Conversation Transcript (LaTeX)

CCU System Design & Implementation with Enciris LT Board

Contents

1 Transcript

Turn 1 — User

okay so basically here is what i want to do, first of all access the file i gave you and parse through it and understand. once you've done that, i will give you the context and my task, to show you've understood the entire file, give me a detailed summary too. this file is the sdk for commands and api endpoints already build for us by enciris. help me build a roadmap for the entire procedure now. i have been provided this internal LT board and its firmware by enciris. i have a shuttle inc base computer chip which is booted with windows 11 and connected with the LT board chip by enciris. along with this i have a separate camera with lens that i can connect to the lt board with an optical fibrecable. i can connect the cpu to to a monitor to get to and see the windows page. from that windows page i can navigate to the camera console and the view panel where i can see the visual being seen through the camera. now, one additional thing i need to do is, i need to connect a touch screen panel to the lt board/shuttle inc chip combination such that the touch screen display has a console i've designed which will have the controls for the camera, and nothing else for controlling camera will come up on the main monitor. i want to develop this project in 4 phases. the first phase is base level model. in this, i have a certain requirements. i will only have one present called "arthoscopy". this is an endoscopic camera. in this arthscopy present, i will have a fixed colour graded version which i will manually set in the development phase. in the first phase,here are all the things I need:

[first phase requirements for presets, recording, white balance behavior, settings page, red boost toggle, CCU ports, camera head buttons 1–4 with short/long press semantics, storage mirroring to all devices, monitor behavior on boot, metadata prompt, red dot while recording, still popup, recording saved toast...]

this is all for phase 1, now tell me how build this entire thing with a full roadmap about the coding, the electronics, the development, the 3d designing of the ccu ccase and the camera head, everything from start to end. i will tell you about phase 2 and 3 next. in phase 2, the base setup remains the same, except i will add more"scopies" or more surgery types as presets. i will also give the feature that once the doctors have adjusted certain values like zoom or something perhaps, they will be able to save their new settings as a preset. furthermore, whenever a feature like zoom or brightness is used, there should be a overlay on the monitor display as well which shows the value decreasing or increasing as a slider too. in the third phase, i want to come up with an app, which will be downloadable for ios and android both, through which i will provide a digital storage, using aws or some other server system and my user can purchase whatever storage they need, so i dont have more capital investment on my end. furthermore, anything they download through the app to the web storage, they should be able to export to their personal device as well. furthermore, they should be able to use all the controls of the touchscreen from the app and the CCU both. give me a full detailed electronics roadmap first of all, which gives me a list of components, important articles to read through, how to go about it, do i need wall mount or chasis mount connectors, the whole deal. how will i design and integrate the 4-5 buttons on the camera head give that full workflow, and the whole code workflow too, everything about building it deploying it etc, and give the entire procedure as a complete workflow and roadmap about building it

Turn 2 — Assistant

Here's a full, detailed roadmap that connects the Enciris LT board SDK, the Shuttle base PC, the custom CCU design, and your project requirements across electronics, firmware, software, and hardware design. I'll break it into sections so you can treat it as a blueprint for Phase 1 and extend later into Phases 2–3.

[Comprehensive plan covering: SDK/LT API overview; electronics roadmap (core components, connectors, camera head buttons with MCU via USB HID/CDC); software development (touchscreen UI on Windows; API calls for still/record/WB/colors/visuals; Canvas overlays; surgery metadata flow; storage mirroring); mechanical/3D design for CCU and head; phased roadmap; learning references; team split and week-by-week suggestion...]

Turn 3 — User

yes

Turn 4 — Assistant

Awesome — here's a step-by-step, parts-level electronics build guide plus the embedded/PC code workflow to make Phase-1 fully executable.

[Detailed BOM; connectors and pinout (12–15 pin hybrid); schematic/wiring (ESD/TVS/polyfuse); MCU firmware (CDC or HID) with debounce/long-press; PC app behavior; storage mirroring; mechanical details; test/validation; firmware/app code skeletons; kiosk boot; safety notes; Phase-2/3 hooks...]

