STM32CubeIDE Code and Pin Connections

Here's a basic code structure for your STM32 project in STM32CubeIDE. I've included comments to explain each section. **You will need to adapt this code to the specifics of your HW220 temperature sensor and OLED display.**

Assumptions:

- **Temperature Sensor (HW220):** I'll assume this is an analog sensor. If it's digital (e.g., I2C, SPI), you'll need to change the temperature reading part of the code significantly.
- **OLED Display:** I'll assume this is an I2C display. If it's SPI, you'll need to change the OLED initialization and data transmission functions.

1. Pin Connections (Reminder - Adapt to your specific components!)

Here's the pinout from the previous response, repeated for convenience. **You MUST adapt this based on the specific pinouts of your sensors and OLED display.**

```
STM32 Nucleo-L476RG       Ultrasonic Sensor (HC-SR04 Example)
--------------------      ---------------------------------
5V ------------------------- VCC
GND ------------------------- GND
PA0 (Example)----------------- Trig
PA1 (Example)----------------- Echo (with voltage divider if needed)

STM32 Nucleo-L476RG       Temperature Sensor (Example - Analog)
--------------------      ---------------------------------
VCC ------------------------- VCC
GND ------------------------- GND
PA2 (Example - ADC Pin)------ Signal

STM32 Nucleo-L476RG       OLED Display (Example - I2C)
--------------------      ---------------------------------
VCC ------------------------- VCC
GND ------------------------- GND
PB8 ------------------------- SCL
PB9 ------------------------- SDA

STM32 Nucleo-L476RG       Buzzer
--------------------      ---------------------------------
```

PA3 (Example)---------- + (Positive, possibly with resistor)
GND ---------- - (Negative)

