CCI Low-Level Module

Table of Contents

1. Overview

The `cci` (Command and Control Interface) module implements the low-level I2C communication protocol for the Lepton thermal sensor. It handles command transmission, status polling, and data transfer according to the Lepton CCI specification.

1.1. Key Features

- I2C-based communication at 7-bit address 0x2A
- Command/response protocol with status polling
- Boot detection and timeout handling

- Register read/write operations
- Block buffer transfers for large data
- Comprehensive error handling

2. CCI Protocol

The CCI protocol uses a master-slave I2C communication:

2.1. Register Map

Address	Register	Purpose
0x0002	STATUS	Command status and busy flag
0x0004	COMMAND	Command ID to execute
0x0006	DATA_LENGTH	Data length in 16-bit words
0x0008-0x0026	DATA_0 to DATA_15	Data buffer (max 16 words)
0xF800	BLOCK_BUF_0	Large data block buffer 0
0xFC00	BLOCK_BUF_1	Large data block buffer 1

2.2. Command Execution Flow

1. Write command ID to COMMAND register
2. (Optional) Write data to DATA registers
3. Poll STATUS register until not busy
4. Check for errors in STATUS
5. (Optional) Read response from DATA registers

2.3. Status Register

Bit	Meaning
15	Busy: 1 = command executing, 0 = ready
8-15	Error code (when busy = 0)
0-7	Boot status / command result

3. SDK Modules

The CCI interface is organized into SDK modules:

3.1. AGC Module (0x0100)

Automatic Gain Control configuration:

- Enable/disable AGC
- Set AGC policy (HEQ, Linear)
- Configure ROI
- Histogram statistics
- HEQ parameters (dampening, clip limits, scaling)

3.2. SYS Module (0x0200)

System commands:

- Ping (connectivity test)
- Read serial number
- Get uptime
- Read temperatures (FPA, AUX)
- Telemetry control
- Scene statistics
- Part number

3.3. VID Module (0x0300)

Video configuration:

- Video output format (RAW14, RGB888)
- Video source (normal, ramp, constant)
- Pseudo-color LUT
- Focus metrics
- Video freeze
- User-defined color LUT

3.4. OEM Module (0x0800)

OEM-specific features:

- Power control
- GPIO configuration
- Reboot command
- FFC (Flat Field Correction) modes

3.5. RAD Module (0x0E00)

Radiometry (Lepton 3.5 only):

- TLinear enable/disable
- TLinear resolution
- Spotmeter configuration
- Flux linear parameters
- Emissivity and scene parameters

4. Key Functions

4.1. Initialization

```
Lepton_Error_t CCI_Init(CCI_t *p_Interface);
```

Initialize CCI interface with I2C callbacks.

4.2. Boot Detection

```
Lepton_Error_t CCI_WaitForBoot(CCI_t *p_Interface, Lepton_Result_t *p_Status);
```

Wait for sensor boot (polls STATUS until boot complete).

4.3. Command Execution

```
Lepton_Error_t CCI_ExecuteCommand(
    CCI_t *p_Interface,
    uint16_t Command,
    void *p_DataIn,
    size_t DataInLength,
    void *p_DataOut,
    size_t DataOutLength,
    Lepton_Result_t *p_Status
);
```

Execute a CCI command with optional input/output data.

Parameters:

- `Command`: CCI command ID (e.g., `CCI_CMD_SYS_GET_FPA_TEMP`)
- `p_DataIn`: Input data buffer (or NULL)

- `DataInLength`: Input data size in bytes
- `p_DataOut`: Output data buffer (or NULL)
- `DataOutLength`: Output data size in bytes
- `p_Status`: Optional status result

4.4. Register Access

```
Lepton_Error_t CCI_ReadRegister(CCI_t *p_Interface, uint16_t Register, uint16_t
*p_Value);
Lepton_Error_t CCI_WriteRegister(CCI_t *p_Interface, uint16_t Register, uint16_t
Value);
```

Direct register read/write (16-bit values).

4.5. Block Transfers

For large data (> 32 bytes):

```
Lepton_Error_t CCI_WriteBlockBuffer(CCI_t *p_Interface, void *p_Data, size_t Length);
Lepton_Error_t CCI_ReadBlockBuffer(CCI_t *p_Interface, void *p_Data, size_t Length);
```

5. Error Handling

CCI functions return:

- `LEPTON_ERR_OK`: Success
- `LEPTON_ERR_TIMEOUT`: Command timeout
- `LEPTON_ERR_FAIL`: CCI error code returned
- `LEPTON_ERR_INVALID_ARG`: Invalid parameters

The `Lepton_Result_t` structure contains:

```
typedef struct {
    uint16_t StatusCode;  // CCI status register value
    uint8_t ErrorCode;    // Extracted error code (bits 8-15)
} Lepton_Result_t;
```

6. Timeout Configuration

CCI operations have timeouts:

- Boot wait: 5 seconds
- Command execution: 1 second
- Status polling: 100ms intervals

7. I2C Callbacks

The CCI module requires application-provided I2C functions:

```
typedef Lepton_Error_t (*CCI_I2C_Read_t)(uint8_t Address, uint16_t Register, uint8_t
*p_Data, size_t Length);
typedef Lepton_Error_t (*CCI_I2C_Write_t)(uint8_t Address, uint16_t Register, uint8_t
*p_Data, size_t Length);
```

Example Implementation:

```
Lepton_Error_t my_i2c_read(uint8_t addr, uint16_t reg, uint8_t *data, size_t len) {
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (addr << 1) | I2C_MASTER_WRITE, true);
    i2c_master_write_byte(cmd, reg >> 8, true);
    i2c_master_write_byte(cmd, reg & 0xFF, true);
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (addr << 1) | I2C_MASTER_READ, true);
    i2c_master_read(cmd, data, len, I2C_MASTER_LAST_NACK);
    i2c_master_stop(cmd);
    esp_err_t ret = i2c_master_cmd_begin(I2C_NUM_0, cmd, 1000 / portTICK_PERIOD_MS);
    i2c_cmd_link_delete(cmd);
    return (ret == ESP_OK) ? LEPTON_ERR_OK : LEPTON_ERR_FAIL;
}
```

8. Usage Example

```
CCI_t cci_interface = {
    .I2C_Read = my_i2c_read,
    .I2C_Write = my_i2c_write,
    .Address = 0x2A
};

Lepton_Result_t status;

// Initialize
CCI_Init(&cci_interface);

// Wait for boot
CCI_WaitForBoot(&cci_interface, &status);
```

```
// Read FPA temperature
uint16_t fpa_temp;
CCI_GetFPATemp(&cci_interface, &fpa_temp, &status);

// Execute custom command
uint16_t data_out[2];
CCI_ExecuteCommand(&cci_interface, CCI_CMD_SYS_GET_SCENE_STATISTICS,
                   NULL, 0, data_out, sizeof(data_out), &status);
```

9. See Also

- CCI Commands - High-level command interface
- Main Driver - Complete driver API

10. License

Website: https://www.kampis-elektroecke.de