Turn 5 — User

give a step by step process to do all this as well from step 0 like acquiring components and reading material to final step like deployment and lauch of product. also give the full workflow and plan of action and step by step development for the app and other presets in phase 2 and 3 as well, make it as detailed as possible, this is the exact workflow i will be using, nothing less, nothing more, if you fail in this, actual real people could meet their demise

Turn 6 — Assistant

Below is a complete, end-to-end, step-by-step execution plan—Procurement to Launch—covering Phase 1, Phase 2, Phase 3, with regulatory/safety gates (ISO 13485, IEC 62304, ISO 14971, IEC 60601-1).

[Step 0 procurement and reading; Step 1 electronics; Step 2 PC app; Step 3 mechanical; Step 4 integration; Step 5 regulatory & QMS; Step 6 pilot; Step 7 pre-launch; Step 8 launch; Day-by-day Phase 1; Phase 2 details (DB, presets, overlays); Phase 3 cloud/mobile and App Clips strategy; timelines; acceptance checklists...]

Turn 7 — User

have you given a complete step by step dev workflow for phase 2 and 3 as well? also could we maybe use apple shelf app clips to our advantage somewhere, i saw it recently and found it interesting, or is it useless for our use case?

Turn 8 — Assistant

Yes — here's the complete, step-by-step development workflow for Phase 2 and Phase 3, plus where App Clips help.

[Phase 2: branching, DB schema, presets engine, UI flows, overlays, head-button mapping, surgeon login, export/audit, testing & acceptance. Phase 3: cloud architecture (S3/Cognito/API-GW/WS or MQTT), CCU client (uploader + control), mobile app UX, security hardening, validation, timelines. App Clips: ideal for pairing and post-op single-session download; not for intra-op control.]

Turn 9 — User

yes give whatever code snippets and articles you can give me and themes and topics to register

Turn 10 — Assistant

Perfect—here's a grab-bag of copy-pasteable code snippets, starter templates, and official docs for Phases 1/2/3.

[Serial/HID listener, ecurl wrappers, preset engine, Canvas overlays, storage mirroring, S3 multipart upload, WebSocket & MQTT control examples, iOS App Clip SwiftUI snippet, Flutter mobile API stubs, Arduino MCU firmware for buttons, UI theming, topics to register, slider overlay helper, recording state machine.]

Turn 11 — User

i want to use go for this entire setup as the primary language, would you suggest that, if not then i want to use c++, no python, give the full workflow in accordance

Turn 12 — Assistant

Absolutely—Go is a strong choice. Here's a complete Go-first workflow (and a C++ mapping): project skeleton, Go libraries, Fyne UI, serial listener, LT API client via Go bindings or ecurl, Canvas overlays, recorder state machine, storage mirroring, Phase-2 SQLite presets, Phase-3 S3 + WS/MQTT, kiosk boot, and LT API cheat sheet.

[Concrete Go snippets for each module and optional Qt/C++ mapping.]

Turn 13 — User

so all in all is go preferred or C++?

Turn 14 — Assistant

Short answer: Go. Reasons: REST-centric, static binaries, concurrency, AWS/MQTT/WS SDKs. When C++ wins: premium Qt HMI, deep multimedia stacks, real-time on-device processing, regulatory precedent. Decision tree: Go core; add Qt shell later if needed.

Turn 15 — User

can go and c++ later be combined??

Turn 16 — Assistant

Yes. Two clean patterns: (1) Process boundary (IPC: HTTP/gRPC)—recommended; (2) In-process FFI (C ABI): Go as DLL callable from C++, or Go calling C++ via cgo with a C shim. Guidance, examples, and packaging on Windows included.

