



Department of Electronic & Telecommunication Engineering,
University of Moratuwa,
Sri Lanka.

Software Report Handheld RFID Reader

Group Members

Index No	Name
220235V	Ilukkumbura I.M.E.I.B.
220212A	Hapuarachchi H.A.D.N.D.
220162T	Fernando C.S.R.
220221B	Hathurusingha HA.R.
220420J	Nawarathne MA.A.K.
220700T	Wickramasinghe S.D.
220089B	Cooray M.S.T.
220163X	Fernando D.S.

Submitted in partial fulfillment of the requirements for the module
EN 2161 Electronic Design Realization

5/22/2025

Contents

1	PCB Design	2
2	Enclosure Design	2
3	Web App Development	2
4	Database Implementation	2
5	Schema	2
6	Firmware Development	8
6.1	Atmega32u4 Programming Code (.cpp)	8
6.2	PN5180 Programming Code (.h)	11
6.3	SPI Programming Code	13
6.3.1	.h	13
6.3.2	.cpp	13
6.4	UART Programming Code	15
6.4.1	.h	15
6.4.2	.cpp	15

1 PCB Design

Altium Designer - <https://www.altium.com/education/students>

2 Enclosure Design

SolidWorks

3 Web App Development

The web app was developed using Node.js.

Link: <https://uniofmoramy.sharepoint.com/:f/g/personal/wickramasinghesd22uomlk/Emx1S-QYZOLAi52qVPWJZnoBJQriCgSUMRu-N7Q6K0C1tQ?e=sf9qnY>

4 Database Implementation

The database was implemented using Supabase(PostgreSQL).

