Camera Pipeline Architecture

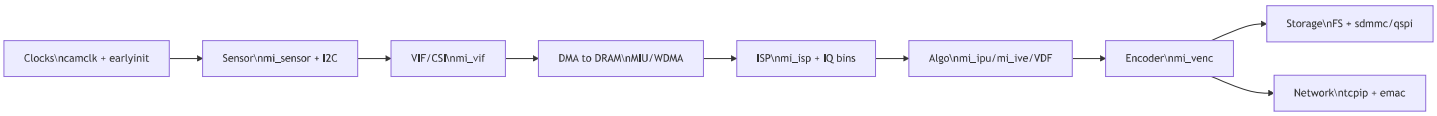
This document summarizes the end-to-end camera pipeline for the `proj` stack: clock setup → sensor init → DMA → ISP/Algo → encode → storage/network.

Overview

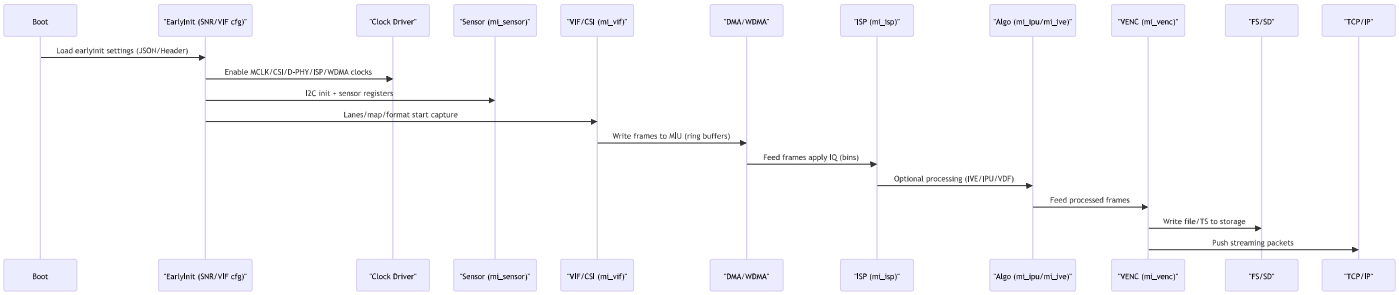
- Control path: Early-init parses SNR/VIF/APP config → enables clocks/padmux → programs sensor over I2C → creates VIF/ISP/Algo/VENC graphs.
- Data path: Sensor → VIF/CSI → DMA to DRAM → ISP (IQ) → optional Algo (IPU/IVE/VDF) → Encoder → File system or Network.
- Product/IP: MI libraries (mi_sensor, mi_vif, mi_isp, mi_ipu, mi_ive, mi_venc, mi_sys, etc.) provided under `libs/product/...`; board clocks and pads live under `hdrs/...` and drivers under `sc/driver/sysdriver`.

