SOLID OXIDE ELECTROLYSIS CELLS FOR ASTA EPSILON CHEMICALS

Optimizations:

1. **Cathode (Porous GDC)**
   - **Active Layer**: Reduce thickness slightly (e.g., from 13 microns to ~10 microns) to increase catalyst utilization and reduce resistance while maintaining sufficient reaction surface area.
   - **Diffusion Layer**: Optimize thickness for gas transport, potentially reducing it from 280 microns to ~200 microns to minimize gas diffusion path resistance without compromising flow.
2. **Anode (Porous LSF)**
   - Optimize thickness to balance gas diffusion and conductivity. Consider reducing from 25 microns to ~20 microns to lower resistance while maintaining diffusion efficiency.
3. **Electrolyte (Non-porous ScSZ)**
   - Minimize thickness further, e.g., reduce from 3 microns to ~2 microns, to decrease ohmic resistance, as thinner electrolytes improve ion conductivity without compromising gas impermeability.
4. **Barrier Layer (GDC)**
   - Maintain or slightly reduce thickness (e.g., from 2 microns to ~1.5 microns) if structural stability and anode durability remain unaffected.
5. **Seals (Glass-based Ceramics)**
   - Review thermal expansion matching and minimize thickness (e.g., from 320 microns to ~250 microns) to improve thermal efficiency and reduce stack volume.
6. **Interconnects (Crofer 22 Steel)**
   - Maintain current thickness (500 microns), as reducing it may compromise mechanical stability. Ensure optimal LSF coating for better conductivity.
7. **End Plates (318 Stainless Steel)**
   - Review mechanical requirements and reduce thickness if possible (e.g., from 30,000 microns to ~20,000 microns) to lower stack weight and improve thermal efficiency while maintaining durability.

Additional Considerations:

1. **Thermal Management**
    - Ensure adequate heat dissipation and uniform temperature distribution to prevent thermal stresses and improve hydrogen production efficiency.
2. **Gas Flow Optimization**
    - Optimize the flow channels to reduce pressure drops and improve reactant gas distribution across the active area of the stack.
3. **Material Quality**
    - Use high-purity and defect-free materials to improve performance and durability.
4. **Operating Parameters**
    - Fine-tune operating conditions such as temperature, pressure, and gas composition for the 95:5 water-hydrogen feed gas to maximize electrochemical performance.

SPECIFIC DIMENSIONS

1. Cathode (Porous GDC)

- **Active Layer Thickness**: Reduced to **10 µm (0.01 mm)**.
- **Diffusion Layer Thickness**: Reduced to **200 µm (0.2 mm)**.
- **Total Cathode Thickness**: 10+200=210 µm (0.21 mm)

2. Anode (Porous LSF)

- **Thickness**: Reduced to **20 µm (0.02 mm)**.

3. Electrolyte (Non-porous ScSZ)

- **Thickness**: Reduced to **2 µm (0.002 mm)**.

4. Barrier Layer (GDC)

- **Thickness**: Reduced to **1.5 µm (0.0015 mm)**.

5. Seals (Glass-based Ceramics, SiO_2)

- **Thickness**: Reduced to **250 µm (0.25 mm)**.

6. Interconnects (Crofer 22 Steel)

- **Thickness**: Retained at **500 µm (0.5 mm)**.
- **LSF Coating Thickness**: Retained at **1 µm (0.001 mm)** on each side.
- **Total Interconnect Thickness (including coating)**: 500+2×1=502 µm (0.502 mm)

7. End Plates (318 Stainless Steel)

- **Thickness**: Reduced to **20,000 µm (20 mm)**.

Cell Assembly Height Calculation (Per Cell)

For each cell:

1. **Cathode**: 0.21 mm
2. **Anode**: 0.02 mm
3. **Electrolyte**: 0.002 mm
4. **Barrier Layer**: 0.0015 mm
5. **Seals (one layer per side)**: 0.25 mm×2=0.5 mm
6. **Interconnects**: 0.502 mm

Total Stack Height per Cell:
0.21+0.02+0.002+0.0015+0.5+0.502=1.2355 mm

Total Stack Height (3 Cells + End Plates)

For a 3-cell stack:

- **Cell Layers**: 1.2355 mm×3=3.7065 mm
- **End Plates (top and bottom)**: 20 mm×2=40 mm

Total Stack Height:
3.7065+40=43.7065 mm (≈43.71 mm)

Final Dimensions

- **Active Area**: 0.0016 m2 (40 mm×40 mm)
- **Total Stack Height**: 43.71 mm
- **Number of Cells**: 3.

PLC CONTROL FOR SOEC

Functional Requirements:

1. **Temperature Monitoring and Control**:
   - Operating temperatures of **600–700°C** with precise regulation and error logging.
2. **Error Monitoring and Logging**:
   - Monitor real-time stack parameters (voltage, current, pressure, temperature, etc.) and store error logs for troubleshooting.
3. **Cooling System Management**:
   - Control bidirectional liquid cooling flows and adjust cooling rates as needed.
4. **Gas Flow Monitoring**:

- Regulate input gas (temperature, pressure, and composition) and monitor output gas flow rates.

