Multi-Sensor Security System

Embedded Security Project Report

Name: Milan Jani
Enrollment No: 92301733041
Domain: Embedded Systems – Security Sector
Microcontroller: MSP430F5529 LaunchPad
IDE: Code Composer Studio (CCS)
Date: December 2025

---

1. Project Introduction

1.1 Overview

This project is a **comprehensive security system** that combines multiple sensors and actuators to protect a home or building. The system can detect: - Intruders (motion detection) - Fire/flames - Gas leaks - Emergency situations (panic button)

It also has a **password-protected door lock** that can be controlled remotely via UART communication.

1.2 Why This Project?

Real-world application: Can be used in homes, offices, labs

Multi-threat detection: Handles 4 different types of dangers

Interrupt-based design: Fast response time (hardware reacts immediately)

Priority system: Fire is handled first, then other threats

User-friendly: LCD shows real-time status, password entry is simple

1.3 Key Features Implemented

IR Motion Sensor - Detects intruders
Flame Sensor - Detects fire instantly
MQ-6 Gas Sensor - Detects LPG/propane leaks
UART Password Lock - Remote door control
SOS Panic Button - Emergency alert
I2C LCD Display - Shows gas levels and system status
Visual Indicators - LEDs for unlock confirmation

---

2. Hardware Architecture

2.1 Components Used

Component	Model/Type	Purpose
Microcontroller	MSP430F5529 LaunchPad	Main brain of the system
IR Sensor	Digital PIR/IR	Detects human motion
Flame Sensor	Digital Flame Detector	Detects fire/flames
Gas Sensor	MQ-6 (Analog)	Detects LPG gas leaks
Servo Motor	SG90 (180°)	Locks/unlocks door
LCD Display	16x2 I2C LCD	Shows status messages
Buzzer	Active Buzzer	Alarm sound
Push Button	Tactile Switch	SOS panic button
LED	Green LED	Unlock indicator

2.2 Pin Configuration

```
MSP430F5529 Pin Mapping:
├── PORT 1
│   ├── P1.0 → Green LED (unlock indicator)
│   ├── P1.3 → IR Sensor (input with interrupt)
│   └── P1.6 → Buzzer (output)
├── PORT 2
│   ├── P2.0 → Servo Motor PWM (TA1.1)
│   ├── P2.1 → SOS Button (input with interrupt)
│   └── P2.7 → Flame Sensor (input with interrupt)
├── PORT 3
│   ├── P3.0 → I2C SDA (LCD data)
│   └── P3.1 → I2C SCL (LCD clock)
├── PORT 4
│   ├── P4.4 → UART TX (transmit)
│   └── P4.5 → UART RX (receive)
└── PORT 6
    └── P6.0 → MQ-6 Gas Sensor (analog ADC input)
```

2.3 Circuit Block Diagram

```
            ┌──────────────────────────────────┐
            │   MSP430F5529 LaunchPad          │
            │   (Main Controller)               │
            └──────────────────────────────────┘
                       ▲  ▲  ▲  ▲
          ┌────────────┘  │  │  └────────────┐
          │               │  │               │
     ┌────▼────┐    ┌─────▼──▼─────┐   ┌─────▼──────┐
     │ Sensors │    │   Display    │   │ Actuators  │
     │ (Input) │    │  & Comms     │   │  (Output)  │
     └─────────┘    └──────────────┘   └────────────┘
          │               │                  │
     ├─ IR Sensor     ├─ LCD 16x2       ├─ Buzzer
     ├─ Flame         ├─ UART           ├─ Servo Motor
     ├─ Gas MQ-6      └─ Green LED      └─ LED Indicator
     └─ SOS Button
```

---

3. Software Architecture

3.1 Code Structure (Modular Design)

The code is split into **5 separate files** for better organization:

```
Project Files:
├── main_new.c        → Main program (initialization + event loop)
├── sensors.c         → All sensors (IR, gas, flame, SOS) + ADC + timers
├── uart.c            → UART communication + password handling
├── servo.c           → Servo motor PWM control
├── lcd.c             → I2C LCD display functions
└── Config.txt        → Hardware documentation
```

Why separate files? - Easier to test individual parts - Can reuse code in other projects - Debugging is simpler (if LCD fails, check lcd.c only)

