ESP32 Winbond W25Q32JVSSIQ Flash Memory Controller

A comprehensive FreeRTOS-based firmware for ESP32 to control Winbond W25Q32JVSSIQ SPI flash memory (4MB) with Bluetooth Serial interface and advanced ring buffer functionality.

📋 Table of Contents

✨ Features

- **Bluetooth Serial Control**: Full wireless control via Bluetooth Serial interface
- **Ring Buffer Management**: Circular buffer system with automatic sector erasing and wrapping
- **Auto-Write System**: Automated random data generation with timestamps
- **Read/Write/Erase Operations**: Complete flash memory management
- **FreeRTOS Architecture**: Multi-task design with thread-safe SPI access
- **Full Flash Dump**: Read entire 4MB flash memory with stop capability
- **Pause/Resume Control**: Automatic write pausing during read operations
- **Comprehensive Logging**: Detailed serial output for debugging and monitoring

🔧 Hardware Requirements

- **Microcontroller**: ESP32 DOIT DevKit V1 (or compatible)
- **Flash Memory**: Winbond W25Q32JVSSIQ (4MB SPI Flash)
- **Connections**: 4 wires for SPI communication (CLK, MISO, MOSI, CS)

📌 Pin Configuration

ESP32 Pin	Flash Pin	Function
GPIO 14	CLK	SPI Clock
GPIO 12	MISO	Master In Slave Out
GPIO 13	MOSI	Master Out Slave In
GPIO 26	CS	Chip Select

Power Connections:

- Flash VCC → ESP32 3.3V
- Flash GND → ESP32 GND

📥 Installation

Prerequisites

- PlatformIO installed
- USB cable for ESP32 programming
- Bluetooth Serial terminal app (e.g., Serial Bluetooth Terminal for Android)

Build and Upload

1. **Clone the repository:**

```
git clone https://github.com/Bamamou/ESP32_Winbond_W25Q32JVSSIQ.git
cd ESP32_Winbond_W25Q32JVSSIQ
```

2. **Build the project:**

```
platformio run
```

3. **Upload to ESP32:**

```
platformio run --target upload
```

4. **Monitor Serial output (optional):**

```
platformio device monitor
```

🚀 Getting Started

1. Connect Hardware

Wire the Winbond flash chip to your ESP32 according to the pin configuration.

2. Upload Firmware

Upload the firmware using PlatformIO (see Installation).

3. Connect via Bluetooth

1. Power on the ESP32
2. Open your Bluetooth Serial terminal app
3. Search for device named **"ESP32-Flash"**
4. Connect to the device
5. Type `help` to see all available commands

4. Initialize Ring Buffer (Optional)

If you want to use ring buffer features:

```
ringinit
```

This scans the flash memory and sets the write position (takes ~3 minutes).

5. Start Using

You're ready! Try some basic commands:

```
info                    # Show flash chip information
write 0 Hello World    # Write a string
read 0                 # Read the string back
```

📖 Command Reference

All commands are case-insensitive and sent via Bluetooth Serial. Addresses are in hexadecimal format.

Write Commands

`write <addr> <data>`

Write a string to flash memory at the specified address.

Example:

```
write 0 Hello World
write 1000 Temperature:25.5C
```

Notes:

- Address in hex (e.g., `1000` = 0x1000)
- Automatically erases sector before writing
- Adds null terminator to string

`writeb <addr> <byte1,byte2,...>`

Write raw bytes to flash memory.

Example:

```
writeb 0 65,66,67,68,69          # Writes "ABCDE"
writeb 2000 0,1,2,3,4,5,6,7      # Writes 0x00 to 0x07
writeb 3000 255,128,64,32        # Writes 0xFF,0x80,0x40,0x20
```

Notes:

- Bytes separated by commas
- Values: 0-255
- Max 256 bytes per command

Read Commands

`read <addr>`

Read a null-terminated string from flash memory.

Example:

```
read 0
read 1000
```

Output:

```
[BT] Result: Hello World
```

`readb <addr> <length>`

Read raw bytes from flash memory.

Example:

```
readb 0 16          # Read 16 bytes
readb 1000 32       # Read 32 bytes
```

Output:

```
[BT] Result: 48 65 6C 6C 6F 20 57 6F 72 6C 64 00 FF FF FF FF
```

Notes:

- Max 256 bytes per read
- Output in hexadecimal format

`readrange <start> <end>`

Read a range of addresses.

Example:

```
readrange 0 FF                # Read 256 bytes (0x00 to 0xFF)
readrange 1000 10FF           # Read 256 bytes (0x1000 to 0x10FF)
```

Notes:

- Max 256 bytes range
- Both addresses in hex

`readall`

Dump entire 4MB flash memory to Bluetooth.