5. **Communications**:
   - Integration with higher-level systems via **MODBUS, CAN, or Ethernet**.
6. **Durability**:
   - Must handle industrial conditions with high reliability.
7. **Expandable I/O**:
   - Support multiple sensors and actuators for gas analysis, flowmeters, heaters, pumps, etc.

Suggested Sensors and Peripherals

To work with the controller:

1. **Temperature Sensors**: Type K thermocouples or PT100 RTDs.
2. **Pressure Sensors**: Industrial-grade pressure transducers.
3. **Gas Flow Meters**: Thermal mass flowmeters for input/output gases.
4. **Gas Analyzer**: Tuneable diode laser spectrometry (TDLS) for H_2 and H_2O analysis.
5. **Liquid Flow Control**: Solenoid valves with feedback for bidirectional liquid cooling.
6. **Actuators**: Relays or stepper motors for flow regulation.

Key Features of TMS320F28027

1. **High-Performance Core**:
   - **32-bit CPU** with up to 60 MHz clock speed for fast real-time processing.
   - Optimized for control algorithms like PID, making it suitable for temperature regulation and cooling system management.
2. **Analog Integration**:
   - **12-bit ADC** with up to 16 channels for accurate sensing of temperature, pressure, and other parameters.
   - ADCs are fast enough to handle real-time data from sensors like RTDs, thermocouples, and flowmeters.
3. **PWM Control**:
   - Multiple **PWM modules** for precise control of pumps, valves, or heaters.
   - Ideal for driving bidirectional liquid cooling systems or gas flow actuators.
4. **Communication Interfaces**:

- Supports **I²C, SPI, UART, and CAN**, making it compatible with various industrial sensors and actuators.
- Integrates seamlessly with external communication modules for protocols like MODBUS.

5. **Energy Efficiency**:
   - Low power consumption compared to PLCs, which is useful for reducing overall energy use in your SOEC system.
6. **Expandability**:
   - Works well with TI's development ecosystem, including Code Composer Studio and TI software libraries, to simplify programming and expand functionality.
7. **Cost-Effective**:
   - The TMS320F28027 is much more affordable than PLCs or high-end controllers like NI CompactRIO, making it ideal for budget-conscious applications.

Application in Your SOEC Stack System

Advantages

- **Temperature Monitoring and Control**: The built-in ADC can monitor temperature sensors like RTDs or thermocouples with external conditioning circuits.
- **Cooling Procedures**: PWM modules can precisely control pumps and valves in the bidirectional liquid cooling system.
- **Gas Flow Regulation**: Communication interfaces allow integration with mass flow controllers, pressure transducers, and gas analysers.
- **Error Monitoring and Logging**: The microcontroller can manage error handling, event logging, and reporting via CAN or UART to a host system.
- **Compact Design**: The small footprint makes it suitable for embedding directly in the stack's control system.

Challenges

1. **Limited Built-in Communication Protocols**:
   - Lacks native support for higher-level industrial protocols like MODBUS or PROFINET. This can be addressed by using external communication modules or stacks (e.g., MODBUS over UART).
2. **Development Complexity**:
   - Unlike PLCs, it requires custom firmware development, which may increase engineering time and effort.
3. **Scalability**:

- While the TMS320F28027 is adequate for a small SOEC stack, scaling to larger systems with more sensors/actuators might require additional I/O expanders or moving to a more powerful microcontroller (e.g., other C2000 series chips).

The **Nextion Intelligent NX8048P070-011C 7.0″ HMI Capacitive Touch Display** is compatible with the **TMS320F28027**, as it is designed to work with microcontrollers via **UART communication**, which the TMS320F28027 supports. Here's how it can be integrated and its advantages:

Compatibility Analysis

1. Communication Interface

- The Nextion Intelligent Display uses **UART** (serial communication) to communicate with the host microcontroller.
- The TMS320F28027 has **SCI (Serial Communication Interface)** modules that can easily interface with the Nextion display. You can configure the baud rate, data bits, and other UART settings on the TMS320F28027 to match the Nextion's requirements.

2. Memory and Processing

- The Nextion display **offloads all GUI rendering tasks** from the microcontroller, so the TMS320F28027 doesn't need to handle graphic-intensive operations.
- You can design the GUI using Nextion's **Nextion Editor Software**, upload it to the display, and then use simple serial commands from the microcontroller to update data or respond to touch inputs.

3. Power Requirements

- The Nextion NX8048P070-011C requires **5V power**, which must be supplied either directly from the power source or through a voltage regulator if the TMS320F28027 system operates at a different voltage level.

4. Programming

- The TMS320F28027 would send/receive data to/from the Nextion display using a **simple ASCII or binary protocol** over UART.
- Example: If you want to update a text box on the Nextion screen, the TMS320F28027 sends a predefined command string like:

Arduino : t0.txt="Hello"

- Similarly, touch events from the display are sent as serial messages to the microcontroller, which can process them to trigger corresponding actions.