5 Schema

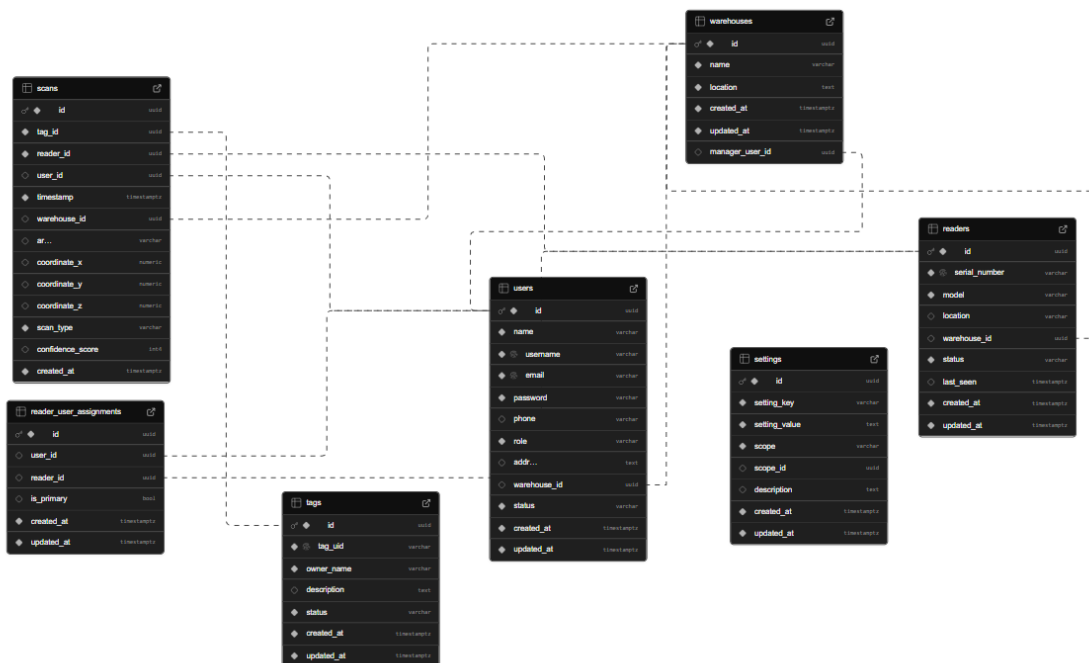


Figure 1: Schema

```
1  -- Drop tables if they exist for clean migration
2  DROP TABLE IF EXISTS settings CASCADE;
3  DROP TABLE IF EXISTS scans CASCADE;
```

```

4 DROP TABLE IF EXISTS tags CASCADE;
5 DROP TABLE IF EXISTS reader_user_assignments CASCADE;
6 DROP TABLE IF EXISTS readers CASCADE;
7 DROP TABLE IF EXISTS users CASCADE;
8 DROP TABLE IF EXISTS warehouses CASCADE;
9
10 -- Create warehouses table (removed the manager_user_id initially to
    ↳ avoid circular reference)
11 CREATE TABLE warehouses (
12     id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
13     name VARCHAR(255) NOT NULL,
14     location TEXT NOT NULL,
15     created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
16     updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
17 );
18
19 -- Create users table
20 CREATE TABLE users (
21     id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
22     name VARCHAR(255) NOT NULL,
23     username VARCHAR(100) NOT NULL UNIQUE,
24     email VARCHAR(255) NOT NULL UNIQUE,
25     password VARCHAR(255) NOT NULL,
26     phone VARCHAR(20),
27     role VARCHAR(20) NOT NULL CHECK (role IN ('admin', 'manager', '
    ↳ employee')),
28     address TEXT,
29     warehouse_id UUID REFERENCES warehouses(id) ON DELETE SET NULL,
30     status VARCHAR(20) NOT NULL DEFAULT 'active' CHECK (status IN ('
    ↳ active', 'suspended', 'inactive')),
31     created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
32     updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
33 );
34
35 -- Now add the manager_user_id to warehouses after users table exists
36 ALTER TABLE warehouses ADD COLUMN manager_user_id UUID;
37 ALTER TABLE warehouses ADD CONSTRAINT fk_warehouses_manager
38     FOREIGN KEY (manager_user_id) REFERENCES users(id) ON DELETE SET
    ↳ NULL;
39
40 -- Create readers table
41 CREATE TABLE readers (
42     id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
43     serial_number VARCHAR(100) NOT NULL UNIQUE,
44     model VARCHAR(100) NOT NULL,
45     location VARCHAR(255) NOT NULL,
46     warehouse_id UUID REFERENCES warehouses(id) ON DELETE SET NULL,
47     status VARCHAR(20) NOT NULL DEFAULT 'active' CHECK (status IN ('
    ↳ active', 'maintenance', 'retired')),
48     last_seen TIMESTAMPTZ,
49     created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
50     updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
51 );
52
53 -- Create reader_user_assignments table for many-to-many relationship
54 -- This is better than having assigned_user_id in readers table when
    ↳ multiple users can be assigned
55 CREATE TABLE reader_user_assignments (
56     id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

```

```

57         user_id UUID REFERENCES users(id) ON DELETE CASCADE,
58         reader_id UUID REFERENCES readers(id) ON DELETE CASCADE,
59         is_primary BOOLEAN DEFAULT false,
60         created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
61         updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
62         UNIQUE(reader_id, user_id)
63     );
64
65     -- Create tags table
66     CREATE TABLE tags (
67         id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
68         tag_uid VARCHAR(100) NOT NULL UNIQUE,
69         owner_name VARCHAR(255) NOT NULL,
70         description TEXT,
71         status VARCHAR(20) NOT NULL DEFAULT 'active' CHECK (status IN ('
            ↳ active', 'damaged', 'retired')),
72         created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
73         updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
74     );
75
76     -- Create scans table
77     CREATE TABLE scans (
78         id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
79         tag_id UUID NOT NULL REFERENCES tags(id) ON DELETE CASCADE,
80         reader_id UUID NOT NULL REFERENCES readers(id) ON DELETE CASCADE,
81         user_id UUID REFERENCES users(id) ON DELETE SET NULL,
82         timestamp TIMESTAMPTZ NOT NULL DEFAULT NOW(),
83         warehouse_id UUID REFERENCES warehouses(id) ON DELETE SET NULL,
84         area VARCHAR(255),
85         coordinate_x DECIMAL(10, 2),
86         coordinate_y DECIMAL(10, 2),
87         coordinate_z DECIMAL(10, 2),
88         scan_type VARCHAR(20) NOT NULL CHECK (scan_type IN ('entry', 'exit',
            ↳ 'inventory')),
89         confidence_score INTEGER CHECK (confidence_score BETWEEN 0 AND 100),
90         created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
91     );
92
93     -- Create settings table
94     CREATE TABLE settings (
95         id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
96         setting_key VARCHAR(100) NOT NULL,
97         setting_value TEXT NOT NULL,
98         scope VARCHAR(50) NOT NULL DEFAULT 'global' CHECK (scope IN ('global
            ↳ ', 'reader', 'warehouse')),
99         scope_id UUID, -- This will store reader_id or warehouse_id based on
            ↳ scope
100         description TEXT,
101         created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
102         updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
103     );
104
105     -- Add a partial unique constraint that treats NULL scope_id as a
        ↳ specific value
106     CREATE UNIQUE INDEX idx_settings_unique ON settings (setting_key, scope,
        ↳ COALESCE(scope_id, '00000000-0000-0000-0000-000000000000'::UUID))
        ↳ ;
107
108     -- Create functions and triggers for updated_at timestamps

