2022 - 2026

PORTFOLIO

Grace O. Zongo

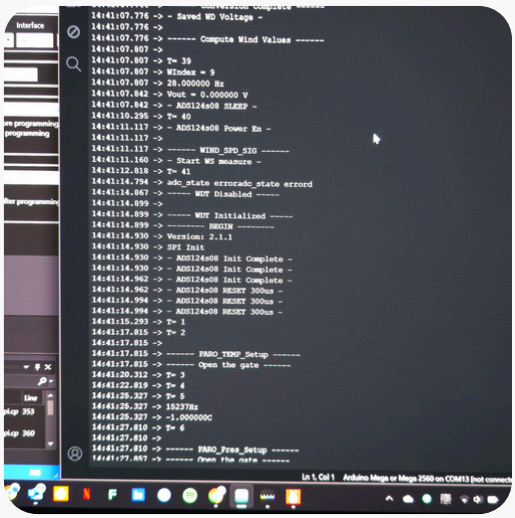
GRACE O. ZONGO

ELECTRICAL ENGINEERING AT UW-MADISON

+ 1 608 707 2473
gzongo@wisc.edu
https://zongr-ce.github.io/index.html

Sensors Drivers in C++ : Automatic Weather Station

```
class BAROMETER_25SAX {
    public:

        // initialize temp and pres
        void init_baro_temp( void );
        void init_baro_pres( void );

        // measure frequency coming through the multiplexer
        void measure_freq();

        // set temperature and pressure frequency
        void set_baro_temp_freq( void );
        void set_baro_pres_freq( void );

        // used to switch from Temp frequency measurement to pres frequency measurement
        void disable_mux( void );

        // calc temp and press
        void calc_temp( void );
        void calc_pres( void );

        // stop channeling barometer and pressure frequency through multiplexer, and disa
        void stop_baro( void );

        void TS_prescaler(int prescaler);

        // send measure pressure and temperature to the irridium
        double get_pres( void );
        double get_temp( void );
    private:
```



```
14:41:07.776 -> - Saved WD Voltage -
14:41:07.776 ->
14:41:07.776 -> ------ Compute Wind Values ------
14:41:07.807 ->
14:41:07.807 -> T= 39
14:41:07.807 -> WIndex = 9
14:41:07.807 -> 28.000000 Hz
14:41:07.842 -> Vout = 0.000000 V
14:41:07.842 -> - ADS124s08 SLEEP -
14:41:10.295 -> T= 40
14:41:11.117 -> - ADS124s08 Power En -
14:41:11.117 ->
14:41:11.117 -> ------ WIND_SPD_SIG ------
14:41:11.160 -> - Start WS measure -
14:41:12.818 -> T= 41
14:41:14.794 -> adc_state erroradc_state errord
14:41:14.867 -> ------ WDT Disabled ------
14:41:14.899 ->
14:41:14.899 -> ------ WDT Initialized ------
14:41:14.899 -> -------- BEGIN --------
14:41:14.930 -> Version: 2.1.1
14:41:14.930 -> SPI Init
14:41:14.930 -> - ADS124s08 Init Complete -
14:41:14.930 -> - ADS124s08 Init Complete -
14:41:14.962 -> - ADS124s08 Init Complete -
14:41:14.962 -> - ADS124s08 RESET 300us -
14:41:14.994 -> - ADS124s08 RESET 300us -
14:41:14.994 -> - ADS124s08 RESET 300us -
14:41:15.293 -> T= 1
14:41:17.815 -> T= 2
14:41:17.815 ->
14:41:17.815 -> ------ PARO_TEMP_Setup ------
14:41:17.815 -> ------ Open the gate ------
14:41:20.312 -> T= 3
14:41:22.819 -> T= 4
14:41:25.327 -> T= 5
14:41:26.327 -> 19237Hz
14:41:26.327 -> -1.000000C
14:41:27.810 -> T= 6
14:41:27.810 ->
14:41:27.810 -> ------ PARO_Pres_Setup ------
14:41:27.847 -> ------ Open the gate ------
```

What?