Advantages of Using the Nextion NX8048P070-011C

1. **Ease of Integration**:
    - Requires minimal processing power or memory from the TMS320F28027.

- Simplifies GUI design with a drag-and-drop editor.

2. **Touchscreen Functionality**:
   - Offers capacitive touch with smooth operation, suitable for modern, user-friendly interfaces.
3. **Customizable GUI**:
   - GUI elements like buttons, sliders, and charts can be created directly using Nextion's Editor.
4. **Real-Time Data Display**:
   - Ideal for monitoring stack parameters such as temperature, pressure, and flow rates.
5. **Error Handling and Logging**:
   - Can display error codes and logs directly to users for easy debugging and maintenance.

System Integration Diagram

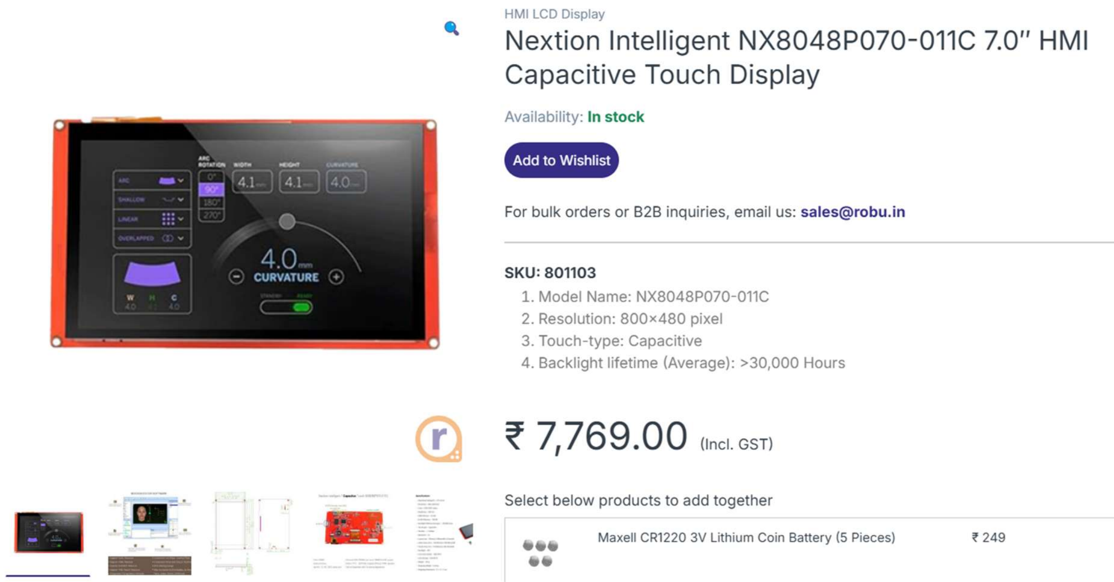
Here's a simplified system setup:

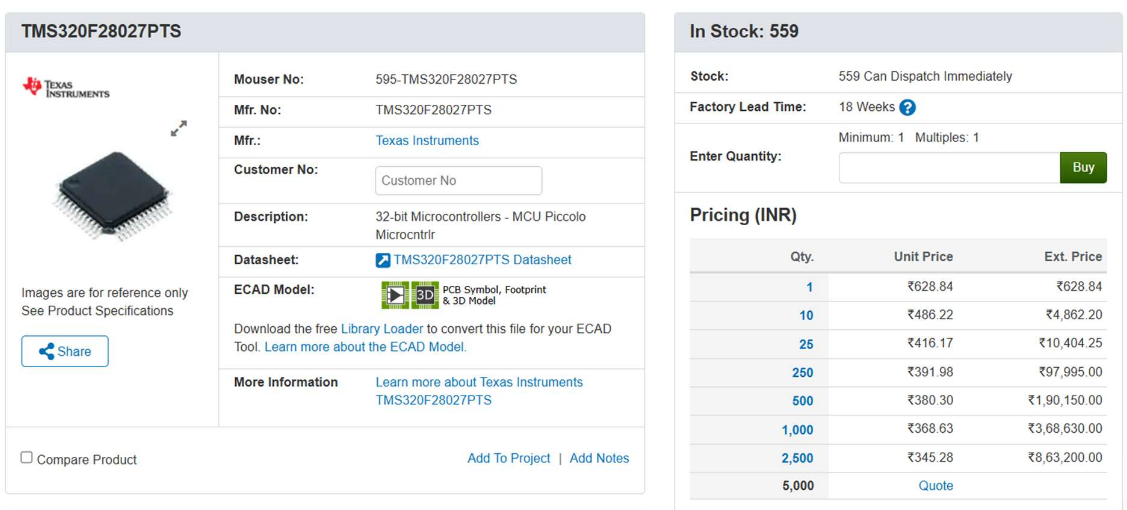
1. **Power Supply**:
   - Provide 5V power to the Nextion display.
   - Ensure the TMS320F28027 power supply matches its operating voltage.
2. **UART Communication**:
   - Connect the Nextion display's TX (transmit) and RX (receive) pins to the TMS320F28027's SCI RX and TX pins.
   - Configure the UART baud rate in the TMS320F28027 firmware to match the Nextion display (default is usually 9600 bps but can be adjusted in the Nextion Editor).
3. **Firmware Development**:
   - Use the TMS320F28027 to send/receive serial commands for:
     - Updating displayed values (e.g., temperature, pressure).
     - Responding to touch inputs (e.g., user actions like turning cooling on/off).