Diagrams

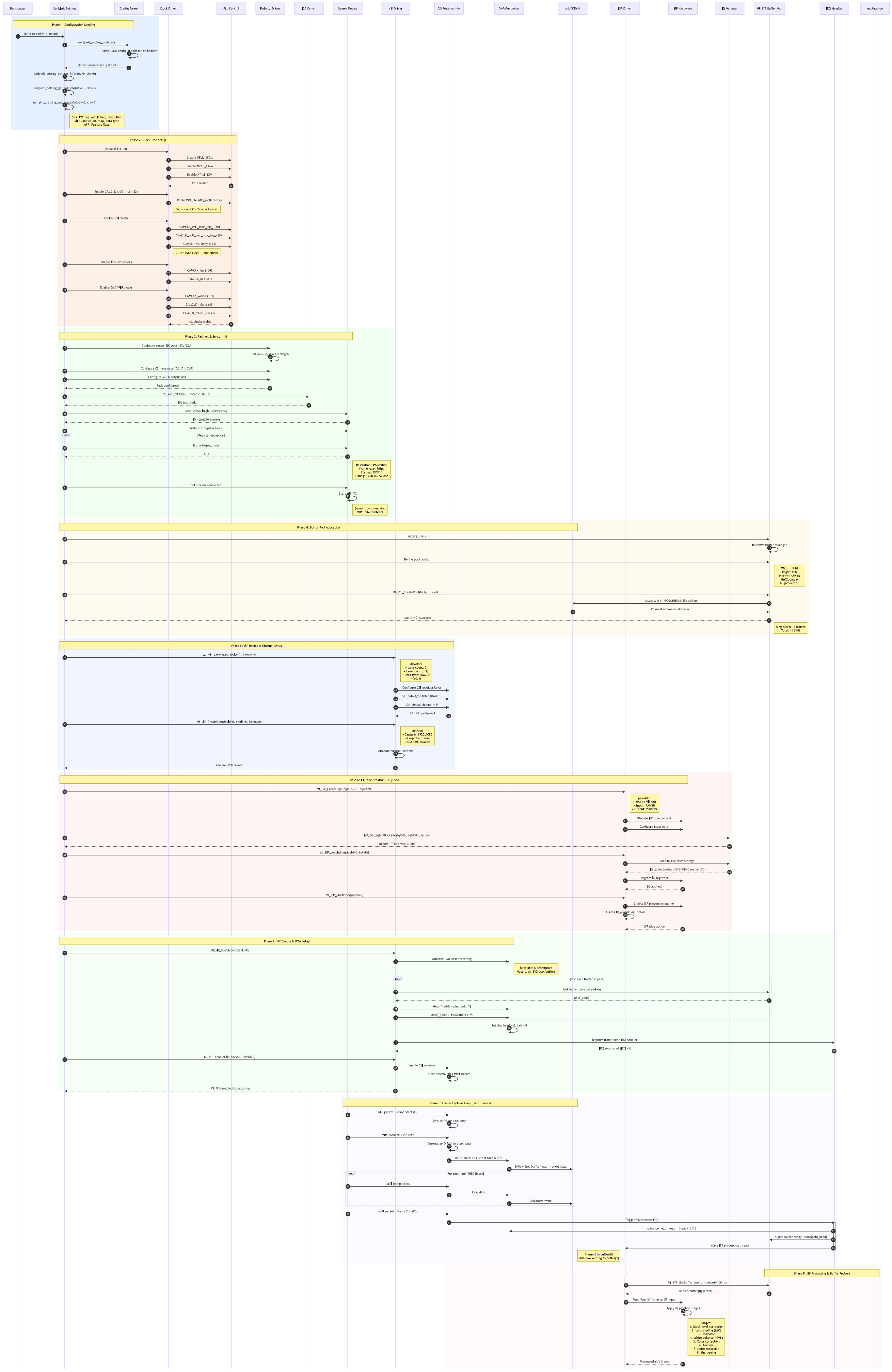
Flow



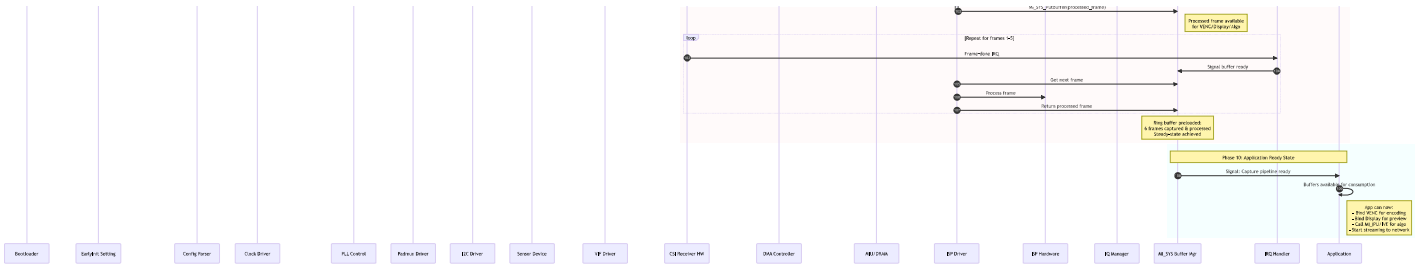
Sequence (High-level)



Detailed Sequence: Earlyinit → Preload Buffer



Bootloader
EarlyInit Setting
Config Parser
Clock Driver
PLL Control
Padmux Driver
I2C Driver
Sensor Device
VIF Driver
CSI Receiver HW
DMA Controller
MIU/DRAM
ISP Driver
ISP Hardware
IQ Manager
MI_SYS Buffer Mgr
IRQ Handler
Application
Phase 1: Configuration Loading
Jump to earlyinit_main()
earlyinit_setting_update()
Parse JSON config or fallback to header
Return parsed config struct
Phase 2: Clock Tree Setup
Phase 3: Padmux & Sensor Init
Phase 4: Buffer Pool Allocation
Phase 5: VIF Device & Channel Setup
Phase 6: ISP Pipe Creation & IQ Load
Phase 7: VIF Enable & DMA Setup
Phase 8: Frame Capture Loop (First Frame)
Phase 9: ISP Processing & Buffer Release



