UPPAAL Verification

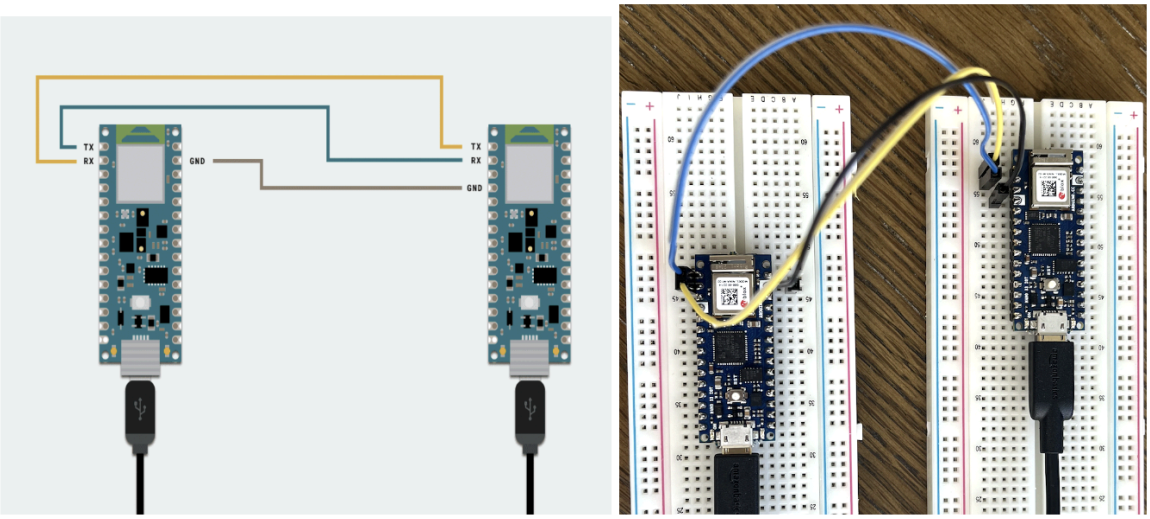
https://www.loom.com/share/48dabdc93f9240f69bb0616c8847d847

Slow Heart Rate Demo: https://www.loom.com/share/a5f25c2a6935400e832769363ad3a0e3

Fast Heart Rate Demo: https://www.loom.com/share/7dde86de33ba4f4e82378845b12a279f

Instruction on how to connect the two Arduino Boards:

To use UART communication between two Arduino boards, connect the TX, RX, and GND pins by crossing TX and RX, and connecting GND to GND between the boards as shown in the figure below:



In addition to the sensing/pacing communication between the two boards, a LED should blink whenever a sensing signal is generated by the heart, and another LED should blink whenever a pacing signal is generated by the pacemaker.

CIS 4410/5410 Project 2 - Milestone 2

System Architecture

Communication Protocol

The two Arduino boards communicate via UART (Serial1) using a simple character-based protocol:

- **'S' (Sense Signal)**: Heart → Pacemaker when heart beats intrinsically
- **'P' (Pace Signal)**: Pacemaker → Heart to trigger a paced beat

Timing Parameters

LRL (Lower Rate Limit): 1500 ms (40 bpm)

URL (Upper Rate Limit): 333 ms (~180 bpm)

VRP (Ventricular Refractory Period): 150 ms

HRI (Hysteresis Rate Interval): 1700 ms (LRL + 200)

minIBI (Minimum Inter-Beat Interval): 400 ms

maxIBI (Maximum Inter-Beat Interval): 2000 ms

UPPAAL Model

The system behavior is formally specified in `Proj_2_M1_V2.xml` using UPPAAL timed automata.

Pacemaker Template

The pacemaker has **4 states**:

1. **SenseReady** (Initial state)
    - Invariant: `xv <= LRL`
    - Ready to sense heart activity or deliver pace
    - Transitions:
        - On `sense?` (heart beat detected) → SenseEvent
        - When `xv >= LRL` (timeout) → PaceEvent

2. **WaitVRP**

    - Invariant: `xv <= VRP`
    - Refractory period after sense or pace
    - Transition: When `xv >= VRP` → SenseReady
3. **SenseEvent**: Processes sense signal, sets hysteresis rate
4. **PaceEvent**: Delivers pace signal, sets lower rate

Heart Template

The heart simulator has **4 states**:

1. **Resting** (Initial state)

    - Invariant: `xh <= minIBI`
    - Post-beat recovery period
    - Transition: When `xh >= minIBI` → SenseReady