```

```

109 CREATE OR REPLACE FUNCTION update_timestamp()
110 RETURNS TRIGGER AS $$
111 BEGIN
112     NEW.updated_at = NOW();
113     RETURN NEW;
114 END;
115 $$ language 'plpgsql';
116
117 -- Create triggers for each table
118 CREATE TRIGGER update_warehouses_timestamp BEFORE UPDATE ON warehouses
119     FOR EACH ROW EXECUTE PROCEDURE update_timestamp();
120 CREATE TRIGGER update_users_timestamp BEFORE UPDATE ON users
121     FOR EACH ROW EXECUTE PROCEDURE update_timestamp();
122 CREATE TRIGGER update_readers_timestamp BEFORE UPDATE ON readers
123     FOR EACH ROW EXECUTE PROCEDURE update_timestamp();
124 CREATE TRIGGER update_tags_timestamp BEFORE UPDATE ON tags
125     FOR EACH ROW EXECUTE PROCEDURE update_timestamp();
126 CREATE TRIGGER update_settings_timestamp BEFORE UPDATE ON settings
127     FOR EACH ROW EXECUTE PROCEDURE update_timestamp();
128 CREATE TRIGGER update_reader_user_assignments_timestamp BEFORE UPDATE ON
    ↪ reader_user_assignments
129     FOR EACH ROW EXECUTE PROCEDURE update_timestamp();
130
131 -- Create indexes for performance
132 CREATE INDEX idx_scans_timestamp ON scans(timestamp);
133 CREATE INDEX idx_scans_tag_id ON scans(tag_id);
134 CREATE INDEX idx_scans_reader_id ON scans(reader_id);
135 CREATE INDEX idx_readers_warehouse ON readers(warehouse_id);
136 CREATE INDEX idx_users_warehouse ON users(warehouse_id);
137 CREATE INDEX idx_tags_status ON tags(status);
138 -- We already have the unique index that acts as our index for these
    ↪ columns
139 CREATE INDEX idx_reader_assignments_user ON reader_user_assignments(
    ↪ user_id);
140 CREATE INDEX idx_reader_assignments_reader ON reader_user_assignments(
    ↪ reader_id);
141
142 -- Create RLS (Row Level Security) policies for Supabase
143 -- Enable RLS on all tables
144 ALTER TABLE warehouses ENABLE ROW LEVEL SECURITY;
145 ALTER TABLE users ENABLE ROW LEVEL SECURITY;
146 ALTER TABLE readers ENABLE ROW LEVEL SECURITY;
147 ALTER TABLE tags ENABLE ROW LEVEL SECURITY;
148 ALTER TABLE scans ENABLE ROW LEVEL SECURITY;
149 ALTER TABLE settings ENABLE ROW LEVEL SECURITY;
150 ALTER TABLE reader_user_assignments ENABLE ROW LEVEL SECURITY;
151
152 -- Comprehensive RLS policies
153
154 -- Warehouse policies
155 CREATE POLICY "Admins can do anything with warehouses" ON warehouses
156     FOR ALL TO authenticated
157     USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));
158
159 CREATE POLICY "Managers can update their warehouses" ON warehouses
160     FOR UPDATE TO authenticated
161     USING (auth.uid() = manager_user_id);
162
163 CREATE POLICY "All users can view warehouses" ON warehouses

```

```

164         FOR SELECT TO authenticated
165         USING (true);
166
167     -- User policies
168     CREATE POLICY "Admins can do anything with users" ON users
169         FOR ALL TO authenticated
170         USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));
171
172     CREATE POLICY "Managers can view and update users in their warehouse" ON
173     ↪ users
174     FOR SELECT TO authenticated
175     USING (
176         (auth.uid() IN (SELECT id FROM users WHERE role = 'manager'))
177         AND
178         (warehouse_id IN (SELECT warehouse_id FROM users WHERE id = auth
179             ↪ .uid())
180         OR warehouse_id IN (SELECT id FROM warehouses WHERE
181             ↪ manager_user_id = auth.uid()))
182     );
183
184     CREATE POLICY "Users can view their own data" ON users
185     FOR SELECT TO authenticated
186     USING (id = auth.uid());
187
188     -- Reader policies
189     CREATE POLICY "Admins can do anything with readers" ON readers
190     FOR ALL TO authenticated
191     USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));
192
193     CREATE POLICY "Managers can manage readers in their warehouse" ON
194     ↪ readers
195     FOR ALL TO authenticated
196     USING (
197         (auth.uid() IN (SELECT id FROM users WHERE role = 'manager'))
198         AND
199         (warehouse_id IN (SELECT id FROM warehouses WHERE
200             ↪ manager_user_id = auth.uid()))
201     );
202
203     CREATE POLICY "Users can view readers they are assigned to" ON readers
204     FOR SELECT TO authenticated
205     USING (
206         id IN (
207             SELECT reader_id FROM reader_user_assignments
208             WHERE user_id = auth.uid()
209         )
210     );
211
212     -- Reader Assignment policies
213     CREATE POLICY "Admins can do anything with reader assignments" ON
214     ↪ reader_user_assignments
215     FOR ALL TO authenticated
216     USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));
217
218     CREATE POLICY "Managers can manage reader assignments in their warehouse
219     ↪ " ON reader_user_assignments
220     FOR ALL TO authenticated
221     USING (
222         (auth.uid() IN (SELECT id FROM users WHERE role = 'manager'))