3.2 Interrupt-Based Design

The system uses **hardware interrupts** instead of continuous polling. This means: - **Faster response**: Sensor detected → Interrupt triggers → ISR runs immediately - **Lower power**: CPU sleeps when idle (Low Power Mode 0) - **No missed events**: Hardware flags are set even if CPU is busy

Interrupts Used:

Interrupt Vector	Source	Trigger Condition
`PORT1_VECTOR`	IR Sensor (P1.3)	Rising edge (motion detected)
`PORT2_VECTOR`	Flame Sensor (P2.7)	Falling edge (fire detected)
`PORT2_VECTOR`	SOS Button (P2.1)	Falling edge (button pressed)
`USCI_A1_VECTOR`	UART RX (P4.5)	Character received
`TIMER0_A0_VECTOR`	Timer A0	Every 0.5 seconds (gas check)

3.3 Priority System

Not all threats are equal. The system handles them in **priority order**:

```
┌──────────────────────────────────────────┐
│ Priority 1: FIRE (Highest)               │  ← Opens door automatically
│ - Overrides everything                   │  ← Buzzer continuous
│ - Door opens to allow escape             │
└──────────────────────────────────────────┘
          ↓ (if no fire)
┌──────────────────────────────────────────┐
│ Priority 2: SOS Panic Button             │  ← Emergency alert
│ - User pressed for help                  │  ← Buzzer pattern
└──────────────────────────────────────────┘
          ↓ (if no fire/SOS)
┌──────────────────────────────────────────┐
│ Priority 3: Gas / Motion / UART          │  ← Normal operations
│ - Gas leak detection                     │
│ - Intruder motion                        │
│ - Password entry                         │
└──────────────────────────────────────────┘
```

Why this order? - **Fire**: Life-threatening, needs immediate escape route - **SOS**: User needs help urgently - **Gas/Motion**: Important but not immediately life-threatening

3.4 Main Program Flow

```
START
  ↓
┌──────────────────────────────┐
│ 1. Initialize Hardware       │
│    - Configure GPIO pins     │
│    - Setup I2C for LCD       │
│    - Initialize LCD          │
│    - Setup Servo PWM         │
│    - Initialize UART         │
│    - Configure ADC for gas   │
│    - Setup Timer A0          │
│    - Enable interrupts       │
└──────────────────────────────┘
```

```
  ↓
┌──────────────────────────────┐
│ 2. Display Welcome Message   │
│    - "SYSTEM READY"          │
│    - Clear after 2 seconds   │
└──────────────────────────────┘
  ↓
┌──────────────────────────────┐
│ 3. Enter Main Loop           │ <────────┐
│    (Infinite While Loop)     │          │
└──────────────────────────────┘          │
  ↓                                       │
┌──────────────────────────────┐          │
│ 4. Check Priority Flags      │          │
│    (Set by interrupts)       │          │
└──────────────────────────────┘          │
  ↓                                       │
  ├──→ Fire Flag? ──YES──→ Open Door ─────┤
  │                        + Buzzer       │
  │                                       │
  ├──→ SOS Flag? ───YES──→ SOS Alarm ─────┤
  │                                       │
  ├──→ Gas Flag? ───YES──→ Check Gas ─────┤
  │                        Level          │
  │                                       │
  ├──→ UART Flag? ──YES──→ Handle ────────┤
  │                        Password       │
  │                                       │
  └──→ IR Flag? ───YES──→ Motion ─────────┤
                          Alert           │
  ↓                                       │
┌──────────────────────────────┐          │
│ 5. Enter Low Power Mode      │          │
│    (Sleep until interrupt)   │ ─────────┘
└──────────────────────────────┘

Interrupt occurs → Wake up → Check flags → Repeat
```

---

4. Feature-by-Feature Explanation

4.1 Feature 1: IR Motion Detection (Intruder Alert)

Purpose

Detects when someone moves near the protected area. Used to catch intruders or unwanted visitors.

How It Works

IR sensor detects motion and sends signal to MCU

Hardware interrupt triggers immediately (no polling needed)

ISR sets a flag

Main loop checks flag and activates buzzer for 2 seconds

Flag cleared after alarm

Algorithm

```
ISR (Interrupt):
  Set motion_flag = 1

Main Loop:
  IF motion_flag:
    Buzzer ON → Wait 2 sec → Buzzer OFF
    Clear flag
```

4.2 Feature 2: Flame/Fire Detection

Purpose

Detects fire or flames immediately. This is the **highest priority** threat because fire can spread fast and block escape routes.