2. **SenseReady**

    - Invariant: `xh <= maxIBI`
    - Ready for intrinsic beat or pace
    - Transitions:
        - When `xh >= currentBeatInterval` → SenseEvent (intrinsic beat)
        - On `pace?` (pacemaker signal) → PacedEvent
3. **SenseEvent**: Sends sense signal to pacemaker
4. **PacedEvent**: Processes pace signal from pacemaker

Verification Properties

All critical safety properties are verified:

A[] not deadlock

A[] Pacemaker.WaitVRP imply Pacemaker.xv <= VRP

A[] (Pacemaker.xv > VRP) imply !Pacemaker.WaitVRP

A[] Pacemaker.xv <= LRL

A[] Heart.xh <= maxIBI

Hardware Setup

Required Components

- 2x Arduino Nano 33 IoT boards
- 1x USB cable for each board (for programming and power)
- 3x Jumper wires (for UART communication)

Pin Connections

Connect the two Arduino boards as follows:

Heart Arduino Pacemaker Arduino

——————————— ————————————————

Pin 0 (RX) ————————► Pin 1 (TX)

Pin 1 (TX) ————————► Pin 0 (RX)

GND ————————————————► GND

WiFi Configuration

The **Pacemaker Controller** requires WiFi for MQTT publishing. Update credentials in:

// controller/include/arduino_secrets.h

#define SECRET_SSID "your_wifi_ssid"

#define SECRET_PASS "your_wifi_password"

#define SECRET_MQTT_USERNAME "cis441-541_2025"

#define SECRET_MQTT_PASSWORD "cukwy2-geNwit-puqced"

Code Logic

Heart Simulator (`heart/src/main.cpp`)

State Machine Logic

The heart implements a simple ventricular model with random intrinsic beats:

1. **Initialization**

    - Creates FreeRTOS task: `TaskHeartStateMachine`
    - Initializes Serial1 at 9600 baud for pacemaker communication
    - Generates random beat interval between minIBI and maxIBI
    - Starts in RESTING state

2. **Main Task Loop** (`TaskHeartStateMachine`)

```
while (true) {
    // Update heart clock
    xh = millis() - stateStartTime;
    // Check for incoming PACE signals
    if (pacemakerSerial.available()) {
        char signal = pacemakerSerial.read();
        if (signal == 'P' && currentState == SENSE_READY) {
            handlePaceSignal();  // Transition to PACED_EVENT
        }
    }
    // State machine transitions
    switch (currentState) {
        case RESTING:
            if (xh >= minIBI) {
```

```
            transitionToState(SENSE_READY);
        }
        break;
    case SENSE_READY:
        if (xh >= currentBeatInterval) {
            transitionToState(SENSE_EVENT);
        }
        break;
    case SENSE_EVENT:
        sendSenseSignal();  // Send 'S' to pacemaker
        currentBeatInterval = generateRandomBeatInterval();
        stateStartTime = millis();  // Reset clock (xh = 0)
        transitionToState(RESTING);
        break;
    case PACED_EVENT:
        currentBeatInterval = generateRandomBeatInterval();
        stateStartTime = millis();  // Reset clock (xh = 0)
        transitionToState(RESTING);
        break;
  }
  vTaskDelay(5 ms);
}
```

3. **Key Functions**

    - `sendSenseSignal()`: Sends 'S' via Serial1
    - `handlePaceSignal()`: Processes received pace command
    - `generateRandomBeatInterval()`: Returns random value in [minIBI, maxIBI]
    - `transitionToState()`: Changes state (clock continues unless explicitly reset)

4. **Clock Semantics**

    - Heart clock `xh` tracks time since last beat
    - Clock is reset (`stateStartTime = millis()`) only when UPPAAL assigns `xh = 0`
    - This occurs in committed states (SENSE_EVENT, PACED_EVENT) before returning to RESTING

Critical Design Points

- Pace signals are **only accepted in SENSE_READY state** (not during RESTING or refractory)
- Random beat intervals simulate natural heart rate variability
- Mutex protection ensures thread-safe state transitions

Pacemaker Controller (`controller/src/main.cpp`)

Architecture

The controller uses **two FreeRTOS tasks**:

1. **TaskHeartPolling** (Priority 2): High-priority state machine for sensing/pacing
2. **TaskMQTT** (Priority 1): Lower-priority task for network communication and monitoring

Task 1: Heart Polling (`TaskHeartPolling`)

Main State Machine Logic:

```
while (true) {

  uint32_t timeInState = millis() - stateStartTime;

  // Check for incoming SENSE signals from heart

  if (heartSerial.available()) {
```

```
        char signal = heartSerial.read();
        if (signal == 'S' && currentState == SENSE_READY) {
            handleSenseSignal();  // Process sense event
        }
    }
    // State-based timing transitions
    switch (currentState) {
        case WAIT_VRP:
            if (timeInState >= VRP) {
                stateStartTime = millis();  // Reset clock
                transitionToState(SENSE_READY);
            }
            break;
        case SENSE_READY:
            if (timeInState >= LRL) {
                sendPaceSignal();  // Timeout -> deliver pace
            }
            break;
    }
    vTaskDelay(5 ms);
}
```