- Developed sensor drivers in C++ to interface with environmental sensors for an automatic weather station.

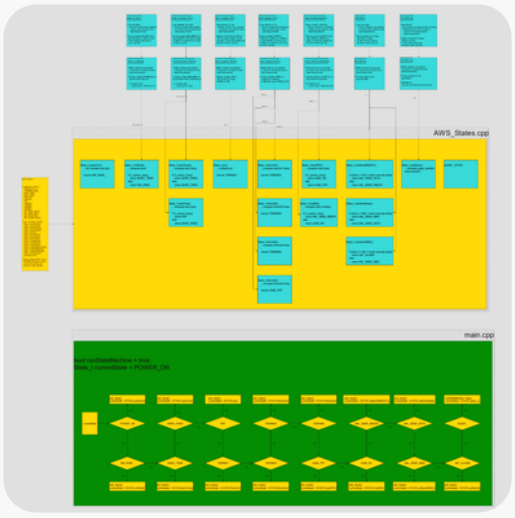
How?

- Implemented low-level C++ drivers to communicate with sensors
- Handled data acquisition, calibration, and error conditions
- Integrated sensor drivers into a larger embedded software system
- Tested functionality on the field

Results

- Successfully collected and processed environmental data (**pressure, humidity, temperature, etc...**)
- Successfully implemented frequency sampling techniques using **timers and Interrupt Service Routines (ISR)**
- Improved **reliability** and **modularity** of the weather station firmware
- Strengthened embedded C++ programming and hardware–software integration skills

Finite State Machine in C++ : Automatic Weather Station




```
(TIMER1_COMPA_vect){ // triggers every 4 seconds, and use the current value of ts_ovf
    RM_WIND_01.meas_wind_freq();
    wind_spd_ovf_flag = true;
}

ISR(TIMER5_OVF_vect) { // used to assist the measure of incoming frequencies in pin PL1
  PARO_BARO_01.measure_freq();
}

ISR(TIMER5_CAPT_vect){ //TIMER5_CAPT_vect  used for counting incoming pulse from the multi
  edge_count++;
}

ISR(PCINT2_vect)
{
  // increments all because there is no time to read which pin is high, pulse width = 16
  adc_01.Increment_drdy_count();
  adc_02.Increment_drdy_count();
  adc_03.Increment_drdy_count();
}

ISR (INT2_vect){  // Wake Up ISR
  if (!(PIND & (1 << INT_RTC))){
    POWER.sleep_OFF();
    error_wdt_reset = false;
    reset_device(); // Resetting the micro controller avoids memory leakage for the va
                    // this function uses the the watchdog timer to trigger a watchdog
  }
```

```
extern volatile bool USART_0_RX_String_Complete;
extern volatile bool USART_1_RX_String_Complete;

extern volatile int t_initial;  // records the time in seconds before we start the measurement cycle in main (t_initial
extern volatile int t;     // keeping track of how long the system has been running in seconds, starts at t = RTC.getPC
extern volatile int ticks;  // keeping track of how long the system has been running in seconds, but starts at 0 instead
extern volatile int cycle_duration; // total time in second for the measurement cycle (cycle_duration = t - t_initial)
extern volatile int prev_t; // to allow us to print ticks on the screen every second.

extern volatile bool error_wdt_reset; //refer to ISR(WDT_vect) in main.cpp to see how this variable is used
extern volatile uint16_t sleep_duration; // in seconds. Computed in the SET_ALARM state, normally = 600 - cycle_duration
                                         // a transmission every 10 minute in real time, so if the device runs for 120 s
                                         // Refer to setAlarm() in AWS_States.cpp to see how the value is computed to pr

//////////////////////////////////////////////////////////////////////////////////
// USART Buffers
extern char Buffer_usart_0[256];
extern char Buffer_usart_1[256];
extern char Iridium_Buffer[120];

/* Enumeration of all possible states in the system */
typedef enum {
    POWER_ON = 0, THERM, RM_WIND_, BARO_TEMP, BARO_PRES,
    HUMI_PRT, HUMI_RH, PRT, THERM01, THERM02, THERM03, THERM04,
    WAS_SEND_SBDWT, WAS_SEND_DATA, WAS_SEND_SBDI, SET_ALARM, SLEEP,
} State_t;

class SYSTEM_STATES{
    public:
        State_t powerOn(void);
        State_t rmWind(void);
        State_t baroTemp(void);
        State_t baroPres(void);
        State_t prt(void);
        State_t therm01(void);
        State_t therm02(void);
        State_t therm03(void);
        State_t therm04(void);
        State_t humPRT(void);
        State_t humRH(void);
        State_t wasSendSBDWT(void);
        State_t wasSendData(void);
        State_t wasSendSBDI(void);
```

