**INSTITUTO FEDERAL**
Minas Gerais
Campus Avançado Ipatinga

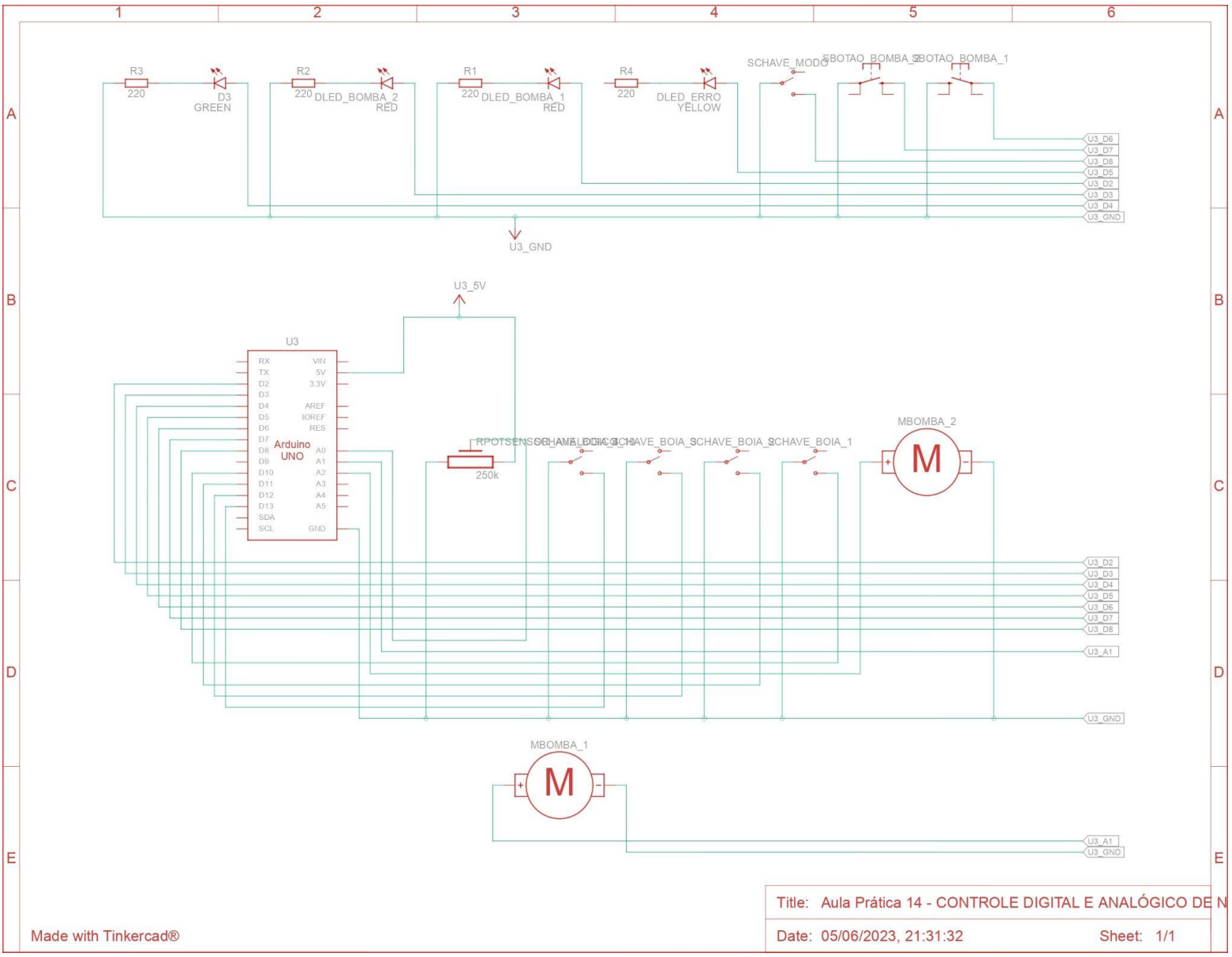
**IFMG – *Campus* Avançado Ipatinga**
**Ministério da Educação – Governo Federal**
Av. Maria Silva, 125, Veneza, Ipatinga / MG – CEP 35164-261.
Tel.: (31) 3829-8615 / Cel.: (31) 99734-7688 – http://www2.ifmg.edu.br/ipatinga

| | | |
|---|---|---|
| **CURSO:** Engenharia Elétrica | **TURNO:** Diurno | **TURMA:** _____ |
| **DISCIPLINA:** Microcontroladores | **NATUREZA DO TRABALHO:** Exercícios | **MÉDIA:** |
| **PROFESSOR:** Sandro Dornellas | **DATA:** ____/____/____ **VALOR:** | **NOTA:** |
| **ALUNO(A):** Adalberto Mariano A. Motta / Camila P. Faria / Samuel O. Caldas | | |