Example:

```
readall
```

Output:

```
[BT] Starting full flash dump (4MB)...
[BT] This will take several minutes...
[BT] Send 'stop' command to abort

========== FLASH MEMORY DUMP START ==========
[00000000] FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

```
[00000010] FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
...
[PROGRESS] 25% - 1024 KB
...
```

Notes:

- Takes several minutes to complete
- 16 bytes per line in hex format
- Progress shown every 64KB
- Pauses ring buffer writes during dump

`stop`

Stop an ongoing `readall` operation.

Example:

```
# While readall is running:
stop
```

Output:

```
[BT] ✓ Read operation stopped by user
Total bytes read: 65536 (0.06 MB)
```

Erase Commands

`erase <addr>`

Erase a 4KB sector at the specified address.

Example:

```
erase 0              # Erase sector 0 (0x0000 - 0x0FFF)
erase 1000           # Erase sector at 0x1000
```

Notes:

- Erases entire 4KB sector (4096 bytes)
- Address automatically aligned to sector boundary
- Sets all bytes in sector to 0xFF

`eraserange <start> <end>`

Erase multiple sectors in a range.

Example:

```
eraserange 0 3FFF           # Erase first 4 sectors (16KB)
eraserange 1000 5000        # Erase sectors from 0x1000 to 0x5000
```

Notes:

- Start/end addresses aligned to sector boundaries
- Shows progress for each sector erased

`eraseall`

Erase the entire 4MB flash chip.

Example:

```
eraseall
```

⚠ **WARNING:** This erases ALL data on the flash chip (1024 sectors). Cannot be undone!

Ring Buffer Commands

The ring buffer provides circular buffer functionality with automatic sector management and wrapping.

`ringinit`

Initialize the ring buffer by scanning flash memory.

Example:

```
ringinit
```

What it does:

- Scans all 1024 sectors (takes ~3 minutes)
- Finds where existing data ends
- Sets write position to first empty sector
- Required before using other ring buffer commands

Output:

```
[RING] Initializing ring buffer...
[RING] Scanning for first empty sector...
[RING] Scanned 100/1024 sectors...
...
[RING] Write position set to sector 45 (address 0x0002D000)
```

Notes:

- Must run after every reboot (state not saved in flash)
- Only needed once per session

`ringwrite <data>`

Write string data to ring buffer.

Example:

```
ringwrite Temperature: 25.5C
ringwrite Sensor reading #123
```

Output:

```
[BT] ✓ Write successful
[BT] Next write position: 0x0002D020
```

Notes:

- Automatically erases sectors when needed
- Wraps to address 0x00000000 when reaching end
- Requires `ringinit` first

`ringwriteb <byte1,byte2,...>`

Write raw bytes to ring buffer.

Example:

```
ringwriteb 65,66,67,68,69
ringwriteb 0,1,2,3,4,5
```

Notes:

- Same as `writeb` but uses ring buffer position

- Max 256 bytes per command

`ringstatus`

Show ring buffer status and configuration.

Example:

```
ringstatus
```

Output (initialized):

```
[RING] Ring Buffer Status:
  Initialized: YES
  Current position: 0x0002D000
  Current sector: 45
  Status: ACTIVE
  Auto-write: DISABLED
  Flash capacity: 0x00400000 (4.00 MB)
  Sector size: 0x00001000 (4096 bytes)
```

Output (not initialized):

```
[RING] Ring Buffer Status:
  Status: NOT INITIALIZED
  Run 'ringinit' to initialize the ring buffer
  Note: Ring buffer state is lost on reboot
```

`ringsetpos <addr>`

Manually set ring buffer write position.

Example:

```
ringsetpos 0            # Set to beginning
ringsetpos 10000        # Set to 0x10000
```

Notes:

- Address automatically aligned to sector boundary
- Use with caution - may overwrite existing data
- Alternative to `ringinit` if you know the position

`ringreset`

Reset ring buffer to the beginning of flash.

Example:

```
ringreset
```

Output:

```
[BT] ✓ Ring buffer reset to 0x00000000
```

Notes:

- Sets position to 0x00000000
- Does NOT erase flash
- Next write will start from beginning

Auto-Write Commands

Automatically generate and write random data to flash memory.

`autostart`

Start automatic data generation.

Example:

```
ringinit        # Initialize first
autostart       # Start auto-writing
```

Output:

```
[BT] ✓ Auto-write started
[BT] Writing random numbers (0-1000) every second
[BT] Use 'autostop' to stop
```

Serial Monitor Output:

```
[AUTO] #1: Wrote 742 at 0x00000000 (time: 12345 ms)
[AUTO] #2: Wrote 156 at 0x00000006 (time: 13345 ms)
```