What?

- Designed a finite state machine (FSM) in C++ to manage operational modes of an automatic weather station.

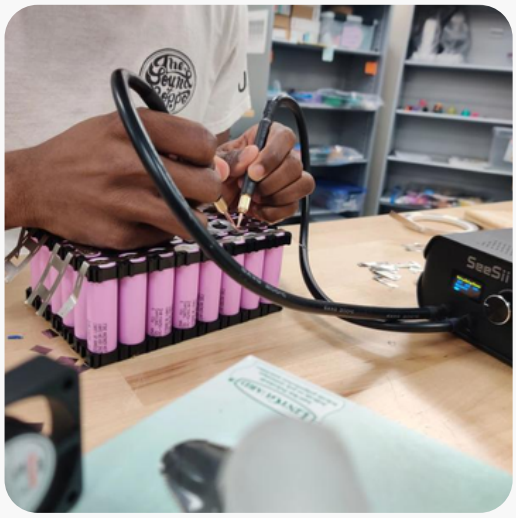
How?

- Defined system states for data collection, processing, and error handling
- Implemented FSM logic in C++ for predictable system behavior
- Integrated FSM with sensor drivers, RTC, and system timing
- Tested transitions and edge cases for robustness

Results

- Achieved reliable and deterministic system control
- AWS independently run in Antarctica for a **full year**
- Implemented sleep mode, reducing system runtime from **10 min** to **1 min 30 s**
- Reduced power consumption during idle mode by **~78%**

Battery Design & Management Systems : Electric Boat



What?

- Contributed to the design and management of a battery system and steering controls for an electric boat, focusing on safety, reliability, and performance.

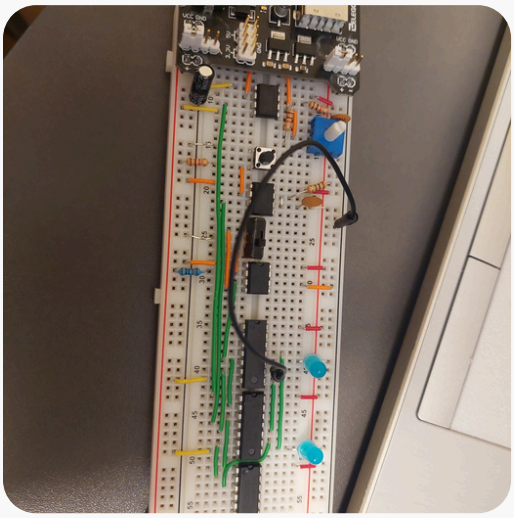
How?

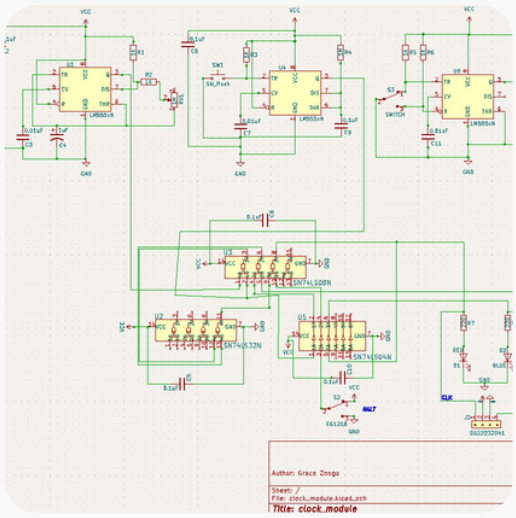
- Assisted in battery packs design and system-level integration
- Analyzed battery characteristics including voltage, current, and capacity
- Supported development of monitoring and safety strategies
- Identified and implemented optimal steering system for a catamaran hull.
- Collaborated with a multidisciplinary team to meet project requirements