Considerations

1. **Signal Level Conversion**:
   - The Nextion display operates at **3.3V or 5V logic levels**. The TMS320F28027 is also a 3.3V device, so direct connections should work without additional level shifters.

2. **UART Port Availability**:
    - If the TMS320F28027 has only one UART and it's being used for other purposes, you might need to use a **UART multiplexer** or a software-based serial implementation.
3. **Software Complexity**:
    - While the Nextion Editor simplifies GUI design, you need to carefully manage serial communication in the TMS320F28027 firmware to handle updates and responses in real time.





1. Sensors Interfacing

a. Temperature Sensors

- **Sensor Type**: RTD (e.g., PT100/PT1000), Thermocouple (e.g., Type K), or digital temperature sensors (e.g., DS18B20).

- **Interface**:
  - RTD/Thermocouple: Connect to the **ADC** via a precision instrumentation amplifier (e.g., INA333) for signal conditioning. Use a 24-bit ADC (e.g., ADS124S08) for high precision if needed.
  - Digital Sensors: Use **UART**, **I2C**, or **1-Wire** protocols for direct interfacing (e.g., DS18B20 via 1-Wire, TMP117 via I2C).
- **Microcontroller Pins**:
  - Analog sensors: Connect to the ADC pins, ensuring proper scaling to match the ADC voltage range (0–3.3V).
  - Digital sensors: Use I2C or UART pins for communication.

b. Pressure Sensors

- **Sensor Type**: Analog (e.g., 4-20mA output) or digital (e.g., I2C/SPI-based).
- **Interface**:
  - For 4-20mA sensors: Use a resistor (250 Ω for 1-5V output) and connect to an **ADC** input.
  - For I2C/SPI sensors: Connect to respective **I2C** or **SPI** pins.
- **Microcontroller Pins**:
  - ADC for analog sensors, or I2C/SPI for digital sensors.

c. Gas Mixture Analysis (Flow & Composition)

- **Sensors**: Use MEMS-based flow sensors (e.g., Honeywell AWM series) and gas concentration sensors (e.g., CO2, H2 sensors from Alphasense or Sensirion).
- **Interface**:
  - Flow sensors: Typically analog output, connected to the ADC.
  - Gas concentration sensors: Use digital interfaces like I2C or SPI.

2. Relays Interfacing

- **Relay Type**: 5V or 12V DC relays for switching pumps, valves, or alarms.
- **Interface**:
  - Use a transistor (e.g., 2N2222) or MOSFET (e.g., IRLZ44N) as a switch to control the relay coil.
  - Place a flyback diode (e.g., 1N4007) across the relay coil to prevent back EMF damage.
- **Microcontroller Pins**:

- Connect GPIO pins to the base/gate of the transistor or MOSFET through a current-limiting resistor (e.g., 1kΩ).
- Use PWM GPIOs for proportional control if relays are replaced with solenoids or motor drivers.

3. Cooling System Control

a. Pumps and Fans

- **Control Method**: Use PWM signals to control pump/fan speed.
- **Interface**:
  - Use a motor driver IC (e.g., L298N or DRV8871) or MOSFET for high-current loads.
- **Microcontroller Pins**:
  - Use the PWM output from the **ePWM module** to drive the control input of the driver IC or MOSFET.

b. Temperature Feedback for Cooling

- Use temperature sensors (as described above) to monitor cooling liquid and control pumps/fans dynamically.

4. Display and Communication

a. 7" Touchscreen (Nextion Intelligent Display NX8048P070-011C)

- **Interface**:
  - Connect the touchscreen via the UART interface.
  - Use an appropriate baud rate (typically 9600 or 115200) to communicate with the display.
- **Microcontroller Pins**:
  - Assign one UART module (TX/RX pins) for display communication.

5. Gas Flow Regulation

a. Input Gas Mixture Setup

- **Actuators**: Use proportional solenoid valves (e.g., Parker Hannifin valves).
- **Interface**:
  - Control valves via PWM signals generated by the ePWM module.
  - Feedback loop from flow sensors to regulate gas mixture dynamically.
- **Microcontroller Pins**:
  - PWM GPIOs for controlling valves.