# Prática 14

## Desenvolvimento do Exercício

### Esquemático do circuito elaborado:



**Código:**

```
/**
 * @file Aula Prática 14.ino
 * @brief Aula Prática 14 - CONTROLE DIGITAL E ANALÓGICO DE NÍVEL DE RESERVATÓRIO
 * @version 1.0
 * @date 2023-06-03
 *
 * A. O controle de nível do reservatório deve ser realizado da seguinte forma:
 *    Quando o nível estiver em 50% a Bomba 1 deverá ligar até que o reservatório encha completamente.
 *    Se a Bomba 1 não conseguir manter o nível no reservatório, existe uma bomba de segurança (Bomba 2),
 *    que deve ser ligada apenas quando o nível estiver muito baixo (abaixo de 25%) e ela deve ser desligada
 *    somente quando o nível de água atingir 75%.
 * B. Deverá haver uma chave seletora, com retenção, para definir o modo Automático/Manual.
 *    Em Automático funcionará conforme descrito acima e, em modo Manual,
 *    dois botões pulsantes farão o controle de ligarem as bombas, ignorando os sensores de nível,
 *    ou seja, enquanto o botão 1 estiver pressionado a Bomba 1 deverá ficar ligada e enquanto o Botão 2 estiver pressionado a Bomba 2 deverá ficar ligada.
 * C. Um LED verde deverá ser usado para sinalização do modo Automático (aceso) / Manual (apagado).
 * D. Utilize botões/chaves com retenção para simular o acionamento das chaves boia e LEDs vermelhos para as bombas.
 *    Lembrando que quando o nível estiver em 100% todos as chaves deverão estar acionadas.
 * E. Se a chave do nível superior for acionada sem que a chave do nível inferior esteja acionada um LED amarelo deverá acender indicando o estado de erro.
 * F. Após concluída a primeira etapa, faça uma modificação no controle de nível substituindo os sensores digitais por um sensor analógico.
 *    Utilize um potenciômetro para simular o funcionamento do sensor analógico de nível.
 */

// Define constants for pin numbers
const int BOMBA_1_PIN = A1;
```

```
const int LED_BOMBA_1_PIN = 2;
const int BOTAO_BOMBA_1_PIN = 6;
const int BOMBA_2_PIN = A2;
const int LED_BOMBA_2_PIN = 3;
const int BOTAO_BOMBA_2_PIN = 7;
const int LED_MODO_AUTOMATICO_PIN = 4;
const int CHAVE_SELETORA_PIN = 8;
const int SENSOR_ANALOGICO_PIN = A0;
const int LED_ERRO_SENSOR_PIN = 5;
const int SENSOR_DIGITAL_PINS[] = {10, 11, 12, 13};

// Define constants for water level thresholds
const int NIVEL_BAIXO = 25;
const int NIVEL_MEDIO = 50;
const int NIVEL_ALTO = 75;
const int NIVEL_MAXIMO = 100;

// Define constants for debounce time
const unsigned long DEBOUNCE_DELAY = 50;

/**
 * @class Reservatorio
 * @brief Represents a water reservoir.
 *
 * This class provides methods to set and get the water level of the
reservoir.
 */
class Reservatorio
{
public:
  /**
   * @brief Constructor for the Reservatorio class.
   * @param nivel The initial water level of the reservoir.
   */
  Reservatorio(int nivel)
  {
    this->nivel = nivel;
  }

  /**
   * @brief Sets the water level of the reservoir.
   * @param nivel The new water level of the reservoir.
   */
```

```
  void setNivel(int nivel)
  {
    this->nivel = nivel;
  }

  /**
   * @brief Gets the current water level of the reservoir.
   * @return The current water level of the reservoir.
   */
  int getNivel()
  {
    return this->nivel;
  }

private:
  // Water level
  int nivel;
};

/**
 * @class LED
 * @brief Represents an LED.
 *
 * This class provides methods to turn on and off the LED.
 */
class LED
{
public:
  /**
   * @brief Constructor for the LED class.
   * @param pin The pin number of the LED.
   */
  LED(int pin)
  {
    this->pin = pin;
    pinMode(pin, OUTPUT);
    desligar();
  }

  /**
   * @brief Turns on the LED.
   */
  void ligar()
```

```
  {
    digitalWrite(pin, HIGH);
  }

  /**
   * @brief Turns off the LED.
   */
  void desligar()
  {
    digitalWrite(pin, LOW);
  }

private:
  // Pin number
  int pin;
};

/**
 * @class Bomba
 * @brief Represents a pump.
 *
 * This class provides methods to turn on and off the pump.
 */
class Bomba
{
public:
  /**