Results

- Designed and validated a **dual 18650 Li-ion** battery system supporting **~2.5 hours** of continuous operation at a maximum speed of **8 knots**
- Integrated each battery pack with a **1 kW Blue Robotics T500 thruster**, achieving reliable in-water performance with an average current draw of **~10 A** per motor at full throttle
- Implemented differential thrust control to enable steering without a mechanical rudder
- Contributed to a **2nd place** finish in the Displacement Hull Division at the PEP 2024 Competition

555 timer Clock Module





What?

- Prototyped and designed a clock generation module using a 555 timer to provide a stable timing source for digital circuits.

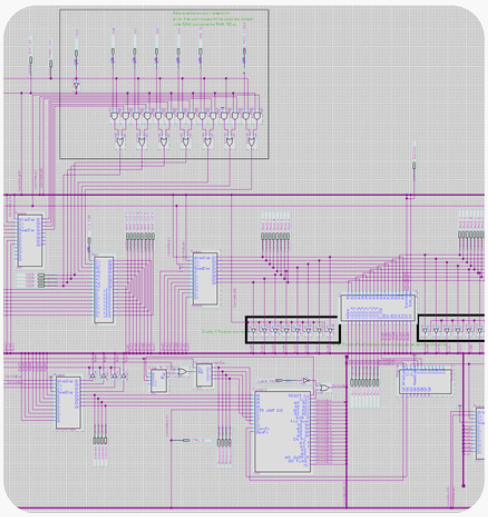
How?

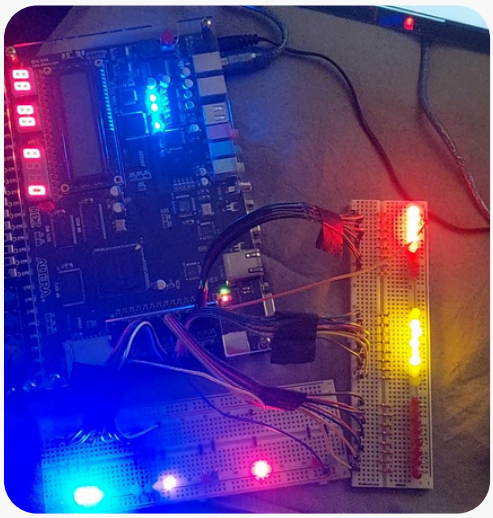
- Configured the **555 timer** in astable mode to generate a periodic clock signal
- Selected resistor and capacitor values to achieve target frequency
- Built and tested the circuit on breadboard and prototype hardware
- Verified output frequency and duty cycle using measurement tools

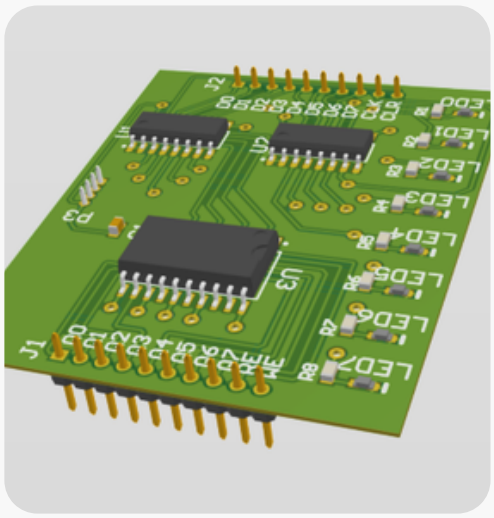
Results

- Generated a stable and reliable clock signal, from manual trigger to **~10 kHz** astable frequency, for use in digital systems
- Reinforced understanding of analog timing circuits and component-level design

8-bit CPU Design : FPGA & PCB Design







What?