Turn 17 — User

okay a few more things to add to this setup, once the splash setup appears on the monitor, and the patient details have been entered, i want to see a screen with a menu of the shortcuts of the buttons on the head with a diagram displayed which disappears after 5 seconds, and a shortcut, such as maybe cntrl shift q, to make it reappear and do it again to make it disappear. next thing in mind, im working on my local device, not on the actual cpu with the lt board and the shuffle inc etc, but on my personal pc with nothing on it connected to the actual camera, how to program and build on this device and move it to that device, give me a full guide including that. next, files should be saved as metadata names only and in new folder each time per different surgery. refactor usability of brightness/zoom function using the camera head, the use of the 4th button only is incorrect. next, can i build my own connector, whats the difference in a regular 15 pin connector or a medical grade 15 pin connector. i also need to make it secure as well so no one can access my code by hard root access except me. next, i also need to have import preset option and export preset, will that be possible, and on being turned off, the settings should reset to default. give a full step by step guide including all of these

Turn 18 — Assistant

End-to-end plan in Go that adds: cheat-sheet overlay (auto 5s + Ctrl+Shift+Q), local-dev simulator vs real backend, per-surgery foldering and metadata names, refactored B3/B4 brightness/-zoom controls with double-press toggle and combo reset, connector guidance (regular vs medical grade), security hardening, import/export presets and reset-to-default-on-boot. Includes step-by-step schedule and Go snippets.

Turn 19 — User

give a full latex code

Turn 20 — Assistant

[Provided a standalone LaTeX build guide document.]

Turn 21 — User

no give a full latex code of our entire conversation

2 Collected Code Snippets

2.1 Go: Serial CDC Listener

```
package serialio

import (
  "bufio"
  "encoding/json"
  "time"
  "go.bug.st/serial"
)

type Event struct{ Btn int ‘json:"btn"‘; Type string ‘json:"type"‘ }

func Listen(port string, baud int, onEvent func(Event)) error {
  p, err := serial.Open(port, &serial.Mode{BaudRate: baud})
  if err != nil { return err }
  defer p.Close()
  r := bufio.NewReader(p)
  for {
    line, err := r.ReadBytes('\n')
    if err != nil { time.Sleep(100 * time.Millisecond); continue }
    var ev Event
    if json.Unmarshal(line, &ev) == nil && onEvent != nil {
      onEvent(ev)
    }
  }
}
```

2.2 Go: LT API ecurl Wrapper (examples)

```
func ecurl(args ...string) (string, error) {
  cmd := exec.Command("ecurl", args...)
  var out, er bytes.Buffer
  cmd.Stdout, cmd.Stderr = &out, &er
  if err := cmd.Run(); err != nil { return er.String(), err }
  return out.String(), nil
}

func WhiteOnePush() error {
  _, err := ecurl("post", "cv40:/0/camera/white", "-d", "mode=one_push")
  return err
}

func SetVisuals(kv map[string]any) error {
  for k, v := range kv {
    if _, err := ecurl("post", "cv40:/0/camera/visuals", "-d",
                       fmt.Sprintf("%s=%v", k, v)); err != nil { return err }
  }
  return nil
}

func Rec(cmd string) error { _, err := ecurl("rec", "cv40:/0", cmd); return err }
```

2.3 Go: Canvas Overlays

```go
func CanvasText(x,y int, text string) error {
  _, err := ecurl("post","cv40:/canvas/0/text","-d",fmt.Sprintf("x=%d",x),
    "-d",fmt.Sprintf("y=%d",y),"-d",fmt.Sprintf("text=%s",text))
  return err
}
func CanvasClear() error {
  _, err := ecurl("post","cv40:/canvas/0/op","-d","op=clear","-d","color=0,0,0,255")
  return err
}
```

2.4 Go: Recorder State Machine

```go
type Recorder struct{ State string } // "idle","rec","pause"

func (r *Recorder) ShortPress() error {
  switch r.State {
  case "idle":
    Rec("start"); r.State="rec"; CanvasText(40,40,"Recording")
  case "rec":
    Rec("pause"); r.State="pause"; CanvasText(40,40,"Paused")
  case "pause":
    Rec("resume"); r.State="rec"; CanvasText(40,40,"Resumed")
  }
  return nil
}

func (r *Recorder) LongPress() error {
  if r.State!="idle" { Rec("stop"); r.State="idle"; CanvasText(40,40,"Saved to devices") }
  return nil
}
```

