

Master Code For R2D2 Alarm Clock

Copy and Paste the code below in its entirety it to an Arduino IDE Sketch

(See Arduino IDE Setup Guide First)

```
#include <WiFi.h>
#include <time.h>
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <TM1637Display.h>
#include <DFRobotDFPlayerMini.h>

// ===== USER SETTINGS =====
const char* ssid    = "NETGEAR69_EXT";
const char* password = "aquatictomato152";

// Time Zone (EST)
long gmtOffset_sec = -18000;
int daylightOffset_sec = 3600;

// ===== PIN DEFINITIONS =====
#define RED_LED_PIN    2
#define WHITE_LED_PIN  7
#define BLUE_LEDS_PIN 10

#define CLK_PIN        20
#define DIO_PIN        21
#define DF_RX_PIN      6
#define DF_TX_PIN      8

#define ROTARY_SW      3
#define ROTARY_DT      4
#define ROTARY_CLK     5

// ===== VARIABLES =====
TM1637Display display(CLK_PIN, DIO_PIN);
HardwareSerial myDFSerial(1);
DFRobotDFPlayerMini myDFPlayer;

// Bluetooth
BLEServer *pServer = NULL;
BLECharacteristic *pCharacteristic = NULL;
```

```

bool deviceConnected = false;
String btMessage = "";
bool msgReceived = false;
bool justConnected = false;

// Time & Audio
int currentHour = 12, currentMin = 0;
int alarmHour = 7, alarmMinute = 30;
int alarmSong = 1;
int playSelection = 1;
bool alarmActive = false;
bool isAlarming = false;
bool musicActive = false;
int currentVolume = 15;

// Encoder State
int lastClk = HIGH;
int lastBtnState = HIGH;
unsigned long btnPressStart = 0;
bool btnPressed = false;

// Display Timers
unsigned long menuIdleTimer = 0;
unsigned long volumeDisplayTimer = 0;
bool showVolumeMode = false;

// Menu System
int currentMode = 0;
unsigned long lastTimeUpdate = 0;
unsigned long lastBlinkTime = 0;
bool ledState = true;

// ===== BLUETOOTH CALLBACKS =====
class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
        deviceConnected = true;
        justConnected = true;
    };
    void onDisconnect(BLEServer* pServer) {
        deviceConnected = false;
        pServer->getAdvertising()->start();
    }
};

```

```

class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
        // FIX: Use Arduino String instead of std::string
        String rxValue = pCharacteristic->getValue();

        if (rxValue.length() > 0) {
            btMessage = rxValue; // Simplify assignment
            msgReceived = true;
        }
    }
};

// ===== SETUP =====
void setup() {
    Serial.begin(115200);

    // 1. Setup LEDs
    pinMode(RED_LED_PIN, OUTPUT);
    pinMode(WHITE_LED_PIN, OUTPUT);
    pinMode(BLUE_LEDS_PIN, OUTPUT);
    digitalWrite(RED_LED_PIN, LOW);
    digitalWrite(WHITE_LED_PIN, LOW);
    digitalWrite(BLUE_LEDS_PIN, LOW);

    // 2. Knob Inputs
    pinMode(ROTARY_SW, INPUT_PULLUP);
    pinMode(ROTARY_DT, INPUT);
    pinMode(ROTARY_CLK, INPUT);
    lastClk = digitalRead(ROTARY_CLK);

    // 3. Display
    display.setBrightness(0x0f);
    uint8_t boot[] = {0x40, 0x40, 0x40, 0x40};
    display.setSegments(boot);
    delay(500);

    // 4. WiFi Start
    WiFi.begin(ssid, password);
    int retry = 0;
    while (WiFi.status() != WL_CONNECTED && retry < 20) {
        delay(200);
        retry++;
    }
    if (WiFi.status() == WL_CONNECTED) {

```

```

    configTime(gmtOffset_sec, daylightOffset_sec, "pool.ntp.org");
    uint8_t wifi[] = {0x1C, 0x10, 0x71, 0x10};
    display.setSegments(wifi);
    delay(500);
}

// 5. MP3 Start
digitalWrite(RED_LED_PIN, LOW);
digitalWrite(WHITE_LED_PIN, LOW);
digitalWrite(BLUE_LEDS_PIN, LOW);
delay(200);

myDFSerial.begin(9600, SERIAL_8N1, DF_RX_PIN, DF_TX_PIN);
delay(500);

if (myDFPlayer.begin(myDFSerial)) {
    Serial.println("✅ DFPlayer Online.");
    myDFPlayer.volume(currentVolume);
    myDFPlayer.play(1);
}

// 6. Bluetooth Start
BLEDevice::init("R2D2_Final");
pServer = BLEDevice::createServer();
pServer->setCallbacks(new MyServerCallbacks());

BLEService *pService = pServer->createService("FFE0");

pCharacteristic = pService->createCharacteristic(
    "FFE1",
    BLECharacteristic::PROPERTY_READ |
    BLECharacteristic::PROPERTY_WRITE |
    BLECharacteristic::PROPERTY_NOTIFY |
    BLECharacteristic::PROPERTY_WRITE_NR
);

pCharacteristic->setCallbacks(new MyCallbacks());
pCharacteristic->addDescriptor(new BLE2902());

pService->start();

BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
pAdvertising->addServiceUUID("FFE0");
pAdvertising->setScanResponse(true);