   * @brief Constructor for the Bomba class.
   * @param pin The pin number of the pump.
   * @param led A pointer to an LED object that is associated with the
pump.
   */
  Bomba(int pin, LED *led) : pin(pin), led(led)
  {
    pinMode(pin, OUTPUT);
    desligar();
  }

  /**
   * @brief Turns on the pump.
   */
  void ligar()
  {
```

```
    digitalWrite(pin, HIGH);
    led->ligar();
  }

  /**
   * @brief Turns off the pump.
   */
  void desligar()
  {
    digitalWrite(pin, LOW);
    led->desligar();
  }

private:
  // Pin number
  int pin;
  // Pointer to an LED object
  LED *led;
};

/**
 * @class Switch
 * @brief Represents a switch with debounce functionality.
 *
 * This class provides methods to check if the switch is on or off, and
also provides
 * functionality to debounce the switch to prevent false readings.
 */
class Switch
{
public:
  /**
   * @brief Constructor for the Switch class.
   * @param pin The pin number of the switch.
   * @param debounceDelay The debounce delay in milliseconds.
   */
  Switch(int pin, unsigned long debounceDelay) : pin(pin),
debounceDelay(debounceDelay)
  {
    pinMode(pin, INPUT_PULLUP);

    estadoSwitch = digitalRead(pin);
    ultimoEstadoSwitch = estadoSwitch;
```

```
    ultimoTempoMudancaEstado = 0;
  }

  /**
   * @brief Checks if the switch is on.
   * @return True if the switch is on, false otherwise.
   */
  bool estaLigado()
  {
    atualizar();
    return (estadoSwitch == LOW);
  }

protected:
  /**
   * @brief Virtual method to be called when the switch is turned on.
   */
  virtual void ligado() {}

  /**
   * @brief Virtual method to be called when the switch is turned off.
   */
  virtual void desligado() {}

  /**
   * @brief Updates the switch state and debounce functionality.
   */
  void atualizar()
  {
    int leitura = digitalRead(pin);

    if (leitura != ultimoEstadoSwitch)
    {
      ultimoTempoMudancaEstado = millis();
    }

    if ((millis() - ultimoTempoMudancaEstado) > debounceDelay)
    {
      if (leitura != estadoSwitch)
      {
        estadoSwitch = leitura;

        if (estadoSwitch == LOW)
```

```
        {
          ligado();
        }
        else
        {
          desligado();
        }
      }
    }

    ultimoEstadoSwitch = leitura;
  }

private:
  // Pin number
  int pin;

  // Button state variables for debounce functionality
  unsigned long debounceDelay;
  int estadoSwitch;
  int ultimoEstadoSwitch;
  unsigned long ultimoTempoMudancaEstado;
};

/**
 * @class Botao
 * @brief A class for a button switch that inherits from the Switch
class.
 *
 * This class provides functionality for a button switch that inherits
from the Switch class.
 */
class Botao : public Switch
{
public:
  Botao(int pin, unsigned long debounceDelay) : Switch(pin,
debounceDelay) {}
};

/**
 * @class ChaveSeletora
 * @brief A class for a selector switch that controls an LED.
 *
```

```
 * This class provides functionality for a selector switch that
controls an LED. It inherits from the Switch class.
 */
class ChaveSeletora : public Switch
{
public:
  /**
   * @brief Constructor for the ChaveSeletora class.
   * @param pin The pin number of the switch.
   * @param debounceDelay The debounce delay in milliseconds.
   * @param led A pointer to the LED object that the switch controls.
   */
  ChaveSeletora(int pin, unsigned long debounceDelay, LED *led) :
Switch(pin, debounceDelay), led(led) {}

protected:
  /**
   * @brief Virtual method to be called when the switch is turned on.
   * @details This method turns on the LED.
   */
  virtual void ligado()
  {
    led->ligar();
  }

  /**
   * @brief Virtual method to be called when the switch is turned off.
   * @details This method turns off the LED.
   */
  virtual void desligado()
  {
    led->desligar();
  }

private:
  LED *led; /**< A pointer to the LED object that the switch controls.
*/
};

/**
 * @class ChaveBoia
 * @brief A class for a water level switch.
 *
```

```
 * This class provides functionality for a water level switch. It