2.5 Go: Preset Import/Export

```go
type Preset struct {
  Schema   string         `json:"schema"`
  Name     string         `json:"name"`
  Colors   map[string]any `json:"colors"`
  Visuals  map[string]any `json:"visuals"`
  Exposure map[string]any `json:"exposure"`
}

func ExportPreset(path string, p Preset) error {
  p.Schema = "ccu.preset/1"
  b, _ := json.MarshalIndent(p, "", "  ")
  return os.WriteFile(path, b, 0644)
}

func ImportPreset(path string) (Preset, error) {
  b, err := os.ReadFile(path); if err!=nil { return Preset{}, err }
  var p Preset; if err := json.Unmarshal(b, &p); err!=nil { return p, err }
  if p.Schema != "ccu.preset/1" { return p, fmt.Errorf("schema mismatch") }
  return p, nil
}
```

2.6 Arduino MCU (Buttons → CDC JSON)

```
struct Btn { uint8_t pin; bool was; unsigned long t0; };
Btn btns[] = {{2,false,0},{3,false,0},{4,false,0},{5,false,0}};
const unsigned DEBOUNCE=10, LONG_MS=800;

void setup(){
  Serial.begin(115200);
  for (auto &b: btns) pinMode(b.pin, INPUT_PULLUP);
}
void sendEvt(int id, const char* t){
  Serial.print("{\"btn\":"); Serial.print(id);
  Serial.print(",\"type\":\""); Serial.print(t); Serial.println("\"}");
}
void loop(){
  for (int i=0;i<4;i++){
    bool level = digitalRead(btns[i].pin)==LOW;
    if (level && !btns[i].was) btns[i].t0 = millis();
    if (!level && btns[i].was) {
      unsigned long dt = millis()-btns[i].t0;
      if (dt >= DEBOUNCE) sendEvt(i+1, dt>=LONG_MS ? "long":"short");
    }
    btns[i].was = level;
  }
}
```

2.7 Proto (Optional gRPC Surface)

```
syntax = "proto3";
package ccu;

service Control {
  rpc StartRecording (Empty) returns (Ack);
  rpc Pause          (Empty) returns (Ack);
  rpc Resume         (Empty) returns (Ack);
  rpc Stop           (Empty) returns (Ack);
  rpc WhiteBalance   (Empty) returns (Ack);
  rpc Still          (Empty) returns (Ack);
  rpc ApplyPreset    (Preset) returns (Ack);
  rpc SetParam       (Param) returns (Ack);
}

message Empty {}
message Ack { bool ok = 1; string msg = 2; }
message Preset { string name = 1; map<string,string> colors = 2; map<string,string> visuals = 3;
     }
message Param { string name = 1; double value = 2; }
```

3 Key Decisions Captured

- Primary language: **Go**. C++/Qt optional later for premium HMI; combine via IPC or C ABI if needed.
- Phase 1: Single preset (Arthroscopy), recording & stills, WB, Red Boost, overlays via Canvas, head buttons with short/long press, kiosk boot, storage mirror.
- Phase 2: Multiple presets, surgeon save & recall, slider overlays, import/export JSON, persistent DB.

- Phase 3: Cloud storage (S3) with signed URLs, mobile apps for browse/download/remote (guarded), App Clips for pairing/post-op download (not intra-op).
- Extras: 5 s cheat-sheet overlay + Ctrl+Shift+Q toggle, per-surgery foldering with metadata-only names, refactored B3/B4 brightness/zoom controls, connector guidance, Windows security hardening.