How It Works

Flame sensor detects IR radiation from fire

Hardware interrupt triggers on detection

ISR sets flame flag (highest priority)

Main loop immediately opens door for escape

Continuous buzzer alarm until flame disappears

Overrides all other operations (including password lock)

Algorithm

```
ISR (Interrupt):
  Set flame_flag = 1

Main Loop (Priority 1):
  IF flame_flag:
    Open door immediately
    Buzzer ON continuously
    LCD: "FIRE ALERT!"
```

Safety Logic

Door opens automatically (escape more important than security)

Continuous alarm to alert everyone nearby

No password needed during fire emergency

4.4 Feature 4: UART Password Door Lock System

Purpose

Remote door control via serial communication. User sends 4-digit password (“1234”) from PC → Door unlocks if correct.

How It Works

- User types password via UART terminal
- UART interrupt validates each digit (‘0’-‘9’ only)
- After 4 digits, compares with correct password
- **Correct**: Servo opens door (180°) → Wait 5 sec → Closes (0°), LCD “CORRECT!”
- **Wrong**: Buzzer beeps twice, LCD “WRONG! Got: xxxx”

Servo Control

- PWM signal controls servo angle (50Hz)
- 1ms pulse = 0° (LOCKED)
- 2ms pulse = 180° (UNLOCKED)
- Needs external 5V power supply

Algorithm

```
UART ISR:
  IF digit ('0'-'9'):
    Store in buffer
    Set flag

Main Loop:
  IF 4 digits received:
    Compare with "1234"
    IF match: Servo open → Wait 5s → Close, LCD "CORRECT!"
    ELSE: Buzzer beep, LCD "WRONG!"
```

4.5 Feature 5: SOS Panic Button

Purpose

Emergency button for manual alarm trigger. Press when you need help or feel threatened.

How It Works

- Push button with pull-up resistor
- Button press → Interrupt triggers → ISR sets SOS flag
- Main loop activates alarm pattern
- Buzzer beeps 3 times, LCD "SOS ACTIVE!"

Algorithm

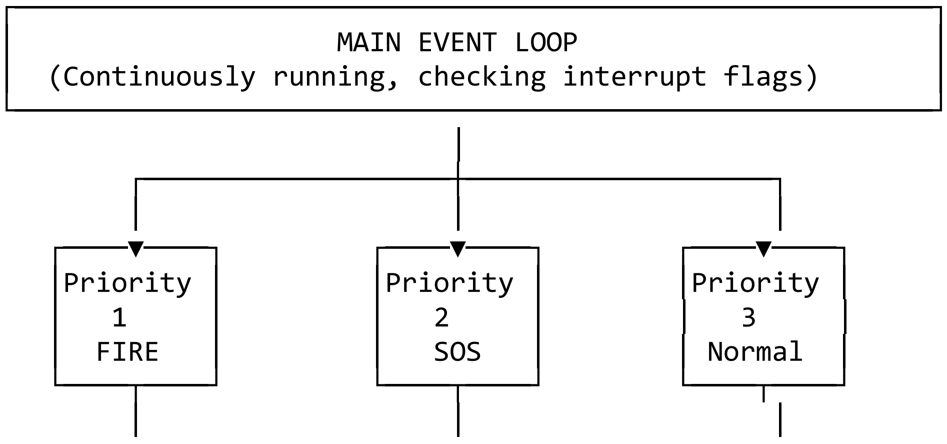
```
ISR:
  Set sos_flag = 1

Main Loop (Priority 2):
  IF sos_flag:
    LCD: "SOS ACTIVE!"
    Buzzer beep 3 times
```

---

5. System Integration

5.1 How Everything Works Together



```
    ├─→ Servo Open       ├─→ Buzzer Pattern ├─→ Gas Check
    ├─→ Buzzer ON        └─→ LCD Message    ├─→ UART Input
    └─→ Override All                         ├─→ Motion Alert
                                            └─→ LCD Update
```

