**AGILE-SCRUM WITH PHASED HARDWARE DEVELOPMENT**

To Ensure Iterative Progress, Testing, And Refinements

**Dk Board – Professional Project Development Document**

**Project Title:**  
**DK Board** – An Open-Source Embedded System for Educational & Industrial Automation

**Version:** Final Optimized Flow  
**Prepared By:** Mathews P Jacob  
**Date:** 30-05-2025

**Table of Contents**

1. Executive Summary
2. Project Overview & Vision
3. Project Development Lifecycle  
   3.1 Phase 1: Planning & Requirement Analysis  
   3.2 Phase 2: Hardware Architecture & PCB Design  
   3.3 Phase 3: Firmware Development & MCU Programming  
   3.4 Phase 4: Web-Based GUI & API Development  
   3.5 Phase 5: Prototype Testing & Iteration  
   3.6 Phase 6: Final Product Release & Documentation
4. System Architecture  
   4.1 Hardware Architecture  
   4.2 Firmware Architecture  
   4.3 Software & Cloud Integration
5. Technical Details & Illustrations  
   5.1 Parallel Execution (FreeRTOS-Based)  
   5.2 Interrupt-Driven & Non-Blocking Execution  
   5.3 Block-Based Programming & JSON Mapping
6. Technical Documentation Structure
7. Conclusion & Next Steps
8. Appendices

**1. Executive Summary**

DKBCode is an ambitious open-source development kit aimed at merging hardware and software into a singular ecosystem for practical embedded system education and industrial prototyping. By leveraging multiple MCUs (ESP32, STM32, ATmega, PIC), advanced sensor networks, and robust communication modules (GSM800L, LoRaWAN, Zigbee, BLE, RF, CAN/LIN), DKBCode offers a comprehensive platform optimized for real-time control, automation, and cloud integration. Our agile — Scrum-inspired — project lifecycle ensures iterative progress, unwavering quality, and continual improvement.

**2. Project Overview & Vision**

**Objective:**  
Provide an affordable, accessible, and modular development platform that empowers students, educators, and IoT innovators to design, test, and deploy real-world automation solutions. DKBCode bridges theoretical learning with practical implementation by emphasizing:

* **Multi-MCU Compatibility:** ESP32, STM32, ATmega, and PIC support.
* **Seamless Wireless Connectivity:** GSM, LoRaWAN, Zigbee, BLE, and RF for robust remote control.
* **Real-Time Automation:** FreeRTOS for multitasking that guarantees simultaneous sensor readings and actuator controls.
* **User-Friendly Interfaces:** A web-based GUI offering a Node-RED–like drag-and-drop programming environment and comprehensive API services.

**Vision:**  
Empower the next generation of engineers with hands-on experimentation by providing a platform that is advanced yet accessible, supported by detailed documentation, community support, and open-source collaboration.

**3. Project Development Lifecycle**

**Phase 1: Project Planning & Requirement Analysis (Weeks 1-2)**

* **Define Core Objectives:** Identify the problems solved by DKBCode.
* **Component Inventory:** List hardware (MCUs, sensors, actuators) and software (C++, Python, Web APIs, GUI frameworks).
* **Scope & Budget Definition:** Establish project constraints, timelines, and resources.
* **Roadmap Creation:** Draft system architecture, data flow models, and firmware specifications.
* **Version Control Setup:** Initialize GitHub repositories.

**Phase 2: Hardware Architecture & PCB Design (Weeks 3-6)**

* **Schematic Finalization:** Create comprehensive schematics for all supported MCUs.
* **Pin Mapping & Peripheral Layout:** Map sensor and actuator connections with dedicated I/O assignments.
* **PCB Design:** Develop a multi-layer PCB focused on power management, minimal interference, and signal integrity.
* **Prototype Fabrication & Testing:** Build and validate the initial PCB.

**Phase 3: Firmware Development & MCU Programming (Weeks 7-12)**

* **Driver Development:** Write modular drivers for sensors, motors, displays, and communication modules.
* **Hardware Abstraction Layer (HAL):** Ensure cross-platform compatibility.
* **Multitasking Integration:** Leverage FreeRTOS on ESP32/STM32 for parallel execution.
* **Data Logging & Debugging:** Integrate data logging and debugging interfaces.
* **Sub-Module Implementation:** Cover sensor interface, stepper motor/servo control, display drivers, wireless stack, GSM connectivity, CAN/LIN, storage, and security.

