```python
# Import necessary modules
from machine import Pin, SPI, I2C
from display import SSD1306_DualSPI
from radio import Radio
import time
import ujson
from utime import localtime, mktime

# Setup SPI for the dual OLED display
spi = SPI(0, sck=Pin(18), mosi=Pin(19))
dc = Pin(20)
res = Pin(21)
cs1 = Pin(17)
cs2 = Pin(5)

# Initialize the dual-screen OLED display
display = SSD1306_DualSPI(256, 64, spi, dc, res, cs1, cs2)

# Initialize user interface buttons
btn_mode = Pin(0, Pin.IN, Pin.PULL_UP)
btn_select = Pin(3, Pin.IN, Pin.PULL_UP)
btn_up = Pin(6, Pin.IN, Pin.PULL_UP)
btn_down = Pin(7, Pin.IN, Pin.PULL_UP)

# Track button states for edge detection
last_button_state_btn_mode = 1
last_button_state_btn_select = 1
last_button_state_btn_up = 1
last_button_state_btn_down = 1

# UI and system state variables
mode = 0
edit_hour = True
flash_state = True
radio_info_toggle = False

# Default alarm and display settings
alarm_time = [6, 30]
alarm_active = False
snooze_minutes = 0
show_24hr = True
alarm_triggered = False
snooze_until = None

# Use a simulated clock that increments every second
sim_time = list(time.localtime())
sim_last_tick = time.ticks_ms()

# Initialize FM radio once at 98.5 MHz
fm = Radio(101.9, 2, False)
#i2c_radio = I2C(1, scl=Pin(27), sda=Pin(26))
# Alternate time zones (simulated)
alternate_timezones = [
    ("UTC-12", -12.0),
    ("UTC-11", -11.0),
    ("UTC-10", -10.0),
    ("UTC-09.5", -9.5),
    ("UTC-09", -9.0),
    ("UTC-08", -8.0),
    ("UTC-07", -7.0),
    ("UTC-06", -6.0),
    ("UTC-05", -5.0),
    ("UTC-04.5", -4.5),
    ("UTC-04", -4.0),
    ("UTC-03.5", -3.5),
    ("UTC-03", -3.0),
    ("UTC-02", -2.0),
    ("UTC-01", -1.0),
    ("UTC+0", 0.0),
    ("UTC+01", 1.0),
    ("UTC+02", 2.0),
    ("UTC+03", 3.0),
    ("UTC+03.5", 3.5),
    ("UTC+04", 4.0),
    ("UTC+04.5", 4.5),
    ("UTC+05", 5.0),
    ("UTC+05.5", 5.5),
    ("UTC+05.75", 5.75),
    ("UTC+06", 6.0),
    ("UTC+06.5", 6.5),
    ("UTC+07", 7.0),
    ("UTC+08", 8.0),
    ("UTC+08.75", 8.75),
    ("UTC+09", 9.0),
    ("UTC+09.5", 9.5),
    ("UTC+10:00", 10.0),
    ("UTC+10.5", 10.5),
    ("UTC+11", 11.0),
    ("UTC+12", 12.0),
    ("UTC+12.75", 12.75),
    ("UTC+13", 13.0),
    ("UTC+14", 14.0)
]


selected_timezone_index = 0

# Path to settings file
SETTINGS_FILE = "settings.json"

# Attempt to load saved settings
try:
    with open(SETTINGS_FILE, "r") as f:
        data = ujson.load(f)
        alarm_time = data.get("alarm_time", alarm_time)
        alarm_active = data.get("alarm_active", alarm_active)
        snooze_minutes = data.get("snooze_minutes", snooze_minutes)
        show_24hr = data.get("show_24hr", show_24hr)
except:
    pass

# Save settings to flash storage
def save_settings():
    try:
        with open(SETTINGS_FILE, "w") as f:
            ujson.dump({
                "alarm_time": alarm_time,
                "alarm_active": alarm_active,
                "snooze_minutes": snooze_minutes,
                "show_24hr": show_24hr
            }, f)
    except:
        pass

# Handle button press edge detection
def button_pressed(button, last_state):
    current_state = button.value()
    if last_state == 1 and current_state == 0:
        time.sleep(0.05)
```

```python
        if button.value() == 0:
            return True, 0
    return False, current_state

# Draw the main clock view
def draw_clock():
    hour = sim_time[3]
    minute = sim_time[4]
    display.text("Freq: {:.1f}".format(fm.Frequency), 140, 10)
    if not show_24hr:
        suffix = "AM" if hour < 12 else "PM"
        hour = hour % 12 or 12
        display.text("Time: {:02d}:{:02d} {}".format(hour, minute, suffix), 10, 10)
    else:
        display.text("Time: {:02d}:{:02d}".format(hour, minute), 10, 10)
    if alarm_active:
        display.text("Alarm: {:02d}:{:02d}".format(*alarm_time), 140, 20)
    display.text("Mode: Clock", 10, 50)

# Draw the alarm time setting view
def draw_alarm_set():
    display.text("Set Alarm:", 10, 5)
    display.text("Hour: {:02d}".format(alarm_time[0]), 140, 10)
    display.text("Min : {:02d}".format(alarm_time[1]), 140, 50)
    display.text("Mode: Alarm Set", 10, 50)

# Draw the FM radio interface
def draw_radio():
    display.text("FM Radio", 10, 10)
    display.text("Freq: {:.1f}".format(fm.Frequency), 140, 10)
    if alarm_active:
        display.text("A", 240, 0)
    if radio_info_toggle:
        display.text("Retro", 180, 20)
    display.text("Mode: Radio", 10, 50)

# Draw the info/settings view
def draw_info():
    display.text("Alarm: {:02d}:{:02d}".format(*alarm_time), 140, 10)
    display.text("Snooze: +{}min".format(snooze_minutes), 140, 50)
    display.text("Mode: Info", 10, 50)

# Draw the manual time change interface
def draw_time_change():
    display.text("New Time: {:02d}:{:02d}".format(sim_time[3], sim_time[4]), 140, 10)
    display.text("Edit: Hour" if edit_hour else "Edit: Minute", 140, 25)
    display.text("Mode: Time Edit", 10, 50)

# Flashing display effect for alarm trigger
def draw_alarm_trigger():
    global flash_state
    flash_state = not flash_state
    if flash_state:
        display.text("!!! WAKE", 20, 15)
        display.text("UP !!!", 140, 15)

# Time shifting for alternate timezones
def get_shifted_time(base_time, offset_hours):