Code Map (key locations)

- Clocks and PLL sources: hdrs/iford_ssc029b_s01a_256_ipc_lh_spinand/camclk.h
  - Examples: `CAMCLK_sr00_mclk` (sensor MCLK), `CAMCLK_csi0_mac_top_i`, `CAMCLK_ssi_dphy`, `CAMCLK_wdma`, `CAMCLK_isp`.
- Early-init configuration: sc/customer/earlyinit_setting/earlyinit_setting/earlyinit_setting.c
  - Accessors: `earlyinit_setting_get_snr_info()`, `earlyinit_setting_get_vif_info()`, `earlyinit_setting_get_app_info()`.
  - Platform scaffolding: sc/customer/earlyinit_setting/platform
- ISP IQ bins: sc/customer/cust_isp/src/Cust_IspBinCfg.c
  - Functions: `ISP_Get_IqBinBlockInfo*`, `ISP_Get_AwbBinBlockInfo*`, `ISP_Get_AlscBinBlockInfo*`, `ISP_Set_ApiBinPath()`.
- Low-level drivers (examples):
  - I2C: sc/driver/sysdriver/i2c
  - EMAC: sc/driver/sysdriver/emac
  - SD/eMMC: sc/driver/sysdriver/sdmmc
  - QSPI/NOR: sc/driver/sysdriver/fsp_qspi
  - Timers/Watchdog/UART/GPIO/etc.: sc/driver/sysdriver
- Middleware and Algo APIs: sc/middleware/algo/pub
  - Examples: `mi_ive.h`, `mi_vg.h`, `mi_md.h`, `ifs_det_api.h`.
- Product MI libraries (linked at build): libs/product/iford_ssc029b_s01a_256_ipc_lh_spinand
  - Notable libs: `mi_sensor.lib`, `mi_vif.lib`, `mi_isp.lib`, `mi_ipu.lib`, `mi_ive.lib`, `mi_venc.lib`, `mi_sys.lib`.

Typical Bring-up Order

1. Enable clocks and padmux for sensor MCLK, CSI/D-PHY, ISP, WDMA per board (see `camclk.h`).
2. Initialize MI_SYS and allocate buffers for VIF/ISP/VENC rings.
3. Create/config `mi_sensor` (bus, mclk, resolution, pixel format) using early-init SNR info.
4. Create/config `mi_vif` (CSI lanes/map, data type, virtual channel) using early-init VIF info.
5. Create/config `mi_isp` (pipe, IQ bin paths); start ISP.
6. Optional: Create `mi_ipu` / `mi_ive` tasks or graphs (MD/OD/IVE operations).
7. Create/config `mi_venc` (codec, profile, bitrate, GOP); bind upstream outputs.
8. For storage: mount FS and write bitstream; for network: bring up TCP/IP and start streaming.

Runtime Data Flow Notes

- VIF writes frames to MIU via DMA (ring buffers). Cache and alignment must match MI_SYS pool settings.
- ISP consumes frames, applies IQ (AWB/ALSC/NR/etc.). IQ bin paths provided by `Cust_IspBinCfg.c` helpers.
- Algo stages (IPU/IVE/VDF) operate on ISP outputs or downscaled taps; ensure buffer formats match API requirements.
- Encoder consumes frames from ISP/Algo; outputs are written to file (FS/SD/QSPI) or streamed over TCP/IP (EMAC).

Configuration Sources

- Early-init JSON/Header: parsed by early-init to populate SNR/VIF/APP sections.
- Kconfig/defconfig: feature selection and library inclusion.
- Board-specific headers and linker scripts: clocks, memory, and sections under `hdrs/...` and `scatter/`.

Troubleshooting Tips

- If sensor does not stream: verify MCLK from `camclk.h`, CSI/D-PHY clocks, and I2C register sequence.
- If frames drop: check MI_SYS pool sizes/alignments and WDMA/MIU bandwidth clocks.
- If ISP effects do not apply: confirm IQ bin paths via `ISP_Get_*BinBlockInfo*` and file presence.
- If storage/network stalls: validate FS mount, SD/QSPI drivers, and EMAC link/IP stack readiness.