```

```

216         AND
217         (reader_id IN (
218             SELECT id FROM readers
219             WHERE warehouse_id IN (
220                 SELECT id FROM warehouses
221                 WHERE manager_user_id = auth.uid()
222             )
223         ))
224     );
225
226     CREATE POLICY "Users can view their reader assignments" ON
227         ↪ reader_user_assignments
228         FOR SELECT TO authenticated
229         USING (user_id = auth.uid());
230
231     -- Tags policies
232     CREATE POLICY "Admins can do anything with tags" ON tags
233         FOR ALL TO authenticated
234         USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));
235
236     CREATE POLICY "All authenticated users can view tags" ON tags
237         FOR SELECT TO authenticated
238         USING (true);
239
240     -- Scans policies
241     CREATE POLICY "Admins can do anything with scans" ON scans
242         FOR ALL TO authenticated
243         USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));
244
245     CREATE POLICY "Managers can view scans in their warehouse" ON scans
246         FOR SELECT TO authenticated
247         USING (
248             (auth.uid() IN (SELECT id FROM users WHERE role = 'manager'))
249             AND
250             (warehouse_id IN (SELECT id FROM warehouses WHERE
251                 ↪ manager_user_id = auth.uid()))
252         );
253
254     CREATE POLICY "Users can view their scans" ON scans
255         FOR SELECT TO authenticated
256         USING (user_id = auth.uid());
257
258     -- Settings policies
259     CREATE POLICY "Admins can do anything with settings" ON settings
260         FOR ALL TO authenticated
261         USING (auth.uid() IN (SELECT id FROM users WHERE role = 'admin'));
262
263     CREATE POLICY "All authenticated users can view global settings" ON
264         ↪ settings
265         FOR SELECT TO authenticated
266         USING (scope = 'global');
267
268     CREATE POLICY "Managers can view and edit settings for their warehouse"
269         ↪ ON settings
270         FOR ALL TO authenticated
271         USING (
272             (auth.uid() IN (SELECT id FROM users WHERE role = 'manager'))
273             AND
274             (

```



```

271         (scope = 'warehouse' AND scope_id IN (
272             SELECT id FROM warehouses WHERE manager_user_id = auth.
273                 ↳ uid())
274     OR
275     (scope = 'reader' AND scope_id IN (
276         SELECT id FROM readers
277         WHERE warehouse_id IN (
278             SELECT id FROM warehouses WHERE manager_user_id =
279                 ↳ auth.uid()
280         )
281     ))
282 );

```

6 Firmware Development

6.1 Atmgea32u4 Programming Code (.cpp)

```

1  #include <avr/io.h>
2  #include <util/delay.h>
3  #include <stdio.h>
4  #include <string.h>
5  #include <stdint.h>
6  #include "mySPI.h"
7  #include "PN5180.h"
8  #include "myUART.h"
9
10 // Define F_CPU if not already defined
11 #ifndef F_CPU
12 #define F_CPU 16000000UL
13 #endif
14
15 // Buffer size for UID string
16 #define UID_STRING_SIZE 32
17
18 // PN5180 instance (NSS = PD1/Pin 2, BUSY = PB6/Pin 10, RST = PB4/Pin 8)
19 PN5180 pn5180(/* NSS / 2, / BUSY / 10, / RST */ 8);
20
21 // Function prototypes
22 int handshake(void);
23 int connectWiFi(void);
24 void uart_transmit(const char *data); // For debug output
25 void uart_transmit_num(int16_t num); // For number output
26
27 int main(void) {
28     // Initialize UART (USART1)
29     USART_Init();
30
31     // Initialize SPI
32     SPI_init();
33     SPI_setConfig();
34
35     // Initialize PN5180
36     pn5180.begin();
37

