Discrete I/O Fundamentals

GPIO configuration and electrical characteristics

Target platform: EVKB i.MX RT1050

Focus: avionics-style “discretes” and robust engineering choices

Outline

What "Discrete I/O" Means in Avionics

- A discrete is a binary signal: an electrical line that is asserted (true) or de-asserted (false)
- Discretes announce aircraft states like WOW, FLT/GRD, FIRE DETECT, DOOR CLOSED, APU READY
- Software interfaces to discretes through GPIO pins

GPIO Capabilities You Use for Discretes

- Configure GPIO as input or output
- Internal pull-ups and pull-downs to define a default state
- Schmitt trigger (hysteresis) for noise rejection
- Open drain mode, drive strength, and slew rate control
- Goal: reliable logic levels and noise immunity

i.MX RT1050 Pin Mux and Pad Configuration

- Each physical pad is routed through a pin mux to a chosen peripheral function (GPIO, UART, SPI, etc.)
- First select GPIO in the mux
- Then configure pad electrical characteristics (pulls, slew, drive, hysteresis)
- Rationale: logic must be reliable, noise tolerant, and compatible with surrounding hardware

Safety-Critical Expectations Around Discretes

- Determinism is expected in safety-critical avionics implementations
- Diagnosability via BIT (Built-In Test) is a typical requirement
- Tolerance to EMI/EMC and ESD is expected (e.g., DO-160)
- DAL software (DO-178C) and hardware (DO-254) processes assume disciplined discrete engineering

Key Terminology (1 of 2)

- GPIO: configurable digital pin used as binary input or output
- VIH / VIL: input high/low thresholds (recognized as logic 1 / logic 0)
- VOH / VOL: guaranteed output high/low voltages under load
- Drive strength: sourcing and sinking capability while maintaining valid logic levels
- Slew rate: edge speed (slower edges generally reduce EMI)

Key Terminology (2 of 2)

- Schmitt trigger (hysteresis): separate rising/falling thresholds to reject noise and slow edges
- Pull-up / pull-down: weak resistors to rails that prevent floating and define defaults
- Push-pull: output can actively drive both high and low
- Open drain: output can only pull low; external or internal pull-up defines high level
- Debounce: filtering to remove mechanical chatter and narrow spikes
- Wired-OR / wired-AND: multiple open-drain outputs share a line; any device pulling low dominates

Electrical Characteristics: Logic Levels and Supplies

- EVKB user I/O domain is 3.3 V (LVCMOS)
- Thresholds are often expressed as fractions of VDDIO (confirm in datasheet)
- Never drive GPIO above its bank supply or below ground beyond allowed limits
- Always respect absolute maximum ratings

Electrical Characteristics: Inputs

- Floating inputs are unsafe and noise sensitive
- Use pull-up for active-low signals and pull-down for active-high signals
- Schmitt trigger is especially valuable for external harnessed signals or RC-filtered inputs
- Glitch filtering can be done in software (sampling and debounce) or hardware where available

Pull-Up / Pull-Down: Why They Exist (1 of 2)

- Define a default state so an input does not float
- Provide a leakage path that improves immunity to external electromagnetic interference
- Help when interfacing logic families (example: raising an otherwise marginal “high” level in some interfacing cases)
- Required for open-collector or open-drain style outputs so the line can return high

Pull-Up / Pull-Down: Additional Practical Reasons (2 of 2)

- Improve effective drive capability for some pins by providing a defined bias path
- Unused CMOS inputs should not be left floating; bias them to a known level to reduce ESD and EMI susceptibility
- For longer lines, biasing and impedance planning helps reduce false switching and reflection issues (proper termination strategy still matters)
- Preset bus idle state (example: I^2C idle high uses pull-ups)
- Improve noise tolerance by avoiding random levels in high-impedance states

Electrical Characteristics: Outputs (Push-Pull vs Open Drain)

- Push-pull output stage can source or sink current (drive high and low)
- Useful for on-board logic at the same voltage domain
- Open-drain outputs pull low via a transistor; a pull-up completes the path to logic high
- Open drain is useful for shared lines, fail-safe wired-OR, and level shifting (with correct external circuitry)
- Real line behavior also includes capacitance from PCB traces and connected devices (affects edges and rise time)

Drive Strength and Slew Rate

- Use the lowest drive strength and slowest slew rate that still meets timing
- Slower edges reduce EMI and ringing
- Faster edges can increase emissions and susceptibility issues

Protection and Aircraft Interfacing Reality

- Aircraft discretes may be based on 28 V nominal systems
- Never connect aircraft-level signals directly to MCU GPIO pins
- Use appropriate external interface circuitry: dividers, TVS, RC filters, isolation (optocoupler or digital isolator)
- For outputs, use transistor or MOSFET stages (often open-drain style) into aircraft discrete networks
- Add EMI and ESD components consistent with DO-160 style robustness expectations