- Designed and implemented a custom 8-bit CPU to understand processor architecture, instruction execution, and control logic at the hardware level.
- Extended the design to a **PCB-based implementation** to explore hardware integration, signal integrity, and real-world constraints beyond FPGA simulation.

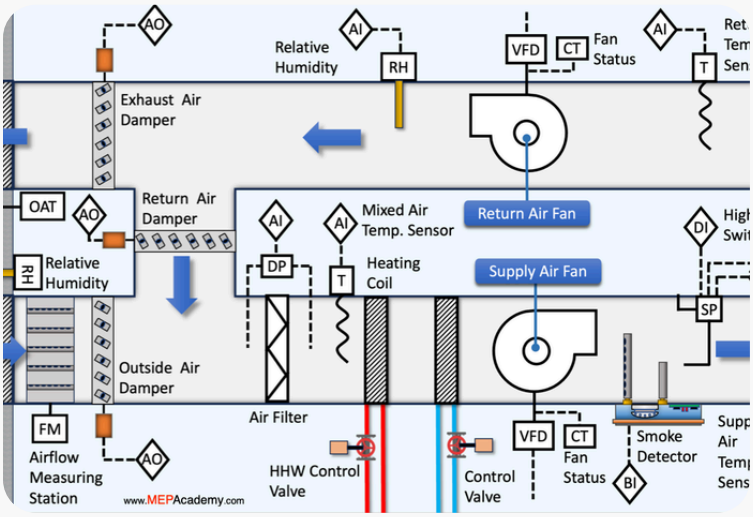
How?

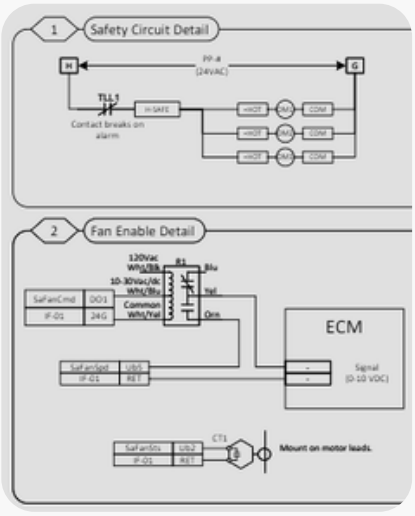
- Designed datapath and control unit using block diagram design in **Intel Quartus**
- Implemented key components including program counter, 8-bit register, RAM, ALU, and control unit from scratch using logic gates ONLY.
- Defined instruction sets supporting addition and subtraction
- Verified functionality using simulation, testbenches, and **waveform analysis**
- Translated digital CPU design into a **PCB-level hardware architecture**

Results

- Successfully executed all implemented instructions on FPGA hardware
- Achieved stable operation at **15 kHz** after resolving timing violations
- Improved understanding of design tradeoffs between **FPGA** and **discrete hardware** implementations
- Strengthened understanding of **computer architecture, FSM-based control, and FPGA debugging**
- Earned **1st place** at Honors Project Competition

HVAC System Wiring Diagram : Alpha Controls & Services





What?

- Developed **HVAC system wiring** diagrams for commercial control systems at Alpha Controls & Services.

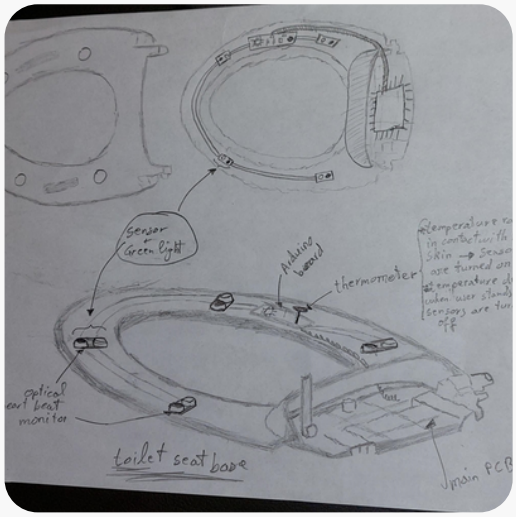
How?