```

```

38 // Configure Pin 12 (PD6) as input with pull-up, Pin 6 (PD7) as
    ↪ output
39 DDRD &= ~(1 << DDD6); // PD6 as input (button)
40 DDRD |= (1 << DDD7); // PD7 as output (LED)
41 PORTD |= (1 << PORTD6); // Enable pull-up on PD6
42 PORTD &= ~(1 << PORTD7); // LED off initially
43
44 // Handshake with ESP
45 for (uint8_t retries = 3; retries > 0; retries--) {
46     USART_TransmitCommand("PING");
47     int response = handshake();
48     if (response == 3) break; // OK received
49     _delay_ms(500);
50 }
51
52 // Connect to WiFi
53 int wifiStatus = connectWiFi();
54 char wifiMsg[32];
55 snprintf(wifiMsg, sizeof(wifiMsg), "WiFi Status: %d\r\n", wifiStatus
    ↪ );
56 uart_transmit(wifiMsg);
57
58 // Main loop
59 while (1) {
60     if (!(PIND & (1 << PIND6))) { // Button pressed (PD6 low)
61         _delay_ms(50); // Debounce
62         if (!(PIND & (1 << PIND6))) { // Confirm button
            ↪ press
63             uint8_t uid[8];
64             if (pn5180.getInventoryFake(uid)) {
65                 char uidStr[17];
66                 for (int i = 7, j = 0; i >= 0; i--,
                    ↪ j += 2) {
67                     sprintf(uidStr + j, "%02X",
                        ↪ uid[i]);
68                 }
69                 char cmd[32];
70                 snprintf(cmd, sizeof(cmd), "SEND:UID
                    ↪ :%s", uidStr);
71                 USART_TransmitCommand(cmd); // Send
                    ↪ UID to NodeMCU
72                 int response = USART_ReceiveCommand
                    ↪ ();
73                 char responseMsg[16];
74                 snprintf(responseMsg, sizeof(
                    ↪ responseMsg), "Response: %d\r\
                    ↪ n", response);
75                 uart_transmit(responseMsg);
76                 if (response == 3) {
77                     PORTD |= (1 << PORTD7); //
                        ↪ LED on
78                     _delay_ms(500);
79                     PORTD &= ~(1 << PORTD7); //
                        ↪ LED off
80                     } else if (response == 2) {
81                         // Handle ERROR (add if
                            ↪ needed)
82                     }
83                 }
            }
        }
    }
}

```

```

84         while (!(PIND & (1 << PIND6))) {
85             _delay_ms(10); // Wait for button
86                 ↪ release
87         }
88     }
89 }
90
91     return 0; // Never reached
92 }
93
94 // UART transmit function for strings
95 void uart_transmit(const char *data) {
96     while (*data) {
97         while (!(UCSR1A & (1 << UDRE1))); // Wait for empty transmit
98             ↪ buffer
99         UDR1 = *data++; // Send character
100     }
101 }
102
103 // UART transmit function for numbers
104 void uart_transmit_num(int16_t num) {
105     char buffer[16];
106     snprintf(buffer, sizeof(buffer), "%d\r\n", num);
107     uart_transmit(buffer);
108 }
109
110 // Handshake with ESP
111 int handshake(void) {
112     _delay_ms(500);
113     USART_TransmitCommand("PING");
114     uart_transmit("Waiting for ESP response...\r\n");
115
116     // Wait for response with timeout (~3 seconds)
117     char response[MAX_BUFFER_SIZE] = {0};
118     for (uint32_t i = 0; i < 3000000; i++) { // Approx 3 seconds at 16
119         ↪ MHz
120         USART_ReadString(response, MAX_BUFFER_SIZE, 1000); // Short
121             ↪ timeout per read
122         if (strlen(response) > 0) {
123             char debugMsg[128];
124             snprintf(debugMsg, sizeof(debugMsg), "Received from
125                 ↪ ESP: %s\r\n", response);
126             uart_transmit(debugMsg);
127             if (strcmp(response, "OK") == 0) {
128                 uart_transmit("ESP communication successful
129                     ↪ .\r\n");
130                 return 3;
131             } else {
132                 uart_transmit("ESP responded incorrectly.\r\n
133                     ↪ n");
134                 return -1;
135             }
136         }
137     }
138     _delay_us(1); // Small delay
139 }
140
141 uart_transmit("Timeout waiting for ESP.\r\n");
142 return -1;

```

```

136     }
137
138     // Connect to WiFi
139     int connectWiFi(void) {
140         const uint32_t maxTimeout = 30000000; // ~30 seconds at 16 MHz
141         const uint32_t retryInterval = 12000000; // ~12 seconds per attempt
142         uint32_t attemptCount = maxTimeout / retryInterval;
143
144         for (uint32_t attempt = 0; attempt < attemptCount; attempt++) {
145             USART_TransmitCommand("WIFI:CONNECT");
146             char response[MAX_BUFFER_SIZE] = {0};
147
148             // Wait for response
149             for (uint32_t i = 0; i < retryInterval; i++) {
150                 USART_ReadString(response, MAX_BUFFER_SIZE, 1000);
151                 ↪ // Short timeout
152                 if (strlen(response) > 0) {
153                     if (strcmp(response, "OK") == 0) {
154                         return 3;
155                     } else if (strcmp(response, "ERROR")
156                             ↪ == 0) {
157                         break;
158                     }
159                 }
160                 _delay_us(1); // Small delay
161             }
162             return -1;
163         }