**Phase 4: Web-Based GUI & API Development (Weeks 13-16)**

* **Visual Programming Interface:** Develop a drag-and-drop logic framework.
* **IoT Dashboard:** Build a responsive web-based GUI for real-time monitoring.
* **API Development:** Create RESTful APIs and secure MQTT/WebSockets channels.
* **OTA Updates & User Authentication:** Incorporate over-the-air updates and role-based access.
* **Future Expansion:** Consider embedded edge AI features for advanced applications.

**Phase 5: Prototype Testing & Iteration (Weeks 17-20)**

* **Integration Testing:** Validate hardware and software interoperability.
* **Performance Metrics:** Measure power consumption, response time, and latency.
* **Refinement & Optimization:** Enhance firmware performance based on test results.
* **Beta Field Testing:** Deploy prototypes in real-world scenarios and gather feedback.

**Phase 6: Final Product Release & Documentation (Weeks 21-24)**

* **Comprehensive Documentation:** Produce detailed guides across hardware setup, firmware development, web GUI utilization, and API integration.
* **Production PCB & Enclosure Design:** Manufacture final PCBs and refine case designs.
* **Community & Open-Source Collaboration:** Build a supportive GitHub repository and user forums.
* **Industry Outreach:** Engage academic and industrial partners and publish case studies.

**4. System Architecture**

**4.1. Hardware Architecture**

* **Core Components:**
  + **MCUs:** ESP32, STM32, ATmega, PIC.
  + **Sensors:** Temperature (LM35, DHT11, DS18B20), Light (LDR, TSL2561), PIR, Current (ACS712, INA219).
  + **Communication Modules:** GSM800L, LoRaWAN, Zigbee, BLE, RF, CAN/LIN.
  + **Displays:** 16x2 LCD, OLED, TFT screens.
* **Power Management:** Voltage regulators and EMI shielding for stable operation.

**4.2. Firmware Architecture**

* **Layered Design:**
* **Hardware Abstraction Layer (HAL):** Standardizes drivers across various MCUs.
* **Peripheral Driver Layer:** Manages sensor readings, motor control, and communication.
* **Logic Processing Layer:** Uses FreeRTOS for scheduling, conditional logic, and event management.
* **Application Layer:** Interfaces with the web GUI and external APIs.

**4.3. Software & Cloud Integration**

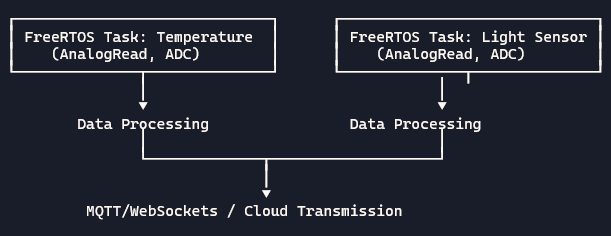
* **Web-Based GUI:** Built with modern frameworks (e.g., React.js, Node.js) offering an intuitive drag-and-drop interface.
* **IoT Communication:** Utilizes RESTful APIs, MQTT, and WebSockets for real-time data streaming.
* **Cloud Connectivity:** Implements online logging and analytics for performance tracking.

**5. Technical Details & Illustrations**

**5.1. Parallel Execution – FreeRTOS-Based Architecture**

**Concept:**  
Running multiple tasks concurrently using FreeRTOS ensures that sensor readings and actuator actions are performed in parallel without CPU bottlenecks.