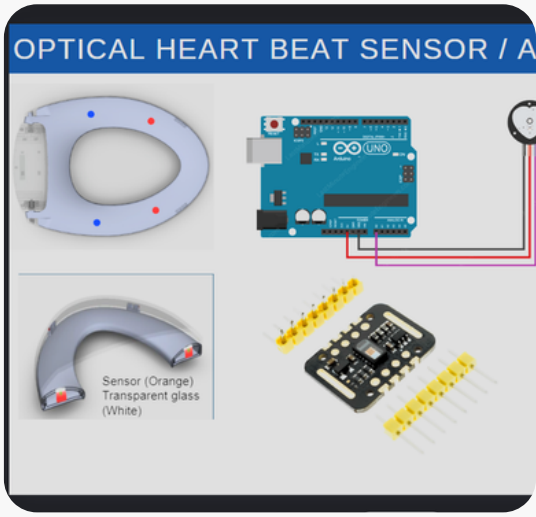
- Created detailed wiring diagrams for **HVAC control systems**
- Ensured compliance with electrical standards and **safety requirements**
- Coordinated with engineers, programmers, and technicians to support installation
- Reviewed designs for clarity and accuracy

Results

- Produced clear, accurate wiring documentation for real installations
- Improved understanding of HVAC control systems and industrial wiring practices

Smart toilet seat : Bemis manufacturing





What?

- Worked on the development of a smart toilet seat system in collaboration with Bemis Manufacturing, focusing on electrical functionality and integration.

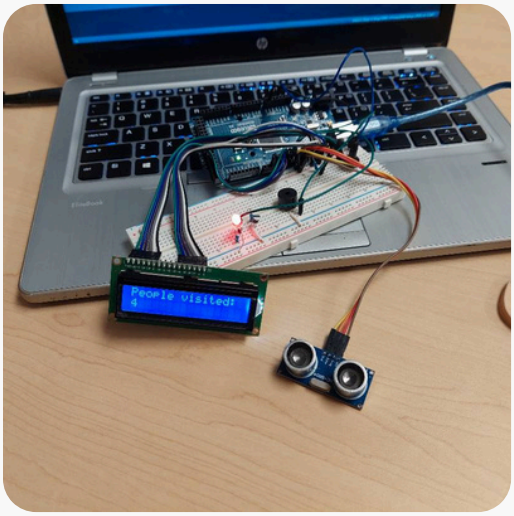
How?

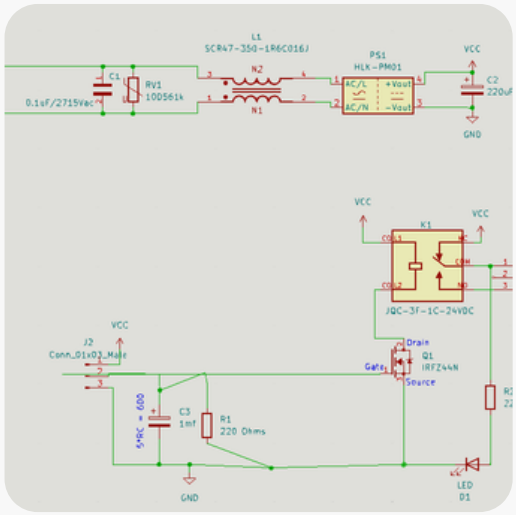
- Contributed to system-level electrical design and component integration
- Assisted with testing of electrical features
- Collaborated with industry engineers to meet product requirements
- Documented design decisions and test results

Results

- Supported development of a functional smart consumer product
- Gained experience working with industry standards and real-world constraints
- Strengthened collaboration and communication skills in an industrial setting

Affordable Occupancy sensor : Prototype & PCB





What?

- Prototyped and developed a low-cost occupancy sensor to detect human presence for energy-efficient automation applications.

How?

- Designed sensor circuitry and signal conditioning for reliable detection
- Created schematic and PCB layout for a compact prototype
- Integrated sensor output with digital processing logic
- Tested sensor accuracy and responsiveness under different conditions

Results

- Successfully detected occupancy with consistent performance
- Delivered a functional, low-cost prototype suitable for developing countries
- Gained experience in sensor integration and PCB prototyping