```

6.2 PN5180 Programming Code (.h)

```

1  /*
2   * PN5180.h
3   *
4   * Created: 5/22/2025 1:47:50 PM
5   * Author: AGRA
6   */
7
8  #ifndef PN5180_H
9  #define PN5180_H
10
11  #include "mySPI.h"
12
13  // PN5180 Commands
14  #define PN5180_WRITE_REGISTER          (0x00)
15  #define PN5180_WRITE_REGISTER_OR_MASK (0x01)
16  #define PN5180_WRITE_REGISTER_AND_MASK (0x02)
17  #define PN5180_READ_REGISTER           (0x04)
18  #define PN5180_WRITE_EEPROM            (0x06)
19  #define PN5180_READ_EEPROM             (0x07)
20  #define PN5180_SEND_DATA               (0x09)
21  #define PN5180_READ_DATA               (0x0A)
22  #define PN5180_LOAD_RF_CONFIG          (0x11)
23  #define PN5180_RF_ON                   (0x16)
24  #define PN5180_RF_OFF                  (0x17)

```

```

25 // 11.9.1, Table 73 PN5180 Register Address Overview
26 #define SYSTEM_CONFIG (0x00)
27 #define IRQ_ENABLE (0x01)
28 #define IRQ_STATUS (0x02)
29 #define IRQ_CLEAR (0x03)
30 #define RX_STATUS (0x13)
31 #define TX_WAIT_CONFIG (0x17)
32 #define TX_CONFIG (0x18)
33 // 11.9.1, Table 76 IRQ_STATUS Register
34 #define RX_IRQ_STAT (1<<0) // End of RF reception IRQ
35 #define TX_IRQ_STAT (1<<1) // End of RF transmission
    ↳ IRQ
36 #define IDLE_IRQ_STAT (1<<2) // Idle IRQ
37 #define RFOFF_DET_IRQ_STAT (1<<6) // RF Field OFF detection
    ↳ IRQ
38 #define RFON_DET_IRQ_STAT (1<<7) // RF Field ON detection IRQ
39 #define TX_RFOFF_IRQ_STAT (1<<8) // RF Field OFF in PCD IRQ
40 #define TX_RFON_IRQ_STAT (1<<9) // RF Field ON in PCD IRQ
41 // 11.9.1 Table 92 RX_STATUS Register
42 #define RX_COLL_POS (1<<19) // These bits show the bit position of the
    ↳ first detected collision in a received frame (7 bits)
43 #define RX_COLLISION_DETECTED (1<<18) // This flag is set to 1, when a
    ↳ collision has occurred
44 // 11.9.1 Table 97 TX_CONFIG Register
45 #define TX_DATA_ENABLE (1<<10) // If set to 1, transmission of data is
    ↳ enabled otherwise only symbols are transmitted.
46
47 // The PN5180 receive buffer can hold a max of 508 bytes
48 // But we only need to transfer 2 bytes + 8 bytes per tag = 10 bytes for
    ↳ a single card
49 #ifndef READ_BUFFER_SIZE
50 #define READ_BUFFER_SIZE 10
51 #endif
52 // Other constants and enums
53 enum ISO15693ErrorCode {
54     EC_NO_CARD = -1,
55     ISO15693_EC_OK = 0,
56     ISO15693_EC_NOT_SUPPORTED = 0x01,
57     ISO15693_EC_NOT_RECOGNIZED = 0x02,
58     ISO15693_EC_OPTION_NOT_SUPPORTED = 0x03,
59     ISO15693_EC_UNKNOWN_ERROR = 0x0F,
60     ISO15693_EC_BLOCK_NOT_AVAILABLE = 0x10,
61     ISO15693_EC_BLOCK_ALREADY_LOCKED = 0x11,
62     ISO15693_EC_BLOCK_IS_LOCKED = 0x12,
63     ISO15693_EC_BLOCK_NOT_PROGRAMMED = 0x13,
64     ISO15693_EC_BLOCK_NOT_LOCKED = 0x14,
65     ISO15693_EC_CUSTOM_CMD_ERROR = 0xA0
66 };
67
68 class PN5180 {
69     public:
70     PN5180(uint8_t nssPin, uint8_t busyPin, uint8_t rstPin);
71     void begin();
72     void hardReset();
73     bool getInventory(uint8_t *uid);
74     bool getInventoryFake(uint8_t *uid);
75     void loadISO15693config();
76     bool activateRF();
77     bool disableRF();

```

```

78     bool checkIdle();
79     void clearIRQ();
80     void setIdle();
81     void activateTransceiver();
82     void sendInventoryCmd();
83     void sendEndOfFrame();
84     bool readReceptionBuffer(uint8_t *buffer, int16_t len);
85     uint32_t readRegister(uint8_t regAddress);
86     bool sendBytes(uint8_t *sendBuffer, size_t sendBufferLen);
87     bool readBytes(uint8_t *recvBuffer, size_t recvBufferLen);
88
89     private:
90     uint8_t _nss, _busy, _rst;
91     // SPISettings _spiSettings;
92     uint16_t _commandTimeout = 800;
93     bool waitUntilAvailable();
94     bool waitUntilBusy();
95     void errorHandler(IS015693ErrorCode errorCode);
96
97     uint8_t* buffer = (uint8_t*)malloc(READ_BUFFER_SIZE);
98 };
99
100 #endif