Getting Started: Capture + Encode (Pseudo-code)

This is API-shaped pseudo-code to illustrate the minimal bring-up path. Replace names with the exact MI APIs available in your product libraries under `libs/product/iford_ssc029b_s01a_256_ipc_lh_spinand`.

```
// 1) Board/clock + system init (earlyinit + MI_SYS)
enable_board_clocks(/* MCLK, CSI/D-PHY, ISP, WDMA per camclk.h */);
padmux_config_for_sensor_and_vif();

MI_SYS_Init();
MI_SYS_BufPoolConfig poolCfg = {/* width/height/pixel format/pool size */};
MI_SYS_CreatePool(&poolCfg, &poolId);

// 2) IQ bins (optional but recommended for real image)
char iqBin[256], iqBak[256]; int iqSize = 0;
ISP_Get_IqBinBlockInfo(iqBin, iqBak, &iqSize);     // from sc/customer/cust_isp
// provide to ISP later when creating the pipe

// 3) Sensor
MI_SENSOR_DevAttr_t snrAttr = {/* bus, mclk, resolution, fmt */};
MI_SENSOR_CreateDev(0, &snrAttr);
MI_SENSOR_Enable(0);

// 4) VIF (CSI front-end)
MI_VIF_DevAttr_t   vifDevAttr = {/* lane num/map, data type */};
MI_VIF_ChnAttr_t   vifChnAttr = {/* size, pixfmt, crop */};
MI_VIF_CreateDev(0, &vifDevAttr);
MI_VIF_CreateChn(0, 0, &vifChnAttr);
MI_VIF_EnableDev(0);
MI_VIF_EnableChn(0, 0);

// 5) ISP (pipe creation + IQ)
MI_ISP_PipeAttr_t ispAttr = {/* bind VIF 0:0, output size/fmt */};
MI_ISP_CreatePipe(0, &ispAttr);
MI_ISP_LoadIqBin(0, iqBin);        // uses path from Cust_IspBinCfg.c
MI_ISP_StartPipe(0);

// (Optional) 6) Algo (IVE/IPU)
// MI_IPU/MI_IVE create tasks/graphs to process ISP output

// 7) Encoder (H.264/H.265/JPEG)
MI_VENC_ChnAttr_t vencAttr = {/* codec, bitrate, gop, profile */};
MI_VENC_CreateChn(0, &vencAttr);

// 8) Bind graph: ISP -> (Algo) -> VENC
MI_SYS_Bind_t b0 = { .eSrcMod = E_MI_MODULE_ID_ISP, .u32SrcDevId = 0, .u32SrcChnId = 0,
           .eDstMod = E_MI_MODULE_ID_VENC, .u32DstDevId = 0, .u32DstChnId = 0 };
MI_SYS_Bind(&b0);

// 9) Run: fetch bitstream and write to FS or socket
MI_VENC_StartRecvPic(0);
```

```
while (running) {
  MI_VENC_Stream_t stream; MI_S32 timeoutMs = 1000;
  if (MI_VENC_GetStream(0, &stream, timeoutMs) == MI_SUCCESS) {
    write_to_file_or_network(stream.pPack, stream.u32PackCount);
    MI_VENC_ReleaseStream(0, &stream);
  }
}

// 10) Tear down (reverse order)
MI_VENC_StopRecvPic(0); MI_VENC_DestroyChn(0);
MI_SYS_UnBind(&b0);
MI_ISP_StopPipe(0); MI_ISP_DestroyPipe(0);
MI_VIF_DisableChn(0,0); MI_VIF_DisableDev(0);
MI_VIF_DestroyChn(0,0); MI_VIF_DestroyDev(0);
MI_SENSOR_Disable(0); MI_SENSOR_DestroyDev(0);
MI_SYS_DestroyPool(poolId); MI_SYS_Exit();
```

Notes

- Early-init sources for SNR/VIF/APP live under: sc/customer/earlyinit_setting.
- IQ bin helpers live under: sc/customer/cust_isp/src/Cust_IspBinCfg.c.
- Clocks/IDs to cross-check in: hdrs/.../camclk.h.

