break;
      }
    case 10:
      if(closeBayLock()){
        return 1;
      }else{
        return 2;
      }

    default:
      return 2;
  }

  delay(100);
}
```

```
}

void loop(){
  // Since there is no description on when to call the functions,
     I directly pasted two fucntions to the loop().
  beginTransitOceanToBay();
  beginTransitBayToOcean();
}
```