```

6.3 SPI Programming Code

6.3.1 .h

```

1  #ifndef MYUART_H
2  #define MYUART_H
3
4  #include <stdint.h>
5
6  #define RX_BUFFER_SIZE 64
7  #define MAX_BUFFER_SIZE 100
8
9  void USART_Init(void);
10 void USART_Transmit(uint8_t data);
11 uint8_t USART_Receive(void);
12 void USART_TxString(const char *str);
13 void USART_ReadString(char *buffer, uint8_t maxLen, uint32_t timeout);
14 int USART_ReceiveCommand(void);
15 void USART_TransmitCommand(const char *cmd);
16
17 #endif

```

6.3.2 .cpp

```

1  /*
2   * mySPI.cpp
3   *
4   * Created: 5/22/2025 1:46:21 PM
5   * Author: AGRA
6   */
7
8  #include <avr/io.h>

```

```

9      #include <util/delay.h>
10     #include <stdio.h>
11     #include <stdint.h>
12     #include "mySPI.h"
13
14     // Define F_CPU if not already defined
15     #ifndef F_CPU
16     #define F_CPU 16000000UL
17     #endif
18
19
20     // Initialize SPI as Master, Mode 0 (CPOL=0, CPHA=0), 125 kHz (F_CPU
21     ↪ /128)
22     void SPI_init(void) {
23         // Set MOSI (PB2), SCK (PB1), and NSS (PD1) as outputs; MISO (PB3)
24         ↪ as input
25         DDRB |= (1 << DDB2) | (1 << DDB1); // MOSI, SCK
26         DDRD |= (1 << DDD1); // NSS
27         DDRB &= ~(1 << DDB3); // MISO as input
28
29         // Configure SPI: Master, Mode 0, 125 kHz
30         SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR1) | (1 << SPR0);
31         SPSR &= ~(1 << SPI2X);
32     }
33
34     // Reapply SPI configuration (for PN5180 compatibility)
35     void SPI_setConfig(void) {
36         // Reapply SPI settings: Master, Mode 0, 125 kHz
37         SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR1) | (1 << SPR0);
38         SPSR &= ~(1 << SPI2X);
39     }
40
41     // Perform bidirectional SPI transfer
42     void SPI_transfer(uint8_t *buffer, size_t length) {
43         for (size_t i = 0; i < length; i++) {
44             SPDR = buffer[i]; // Write byte to SPDR
45             while (!(SPSR & (1 << SPIF))); // Wait for transmission complete
46             buffer[i] = SPDR; // Read received byte
47         }
48     }
49
50     // Disable SPI
51     void SPI_end(void) {
52         SPCR &= ~(1 << SPE); // Disable SPI
53     }
54
55     // Test SPI communication
56     void SPI_testCommunication(void) {
57         SPI_init();
58
59         uint8_t data = 0x55; // Arbitrary test byte
60         uint8_t received;
61
62         // Pull NSS (PD1) low to start communication
63         PORTD &= ~(1 << PORTD1);
64         _delay_ms(1);
65
66         // Send and receive 1 byte

```

```

66     SPDR = data;
67     while (!(SPSR & (1 << SPIF)));
68     received = SPDR;
69
70     // Pull NSS high to end communication
71     PORTD |= (1 << PORTD1);
72 }

```

6.4 UART Programming Code

6.4.1 .h

```

1  #ifndef MYUART_H
2  #define MYUART_H
3
4  #include <stdint.h>
5
6  #define RX_BUFFER_SIZE 64
7  #define MAX_BUFFER_SIZE 100
8
9  void USART_Init(void);
10 void USART_Transmit(uint8_t data);
11 uint8_t USART_Receive(void);
12 void USART_TxString(const char *str);
13 void USART_ReadString(char *buffer, uint8_t maxLen, uint32_t timeout);
14 int USART_ReceiveCommand(void);
15 void USART_TransmitCommand(const char *cmd);
16
17 #endif

```

6.4.2 .cpp

```

1  /*
2   * myUART.cpp
3   *
4   * Created: 5/22/2025 1:44:47 PM
5   * Author: AGRA
6   */
7
8  #include <avr/io.h>
9  #include <util/delay.h>
10 #include <stdio.h>
11 #include <string.h>
12 #include <stdint.h>
13 #include "myUART.h"
14
15 // Define F_CPU if not already defined
16 #ifndef F_CPU
17 #define F_CPU 16000000UL
18 #endif
19
20 // UART buffers
21 volatile static uint8_t rx_buffer[RX_BUFFER_SIZE] = {0};
22 volatile static uint16_t rx_count = 0;
23 volatile static uint8_t uart_tx_busy = 1;
24 static char rxBuffer[MAX_BUFFER_SIZE];