EVKB i.MX RT1050 SDK Building Blocks

- Pin mux and pad control via IOMUXC APIs (or tool-generated pin initialization)
- GPIO driver provides pin init, read, write, and interrupt configuration
- Board helpers initialize pins, clocks, and debug console printing
- Use the SDK board macros for LED and button mapping rather than hardcoding banks and pins

Workflow: Requirements → Pad Configuration → GPIO Logic

- Start with a requirement statement (example: active-low input with debounce + LED output)
- Choose hardware strategy: pull-up, hysteresis, and debounce plan
- Initialize baseline pin mux and pad config (board init or pin tool output)
- Configure GPIO direction and interrupts
- Implement read and write behavior (polling or ISR plus filtering)

Avionics Use Case Example: WOW with Diagnostics

- Input WOW discrete through an isolated interface
- Filter bounce and spikes, then vote samples to determine a robust truth
- Timestamp edges for maintenance logging to a central maintenance computer
- Output FLT/GRD as an open-drain discrete so multiple LRUs can share the line (wired-OR)
- Power-up and periodic BIT detect stuck-at faults
- Record both raw and filtered states with timestamps

Hands-On Labs Overview (What You Build)

- Labs assume a bare-metal project with board pin init, clock init, and debug console init
- LED and button pins are taken from SDK macros so projects are portable and consistent
- Progression: project skeleton $\rightarrow$ output control $\rightarrow$ input with interrupt and debounce $\rightarrow$ voting and timestamping $\rightarrow$ pad audit

Lab 0: Project Skeleton (No Code Shown)

- Initialize pins, clocks, and debug console
- Print a startup banner to prove basic bring-up
- Run an idle loop to keep the firmware alive and observable

Lab 1: Output Lab (User LED)

- Configure the user LED GPIO as a digital output
- Blink or toggle at a known rate
- Reasoning: LED is local at 3.3 V so push-pull is appropriate
- If EMI issues are observed, reduce drive strength and slow slew via pad config

Lab 2: Input Lab (SW8) with Robustness

- Configure SW8 as an input with edge interrupts
- Ensure pull-up and hysteresis are enabled at the pad level for noise immunity
- Implement deterministic debounce (example: resample after a fixed delay)
- Provide feedback by toggling the LED when a stable press is detected

Lab 3: Toward WOW (Timestamp + Voting)

- Timestamp each detected edge using a microsecond time base
- Convert raw active-low input into a logical asserted state
- Use a majority-vote window (example: 3 of 5) to create a robust truth
- Print edge time and voted truth, and reflect truth on LED

Lab 4: Pad Settings You Must Be Able to Audit

- Confirm mux selection matches intended GPIO pad routing
- Inputs: enable pull and Schmitt trigger hysteresis (especially external signals)
- Outputs: choose slow slew and modest drive unless timing demands otherwise
- Use open drain when creating shared aircraft discrete outputs
- Maintain awareness of pad configuration values as device and SDK version specifics

Avionics-Grade Scenarios to Simulate

- WOW with transient spikes: inject brief pulses and show filtering rejects them
- Wired-OR FLT/GRD line: two open-drain outputs share one pull-up line; either can assert low
- Stuck-at BIT: force stuck low or stuck high and detect at power-up or periodically
- EMI tradeoff: observe ringing at high toggle rate, then reduce drive and slow slew and compare behavior

Best Practices (Hardware)

- Match electrical domains: never expose 28 V discretes directly to MCU pins
- Fail-safe states: choose pulls and output types so faults default to the least hazardous meaning
- Enable hysteresis for external discretes
- Respect pull strengths: avoid fighting strong external pulls with internal ones
- Tame EMI using low drive, slow slew, and optional series resistors

Best Practices (Software + Process)

- Debounce deterministically with a stable time base and record raw and filtered state
- BIT for stuck-at and open-circuit; use loopbacks where possible
- Keep ISRs short: clear flags first, defer work to main loop or task
- Document pad-to-bank-to-rail mapping; keep pin configuration under revision control
- Maintain traceability: requirements $\rightarrow$ pad config $\rightarrow$ code $\rightarrow$ tests

Assessment Checklist

- Asserted polarity and fail-safe state documented
- Pad configs set pulls, hysteresis, slew, drive, and open drain where needed
- Debounce and filtering rationale stated (example: 5 ms)
- BIT covers stuck high, stuck low, and open-circuit behaviors
- Edge timestamps logged for maintenance support
- EMI and ESD robustness considered in design assumptions and lab validation

Note for Later ARINC 429 Sessions

- ARINC 429 is differential and unidirectional, not a discrete
- In real systems you still manage enables, health discretes, and label-present lines via GPIO
- All discrete concepts here apply directly to companion GPIO lines around the ARINC transceiver

Implementation Guidance

- Prefer SDK board macros rather than hardcoding banks and pins
- Use the MCUXpresso Pins Tool to generate pin mux and pad configuration
- Explicitly include electrical options you require (pulls, hysteresis, drive, slew, open drain)
- Keep generated pin configuration files under configuration management