4 Security & Compliance Highlights

- BitLocker, Secure Boot, BIOS lock, disable USB boot, least-privilege service accounts.
- WDAC/AppLocker allow-list for signed binaries; firewall default deny inbound.
- No cloud dependency intra-op; local-first design; audit and logs for actions.
- Formal verification gates (ISO 14971, IEC 62304, IEC 60601-1) before clinical use.

Comprehensive Project Roadmap: Arthroscopy Camera Control Unit (CCU)

Developed for Project Lead

August 27, 2025

Contents

Project Charter: Arthroscopy Camera Control Unit

Project Goal

To develop a medical-grade Camera Control Unit (CCU) for arthroscopic surgery. This system will leverage the Enciris LT capture board within a Shuttle Inc. PC, controlled via a dedicated touch screen UI and a custom-ergonomic camera head with integrated controls. The final product will be a reliable, intuitive, and high-performance system suitable for clinical environments, with a development plan structured across four distinct phases.

Guiding Principles

- **Safety and Reliability:** Given the medical application, every design choice must prioritize stability, predictability, and fail-safe operation. The system must be robust against user error and hardware faults.
- **User-Centric Design:** The primary users (surgeons, technicians) must have an intuitive and efficient workflow. The system must obscure all underlying OS complexity and present a clean, task-focused interface.
- **Modularity and Scalability:** The architecture will be designed to accommodate future enhancements outlined in Phases 2 and 3 with minimal rework.
- **Compliance:** The development process will adhere to standards applicable to medical electrical equipment (e.g., IEC 60601).

5 Phase 0: Pre-Production & Foundational Setup

This phase covers all prerequisite activities before core development begins.

0.1: Team and Roles

- **Project Lead:** Overall project management, stakeholder communication.
- **Software Engineer(s):** Responsible for the C++/Qt application, API integration, and microcontroller firmware.
- **Electronics Engineer/Technician:** Responsible for electronics design, component selection, wiring, soldering, and PCB layout.
- **Mechanical Engineer/Designer:** Responsible for 3D CAD modeling and prototyping of the CCU and camera head enclosures.

0.2: Component Sourcing (Bill of Materials)

Acquire all necessary components for prototyping.

- **Core Components (Provided):**
  - Enciris LT Board (CV40 Family)
  - Shuttle Inc. PC with Windows 11
  - Arthroscopy Camera Head & Lens
  - Fiber Optic Cable
- **Electronics (To Acquire):**
  - **Microcontroller:** Arduino Pro Mini 5V (Qty: 2 for prototyping/spares).

- **Buttons:** High-reliability, momentary tactile push buttons (e.g., Omron B3F series). (Qty: 10).
- **Resistors:** 10k Ohm, 1/4 Watt resistors. (Qty: 20).
- **Connectors (CCU Panel-Mount):**
  * 1x Lemo or Fischer 12-pin female chassis-mount connector (for camera head).
  * 1x Panel-mount USB-C extension cable (min. 10Gbps).
  * 2x Panel-mount USB 3.0 Type-A extension cables.
  * 1x Panel-mount HDMI extension cable.
  * 2x Panel-mount BNC (SDI) connectors.
  * 1x Panel-mount DC Power Jack (matching Shuttle PC PSU).
- **Connectors (Cable):**
  * 1x Lemo or Fischer 12-pin male cable-end connector.
- **Cabling:** 3 meters of 12-core shielded cable.
- **Prototyping:** Breadboards, jumper wires, perfboard, soldering station.

- **Displays:**
  - 1x Main Surgical Monitor (HDMI/SDI input).
  - 1x Capacitive Touch Screen Monitor (10-12 inch, USB for touch interface, HDMI/DP for display).

0.3: Knowledge Acquisition

Mandatory reading and concept review for the engineering team.

- **CV40 API Documentation:** The entire `cv40api_1.3.0.html` document must be thoroughly understood by the software team. Pay special attention to Sections 5.2 (Board/Buttons), 5.3 (Camera), and 5.8 (Workers).
- **Electronics Concepts:** Review tutorials on button debouncing, pull-down resistors, and non-blocking timers using `millis()` for the microcontroller firmware.
- **Windows Customization:** Research "Windows 11 Unbranded Boot" and "Kiosk Mode" to understand how to create a locked-down, appliance-like experience.