    shifted = base_time.copy()
    total_minutes = shifted[3] * 60 + shifted[4] + int(offset_hours * 60)
    total_minutes %= 1440
    shifted[3] = total_minutes // 60
    shifted[4] = total_minutes % 60
    return shifted

```

```
# Alternate timezone screen
def draw_alternate_timezones():
    display.text("Alt Timezones:", 10, 5)
    visible = alternate_timezones[selected_timezone_index:selected_timezone_index+2]
    y_offset = 10
    for label, offset in visible:
        shifted = get_shifted_time(sim_time, offset)
        display.text(f"{label}: {shifted[3]:02}:{shifted[4]:02}", 130, y_offset)
        y_offset += 15
    display.text("Mode: TZ View", 10, 50)

# Main loop
while True:
    now_tick = time.ticks_ms()
    if time.ticks_diff(now_tick, sim_last_tick) >= 1000:
        sim_last_tick = now_tick
        sim_time[5] += 1
        if sim_time[5] >= 60:
            sim_time[5] = 0
            sim_time[4] += 1
        if sim_time[4] >= 60:
            sim_time[4] = 0
            sim_time[3] += 1
        if sim_time[3] >= 24:
            sim_time[3] = 0

    current_hour = sim_time[3]
    current_minute = sim_time[4]
    now_minutes = current_hour * 60 + current_minute
    alarm_minutes = alarm_time[0] * 60 + alarm_time[1]
    snooze_minutes_val = snooze_until[0] * 60 + snooze_until[1] if snooze_until else None

    if alarm_active:
        if snooze_until and now_minutes >= snooze_minutes_val:
            snooze_until = None
        if snooze_until is None and now_minutes == alarm_minutes:
            alarm_triggered = True
        elif alarm_triggered and now_minutes != alarm_minutes:
            alarm_triggered = False

    # Check all button presses at top of loop
    pressed_mode, last_button_state_btn_mode = button_pressed(btn_mode, last_button_state_btn_mode)
    pressed_select, last_button_state_btn_select = button_pressed(btn_select, 
last_button_state_btn_select)
    pressed_up, last_button_state_btn_up = button_pressed(btn_up, last_button_state_btn_up)
    pressed_down, last_button_state_btn_down = button_pressed(btn_down, last_button_state_btn_down)

    display.fill(0)

    if alarm_triggered:
        draw_alarm_trigger()
        if pressed_select:
            alarm_triggered = False
            if snooze_minutes > 0:
                snooze_until = [current_hour, (current_minute + snooze_minutes) % 60]
                if snooze_until[1] < current_minute:
                    snooze_until[0] = (snooze_until[0] + 1) % 24

    else:
        if pressed_mode:
            mode = (mode + 1) % 6

        if mode == 0:
            draw_clock()
            if pressed_select and alarm_triggered:
```

```python
                alarm_triggered = False

        elif mode == 1:
            draw_radio()
            if pressed_select:
                radio_info_toggle = not radio_info_toggle

            if pressed_up:
                fm.Mute = True
                fm.ProgramRadio()
                new_freq = fm.Frequency + 0.2
                if new_freq > 108:
                    new_freq = 88.1
                fm.SetFrequency(new_freq)
                time.sleep(0.1)
                fm.Mute = False
                fm.ProgramRadio()

            if pressed_down:
                fm.Mute = True
                fm.ProgramRadio()
                new_freq = fm.Frequency - 0.2
                if new_freq < 88:
                    new_freq = 107.9
                fm.SetFrequency(new_freq)
                time.sleep(0.1)
                fm.Mute = False
                fm.ProgramRadio()

        elif mode == 2:
            draw_alarm_set()
            if pressed_select:
                alarm_active = not alarm_active
                save_settings()

            if pressed_up:
                alarm_time[0] = (alarm_time[0] + 1) % 24
                save_settings()

            if pressed_down:
                alarm_time[1] = (alarm_time[1] + 1) % 60
                save_settings()

        elif mode == 3:
            draw_info()
            if pressed_select:
                snooze_minutes += 5
                if snooze_minutes > 30:
                    snooze_minutes = 0
                save_settings()

            if pressed_up:
                show_24hr = False
                save_settings()

            if pressed_down:
                show_24hr = True
                save_settings()

        elif mode == 4:
            draw_time_change()
            if pressed_select:
                edit_hour = not edit_hour

            if pressed_up or pressed_down:
                if edit_hour:
                    if pressed_up:
                        sim_time[3] = (sim_time[3] + 1) % 24
                    if pressed_down:
                        sim_time[3] = (sim_time[3] - 1) % 24
                else:
                    if pressed_up:
                        sim_time[4] = (sim_time[4] + 1) % 60
                    if pressed_down:
                        sim_time[4] = (sim_time[4] - 1) % 60

        if mode == 5:
            draw_alternate_timezones()
            if pressed_select:
                selected_timezone_index += 1
    if selected_timezone_index >= len(alternate_timezones):
        selected_timezone_index = 0

    display.show()
    time.sleep(0.05)

```