```

```

pAdvertising->setMinPreferred(0x06);
pAdvertising->setMinPreferred(0x12);
BLEDevice::startAdvertising();

// 7. FINAL POWER UP: Show "Std"
digitalWrite(RED_LED_PIN, HIGH);
digitalWrite(WHITE_LED_PIN, HIGH);
digitalWrite(BLUE_LEDS_PIN, HIGH);

uint8_t std[] = {0x6D, 0x78, 0x5E, 0x00};
display.setSegments(std);
delay(2000);
}

// ===== MAIN LOOP =====
void loop() {
    handleKnob();

    if (currentMode == 0) {
        if (showVolumeMode) {
            if (millis() - volumeDisplayTimer > 2000) showVolumeMode = false;
        } else {
            updateTime();
        }

        checkAlarm();
        handleBluetooth();
        handleLEDs();
    } else {
        blinkMenu();
        if (millis() - menuIdleTimer > 15000) currentMode = 0;
    }
}

// ===== KNOB LOGIC =====
void handleKnob() {
    int newClk = digitalRead(ROTARY_CLK);

    if (newClk != lastClk && newClk == LOW) {
        int dtValue = digitalRead(ROTARY_DT);
        int direction = (dtValue != newClk) ? -1 : 1;
        menuIdleTimer = millis();

        if (currentMode == 0) { // Volume

```

```

    currentVolume += direction;
    if (currentVolume > 30) currentVolume = 30;
    if (currentVolume < 0) currentVolume = 0;
    myDFPlayer.volume(currentVolume);
    display.showNumberDec(currentVolume, false, 2, 2);
    uint8_t SEG_UO[] = {0x3E, 0x3F};
    display.setSegments(SEG_UO, 2, 0);
    showVolumeMode = true;
    volumeDisplayTimer = millis();
}
else if (currentMode == 1) { currentHour += direction; if (currentHour > 23) currentHour = 0; if
(currentHour < 0) currentHour = 23; }
else if (currentMode == 2) { currentMin += direction; if (currentMin > 59) currentMin = 0; if
(currentMin < 0) currentMin = 59; }
else if (currentMode == 3) { alarmHour += direction; if (alarmHour > 23) alarmHour = 0; if
(alarmHour < 0) alarmHour = 23; }
else if (currentMode == 4) { alarmMinute += direction; if (alarmMinute > 59) alarmMinute = 0;
if (alarmMinute < 0) alarmMinute = 59; }
else if (currentMode == 5) { alarmSong += direction; if (alarmSong < 1) alarmSong = 1; if
(alarmSong > 99) alarmSong = 1; }
else if (currentMode == 6) { playSelection += direction; if (playSelection < 1) playSelection =
1; if (playSelection > 99) playSelection = 1; }
}
lastClk = newClk;

// BUTTON
int btnState = digitalRead(ROTARY_SW);
if (btnState == LOW && lastBtnState == HIGH) { btnPressStart = millis(); btnPressed = true; }
if (btnState == HIGH && lastBtnState == LOW && btnPressed) {
    btnPressed = false;
    unsigned long duration = millis() - btnPressStart;
    menuIdleTimer = millis();
    if (duration > 50) {
        if (isAlarming || musicActive) { stopAlarm(); lastBtnState = btnState; return; }
        if (duration > 1000) { if (currentMode == 0) { currentMode = 1; showVolumeMode = false; }
else { currentMode = 0; } }
        else { if (currentMode == 0) { alarmActive = !alarmActive; uint8_t SEG_ON[] = {0x3F, 0x54,
0x00, 0x00}; uint8_t SEG_OFF[] = {0x3F, 0x71, 0x71, 0x00}; display.setSegments(alarmActive ?
SEG_ON : SEG_OFF); delay(1000); } else { if (currentMode == 6) {
myDFPlayer.volume(currentVolume); myDFPlayer.play(playSelection); musicActive = true;
currentMode = 0; return; } currentMode++; if (currentMode > 6) currentMode = 0; } }
    }
}
lastBtnState = btnState;