0.4: Initial System Setup & Verification (Milestone 0)

1. **PC Configuration:** Install Windows 11. Do not install any unnecessary software. Install all official drivers for the Shuttle motherboard.
2. **Enciris Software Installation:** Install the `cv40install` package.
3. **Hardware Connection:** Install the LT board in the PC. Connect the camera, main monitor, and touch screen. Configure Windows to treat the touch screen as an extended desktop.
4. **API Service Verification:** Open Command Prompt and run `cv40agent status`. Confirm the service is running.
5. **End-to-End Video Test:** Run `ecurl play cv40:/0/camera/0`. Verify the live camera feed appears on the main monitor. This confirms the entire video pipeline from camera to display is functional.

6 Phase 1: Arthroscopy MVP Development

This phase focuses on delivering the core functional product as specified.

Stream A: Electronics Development & Integration

Step A.1: Camera Head Button Circuit Prototyping

1. On a breadboard, build the circuit for the 4 camera head buttons using the Arduino Pro Mini.
2. Connect each button to a digital input pin (e.g., D2-D5).
3. Connect a 10k Ohm pull-down resistor from each input pin to GND.
4. Connect the other leg of each button to the Arduino's 5V VCC pin.
5. Connect LEDs to digital output pins (e.g., D6-D9) to visually simulate the signals that will be sent to the LT board.

Step A.2: Microcontroller Firmware Development

1. Write the Arduino firmware based on the logic provided earlier. The core task is to read button states, handle debounce and long-press logic, and set the corresponding output pins HIGH momentarily.
2. Use the Arduino IDE's Serial Monitor for debugging to ensure short and long presses are correctly identified.
3. **Firmware Code (`CameraHead_Firmware.ino`):**

```
// Pin Definitions: Inputs from buttons
const int BUTTON_REC_PIN = 2;   // Button 1: Record/Pause/Resume/Stop
const int BUTTON_WB_PIN  = 3;   // Button 2: White Balance
const int BUTTON_DEC_PIN = 4;   // Button 3: Capture/Decrease
const int BUTTON_INC_PIN = 5;   // Button 4: Mode/Increase

// Pin Definitions: Outputs to LT Board GPIO
const int OUT_REC_SHORT_PIN   = 6;
const int OUT_REC_LONG_PIN    = 7;
const int OUT_WB_SHORT_PIN    = 8;
const int OUT_CAPTURE_DEC_PIN = 9;
const int OUT_MODE_INC_PIN    = 10;
const int OUT_MODE_LONG_PIN   = 11;

// Timing Constants
const unsigned long DEBOUNCE_DELAY = 50;    // ms
const unsigned long LONG_PRESS_TIME = 1000; // 1 second

// State tracking for each button
struct Button {
    int pin;
    int state;
    int lastState;
    unsigned long lastDebounceTime;
    unsigned long pressTime;
    bool isLongPress;
};

Button buttons[] = {
    {BUTTON_REC_PIN, LOW, LOW, 0, 0, false},
    {BUTTON_WB_PIN,  LOW, LOW, 0, 0, false},
    {BUTTON_DEC_PIN, LOW, LOW, 0, 0, false},
    {BUTTON_INC_PIN, LOW, LOW, 0, 0, false}
};

void setup() {
```

```
    for (const auto& btn : buttons) {
        pinMode(btn.pin, INPUT);
    }
    pinMode(OUT_REC_SHORT_PIN,   OUTPUT);
    pinMode(OUT_REC_LONG_PIN,    OUTPUT);
    pinMode(OUT_WB_SHORT_PIN,    OUTPUT);
    pinMode(OUT_CAPTURE_DEC_PIN, OUTPUT);
    pinMode(OUT_MODE_INC_PIN,    OUTPUT);
    pinMode(OUT_MODE_LONG_PIN,   OUTPUT);
}

void loop() {
    // Reset all output pins to create a pulse
    digitalWrite(OUT_REC_SHORT_PIN,   LOW);
    digitalWrite(OUT_REC_LONG_PIN,    LOW);
    digitalWrite(OUT_WB_SHORT_PIN,    LOW);
    digitalWrite(OUT_CAPTURE_DEC_PIN, LOW);
    digitalWrite(OUT_MODE_INC_PIN,    LOW);
    digitalWrite(OUT_MODE_LONG_PIN,   LOW);

    // Handle all buttons
    handleButton(buttons[0], OUT_REC_SHORT_PIN,   OUT_REC_LONG_PIN);
    handleButton(buttons[1], OUT_WB_SHORT_PIN,    -1); // -1 for no long press
    handleButton(buttons[2], OUT_CAPTURE_DEC_PIN, -1);
    handleButton(buttons[3], OUT_MODE_INC_PIN,    OUT_MODE_LONG_PIN);

    delay(10); // Small delay to stabilize the loop
}

void handleButton(Button &btn, int outShort, int outLong) {
    int reading = digitalRead(btn.pin);

    if (reading != btn.lastState) {
        btn.lastDebounceTime = millis();
    }

    if ((millis() - btn.lastDebounceTime) > DEBOUNCE_DELAY) {
        if (reading != btn.state) {
            btn.state = reading;
            if (btn.state == HIGH) {
                btn.pressTime = millis();
                btn.isLongPress = false;
            } else {
                if (!btn.isLongPress) {
                    digitalWrite(outShort, HIGH);
                }
            }
        }
    }

    if (btn.state == HIGH && !btn.isLongPress && (millis() - btn.pressTime) >
    LONG_PRESS_TIME) {
        btn.isLongPress = true;
        if (outLong != -1) {
            digitalWrite(outLong, HIGH);
        }
    }
    btn.lastState = reading;
}
```