Key Functions:

1. `handleSenseSignal()` - Process natural heart beat

```
void handleSenseSignal() {
    lastWasPace = false;
    lastWasSense = true;
    currentRI = HRI;  // Use hysteresis rate after sense
    recordBeat(false);  // Log as sensed beat
    stateStartTime = millis();  // Reset clock (xv = 0)
    transitionToState(WAIT_VRP);  // Enter refractory period
}
```

2. `sendPaceSignal()` - Deliver pacing pulse

```
void sendPaceSignal() {
    lastWasPace = true;
    lastWasSense = false;
    currentRI = LRL;  // Use lower rate after pace
    heartSerial.write('P');  // Send pace command
    recordBeat(true);  // Log as paced beat
    stateStartTime = millis();  // Reset clock (xv = 0)
    transitionToState(WAIT_VRP);  // Enter refractory period
}
```

Task 2: MQTT Monitoring (`TaskMQTT`)

Milestone 2 Implementation:

1. **Beat Tracking**

- Circular buffer stores up to 200 beat events
- Each event records: `timestamp`, `wasPaced` flag
- Thread-safe access via `beatMutex` semaphore

2. **Periodic Publishing** (every 5 seconds)

```
void calculateAndPublishMetrics() {
    // Calculate beats in 20-second window
    uint16_t totalBeats = 0;
    uint16_t pacedBeats = 0;
    for (each beat in buffer) {
        if (beat.timestamp >= cutoffTime) {
            totalBeats++;
            if (beat.wasPaced) pacedBeats++;
        }
    }
    // Only publish if we have sufficient data
    if (totalBeats > 1) {
        float avgHeartRate = (totalBeats * 60.0) / WINDOW_SIZE;
        float pacePercentage = (pacedBeats * 100.0) / totalBeats;
        // Publish heartrate data
        publishHeartRate(avgHeartRate, totalBeats, pacedBeats, pacePercentage);
        // Check alarms
        if (pacePercentage > 70%) {
            publishSlowHeartAlarm(pacePercentage);
```

```
        }

        if (avgHeartRate > 60000.0/URL) {

            publishFastHeartAlarm(avgHeartRate);

        }

    }

}
```

3. **MQTT Topics**

    - `cis441-541/heart_racer/pacemaker/heartrate`: Regular metrics
    - `cis441-541/heart_racer/pacemaker/alarm`: Alarm conditions
    - `cis441-541/heart_racer/pacemaker/status`: Normal status

4. **Alarm Conditions**

    - **Slow/Paced Heart**: Triggered when >70% of beats are paced
        - JSON: `{"type":"SLOW_HEART","pace_pct":X.X,"threshold":70}`
    - **Fast Heart**: Triggered when average heart rate exceeds URL threshold
        - JSON: `{"type":"FAST_HEART","avg_bpm":X.XX}`

Configuration Parameters

```
const uint32_t WINDOW_SIZE = 20;      // 20-second averaging window

const uint32_t PUBLISH_INTERVAL = 5;  // Publish every 5 seconds

const uint8_t PACE_THRESHOLD = 70;    // 70% threshold for slow heart alarm
```

MQTT Monitoring

Broker Information

- **Server**: mqtt-dev.precise.seas.upenn.edu
- **Port**: 1883
- **Username**: cis441-541_2025
- **Password**: (see arduino_secrets.h)

Message Formats

Heartrate Data:


```
{
  "window": 20,
  "avg_bpm": 45.5,
  "total_beats": 15,
  "paced_beats": 3,
  "pace_pct": 20.0
}
```


Slow Heart Alarm:


```
{
  "type": "SLOW_HEART",
  "pace_pct": 75.5,
  "threshold": 70
}
```


Fast Heart Alarm:


```
{
  "type": "FAST_HEART",
  "avg_bpm": 185.2
}
```


Monitoring with MQTT Client

To monitor the system on Codio: run the monitor.py script which listens to the mqtt server and prints status updates to the console. Every 5 seconds it prints the average bpm of the heart. If

an alarm is sent it also prints that to the console for a fast alarm if the average bpm is above the high threshold or slow alarm if the pace percentage is above the pace threshold.