6. Testing & Results

6.1 Individual Component Tests

Component	Test Method	Result
IR Sensor	Wave hand near sensor	Buzzer triggers immediately
Flame Sensor	Lighter flame ~50cm away	Door opens, buzzer continuous
Gas Sensor	Hold lighter (no spark)	LCD shows value >1800, alarm
UART	Send “1234” via terminal	Green LED blinks, door opens
Servo	Test file sweep 0→90→180	Code works, hardware power issue
LCD	Display “0123456789”	Shows characters, some garbage
SOS Button	Press button	Buzzer pattern triggers

6.2 Integration Tests

Test 1: Normal Password Entry - Action: Type “1234” on serial terminal - Expected: Green LED blinks, servo opens, LCD “CORRECT!” - Result: LED and LCD work, servo needs external power

Test 2: Fire Override

Action: Trigger flame sensor during password entry

Expected: Password ignored, door opens immediately

Result: Priority system works correctly

Test 3: Multiple Interrupts

Action: Press SOS while gas alarm active

Expected: SOS takes priority (Priority 2 > 3)

Result: Correct priority handling

Test 4: Wrong Password

Action: Type “5678” on serial terminal

Expected: Buzzer beeps twice, LCD “WRONG! Got: 5678”

Result: Validation works, LCD shows some garbage chars

6.3 Known Issues & Solutions

Issue 1: Servo Not Moving

Problem: PWM signal correct, but servo shaft doesn’t rotate

Cause: Insufficient power (breadboard voltage drop)

Solution: Use external 5V power supply (2A recommended)

Issue 2: LCD Garbage Characters

Problem: Password digits show as vertical lines

Cause: I2C timing or character set incompatibility

Solutions Tried:

Added 100μs delays between enable pulses

Manual character comparison (not strcmp)

Need to try: Different I2C address (0x3F instead of 0x27)

Issue 3: Occasional UART Misses

Problem: Sometimes doesn’t receive character

Cause: Oversampling helps but not perfect

Solution: Changed to SMCLK with UCOS16, validates input

---

7. Future Enhancements

7.1 Possible Improvements

I. **WiFi/Bluetooth Module**: Control door from phone app

II. **Camera Integration**: Take photo when intruder detected

III. **SD Card Logging**: Save all events with timestamps

IV. **Battery Backup**: Keep running during power failure

V. **Multiple Passwords**: Support 5-10 different user codes

VI. **Fingerprint Sensor**: Biometric authentication

VII. **SMS Alerts**: Send message when alarm triggered

VIII. **Web Dashboard**: View real-time gas levels remotely

7.2 Code Optimization

I. **Low Power Mode**: Use LPM3 between events (90% power savings)

II. **Watchdog Timer**: Reset system if code hangs

III. **EEPROM Storage**: Save password in non-volatile memory

IV. **Debouncing**: Add software debounce for buttons

V. **Circular Buffer**: Better UART data handling

---

8. Conclusion

8.1 What Was Achieved

This project successfully demonstrates a **complete embedded security system** using the **MSP430F5529** microcontroller. Key achievements:

Multi-sensor integration: 5 different sensors working together
Real-time response: Interrupt-based design ensures <1ms reaction time
Priority handling: Critical threats (fire) handled first
Modular code: Easy to understand and modify
User-friendly: LCD and LED feedback

8.2 Learning Outcomes

Hardware Interfacing: Connected digital, analog, I2C, UART, PWM peripherals

Interrupt Programming: Proper ISR design and flag management

Timing Analysis: Understanding polling intervals, PWM frequencies, ADC conversion

Debugging Skills: Fixed UART oversampling, servo pins, LCD timing issues

System Design: Priority systems, state machines, modular architecture

8.3 Practical Applications

This system can be used in: - **Home Security**: Detect intruders, fire, gas leaks - **Labs/Studios**: Panic button for emergencies - **Storage Units**: Remote access control with monitoring - **Industrial Sites**: Gas leak detection with automated response

---

9. Resources

9.1 Tools Used

IDE: Code Composer Studio v12

Programmer: MSP430 LaunchPad USB

Terminal: PuTTY (9600 baud, 8N1)

End of Report

This system was developed as part of the Embedded Systems (Hardware Cloning) - Security Sector mini-project series.

Milan Jani | MSP430F5529 LaunchPad | December 2025