```
[AUTO] #3: Wrote 889 at 0x0000000C (time: 14345 ms)
...
```

What it does:

- Generates random number (0-1000) every second
- Writes 6 bytes per entry (4-byte timestamp + 2-byte value)
- Automatically handles sector erasing and wrapping
- Runs in background FreeRTOS task on Core 1

Notes:

- Requires `ringinit` first
- Pauses automatically during read operations
- Continues until stopped or power loss

`autostop`

Stop automatic data generation.

Example:

```
autostop
```

Output:

```
[BT] ✓ Auto-write stopped
```

Notes:

- Does not delete written data
- Can restart with `autostart` at any time

Info Commands

`info`

Display flash chip information.

Example:

```
info
```

Output:

```
[INFO] Flash Chip Information:
  JEDEC ID: 0x00004016
  Capacity: 4194304 bytes (4.00 MB)
  Max Pages: 16384
  Sector Size: 4096 bytes
```

Notes:

- JEDEC ID 0x4016 = Winbond W25Q32
- Capacity: 4MB = 4,194,304 bytes
- 1024 sectors × 4096 bytes each

`help`

Show all available commands.

Example:

```
help
```

Output:

```
========== FLASH MEMORY COMMANDS ==========
Write Commands:
  write <addr> <data>     - Write string to address (hex)
  writeb <addr> <bytes>   - Write bytes (e.g., writeb 1000 0,1,2,3)

Read Commands:
  read <addr>             - Read string from address (hex)
  readb <addr> <len>      - Read bytes (e.g., readb 1000 16)
  readrange <start> <end> - Read address range (hex)
  readall                 - Dump entire flash (4MB!)
  stop                    - Stop readall operation

Erase Commands:
  erase <addr>            - Erase sector at address (hex)
  eraserange <start> <end> - Erase address range (hex)
  eraseall                - Erase entire chip (CAUTION!)

Ring Buffer Commands:
  ringinit                - Initialize ring buffer (scan for write pos)
  ringwrite <data>        - Write string to ring buffer
  ringwriteb <b1,b2,...> - Write bytes to ring buffer
  ringstatus              - Show ring buffer position
  ringsetpos <addr>       - Set ring buffer position
  ringreset              - Reset ring buffer to 0x00000000

Auto-Write Commands:
  autostart              - Start auto-writing random numbers
  autostop               - Stop auto-writing

Info Commands:
  info                   - Show flash chip information
  help                   - Show this menu
===========================================
```

🔄 Ring Buffer System

How It Works

The ring buffer implements a circular buffer in flash memory:

1. **Initialization**: Scans flash to find the boundary between data and empty space
2. **Writing**: Writes data sequentially, erasing sectors as needed
3. **Wrapping**: When reaching end (0x3FFFFF), wraps to beginning (0x00000000)
4. **Auto-Erase**: Automatically erases next sector before writing to it

Memory Layout

```
0x00000000  [==========Data==========][====Empty====]  0x003FFFFF
                                       ↑
                                Write Position
```

After wrapping:

```
0x00000000  [==New Data==][==Old Data==][====Empty====]  0x003FFFFF
             ↑
        Write Position (wrapped)
```

Use Cases

- **Data Logging**: Continuous sensor data logging
- **Event Recording**: System events with timestamps
- **Circular Buffers**: FIFO-style data storage
- **Wear Leveling**: Distribute writes across flash

Important Notes

- Ring buffer state (write position) is lost on reboot
- Must run `ringinit` after every power cycle

- Write position is stored in RAM, not flash
- Automatic sector erasing means old data is overwritten

🤖 Auto-Write Feature

Data Format

Each auto-write entry is **6 bytes**:

Bytes	Content	Format
0-3	Timestamp	uint32_t (little-endian)
4-5	Random Number	uint16_t (little-endian)

Example Entry:

```
Hex: 39 30 00 00 E6 02
Timestamp: 0x00003039 = 12345 ms
Value: 0x02E6 = 742
```

Parsing Auto-Write Data

To read auto-write data:

```python
# Python example
timestamp = data[0] | (data[1] << 8) | (data[2] << 16) | (data[3] << 24)
value = data[4] | (data[5] << 8)
print(f"Time: {timestamp} ms, Value: {value}")
```

Performance

- **Write Rate**: 1 entry per second (6 bytes/sec)
- **Flash Capacity**: 4,194,304 bytes
- **Max Entries**: ~699,050 entries
- **Full Time**: ~8 days to fill entire flash
- **Continuous**: Wraps and continues indefinitely

🔬 Technical Details

Memory Specifications

- **Flash Chip**: Winbond W25Q32JVSSIQ
- **Total Size**: 4MB (4,194,304 bytes)
- **Sector Size**: 4KB (4,096 bytes)
- **Total Sectors**: 1,024

- **Page Size**: 256 bytes
- **Max Pages**: 16,384

Address Range

- **Start**: 0x00000000
- **End**: 0x003FFFFF (4,194,303)
- **Sector 0**: 0x00000000 - 0x00000FFF
- **Sector 1**: 0x00001000 - 0x00001FFF
- **Sector 1023**: 0x003FF000 - 0x003FFFFF

FreeRTOS Architecture

Tasks:

Task	Core	Priority	Stack	Purpose
bluetoothTask	0	1	4096	Bluetooth command processing
monitorTask	0	1	2048	System monitoring (every 10s)
autoWriteTask	1	1	4096	Auto data generation

Synchronization:

- Mutex-protected SPI access
- Thread-safe flash operations
- Automatic pause during reads

Memory Usage

- **Flash**: 86.5% (1,133,769 / 1,310,720 bytes)
- **RAM**: 12.3% (40,264 / 327,680 bytes)

Libraries

- **SPIMemory** v3.4.0 (local)
- **BluetoothSerial** v2.0.0 (built-in)
- **SPI** v2.0.0 (built-in)

🔍 Troubleshooting

Bluetooth Connection Issues

Problem: Cannot connect to "ESP32-Flash"

Solutions:

1. Ensure ESP32 is powered on
2. Check serial monitor for "Bluetooth device ready" message
3. Forget device and reconnect

4. Restart Bluetooth on phone
5. Try different Bluetooth terminal app

Ring Buffer Not Writing

Problem: `ringwrite` fails with error

Solutions:

1. Run `ringinit` first (required after every reboot)
2. Check `ringstatus` to verify initialization
3. Ensure flash is not write-protected
4. Verify flash connections with `info` command

Flash Not Detected

Problem: `info` shows incorrect JEDEC ID or zeros

Solutions:

1. Check all 4 SPI wire connections (CLK, MISO, MOSI, CS)
2. Verify 3.3V power supply (not 5V!)
3. Check GND connection
4. Ensure GPIO pins match configuration
5. Try adding pull-up resistors (10kΩ) on CS line

Auto-Write Not Starting

Problem: `autostart` command does nothing

Solutions:

1. Run `ringinit` before `autostart`
2. Check `ringstatus` for "Auto-write: ENABLED"
3. Monitor serial output for "[AUTO]" messages
4. Stop and restart: `autostop` then `autostart`

Read Operations Return 0xFF

Problem: Reading returns all 0xFF bytes

Solutions:

- 0xFF means erased/empty flash - this is normal
- Write data first with `write` or `writeb`
- Check that correct address is being read
- Verify data was actually written successfully

Memory Full

Problem: Flash memory is full

Solutions:

1. Erase specific sectors: `erase <addr>`
2. Erase all: `eraseall` (⚠ deletes everything)
3. Use ring buffer mode (automatically overwrites old data)

📊 Example Workflows

Basic Data Storage

```
# Connect via Bluetooth
# Write some data
write 0 Sensor1:Temperature
write 100 Sensor2:Humidity
write 200 Sensor3:Pressure

# Read it back
read 0
read 100
read 200

# Erase when done
erase 0
```

Continuous Data Logging

```
# Initialize ring buffer
ringinit

# Start automatic logging
autostart

# Check status periodically
ringstatus

# Stop when needed
autostop

# Dump all data
readall
```

Manual Ring Buffer Usage

```
# Initialize
ringinit

# Write multiple entries
ringwrite Reading #1: 25.5C
ringwrite Reading #2: 26.1C
ringwrite Reading #3: 25.8C

# Check position
ringstatus

# Continue writing (wraps automatically)
ringwrite Reading #4: 26.5C
```

Flash Memory Analysis

```
# Get chip info
info

# Check specific address
readb 0 256

# Dump entire flash
readall

# Stop dump if needed
stop
```

📝 License

This project is open source. Feel free to use, modify, and distribute.

🤝 Contributing

Contributions are welcome! Please feel free to submit pull requests or open issues.

📧 Contact

- **Author**: Bamamou
- **Repository**: ESP32_Winbond_W25Q32JVSSIQ

🎯 Quick Command Cheat Sheet

```
# Info
help                        # Show all commands
info                        # Flash chip info

# Basic Write/Read
write 0 Hello              # Write string
read 0                     # Read string
writeb 0 65,66,67          # Write bytes
readb 0 16                 # Read 16 bytes

# Erase
erase 0                    # Erase sector
eraseall                   # Erase everything

# Ring Buffer
ringinit                   # Initialize (required once per boot)
ringwrite Data             # Write to ring buffer
ringstatus                 # Check status

# Auto-Write
autostart                  # Start auto logging
autostop                   # Stop auto logging

# Advanced
readall                    # Dump all 4MB
stop                       # Stop dump
ringsetpos 0               # Set position
```

Enjoy using your ESP32 Flash Memory Controller! 🚀