inherits from the Switch class.
 */
class ChaveBoia : public Switch
{
public:
  ChaveBoia(int pin, unsigned long debounceDelay) : Switch(pin,
debounceDelay) {}
};

/**
 * @class SensorNivel
 * @brief An interface for water level sensors.
 *
 * This class provides an interface for reading the water level from a
sensor.
 */
class SensorNivel
{
public:
  /**
   * @brief Virtual method to read the water level from the sensor.
   * @return An integer representing the water level read from the
sensor.
   */
  virtual int lerNivel() = 0;
};

/**
 * @class SensorNivelDigital
 * @brief Represents a digital water level sensor.
 *
 * This class provides methods to read the water level from a digital
sensor.
 */
class SensorNivelDigital : public SensorNivel
{
public:
  /**
   * @brief Constructor for the SensorNivelDigital class.
   * @param pPinos An array of integers representing the pins of the
digital sensor.
   * @param numPinos The number of pins in the array.
```

```
   * @param debounceDelay The debounce delay in milliseconds.
   * @param ledErroSensor A pointer to an LED object that will be used
to indicate errors.
   */
  SensorNivelDigital(const int *pPinos, int numPinos, unsigned long
debounceDelay, LED *ledErroSensor)
      : pPinos(pPinos), numPinos(numPinos),
debounceDelay(debounceDelay), ledErroSensor(ledErroSensor)
  {
    chavesBoia = new ChaveBoia *[numPinos];
    for (int i = 0; i < numPinos; i++)
    {
      chavesBoia[i] = new ChaveBoia(pPinos[i], debounceDelay);
    }
  }

  /**
   * @brief Virtual method to read the water level from the digital
sensor.
   * @return An integer representing the water level read from the
sensor.
   */
  virtual int lerNivel()
  {
    if (chavesInvalidas())
    {
      ledErroSensor->ligar();
      return 100;
    }
    else
    {
      ledErroSensor->desligar();
      return lerChaves();
    }
  }

private:
  const int *pPinos;
  const int numPinos;
  LED *ledErroSensor;
  ChaveBoia **chavesBoia;
  unsigned long debounceDelay;
```

```
  /**
   * @brief Method to check if the digital sensor is in an invalid
state.
   * @return True if the sensor is in an invalid state, false
otherwise.
   */
  bool chavesInvalidas()
  {
    for (int i = numPinos - 1; i >= 1; i--)
    {
      if (chavesBoia[i]->estaLigado() && !chavesBoia[i -
1]->estaLigado())
      {
        return true;
      }
    }
    return false;
  }

  /**
   * @brief Method to read the water level from the digital sensor.
   * @return An integer representing the water level read from the
sensor.
   */
  int lerChaves()
  {
    for (int i = 0; i < numPinos; i++)
    {
      if (!chavesBoia[i]->estaLigado())
      {
        return map(i, 0, numPinos, 0, 100);
      }
    }

    return 100;
  }
};

/**
 * @class SensorNivelAnalogico
 * @brief Represents an analog water level sensor.
 *
```

```
 * This class provides methods to read the water level from an analog
sensor.
 */
class SensorNivelAnalogico : public SensorNivel
{
public:
  /**
   * @brief Constructor for the SensorNivelAnalogico class.
   * @param pin An integer representing the pin of the analog sensor.
   */
  SensorNivelAnalogico(int pin) : pin(pin) {}

  /**
   * @brief Virtual method to read the water level from the analog
sensor.
   * @return An integer representing the water level read from the
sensor.
   */
  virtual int lerNivel()
  {
    return map(analogRead(pin), 0, 1023, 0, 100);
  }

private:
  const int pin;
};

/**
 * @class SistemaDeControle
 * @brief Represents a control system for water pumps and reservoir.
 *
 * This class provides methods to control the water pumps and reservoir
based on the water level read from a sensor.
 */
class SistemaDeControle
{
public:
  /**
   * @brief Constructor for the SistemaDeControle class.
   * @param reservatorio A pointer to a Reservatorio object.
   * @param bomba1 A pointer to a Bomba object representing pump 1.
   * @param bomba2 A pointer to a Bomba object representing pump 2.
```