2. STM32CubeIDE Code

```
#include "main.h"
#include <stdio.h>  // For sprintf

// Define pin mappings (Adapt these if needed!)
#define ULTRASONIC_TRIG_PIN     GPIOA, GPIO_PIN_0
#define ULTRASONIC_ECHO_PIN     GPIOA, GPIO_PIN_1
#define TEMP_SENSOR_PIN         GPIOA, GPIO_PIN_2  // Analog input
#define OLED_SCL_PIN            GPIOB, GPIO_PIN_8
#define OLED_SDA_PIN            GPIOB, GPIO_PIN_9
#define BUZZER_PIN              GPIOA, GPIO_PIN_3

// Function prototypes
void SystemClock_Config(void);
void Error_Handler(void);
void ultrasonic_init(void);
uint32_t ultrasonic_getDistance(void);
void temp_sensor_init(void);
float temp_sensor_getTemp(void);
void oled_init(void);
void oled_display_text(char *text);
void buzzer_init(void);
void buzzer_on(void);
void buzzer_off(void);
void delay_us(uint16_t us);

// Global variables
TIM_HandleTypeDef htim1;       // Timer for ultrasonic sensor
ADC_HandleTypeDef hadc1;       // ADC for temperature sensor (if analog)
I2C_HandleTypeDef hi2c1;       // I2C for OLED display
TIM_HandleTypeDef htim3;       // Timer for 1-hour sitting timer
TIM_HandleTypeDef htim4;       // Timer for 30-second absence timer

volatile uint8_t userPresent = 0;
```

```
volatile uint32_t sittingTime = 0;
volatile uint32_t absenceTime = 0;
volatile uint8_t takeBreak = 0;

//char oled_buffer[20]; //removed global

int main(void) {
  HAL_Init();
  SystemClock_Config();

  // Initialize peripherals
  ultrasonic_init();
  temp_sensor_init(); // Initialize ADC if HW220 is analog
  oled_init();
  buzzer_init();
  //initialize timers
  HAL_TIM_Base_Start_IT(&htim3); //start 1 hour timer
  HAL_TIM_Base_Start_IT(&htim4); //start 30 sec timer

  oled_display_text("Waiting for user...");

  while (1) {
    uint32_t distance = ultrasonic_getDistance();

    if (distance < 100) { // Adjust the threshold as needed (e.g., 100 cm)
      userPresent = 1;
      absenceTime = 0; //reset absence timer
      HAL_TIM_Base_Start_IT(&htim3); //restart 1 hour timer
      float temperature = temp_sensor_getTemp();
      char oled_buffer[20]; //local variable
      sprintf(oled_buffer, "Temp: %.2f C", temperature);
      oled_display_text(oled_buffer);

      if (temperature >= 35 && temperature <= 40) {
        oled_display_text("Moderate temp,\nstay hydrated");
      } else if (temperature > 40) {
        oled_display_text("Too hot, turn on AC");
      }
```

```
      if (takeBreak) {
        buzzer_on();
        oled_display_text("Take a break!");
        HAL_Delay(5000); // Buzz for 5 seconds
        buzzer_off();
        takeBreak = 0;
      }
    } else {
      userPresent = 0;
      HAL_TIM_Base_Stop_IT(&htim3); //stop 1 hour timer when user not present
      absenceTime++;
      if(absenceTime >= 30){
          absenceTime = 0;
          sittingTime = 0;
          oled_display_text("Waiting for user...");
      }
    }

    HAL_Delay(500); // Add a short delay to control loop speed
  }
}

void ultrasonic_init(void) {
  // Timer 1 initialization for ultrasonic sensor
  __HAL_RCC_TIM1_CLK_ENABLE();

  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 79; // 80 MHz / 80 = 1 MHz (1 us timer)
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 0xFFFF; // Max period
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK) {
    Error_Handler();
  }
}

uint32_t ultrasonic_getDistance(void) {
```

```
  uint32_t distance = 0;
  uint32_t duration = 0;

  // Trigger pulse
  HAL_GPIO_WritePin(ULTRASONIC_TRIG_PIN, GPIO_PIN_SET);
  delay_us(10); // 10 us trigger pulse
  HAL_GPIO_WritePin(ULTRASONIC_TRIG_PIN, GPIO_PIN_RESET);

  // Measure echo pulse
  while (HAL_GPIO_ReadPin(ULTRASONIC_ECHO_PIN) == GPIO_PIN_RESET); // Wait for
echo start
  uint32_t startTick = HAL_GetTick();
  while (HAL_GPIO_ReadPin(ULTRASONIC_ECHO_PIN) == GPIO_PIN_SET);   // Wait for
echo end
  uint32_t endTick = HAL_GetTick();
  duration = endTick - startTick;

  // Calculate distance (speed of sound: 343 m/s,  *100 to convert to cm)
  distance = (duration * 343) / 2000; //distance in mm
  return distance/10; //distance in cm
}

void temp_sensor_init(void) {
  // Initialize ADC for temperature sensor (if analog)
  __HAL_RCC_ADC1_CLK_ENABLE();

  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV8; //asynchronous clock
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE;
  hadc1.Init.LowPowerAutoWait = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.NbrOfDiscConversion = 0;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
```

```
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.Overrun = ADC_OVR_DATA_OVERWRITTEN;