Minimal CLI Command Handler (Pseudo-code)

Provides simple commands to apply IQ bins and control streaming.

```
typedef void (*cli_fn)(int argc, char **argv);
typedef struct { const char *name; const char *help; cli_fn fn; } cli_cmd_t;

// Forward decls from previous sections
void StartIqTask(void);
void PostIqChange(const char *path, uint32_t mask);
void StartStreamer(const char *ip);
void StopStreamer(void); // implement to signal streamer task to exit and close socket

static void cmd_iq(int argc, char **argv)
{
  if (argc < 2) {
    printf("usage: iq apply <path> [mask]\n");
    return;
  }
  if (strcmp(argv[1], "apply") == 0 && argc >= 3) {
    const char *path = argv[2];
    uint32_t mask = (argc >= 4) ? strtoul(argv[3], NULL, 0) : 0xFFFFFFFFu;
    PostIqChange(path, mask);
    printf("IQ apply queued: %s mask=0x%08X\n", path, mask);
  } else {
    printf("unknown iq subcmd\n");
  }
}

static void cmd_stream(int argc, char **argv)
{
  if (argc < 2) {
    printf("usage: stream <start|stop> [ip]\n");
    return;
  }
  if (strcmp(argv[1], "start") == 0) {
    const char *ip = (argc >= 3) ? argv[2] : "192.168.1.100";
    StartStreamer(ip);
    printf("stream start to %s\n", ip);
  } else if (strcmp(argv[1], "stop") == 0) {
    StopStreamer();
    printf("stream stop\n");
  } else {
    printf("unknown stream subcmd\n");
  }
}

static void cmd_help(int argc, char **argv);

static cli_cmd_t g_cmds[] = {
```

```
  {"iq",     "iq apply <path> [mask]", cmd_iq},
  {"stream", "stream <start|stop> [ip]", cmd_stream},
  {"help",   "show help", cmd_help},
};

static void cmd_help(int argc, char **argv)
{
  (void)argc; (void)argv;
  for (unsigned i = 0; i < sizeof(g_cmds)/sizeof(g_cmds[0]); ++i)
    printf("%s - %s\n", g_cmds[i].name, g_cmds[i].help);
}

void cli_dispatch(const char *line)
{
  // naive split
  char *argv[8]; int argc = 0; char buf[128];
  strncpy(buf, line, sizeof(buf)-1); buf[sizeof(buf)-1] = '\0';
  char *tok = strtok(buf, " \t\r\n");
  while (tok && argc < 8) { argv[argc++] = tok; tok = strtok(NULL, " \t\r\n"); }
  if (argc == 0) return;
  for (unsigned i = 0; i < sizeof(g_cmds)/sizeof(g_cmds[0]); ++i) {
    if (strcmp(argv[0], g_cmds[i].name) == 0) { g_cmds[i].fn(argc, argv); return; }
  }
  printf("unknown cmd, try 'help'\n");
}

void cli_init_example(void)
{
  // Ensure background tasks are ready
  StartIqTask();
  // Optionally: create a UART/console RX task that calls cli_dispatch() per received line
}
```

Example CLI lines

- `iq apply /firmware/isp/iq.bin 0x0000FFFF`
- `stream start 192.168.1.50`
- `stream stop`

Console CLI Task (UART RX, FreeRTOS-style Pseudo-code)

Wire the CLI to a serial console; adapt `uart_*` calls to your driver in sc/driver/sysdriver/uart.

```c
#define CLI_LINE_MAX 128

static void vUartCliTask(void *arg)
{
  (void)arg;
  char line[CLI_LINE_MAX];
  size_t len = 0;

  // init UART here if not already
  // uart_init(115200);

  for (;;) {
    int ch = uart_getchar_blocking(); // replace with your UART API
    if (ch < 0) { vTaskDelay(pdMS_TO_TICKS(1)); continue; }

    if (ch == '\r' || ch == '\n') {
      if (len > 0) {
        line[len] = '\0';
        cli_dispatch(line);
        len = 0;
      }
      continue;
    }

    if (len + 1 < sizeof(line)) {
      line[len++] = (char)ch;
    } else {
      // overflow: terminate and dispatch, then reset
      line[len] = '\0';
      cli_dispatch(line);
      len = 0;
    }
  }
}

void StartCliUart(void)
{
  xTaskCreate(vUartCliTask, "cli-uart", 2048, NULL, tskIDLE_PRIORITY+1, NULL);
}
```

Notes

- Replace `uart_getchar_blocking()` with your HAL (e.g., ring-buffer RX + semaphore). For non-blocking drivers, add a small delay inside the loop.