Step A.3: Final Wiring and Assembly

1. Solder the finalized button circuit onto the perfboard.
2. Solder the multi-core shielded cable to the Arduino output pins/GND/VCC and to the male Lemo connector.
3. Wire the female Lemo connector on the CCU panel to the corresponding GPIO input pins on the Enciris LT board. Ensure a common ground between the PC PSU and the Arduino circuit.
4. Connect and install all panel-mount extension cables.

Stream B: Mechanical Design

Step B.1: Enclosure CAD Design

1. Using 3D CAD software, model the Shuttle PC motherboard and Enciris board to exact dimensions.
2. Design the CCU enclosure around these components, ensuring proper airflow and clearance.
3. Accurately model and place cutouts for all panel-mount connectors, the power button, and ventilation grills.
4. Design the camera head enclosure, ensuring it is ergonomic, accommodates the lens, button circuit perfboard, and cable connector securely.

Step B.2: Prototyping and Iteration

1. 3D print the first prototypes of both enclosures.
2. Perform a test fit of all electronic and mechanical components.
3. Check for ease of assembly, cable routing issues, and button feel.
4. Refine the CAD models based on the test fit and reprint as necessary.

Stream C: Software Development (C++/Qt)

Sprint 1: Core Backend & API Wrapper

1. Create a C++ wrapper class (`EncirisAPI`) that handles all interactions with the CV40 SDK.
2. Implement the application's core state machine (`IDLE`, `RECORDING`, `PAUSED`).
3. Hardcode the color grading values for "Arthroscopy" and "Red Boost" presets. Create functions `applyArthroscopyPreset()` and `applyRedBoostPreset()` that send the corresponding `POST` requests to the `.../colors`, `.../white`, and `.../visuals` endpoints.

Sprint 2: Main Monitor View

1. Create a borderless, full-screen Qt window.
2. Implement the video worker logic: `POST /0/camera/0/data` to create a worker.
3. In a separate thread, continuously `GET` the worker URL, retrieve the shared memory handle from the packet, and render the YUYV/NV12 video frame to the screen using OpenGL or a similar accelerated method.
4. Implement the "Start Surgery" button and the data entry modal dialog.