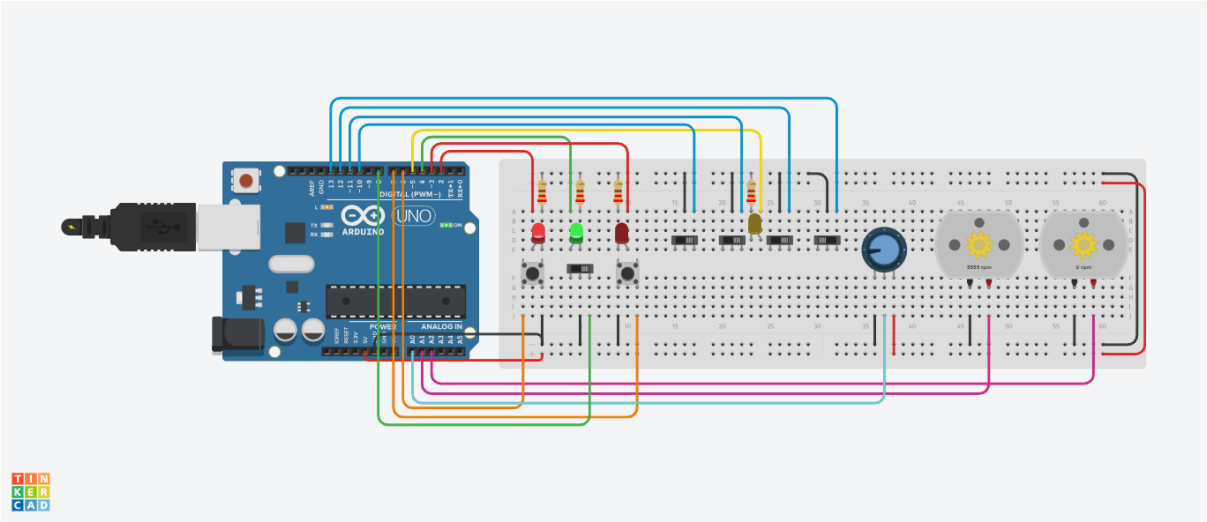
```
   * @param chaveSeletora A pointer to a ChaveSeletora object
representing the mode selector switch.
   * @param botaoBomba1 A pointer to a Botao object representing button
1 for manual mode.
   * @param botaoBomba2 A pointer to a Botao object representing button
2 for manual mode.
   * @param sensorNivel A pointer to a SensorNivel object representing
the water level sensor.
   */
  SistemaDeControle(Reservatorio *reservatorio,
                    Bomba *bomba1,
                    Bomba *bomba2,
                    ChaveSeletora *chaveSeletora,
                    Botao *botaoBomba1,
                    Botao *botaoBomba2,
                    SensorNivel *sensorNivel)
      : reservatorio(reservatorio),
        bomba1(bomba1),
        bomba2(bomba2),
        chaveSeletora(chaveSeletora),
        botaoBomba1(botaoBomba1),
        botaoBomba2(botaoBomba2),
        sensorNivel(sensorNivel) {}

  /**
   * @brief Method to update the control system.
   *
   * This method checks the mode selector switch and updates the water
pumps and reservoir accordingly.
   */
  void atualizar()
  {
    reservatorio->setNivel(sensorNivel->lerNivel());

    if (chaveSeletora->estaLigado())
    {
      modoAutomatico(reservatorio->getNivel());
    }
    else
    {
      modoManual();
    }
  }
```

```
private:
  Reservatorio *reservatorio;
  Bomba *bomba1;
  Bomba *bomba2;
  ChaveSeletora *chaveSeletora;
  Botao *botaoBomba1;
  Botao *botaoBomba2;
  SensorNivel *sensorNivel;

  /**
   * @brief Method to control the system in automatic mode.
   *
   * This method reads the water level from the sensor and turns on/off
the pumps accordingly.
   */
  void modoAutomatico(int nivel)
  {
    // Bomba 1 - 50% ~ 100%
    if (nivel < NIVEL_MEDIO)
    {
      // Water level is between 50% and 100% - turn on pump 1
      bomba1->ligar();
    }
    else if (nivel == NIVEL_MAXIMO)
    {
      // Water level is 100% - turn off pump 1
      bomba1->desligar();
    }

    // Bomba 2 - 0% ~ 25%
    if (nivel < NIVEL_BAIXO)
    {
      // Water level is very low - turn on pump 2
      bomba2->ligar();
    }
    else if (nivel >= NIVEL_ALTO)
    {
      // Water level is above 75% - turn off pump 2
      bomba2->desligar();
    }
  }
```

```
  /**
   * @brief Method to control the system in manual mode.
   *
   * This method reads the state of the manual mode buttons and turns
on/off the pumps accordingly.
   */
  void modoManual()
  {
    // Check if button 1 is pressed
    if (botaoBomba1->estaLigado())
    {
      bomba1->ligar();
    }
    else
    {
      bomba1->desligar();
    }

    // Check if button 2 is pressed
    if (botaoBomba2->estaLigado())
    {
      bomba2->ligar();
    }
    else
    {
      bomba2->desligar();
    }
  }
};

// Create objects for LEDs
LED ledBomba1(LED_BOMBA_1_PIN);
LED ledBomba2(LED_BOMBA_2_PIN);
LED ledModoAutomatico(LED_MODO_AUTOMATICO_PIN);
LED ledErroSensor(LED_ERRO_SENSOR_PIN);

// Create objects for reservoir, and pumps
Reservatorio reservatorio(0);
Bomba bomba1(BOMBA_1_PIN, &ledBomba1);
Bomba bomba2(BOMBA_2_PIN, &ledBomba2);

// Create objects for buttons and mode selector switch
```

```
ChaveSeletora chaveSeletora(CHAVE_SELETORA_PIN, DEBOUNCE_DELAY,
&ledModoAutomatico);
Botao botaoBomba1(BOTAO_BOMBA_1_PIN, DEBOUNCE_DELAY);
Botao botaoBomba2(BOTAO_BOMBA_2_PIN, DEBOUNCE_DELAY);

// Create objects for water level sensors
const int numSensores = sizeof(SENSOR_DIGITAL_PINS) /
sizeof(SENSOR_DIGITAL_PINS[0]);
SensorNivelDigital sensorNivel(SENSOR_DIGITAL_PINS, numSensores,
DEBOUNCE_DELAY, &ledErroSensor);
// SensorNivelAnalogico sensorNivel(SENSOR_ANALOGICO_PIN);

// Create object for control system
SistemaDeControle sistemaDeControle(&reservatorio,
                                    &bomba1,
                                    &bomba2,
                                    &chaveSeletora,
                                    &botaoBomba1,
                                    &botaoBomba2,
                                    &sensorNivel);

void setup() {}

void loop()
{
  sistemaDeControle.atualizar();
}
```