- ADC or digital communication for flow sensor feedback.

b. Output Gas Flow Metering

- **Sensors**: Use digital gas flow meters (e.g., SFM3000 series from Sensirion).
- **Interface**:
  - Connect via I2C or UART for data exchange.
- **Microcontroller Pins**:
  - I2C or UART pins for data acquisition.

6. Error Logging and Data Storage

- Use an external EEPROM or SD card module for error logs.
- **Interface**:
  - Use SPI or I2C for EEPROM (e.g., 24C64) or SD card modules.
- **Microcontroller Pins**:
  - Assign an SPI or I2C module for storage purposes.

7. Power Supply and Protection

- **Power Supply**:
  - Provide a stable 3.3V for the microcontroller and peripherals using a buck converter (e.g., LM2596).
  - Ensure isolation for high-voltage interfaces using optocouplers (e.g., PC817) or isolated ADCs.
- **Protection**:
  - Use TVS diodes, fuses, and ESD protection on input/output lines to prevent damage.

Pin Assignment Example for TMS320F28027 (64-pin)

Peripheral	Microcontroller Pins	Notes
Temperature Sensors	ADCIN0–ADCIN7	Use analog input pins for RTD/Thermocouples.
Pressure Sensors	ADCIN8–ADCIN15 / I2C	Analog or I2C-based communication.
Relays	GPIO16–GPIO31	Drive via transistor/MOSFET circuits.
Cooling Fans/Pumps	ePWM1A, ePWM2A	Use PWM outputs for proportional control.

Touchscreen Display	UART0 (TX, RX)	Communicate with the Nextion display.
Gas Flow Sensors	ADCIN16–ADCIN23 / I2C	Use for flow or concentration sensing.
SD Card/EEPROM	SPI0 (MISO, MOSI, CLK)	For error log storage.

Key Features of the Circuit

1. **Power Supply and Regulation**
2. **Temperature Sensor Interfacing**
3. **Pressure and Flow Sensor Interfacing**
4. **Relay and Cooling System Interfacing**
5. **Touchscreen Display Interface**
6. **Error Logging and Storage**
7. **Protection Features**

Detailed Circuit Schematic

1. Power Supply Circuit

- **Purpose**: Provide 3.3V for the microcontroller and sensors, 5V/12V for relays, and peripherals.
- **Components**:
  - **Input**: 24V DC (industrial standard) or 12V DC.
  - **Buck Converter**: LM2596 (adjustable, for 3.3V and 5V outputs).
  - **Capacitors**: 100 µF and 10 µF (for decoupling).
  - **Diode**: 1N5822 (Schottky, for reverse polarity protection).
  - **Ferrite Bead**: To suppress high-frequency noise.

2. Temperature Sensor Interfacing

- **Sensors**: RTD (PT100/PT1000) with 4-20mA output or thermocouple.
- **Signal Conditioning**:
  - **Instrumentation Amplifier**: INA333 (for RTD/thermocouple voltage amplification).
  - **Resistor**: Precision resistor (250 Ω) for 4-20mA conversion.
  - **Low-pass Filter**: R-C filter with 10 kΩ and 0.1 µF capacitor.

- **ADC Inputs**: Connect to TMS320F28027 ADC pins (e.g., ADCINA0).

3. Pressure and Flow Sensors

- **Sensors**: Honeywell AWM series or Sensirion digital sensors.
- **Interface**:
  - Analog: Use ADC pins with appropriate scaling (e.g., voltage divider).
  - Digital (I2C/SPI): Direct connection to I2C/SPI pins on TMS320F28027.
  - **Pull-up Resistors**: 4.7 kΩ for I2C lines (SDA, SCL).

4. Relay and Cooling System Control

- **Relays**: 12V DC relay modules (e.g., Songle SRD-12VDC-SL-C).
- **Driver Circuit**:
  - **Transistor**: 2N2222 or MOSFET IRLZ44N (logic-level).
  - **Flyback Diode**: 1N4007 across relay coil.
  - **Base Resistor**: 1 kΩ for transistor base.
- **Cooling System (Pumps/Fans)**:
  - **Motor Driver IC**: DRV8871 (PWM input for fan/pump speed control).
  - Connect ePWM pins (e.g., ePWM1A, ePWM2A) to control the driver.

5. Touchscreen Display Interface

- **Display**: Nextion NX8048P070-011C.
- **Interface**:
  - **UART Connection**: TX, RX lines of TMS320F28027 to display TX/RX pins.
  - **Voltage Level Shifter**: Use resistors or TXB0104 bidirectional level shifter if display requires 5V logic.

6. Error Logging and Storage

- **Storage Module**: SD card (SPI interface) or EEPROM (I2C).
- **Components**:
  - **Micro SD Card Module**: SPI (e.g., Catalex MicroSD Module).
  - **EEPROM**: AT24C64 (I2C-based).
  - **Pull-up Resistors**: 4.7 kΩ for I2C lines.

7. Protection Features

- **TVS Diodes**: SMAJ5.0A for transient voltage suppression.
- **Fuses**: 1A for microcontroller power input.

- **Optocouplers**: PC817 for isolating relay control signals.