```

```
}
```

```
// ===== HELPERS =====
```

```
void updateTime() {
    if (millis() - lastTimeUpdate > 1000) {
        lastTimeUpdate = millis();
        struct tm timeinfo;
        if (getLocalTime(&timeinfo)) { currentHour = timeinfo.tm_hour; currentMin = timeinfo.tm_min; }
        int dispH = currentHour;
        bool isPM = (dispH >= 12);
        if (dispH == 0) dispH = 12;
        if (dispH > 12) dispH -= 12;
        bool blink = (millis() / 500) % 2 == 0;
        display.showNumberDecEx((dispH * 100) + currentMin, (isPM || blink) ? 0x40 : 0x00, true);
    }
}
```

```
void blinkMenu() {
    bool on = (millis() / 300) % 2 == 0;
    int val = 0;
    bool isSetPM = false;
    if (currentMode == 1 || currentMode == 2) { val = (currentHour * 100) + currentMin; if
(currentHour >= 12) isSetPM = true; }
    else if (currentMode == 3 || currentMode == 4) { val = (alarmHour * 100) + alarmMinute; if
(alarmHour >= 12) isSetPM = true; }
    else if (currentMode == 5) { if (on) { display.showNumberDec(alarmSong, true, 2, 2); uint8_t
SEG_SO[] = {0x6D, 0x5C}; display.setSegments(SEG_SO, 2, 0); } else display.clear(); return; }
    else if (currentMode == 6) { if (on) { display.showNumberDec(playSelection, true, 2, 2); uint8_t
SEG_PL[] = {0x73, 0x38}; display.setSegments(SEG_PL, 2, 0); } else display.clear(); return; }
```

```
    int dispH = (currentMode == 3 || currentMode == 4) ? alarmHour : currentHour;
    if (dispH == 0) dispH = 12;
    if (dispH > 12) dispH -= 12;
    val = (dispH * 100) + ((currentMode == 3 || currentMode == 4) ? alarmMinute : currentMin);
    uint8_t pmMask = isSetPM ? 0x40 : 0x00;
    if (on) { display.showNumberDecEx(val, pmMask, true); } else { if (currentMode == 1 ||
currentMode == 3) display.showNumberDec(val % 100, true, 2, 2); else
display.showNumberDec(val / 100, true, 2, 0); }
}
```

```
void checkAlarm() {
    if (alarmActive && !isAlarming) {
        struct tm timeinfo;
```

```

    bool match = false;
    if (getLocalTime(&timeinfo)) { if (timeinfo.tm_hour == alarmHour && timeinfo.tm_min ==
alarmMinute && timeinfo.tm_sec == 0) match = true; }
    else { if (currentHour == alarmHour && currentMin == alarmMinute && millis() % 60000 <
1000) match = true; }
    if (match) { isAlarming = true; musicActive = true; myDFPlayer.volume(currentVolume);
myDFPlayer.loop(alarmSong); }
    }
}

```

```

void stopAlarm() {
    isAlarming = false;
    musicActive = false;
    myDFPlayer.pause();
    digitalWrite(RED_LED_PIN, HIGH);
    digitalWrite(WHITE_LED_PIN, HIGH);
    digitalWrite(BLUE_LEDS_PIN, HIGH);
}

```

```

void handleLEDs() {
    if (isAlarming) {
        if (millis() - lastBlinkTime > 100) { lastBlinkTime = millis(); ledState = !ledState;
digitalWrite(RED_LED_PIN, ledState); digitalWrite(WHITE_LED_PIN, ledState);
digitalWrite(BLUE_LEDS_PIN, ledState); }
        else { digitalWrite(RED_LED_PIN, HIGH); digitalWrite(WHITE_LED_PIN, HIGH);
digitalWrite(BLUE_LEDS_PIN, HIGH); }
    }
}

```

```

void handleBluetooth() {
    if (justConnected) { uint8_t cnct[] = {0x39, 0x54, 0x58, 0x78}; display.setSegments(cnct);
delay(1500); justConnected = false; }
    if (msgReceived) { uint8_t recd[] = {0x50, 0x79, 0x58, 0x5E}; display.setSegments(recd);
delay(1000); msgReceived = false; }
}

```

```

if (btMessage != "") {
    btMessage.toUpperCase();
    if (btMessage.startsWith("STOP")) stopAlarm();
    else if (btMessage.startsWith("ALARM ")) {
        String t = btMessage.substring(6);
        int split = t.indexOf('.');
        if (split > 0) {
            alarmHour = t.substring(0, split).toInt();
            alarmMinute = t.substring(split+1).toInt();
            alarmActive = true;
        }
    }
}

```

```
        display.showNumberDecEx((alarmHour*100)+alarmMinute, 0x40, true);
        delay(2000);
    }
}
else if (btMessage.startsWith("PLAY ")) {
    int s = btMessage.substring(5).toInt();
    myDFPlayer.play(s);
    musicActive = true;
}
else if (btMessage.toInt() > 0) {
    int s = btMessage.toInt();
    myDFPlayer.play(s);
    musicActive = true;
}
btMessage = "";
}
}
```