**Simulação:**
https://www.tinkercad.com/things/9sHoR675mbz



## Conclusão

Nesta atividade prática, aplicamos os conhecimentos teóricos adquiridos durante a disciplina de Microcontroladores para programar e montar um sistema de controle de nível de reservatório utilizando a plataforma Arduino. O sistema foi projetado para controlar o nível de água em um reservatório utilizando duas bombas e sensores de nível digital ou analógico. O sistema pode operar em modo automático ou manual, selecionado por meio de uma chave seletora.

No modo automático, o sistema controla o nível de água no reservatório ligando e desligando as bombas com base nas leituras dos sensores de nível. No modo manual, o usuário pode controlar as bombas diretamente por meio de botões.

Durante a atividade, refatoramos o código para melhorar sua legibilidade e modularidade. Criamos classes para representar os componentes do sistema, como o reservatório, as bombas, os LEDs e os sensores de nível. Também criamos uma classe para representar o sistema de controle e implementamos métodos para os modos automático e manual.

Em resumo, esta atividade prática nos permitiu aplicar nossos conhecimentos teóricos em um projeto concreto e desenvolver habilidades práticas na programação e montagem de sistemas microcontrolados. Aprendemos como projetar e implementar um sistema de controle utilizando a plataforma Arduino e como refatorar o código para melhorar sua legibilidade e modularidade. A atividade foi desafiadora e gratificante, e estamos satisfeitos com os resultados alcançados.