  if (HAL_ADC_Init(&hadc1) != HAL_OK) {
    Error_Handler();
  }

    // Configure the ADC channel for the temperature sensor pin
  ADC_ChannelConfTypeDef sConfig = {0};
  sConfig.Channel = ADC_CHANNEL_5;  //  PA2 is ADC_IN5.  Check your board's
datasheet!
  sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
  sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5; //adjust sampling time
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
}

float temp_sensor_getTemp(void) {
  // Read temperature from the sensor (ADC if analog)
  HAL_ADC_Start(&hadc1);
  HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
  uint32_t adcValue = HAL_ADC_GetValue(&hadc1);
  HAL_ADC_Stop(&hadc1);

  // Convert ADC value to temperature (This is VERY sensor-specific!)
  //  You'll need to calibrate this based on your HW220's datasheet.
  //  The following is a placeholder example:
  float temperature = (adcValue * 0.08056) - 50; // Replace with your calibration
formula
  return temperature;
}

void oled_init(void) {
  // Initialize I2C for OLED display
  __HAL_RCC_I2C1_CLK_ENABLE();

  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x00000204;  // Fast Mode 400kHz (adjust as needed)
```

```
hi2c1.Init.OwnAddress1 = 0;
hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
hi2c1.Init.OwnAddress2 = 0;
hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
if (HAL_I2C_Init(&hi2c1) != HAL_OK) {
  Error_Handler();
}

// OLED Initialization sequence (Adapt for your display!)
uint8_t init_sequence[] = {
    0xAE,       // Display off
    0x00,       // Set Memory Addressing Mode
                // 00=Horizontal Addressing Mode; 01=Vertical Addressing Mode;
                // 10=Page Addressing Mode (RESET); 11=Invalid
    0x10,       // Set Column Address Start Address for Page Addressing Mode.
    0x40,       // Set Page Start Address for Page Addressing Mode.
    0xB0,       // Set PAGE0 (0xB0~B7)
    0x81,       // Set contrast control register
    0xCF,       // Contrast 0x00~0xFF
    0xA1,       // Set segment re-map 0 to 127
    0xA4,       // Disable entire display on
    0xA6,       // Set normal display
    0xA8,       // Set multiplex ratio(1 to 64)
    0x3F,       // 1/64 duty
    0xC8,       //Com scan direction
    0xD3,       // Set display offset
    0x00,       // Offset
    0xD5,       // Set display clock divide ratio/oscillator frequency
    0x80,       // Set divide ratio, High frequency oscillator
    0xD9,       // Set pre-charge period
    0xF1,       // Set pre-charge period
    0xDA,       // Set com pins hardware configuration
    0x12,
    0xDB,       // Set vcomh
    0x20,       // Vcomh
    0x20,       //0x20,0x21,0x22
```

```
    0xA7,       // Set normal display
    0xAF        // Display on
  };

  HAL_I2C_Master_Transmit(&hi2c1, 0x78, init_sequence, sizeof(init_sequence),
HAL_MAX_DELAY); // OLED address is 0x3C << 1 = 0x78
}

void oled_display_text(char *text) {
   // Clear the OLED buffer
   uint8_t clear_buffer[128 * 8];  // 128 columns, 8 pages
   memset(clear_buffer, 0, sizeof(clear_buffer));
   HAL_I2C_Mem_Write(&hi2c1, 0x78, 0x40, I2C_MEMADD_SIZE_8BIT, clear_buffer,
sizeof(clear_buffer), HAL_MAX_DELAY);

   // Set cursor to the beginning of the display
   uint8_t set_cursor[] = {0xB0, 0x00, 0x10};  // Page 0, Column 0
   HAL_I2C_Master_Transmit(&hi2c1, 0x78, set_cursor, sizeof(set_cursor),
HAL_MAX_DELAY);

   // Send the text
   uint8_t data_type = 0x40;  // Data byte
   while (*text) {
      HAL_I2C_Mem_Write(&hi2c1, 0x78, data_type, I2C_MEMADD_SIZE_8BIT,
(uint8_t*)text, 1, HAL_MAX_DELAY);
      text++;
   }
}

void buzzer_init(void) {
  // Initialize buzzer pin as output
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  GPIO_InitStruct.Pin = GPIO_PIN_3;  // Example: PA3
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct); // Use the correct GPIO port
  buzzer_off(); // Ensure buzzer is off initially
}
```

```
void buzzer_on(void) {
  HAL_GPIO_WritePin(BUZZER_PIN, GPIO_PIN_SET);
}

void buzzer_off(void) {
  HAL_GPIO_WritePin(BUZZER_PIN, GPIO_PIN_RESET);
}

void Error_Handler(void) {
  // Handle errors (e.g., blink an LED, stop the program)
  while (1) {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5); // Example: Toggle LED on PA5
    HAL_Delay(100);
  }
}

void delay_us(uint16_t us) {
  //using timer
  __HAL_TIM_SET_COUNTER(&htim1, 0); //set counter to 0
  while(__HAL_TIM_GET_COUNTER(&htim1) < us);
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  //1 hour timer
  if(htim->Instance == TIM3){
    sittingTime++;