Bill of Materials (BOM)

Component	Part Number/Description	Quantity	Purpose
Microcontroller	TMS320F28027PTS	1	Main control unit.
Buck Converter	LM2596	2	Power supply (3.3V, 5V).
Temperature Sensor	PT100/PT1000 or TMP117	Varies	Temperature monitoring.
Pressure Sensor	Honeywell AWM or Sensirion SPI	Varies	Gas pressure monitoring.
Relays	SRD-12VDC-SL-C	4-6	Switching pumps/valves.
Transistor	2N2222 or IRLZ44N	4-6	Relay driver.
Diodes	1N4007	6	Flyback protection.
Display	Nextion NX8048P070-011C	1	User interface.
SD Card Module	Catalex MicroSD	1	Data logging.
Motor Driver IC	DRV8871	2	Cooling pump/fan control.
Capacitors	100 µF, 10 µF	6 each	Power supply decoupling.
Resistors	1 kΩ, 10 kΩ	10 each	Signal conditioning.
TVS Diodes	SMAJ5.0A	4	Voltage suppression.
Optocouplers	PC817	4	Signal isolation.
Ferrite Bead	Laird FB43	2	Noise suppression.

ALGORITHM AND PSEUDOCODE FOR TMS320F28027PTS

Algorithm for TMS320F28027PTS

1. **Initialize System**:
    - Configure GPIO pins, ADCs, PWMs, UART, and I2C/SPI interfaces.
    - Set up interrupts and timers.
    - Initialize sensors, relays, and the cooling system.
2. **Continuous Monitoring**:

- Read data from sensors (temperature, pressure, flow, gas composition).
- Analyze sensor values for errors or out-of-range conditions.

3. **Temperature Regulation**:
   - Adjust cooling system (bidirectional liquid flow) using PWM to maintain the temperature between 600°C–700°C.
4. **Gas Flow Regulation**:
   - Control gas input/output flow rates based on stack performance using relays and motor drivers.
5. **Error Handling**:
   - Detect anomalies (e.g., overpressure, overheating, sensor failure).
   - Log errors in storage and display them on the touchscreen.
6. **Touchscreen Interaction**:
   - Display real-time stack data (temperature, pressure, flow rates).
   - Process user commands via the touchscreen (e.g., adjusting operating conditions).
7. **Data Logging**:
   - Store operating data and error logs in the SD card.
8. **Safety Shutdown**:
   - Initiate an emergency shutdown in critical failure scenarios.

```
// Pseudocode for TMS320F28027PTS


// Libraries and Headers

#include "device.h"

#include "adc.h"

#include "pwm.h"

#include "uart.h"

#include "i2c.h"

#include "spi.h"


// Global Variables
```

```
float temperatureSensors[3];       // Array for temperature
data

float pressureSensor;              // Pressure sensor data

float flowRate;                    // Flow sensor data

int errorLog[10];                  // Error log storage

bool emergencyShutdown = false;    // Emergency shutdown flag


// Initialization

void initSystem() {

    configureGPIO();               // Configure GPIO pins

    configureADC();                // Initialize ADC for
sensor readings

    configurePWM();                // Initialize PWM for
cooling control

    configureUART();               // Initialize UART for
touchscreen

    configureI2C();                // Initialize I2C for
EEPROM/Sensors

    configureSPI();                // Initialize SPI for SD
card

    initDisplay();                 // Initialize touchscreen

}


// Read Sensors

void readSensors() {

    for (int i = 0; i < 3; i++) {

        temperatureSensors[i] = readADC(i);  // Read
temperature sensors

    }

    pressureSensor = readADC(3);             // Read pressure
sensor

    flowRate = readFlowSensor();             // Read flow
sensor via I2C/SPI

}
```

```
// Control Cooling System
void regulateTemperature() {
    float avgTemp = (temperatureSensors[0] +
temperatureSensors[1] + temperatureSensors[2]) / 3.0;
    if (avgTemp > 700.0) {
        setPWM(coolingFanPin, HIGH_SPEED);   // Increase
cooling speed
    } else if (avgTemp < 600.0) {
        setPWM(coolingFanPin, LOW_SPEED);    // Decrease
cooling speed
    } else {
        setPWM(coolingFanPin, NORMAL_SPEED); // Maintain
normal speed
    }
}

// Gas Flow Control
void regulateGasFlow() {
    if (pressureSensor > MAX_PRESSURE || flowRate > MAX_FLOW)
{
        stopGasInput();                      // Shut off gas
input relay
    } else if (pressureSensor < MIN_PRESSURE) {
        startGasInput();                     // Enable gas
input relay
    }
}

// Error Detection and Logging
void checkErrors() {
    if (temperatureSensors[0] > MAX_TEMP || pressureSensor >
MAX_PRESSURE) {
```

```
        errorLog[logIndex++] = CRITICAL_ERROR; // Log critical
error
        emergencyShutdown = true;            // Set shutdown
flag
    }
    if (sensorFail()) {
        errorLog[logIndex++] = SENSOR_ERROR; // Log sensor
error
    }
}

// Touchscreen Display Update
void updateDisplay() {
    sendUART("Temp1:", temperatureSensors[0]);
    sendUART("Temp2:", temperatureSensors[1]);
    sendUART("Temp3:", temperatureSensors[2]);
    sendUART("Pressure:", pressureSensor);
    sendUART("Flow Rate:", flowRate);
}

// Main Loop
void mainLoop() {
    while (!emergencyShutdown) {
        readSensors();                       // Read all
sensors
        regulateTemperature();               // Control
cooling system
        regulateGasFlow();                   // Control gas
flow
        checkErrors();                       // Detect and
handle errors
        updateDisplay();                     // Update
touchscreen
```

```
        logData();                              // Log data to SD
card
    }
    initiateShutdown();                         // Safety
shutdown procedure
}

// Emergency Shutdown
void initiateShutdown() {
    stopAllProcesses();                         // Stop pumps,
relays, etc.
    displayError("Emergency Shutdown");         // Notify user
    logError("Critical Failure");               // Log failure
}

// Main Function
int main() {
    initSystem();                               // Initialize
system components
    mainLoop();                                 // Start main
loop
    return 0;
}
```