```



```

25     static uint8_t index = 0;
26
27     // UART transmit function (for debug output, consistent with previous
    ↪ files)
28     static void uart_transmit(const char *data) {
29         while (*data) {
30             while (!(UCSR1A & (1 << UDRE1))); // Wait for empty transmit
    ↪ buffer
31             UDR1 = *data++; // Send character
32         }
33     }
34
35     // Initialize USART1: 9600 baud, 8-bit, 1 stop bit, no parity,
    ↪ asynchronous
36     void USART_Init(void) {
37         // Set baud rate to 9600 (16 MHz -> UBRR1 = 103)
38         UBRR1 = 103;
39
40         // Enable transmitter and receiver
41         UCSR1B = (1 << RXEN1) | (1 << TXEN1);
42
43         // Set frame format: 8-bit data, 1 stop bit, no parity, asynchronous
44         UCSR1C = (1 << UCSZ11) | (1 << UCSZ10);
45         UCSR1B &= ~(1 << UCSZ12); // Ensure 8-bit data
46         UCSR1C &= ~((1 << UMSEL11) | (1 << UMSEL10) | (1 << UPM11) | (1 <<
    ↪ UPM10) | (1 << USBS1));
47
48         // Normal speed (disable double speed)
49         UCSR1A &= ~(1 << U2X1);
50
51         // Debug output
52         char msg[] = "USART1 initialized\r\n";
53         uart_transmit(msg);
54     }
55
56     // Transmit one byte
57     void USART_Transmit(uint8_t data) {
58         while (!(UCSR1A & (1 << UDRE1))); // Wait for empty transmit buffer
59         UDR1 = data;
60     }
61
62     // Receive one byte
63     uint8_t USART_Receive(void) {
64         while (!(UCSR1A & (1 << RXC1))); // Wait for data
65         return UDR1;
66     }
67
68     // Transmit a null-terminated string
69     void USART_TxString(const char *str) {
70         while (*str) {
71             USART_Transmit(*str++);
72         }
73     }
74
75     // Receive a string into buffer (non-blocking, with timeout)
76     void USART_ReadString(char *buffer, uint8_t maxLen, uint32_t timeout) {
77         uint8_t index = 0;
78         buffer[0] = '\0'; // Initialize buffer
79

```

```

80 // Timeout loop (~10 ms at 16 MHz, adjusted for cycles)
81 while (timeout > 0) {
82     if (UCSR1A & (1 << RXC1)) {
83         char c = USART_Receive();
84         if (c == '\n') {
85             buffer[index] = '\0'; // Null-terminate
86             return;
87         } else if (index < maxLen - 1) {
88             buffer[index++] = c;
89         } else {
90             buffer[0] = '\0'; // Overflow, return empty
91                                     ↪ string
92             return;
93         }
94     }
95     _delay_us(1); // ~1 s delay per iteration
96     timeout--;
97 }
98 buffer[0] = '\0'; // Timeout, return empty string
99
100 // Receive and parse command
101 int USART_ReceiveCommand(void) {
102     char response[MAX_BUFFER_SIZE];
103     USART_ReadString(response, MAX_BUFFER_SIZE, 10000); // ~10 ms
104                                     ↪ timeout
105
106     // Debug output
107     char debugMsg[128];
108     snprintf(debugMsg, sizeof(debugMsg), "Received: %s\r\n", response);
109     uart_transmit(debugMsg);
110
111     if (strlen(response) == 0) {
112         return 0; // No complete message
113     }
114
115     // Compare response
116     if (strcmp(response, "OK") == 0) {
117         return 1; // Success
118     } else if (strcmp(response, "ERROR") == 0) {
119         return 2; // Failure
120     } else if (strncmp(response, "DA", 2) == 0) {
121         return 3; // Data received
122     } else {
123         return -1; // Unknown response
124     }
125 }
126
127 // Transmit command with "CMD:" prefix and '\n'
128 void USART_TransmitCommand(const char *cmd) {
129     char buffer[MAX_BUFFER_SIZE] = {0};
130     uint8_t len = strlen(cmd);
131
132     // Add "CMD:" prefix and '\n' if needed
133     if (len > 0 && cmd[len - 1] != '\n') {
134         snprintf(buffer, MAX_BUFFER_SIZE - 1, "CMD:%s\n", cmd);
135     } else {
136         snprintf(buffer, MAX_BUFFER_SIZE - 1, "CMD:%s", cmd);
137     }

```

```
137
138     // Ensure null-termination
139     buffer[MAX_BUFFER_SIZE - 1] = '\0';
140
141     // Send via UART
142     USART_TxString(buffer);
143 }
```