**Diagram:**



Code Snippet:

#include < Arduino\_FreeRTOS.h >

#define TEMP\_SENSOR\_PIN A0

#define LDR\_SENSOR\_PIN A1

TaskHandle\_t tempTaskHandle, ldrTaskHandle;

void readTemperatureTask(void \* pvParameters) {

  while (true) {

    int temp = analogRead(TEMP\_SENSOR\_PIN);

    Serial.println("Temperature: " + String(temp));

    vTaskDelay(1000 / portTICK\_PERIOD\_MS);

  }

}

void readLDRTask(void \* pvParameters) {

  while (true) {

    int light = analogRead(LDR\_SENSOR\_PIN);

    Serial.println("Light Level: " + String(light));

    vTaskDelay(500 / portTICK\_PERIOD\_MS);

  }

}

void setup() {

  Serial.begin(115200);

  xTaskCreate(readTemperatureTask, "Temp Task", 2048, NULL, 1, & tempTaskHandle);

  xTaskCreate(readLDRTask, "LDR Task", 2048, NULL, 1, & ldrTaskHandle);

}

void loop() {

  // Main loop remains free for additional tasks

}

**5.2. Interrupt-Driven & Non-Blocking Execution**

**Concept:**  
Interrupt service routines (ISRs) handle time-critical events, ensuring that the main program continues without blocking.

**Example: External Interrupt for a Button**

#define BUTTON\_PIN 12

void IRAM\_ATTR handleButtonPress() {

  Serial.println("Button Pressed!");

}

void setup() {

  pinMode(BUTTON\_PIN, INPUT\_PULLUP);

  attachInterrupt(digitalPinToInterrupt(BUTTON\_PIN), handleButtonPress, FALLING);

}

void loop() {

  // Main loop is free-running

}

Diagram:



**5.3. Block-Based Programming & JSON Mapping**

**Concept:**  
A visual programming interface allows users to create automation flows; these are translated into structured JSON commands for firmware execution.

**Example JSON for a Stepper Motor Block:**

{

  "block\_id": "motor1",

    "type": "ACTUATOR",

      "name": "Stepper Motor",

        "inputs": {

    "direction": "clockwise",

      "speed": 500

  },

  "connections": ["sensor\_block", "logic\_block"]

}

Flow Diagram:

[Sensor Block] → [IF Condition: Temp > 30°C]

↓

[JSON Mapping] [Timer: 5s Delay]

↓

[Actuator Block: Cooling Fan ON]

**6. Technical Documentation Structure**

Each technical guide provides a clear learning progression:

1. **Introduction & System Overview:** Explaining supported MCUs, sensors, and key features with diagrams.
2. **Hardware Setup & Wiring Guide:** Detailed PCB layouts, pin mapping diagrams, and wiring schemes including safety guidelines.
3. **Firmware Development Guide:** Step-by-step instructions for flashing firmware, setting up communication protocols, and FreeRTOS task management.
4. **Web-Based GUI & API Integration:** How to set up the drag-and-drop interface, API examples, and JSON mappings.
5. **Troubleshooting & FAQs:** Common issues, debugging techniques, and performance profiling recommendations.

**7. Conclusion & Next Steps**

DKBCode aims to revolutionize embedded system education and industrial prototyping by providing a robust, flexible, and user-friendly platform.  
**Next Steps:**

* Finalize prototype implementation and prepare for beta testing.
* Enhance documentation with interactive diagrams and video tutorials.
* Engage community feedback through GitHub and other open-source forums.
* Plan for industry outreach and collaborative research.

**8. Appendices**

* **Appendix A: Code Samples** – Modular code snippets for sensor reading, motor control, interrupt handling, etc.
* **Appendix B: Diagrams & Illustrations** – Additional block diagrams, PCB layout sketches, and flowcharts.
* **Appendix C: Reference Materials** – Datasheets, external resources, and community links.

Instructions to Create Downloadable Files

**To Create a Microsoft Word Document:**

1. **Copy the entire content above.**
2. **Open Microsoft Word or LibreOffice Writer.**
3. **Paste the content into a new document.**
4. **Format as needed (add headers, adjust styles, insert images/diagrams).**
5. **Save the document as a .docx file.**

**To Create a PowerPoint Presentation:**

1. **Open Microsoft PowerPoint.**
2. **Create a new presentation.**
3. **For each high-level section (e.g., Executive Summary, Project Lifecycle, System Architecture, Technical Details, Conclusion), create a separate slide.**
4. **Copy the relevant content from the document above and paste into the appropriate slides.**
5. **Enhance your slides with images or diagrams (use the flow diagrams provided above by recreating them as shapes or inserting screenshots).**
6. **Save the presentation as a .pptx file.**