CODE IN C FOR TMS320F28027PTS

```
#include "F2802x_Device.h"  // Include device header file
#include "adc.h"          // Include ADC library
#include "pwm.h"          // Include PWM library
#include "uart.h"         // Include UART library
#include "i2c.h"          // Include I2C library
#include "spi.h"          // Include SPI library
#include "gpio.h"         // Include GPIO library
```

```
// Define constants
#define MAX_TEMP 700.0
#define MIN_TEMP 600.0
#define MAX_PRESSURE 10.0   // Example value (bar)
#define MIN_PRESSURE 5.0    // Example value (bar)
#define MAX_FLOW 100.0      // Example value (L/min)
#define COOLING_PWM_PIN 1   // Example PWM pin for cooling
#define GAS_RELAY_PIN 2     // Example GPIO pin for gas relay

// Global variables
float temperatureSensors[3] = {0.0};  // Temperature sensor readings
float pressureSensor = 0.0;           // Pressure sensor reading
float flowRate = 0.0;                 // Flow rate sensor reading
int errorLog[10] = {0};               // Error log
bool emergencyShutdown = false;       // Emergency shutdown flag
int logIndex = 0;                     // Error log index

// Function prototypes
void initSystem(void);
void readSensors(void);
void regulateTemperature(void);
void regulateGasFlow(void);
void checkErrors(void);
void updateDisplay(void);
void logData(void);
void initiateShutdown(void);
```

```
// Initialize system peripherals
void initSystem(void) {
    DisableDog();                // Disable watchdog timer
    InitSysCtrl();               // Initialize system control
    DINT;                        // Disable CPU interrupts
    InitGpio();                  // Initialize GPIO pins
    InitAdc();                   // Initialize ADC
    InitPwm();                   // Initialize PWM
    InitUart();                  // Initialize UART for
display
    InitI2C();                   // Initialize I2C for sensors
    InitSpi();                   // Initialize SPI for SD card
    EINT;                        // Enable global interrupt
    ERTM;                        // Enable real-time interrupt
}

// Read sensor data
void readSensors(void) {
    for (int i = 0; i < 3; i++) {
        temperatureSensors[i] = ReadAdc(i);  // Read
temperature sensors via ADC
    }
    pressureSensor = ReadAdc(3);             // Read pressure
sensor via ADC
    flowRate = ReadFlowSensor();             // Read flow
sensor via I2C or SPI
}

// Regulate temperature using cooling system
void regulateTemperature(void) {
    float avgTemp = (temperatureSensors[0] +
temperatureSensors[1] + temperatureSensors[2]) / 3.0;
```

```
    if (avgTemp > MAX_TEMP) {
        SetPwmDutyCycle(COOLING_PWM_PIN, 80);  // Increase cooling system speed (80%)
    } else if (avgTemp < MIN_TEMP) {
        SetPwmDutyCycle(COOLING_PWM_PIN, 40);  // Decrease cooling system speed (40%)
    } else {
        SetPwmDutyCycle(COOLING_PWM_PIN, 60);  // Maintain normal cooling speed (60%)
    }
}

// Regulate gas flow
void regulateGasFlow(void) {
    if (pressureSensor > MAX_PRESSURE || flowRate > MAX_FLOW) {
        GpioDataRegs.GPASET.bit.GAS_RELAY_PIN = 0;  // Turn off gas input relay
    } else if (pressureSensor < MIN_PRESSURE) {
        GpioDataRegs.GPACLEAR.bit.GAS_RELAY_PIN = 1; // Turn on gas input relay
    }
}

// Check for errors and log them
void checkErrors(void) {
    if (temperatureSensors[0] > MAX_TEMP || pressureSensor > MAX_PRESSURE) {
        errorLog[logIndex++] = 1; // Log critical error (code 1)
        emergencyShutdown = true;
    }
    if (temperatureSensors[0] < 0 || pressureSensor < 0) {
```

```
        errorLog[logIndex++] = 2; // Log sensor failure (code
2)
    }
    if (logIndex >= 10) {
        logIndex = 0; // Reset log index if it exceeds
capacity
    }
}

// Update touchscreen display via UART
void updateDisplay(void) {
    SendUart("Temp1: ", temperatureSensors[0]);
    SendUart("Temp2: ", temperatureSensors[1]);
    SendUart("Temp3: ", temperatureSensors[2]);
    SendUart("Pressure: ", pressureSensor);
    SendUart("Flow Rate: ", flowRate);
}

// Log data to SD card (via SPI)
void logData(void) {
    WriteToSDCard(temperatureSensors, pressureSensor,
flowRate);
}

// Emergency shutdown procedure
void initiateShutdown(void) {
    GpioDataRegs.GPACLEAR.all = 0xFFFFFFFF;  // Turn off all
GPIOs
    SetPwmDutyCycle(COOLING_PWM_PIN, 0);    // Stop cooling
system
    SendUart("Emergency Shutdown Initiated\n", 0);
    WriteToSDCardErrorLog(errorLog, logIndex);
```

```
}

// Main loop
void main(void) {
    initSystem();  // Initialize system peripherals

    while (!emergencyShutdown) {
        readSensors();           // Read all sensors
        regulateTemperature();  // Control cooling system
        regulateGasFlow();       // Control gas flow
        checkErrors();           // Detect and handle errors
        updateDisplay();         // Update touchscreen display
        logData();               // Log data to SD card
        DELAY_US(1000000);       // Delay for 1 second
    }

    initiateShutdown();          // Perform emergency shutdown
}
```