Building and Running

Prerequisites

- PlatformIO (install via VSCode extension or CLI)
- Two Arduino Nano 33 IoT boards
- USB cables and jumper wires

Build Process

1. **Build Heart Simulator**

   cd heart

   pio run

2. **Build Pacemaker Controller**

   cd controller

   pio run

Upload Process

1. **Upload to Heart Board**

   cd heart

   pio run --target upload

   - Connect first Arduino via USB
   - Wait for upload to complete
   - Monitor output: `pio device monitor -b 115200`

2. **Upload to Pacemaker Board**

   cd controller

   pio run --target upload

   - Connect second Arduino via USB

- Ensure WiFi credentials are set in `arduino_secrets.h`
- Wait for upload to complete
- Monitor output: `pio device monitor -b 115200`

Expected Serial Output

Heart Board:

[Heart] Heart Simulator Starting...

[Heart] minIBI = 150 ms, maxIBI = 200 ms

[Heart] Serial1 initialized for pacemaker communication

[Heart] Initial beat interval: 175

[Heart] State machine initialized -> RESTING

[Heart] HeartStateMachine task created

[System] Task initialized, starting scheduler...

[Heart] HeartStateMachine task started

[Heart] minIBI reached (xh=152) -> SENSE_READY

[Heart] Intrinsic beat time reached (xh=176) -> SENSE_EVENT

[Heart] SENSE signal sent to pacemaker

[Heart] Sense signal sent -> RESTING

[Heart] Next beat interval: 188

Pacemaker Board:

[Pacemaker] VVI Mode Controller Starting...

[Network] Connecting to WiFi.....

[Network] WiFi connected

[Network] IP address: 192.168.1.100

[Pacemaker] State mutex created

[Pacemaker] Beat mutex created

[Pacemaker] State machine initialized -> SENSE_READY

[Pacemaker] HeartPolling task created

[Network] MQTT task created

[System] All tasks initialized, starting scheduler...

[Pacemaker] HeartPolling task started

[MQTT] Task started

[MQTT] Connecting to broker: mqtt-dev.precise.seas.upenn.edu

[MQTT] Connected to broker

[Pacemaker] SENSE detected

[Pacemaker] Sense event processed -> WAIT_VRP

[Pacemaker] State change: SENSE_READY -> WAIT_VRP

[Pacemaker] VRP complete (xv=152 ms)

[Pacemaker] State change: WAIT_VRP -> SENSE_READY

Verification Results

UPPAAL Model Verification

All formal properties have been verified in UPPAAL (see `Proj_2_M1_V2.xml`):

Recording of verification: https://www.loom.com/share/48dabdc93f9240f69bb0616c8847d847

Property	**Result**	**Description**
`A[] not deadlock`	✓ Pass	System never enters deadlock state

Property	Result	Description
`A[] Pacemaker.WaitVRP imply Pacemaker.xv <= VRP`	✓ Pass	VRP invariant respected
`A[] (Pacemaker.xv > VRP) imply !Pacemaker.WaitVRP`	✓ Pass	Leaves VRP when timer expires
`A[] Pacemaker.xv <= LRL`	✓ Pass	Pacemaker clock bounded by LRL
`A[] Heart.xh <= maxIBI`	✓ Pass	Heart clock bounded by maxIBI

Real-Time Testing Results

Test 1: Normal Heart Rhythm

- Heart beats naturally at ~400-2000ms intervals (based on min and max IBI with random values in between)
- Pacemaker senses all beats successfully
- Pace percentage: 0-10%
- Status: NORMAL

Test 2: Slow Heart (Simulated)

Slow Heart Rate Demo: https://www.loom.com/share/a5f25c2a6935400e832769363ad3a0e3

- Modified minIBI/maxIBI to 2000ms+
- Pacemaker delivers pacing pulses at LRL (1500ms)
- Pace percentage: >70%
- Alarm: SLOW_HEART triggered correctly

Test 3: Fast Heart (Simulated)

Fast Heart Rate Demo: https://www.loom.com/share/7dde86de33ba4f4e82378845b12a279f

- Modified minIBI/maxIBI to 100ms
- Average heart rate exceeds URL threshold
- Alarm: FAST_HEART triggered correctly

Project Structure

CIS-4410-Project-2-Milestone-1/

```
├── heart/                    # Heart simulator project
│   ├── src/
│   │   └── main.cpp          # Heart state machine implementation
│   ├── include/
│   └── platformio.ini        # Build configuration
├── controller/               # Pacemaker controller project
│   ├── src/
│   │   └── main.cpp          # Pacemaker + MQTT implementation
│   ├── include/
│   │   └── arduino_secrets.h  # WiFi/MQTT credentials
│   └── platformio.ini        # Build configuration
├── Proj_2_M1_V2.xml          # UPPAAL model (verified)
└── PROJECT_REPORT.md         # This document
```