- If you have a POSIX-like console, you can substitute with `read(0, ...)` and accumulate until `\n`.
- Ensure CLI and IQ/streamer tasks have appropriate priorities to avoid starving capture/encode.

RTOS Task Templates (FreeRTOS-style Pseudo-code)

These examples assume FreeRTOS (present as `freertos.lib`) or an OS wrapper compatible with it. Adapt API names if you use `cam_os_wrapper` primitives.

Periodic IQ Update Task

```c
// Periodically push updated IQ parameters to ISP (e.g., from file/flash or remote cmd)
typedef struct {
  char iqPath[256];
  uint32_t applyMask; // which modules to reapply
} IQUpdateMsg;

static QueueHandle_t gIqQ;

static void vIqUpdateTask(void *arg)
{
  const TickType_t period = pdMS_TO_TICKS(2000);
  IQUpdateMsg msg;
  (void)arg;

  for (;;) {
    if (xQueueReceive(gIqQ, &msg, 0) == pdPASS) {
      // On-demand update (e.g., from CLI/network)
      MI_ISP_LoadIqBin(0, msg.iqPath);
      MI_ISP_Reapply(0, msg.applyMask);
    } else {
      // Periodic refresh (optional)
      char iqBin[256], iqBak[256]; int iqSize = 0;
      ISP_Get_IqBinBlockInfo(iqBin, iqBak, &iqSize);
      MI_ISP_LoadIqBin(0, iqBin);
    }
    vTaskDelay(period);
  }
}

void StartIqTask(void)
{
  gIqQ = xQueueCreate(4, sizeof(IQUpdateMsg));
  configASSERT(gIqQ);
  xTaskCreate(vIqUpdateTask, "iq-upd", 2048, NULL, tskIDLE_PRIORITY+2, NULL);
}

// Example of posting an on-demand IQ change from another context
void PostIqChange(const char *path, uint32_t mask)
{
  IQUpdateMsg m = {0};
  strncpy(m.iqPath, path, sizeof(m.iqPath)-1);
  m.applyMask = mask;
```

```
  xQueueSend(gIqQ, &m, 0);
}
```

Network Streamer with Retransmit/Backoff

```
// Pulls encoded frames and sends over a socket with simple backoff on failures
static int gSock = -1;

static bool net_connect(const char *ip, uint16_t port)
{
  // Use your tcpip stack here to open and connect socket
  // return true on success
}

static void vStreamerTask(void *arg)
{
  const char *ip = (const char *)arg;
  uint16_t port = 8554; // example
  TickType_t backoff = pdMS_TO_TICKS(200);
  const TickType_t backoffMax = pdMS_TO_TICKS(5000);

  for (;;) {
    if (gSock < 0) {
      if (net_connect(ip, port)) {
        backoff = pdMS_TO_TICKS(200);
      } else {
        vTaskDelay(backoff);
        backoff = (backoff < backoffMax) ? backoff * 2 : backoffMax;
        continue;
      }
    }

    // fetch encoded bitstream
    MI_VENC_Stream_t stream; MI_S32 timeoutMs = 100;
    if (MI_VENC_GetStream(0, &stream, timeoutMs) == MI_SUCCESS) {
      bool ok = socket_send_all(gSock, stream.pPack, stream.u32PackCount);
      MI_VENC_ReleaseStream(0, &stream);
      if (!ok) {
        socket_close(gSock); gSock = -1; // force reconnect
      }
    } else {
      // no frame; yield briefly
      vTaskDelay(pdMS_TO_TICKS(5));
    }
  }
}

void StartStreamer(const char *ip)
{
```

```
  xTaskCreate(vStreamerTask, "net-strm", 4096, (void*)ip, tskIDLE_PRIORITY+3, NULL);
}
```

Notes

- Tune task priorities and stack sizes to your system; ensure MI_SYS pool sizes support sustained throughput.
- For OS wrappers, map `xTaskCreate` / `vTaskDelay` / `QueueHandle_t` to your RTOS abstraction (e.g., `cam_os_wrapper`).
- Replace `socket_*` helpers with your TCP/IP stack's APIs; consider zero-copy send if available.