COST ANALYSIS FOR ELECTRONICS CIRCUIT

Category	**Component**	**Part Number**	**Quantity**	**Cost (INR)**
Microcontroller	TMS320F28027PTS (TI C2000 Piccolo MCU)	TMS320F28027PTS	1 (for commercial)	628~
Development Tools	LaunchPad Development Kit for TMS320F28027	LAUNCHXL-F28027	1	3002
Pressure Sensors	Pressure Transducer (0–15 bar, Analog Output)	Honeywell PX3AG1BS015BSAAX	2	2450
Temperature Sensors	Thermocouple Amplifier for Type-K Thermocouples	MAX6675	3	199*3 = 597

	Type-K Thermocouples (Stainless Steel)	-	3	50*3 = 150
Flow Sensors	Liquid Flow Sensor (Bidirectional)	FS300A	2	558*2 = 1116
	Gas Mass Flow Sensor	Honeywell AWM5000 Series	2	7500*2 = 15000
Voltage Sensor	Voltage Divider Resistor Network	100kΩ + 1kΩ Resistors	1	15
Current Sensors	High-Current Sensor	ACS758LCB-100B-PFF-T	1	582
	Precision High-Side Current Monitor	ZXCT1009	1	40
Actuators	Proportional Solenoid Valve for Gas Control	ASCO 090	2	1900*2 = 3800
	DC Water Pump	Generic 12V, 10 L/min Pump	1	372
Relays	5V Relay	Omron G2R-1-S DC5	2	459*2 = 918
MOSFETs	N-Channel MOSFET	IRF540N	4	25*4 = 100
Power Supplies	5V Power Supply Module	Mean Well LRS-50-5	1	1769
Touchscreen Display	7" HMI Capacitive Touch Display	Nextion NX8048P070-011C	1	7769
Communication Cable	USB-to-UART Cable	FTDI TTL-232R-5V	1	2261
I2C Expander	I2C GPIO Expander	PCF8574	2	75*2 = 150
UART Multiplexer	Quad UART with SPI/I2C Interface	MAX14830	1	544
Data Logging	MicroSD Card Reader Module for SPI	-	1	25
	16GB MicroSD Card	-	1	

Passive Components	Resistors	Various (1kΩ, 10kΩ, 100kΩ, etc.)	Assorted	9, 10, 39
	Capacitors	Various (10nF, 100nF, 10µF, etc.)	Assorted	6, 199, 25
	Diodes	1N4007	10	10*1 = 10
Connectors	Screw Terminals	5mm pitch	10	6*10 = 60
	Pin Headers (Male and Female)	-	20 pairs	71*20 = 1420
Cooling System	Liquid Cooling Tubes	Silicone Tubing (6mm ID)	2 meters	41*2 = 82
	Cooling Radiator	120mm Radiator	1	2369
Enclosure	Metal Enclosure	Custom size	1	1200
Cables/Wires	Assorted Wires for Power and Signal	22AWG, 16AWG	10 meters	49, 18

Total Estimated Cost for Prototype Electronics: INR 46156

Total Estimated Cost for Prototype Chemicals: INR 27000

Grand Total of Estimated Costs: INR 73156