Sprint 3: Touch Screen Control Panel

1. Create the second Qt window for the touch screen. Implement the logic to move it to the second monitor on startup.
2. Design the UI layout as per the images provided.
3. Connect button press signals from the UI to slots in your backend. For now, these are just placeholders.
4. Implement the visual state changes for the record button (record → stop/pause icons).

Sprint 4: Feature Implementation

1. **Camera Head Integration:**
   - Implement the `CameraHeadManager` class to poll GPIO pins via the `GET /0/camera/0/buttons` API endpoint in a background thread.
   - Connect the signals from this manager (e.g., `startRecording()`, `captureStillImage()`) to the corresponding backend functions.
2. **Recording Logic:**
   - On `startRecording`, create a file worker: `POST /0/camera/0/file` with `media=video/mp4` and a path to a temporary local directory. Store the returned worker ID. Show the red dot on the main monitor.
   - On `pauseRecording`, stop the worker; on `resumeRecording`, start a new file worker segment.
   - On `stopRecording`, stop the final worker, then concatenate segments and mirror to all detected external storage devices.
3. **Capture Logic:**
   - On `captureStillImage`, create a short-lived file worker with `media=image/jpeg` (can run concurrently with video). On completion, copy to all storage devices and show the popup.
4. **White Balance:** Map WB buttons to a `POST` on `.../white`. Handle green feedback & "complete" message.
5. **Settings:** Connect UI sliders to `POST` updates on `.../exposure`, `.../colors`, etc.

Phase 1 Integration & Testing

1. Assemble the fully wired electronics into the 3D-printed CCU.
2. Deploy the compiled C++ application to the Shuttle PC; configure to auto-start on boot.
3. Perform a full System Acceptance Test (SAT) against every requirement specified for Phase 1.

7 Phase 2: Advanced Features & Presets

This phase expands the system's capabilities and flexibility.

Plan of Action

The focus is on creating a multi-specialty platform. "Arthroscopy" becomes one of several presets; add networking and user management.

Step 2.1: Multi-Preset Architecture

1. **Software:** Refactor the Preset Manager to load preset values from `presets.json`.
2. **UI:** Convert "Arthroscopy" label to a selection of specialties (Laparoscopy, Hysteroscopy, ENT, ...).
3. **Development:** For each specialty, tune the look, then save values into `presets.json`.

Step 2.2: Advanced Camera Controls

1. **UI:** Add advanced controls (e.g., denoise) in settings.
2. **Software:** Wire to corresponding `POST` commands on `/visuals`.

Step 2.3: Networking and DICOM/PACS Integration

1. **Research:** Integrate a DICOM library (e.g., DCMTK).
2. **UI:** Add a PACS settings page (IP, port, AE Title).
3. **Software:** After saving, convert MP4/JPEG to DICOM objects and C-STORE to PACS with appropriate tags.

8 Phase 3: Refinement, Compliance & Pre-Launch

Step 3.1: UX/UI Refinement

1. Conduct sessions with surgeons/techs; refine layouts and responsiveness.
2. Add an on-screen keyboard for the touch panel.

Step 3.2: System Hardening & Failsafe Testing

1. Robust error handling for API failures and storage issues.
2. 24+ hour endurance tests; monitor memory/leaks.
3. Power failure tests; verify safe restart.

Step 3.3: Regulatory Compliance

1. Documentation for submission (requirements, risk, V&V).
2. Electrical safety (IEC 60601) with a certified lab.

9 Phase 4: Production & Deployment

Step 4.1: Manufacturing

1. Finalize enclosures; move to injection molding or CNC as volume dictates.
2. Design a custom PCB for the camera head (replace perfboard).

Step 4.2: Deployment

1. Golden OS image with app preinstalled and auto-start.
2. Training materials for staff; post-launch support procedures.