    if(sittingTime >= 3600){ //1 hour = 3600 seconds
      takeBreak = 1;
      sittingTime = 0;
    }
  }

  //30 sec absence timer
  if(htim->Instance == TIM4){
    if(!userPresent){
      absenceTime++;
      if(absenceTime >= 30){
```

```
      absenceTime = 0;
      sittingTime = 0;
      userPresent = 0;
      oled_display_text("Waiting for user...");
     }
    }
    else{
     absenceTime = 0;
    }
  }
}



void SystemClock_Config(void) {
  // Configure the system clock (80 MHz for Nucleo-L476RG)
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillator according to the specified
   * parameters in the RCC_OscInitTypeDef structure.
   */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_9; // 16 MHz
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 40;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV4; // 80 MHz
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
   */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK |
```

```
                                RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK) {
    Error_Handler();
  }
  /** Enable the main CPU cache:
   * Use the following if your device has a cache (check the datasheet)
   */
  /*if (HAL_RCCEx_EnableInstructionCache() != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_RCCEx_EnableDataCache() != HAL_OK)
  {
    Error_Handler();
  }*/
}

void HAL_MspInit(void) {
  // Low-level initialization (clock, GPIO)
  __HAL_RCC_SYSCFG_CLK_ENABLE();
  __HAL_RCC_PWR_CLK_ENABLE();

  // GPIO Initialization
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  //Ultrasonic TRIG pin
  __HAL_RCC_GPIOA_CLK_ENABLE();
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  //Ultrasonic ECHO pin
```

```
GPIO_InitStruct.Pin = GPIO_PIN_1;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

//Temperature Sensor pin
GPIO_InitStruct.Pin = GPIO_PIN_2;
GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

//Buzzer Pin
GPIO_InitStruct.Pin = GPIO_PIN_3;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

//OLED SCL and SDA pins
__HAL_RCC_GPIOB_CLK_ENABLE();
GPIO_InitStruct.Pin = GPIO_PIN_8 | GPIO_PIN_9;
GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;  // Open Drain for I2C
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

//TIM1 init
__HAL_RCC_TIM1_CLK_ENABLE();

//TIM3 init (1 hour timer)
__HAL_RCC_TIM3_CLK_ENABLE();
htim3.Instance = TIM3;
htim3.Init.Prescaler = 64000;  // 80MHz / 64000 = 1250 Hz (1ms per tick)
htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
htim3.Init.Period = 65535; // Maximum period
htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim3.Init.RepetitionCounter = 0;
htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
```

```
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  //enable interrupt
  HAL_NVIC_EnableIRQ(TIM3_IRQn);
  HAL_NVIC_SetPriority(TIM3_IRQn, 0, 0);

  //TIM4 init (30 sec timer)
    __HAL_RCC_TIM4_CLK_ENABLE();
  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 64000;  // 80MHz / 64000 = 1250 Hz (1ms per tick)
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 65535; // Maximum period
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.RepetitionCounter = 0;
  htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }
  //enable interrupt
  HAL_NVIC_EnableIRQ(TIM4_IRQn);
  HAL_NVIC_SetPriority(TIM4_IRQn, 0, 0);
}



// Timer interrupt handler
void TIM3_IRQHandler(void) {
  HAL_TIM_IRQHandler(&htim3);
}

void TIM4_IRQHandler(void) {
  HAL_TIM_IRQHandler(&htim4);
}
```

3. Explanation

- **Includes:** Includes necessary header files.
- **Defines:** Defines pin mappings for easy reference. **Adapt these!**
- **Function Prototypes:** Declares functions used in the code.
- **Global Variables:** Declares global variables for timer counters and states.
- **main() Function:**
    - Initializes the microcontroller and peripherals.
    - Enters the main loop:
        - Reads distance from the ultrasonic sensor.
        - Reads temperature from the temperature sensor.
        - Displays information on the OLED.
        - Controls the buzzer.
- **ultrasonic_init():** Configures Timer 1 to generate the trigger pulse for the ultrasonic sensor and measure the echo pulse.
- **ultrasonic_getDistance():** Sends the trigger pulse, measures the echo pulse duration, and calculates the distance.
- **temp_sensor_init():** Configures the ADC for reading the temperature sensor. **Modify this if your HW220 is not analog!**
- **temp_sensor_getTemp():** Reads the temperature from the sensor. **You MUST adapt this function to your HW220 sensor's specific output and