Take-home assignment

Implementation of a sequential network

Instructions

Please read the instructions carefully. The specification begins on page 3.

After your submission, our team will review and decide if we can go to the technical interview part. The technical interview itself is an introduction and discussion of your solution, possible changes and further development options.

What you get

- This file with ***instructions*** and ***specification***
- ***Public API*** for sequential network and condition selector modules provided as interface header files

Delivery

- Submit your solution in a **public GitHub** or **GitLab** repository
- Implementation in **C programming language** of
    - sequential network emulator that interprets the uint16_t array
    - simple condition selector
    - integrate them together and write a simple program to drive them
    - fill the program memory with the required data to make the elevator controller really work
- **README** file with instructions for **building** and **testing** your solution

Evaluation criteria

- Overall code quality
- Build and test environment
- Verification & validation strategy
    - *Add some of the following test cases to your code: call for the actual floor, multiple pending calls, new call during movement, request door re-open by pressing the call button for the actual floor during close in progress…*
- Documentation, error handling and safety concerns
    - *E.g.: Be sure to never go down if the elevator is on floor0, never open the door during movement or if badly positioned…*
    - Please provide the code with documentation, following a – preferably safety – coding guideline

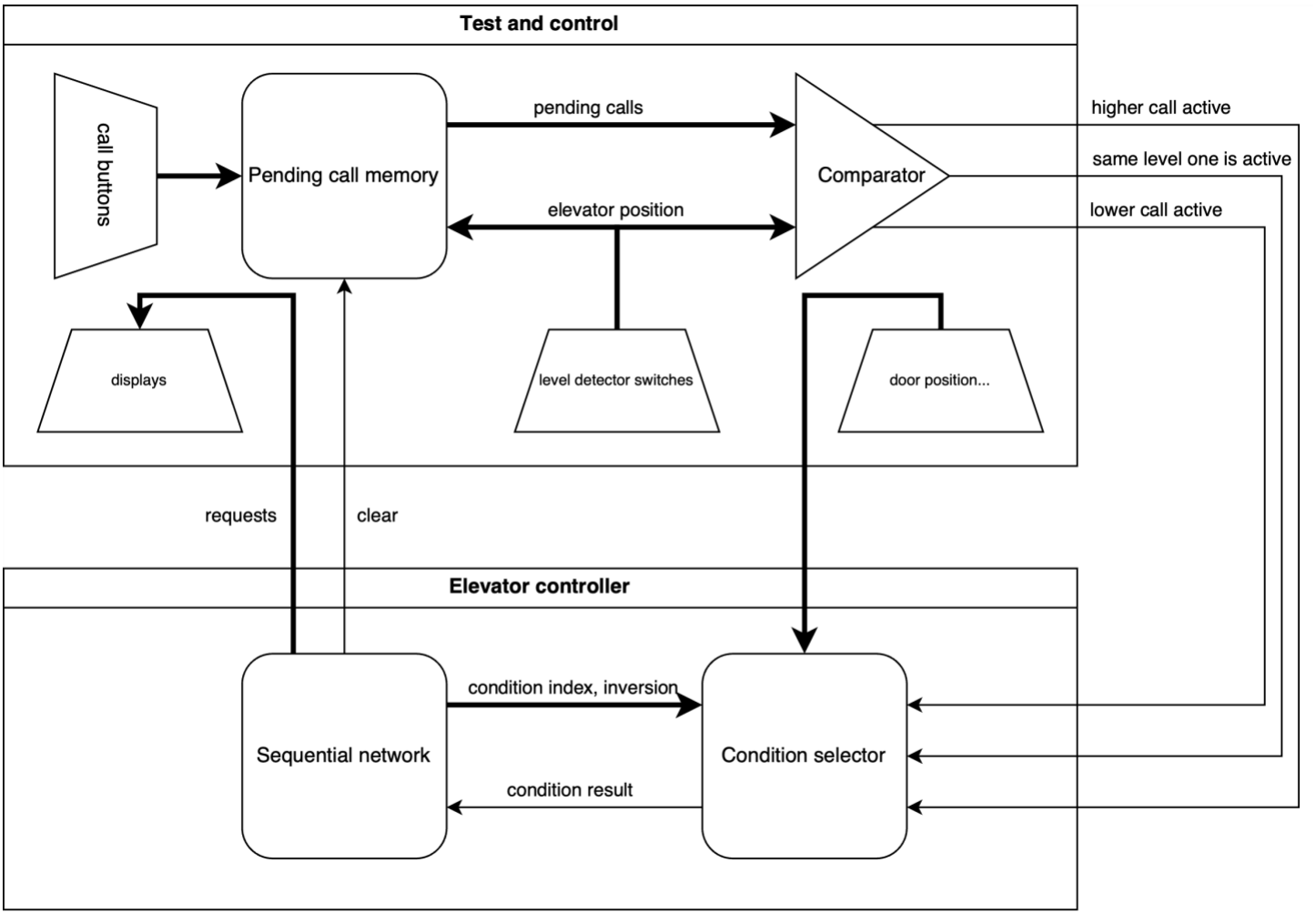
Frequently asked questions

- Can I use AI tools?
  - Yes, it is allowed to use AI chat (ChatGPT, Claude, Gemini, etc.) or coding agents (Copilot, Windsurf, Cursor, etc.)
- How much time do I have?
  - The implementation shall not take more than 2-3 hours
  - If you've spent more than 5 hours, you're probably overthinking
- I have no GitHub account, and I don't want to create one
  - ZIP your solution and find a way to send it to us
  - Still, we'd be happy to receive a Git repository, even if it's a zipped version of your local working set

Specification

HW engineers are working on a 6-floor elevator controller device. Your task is to support them by implementation of an emulator of it and by supplying the binary code – let's call firmware – what will run on it. Optionally, test the implemented emulator on your own.

Modules



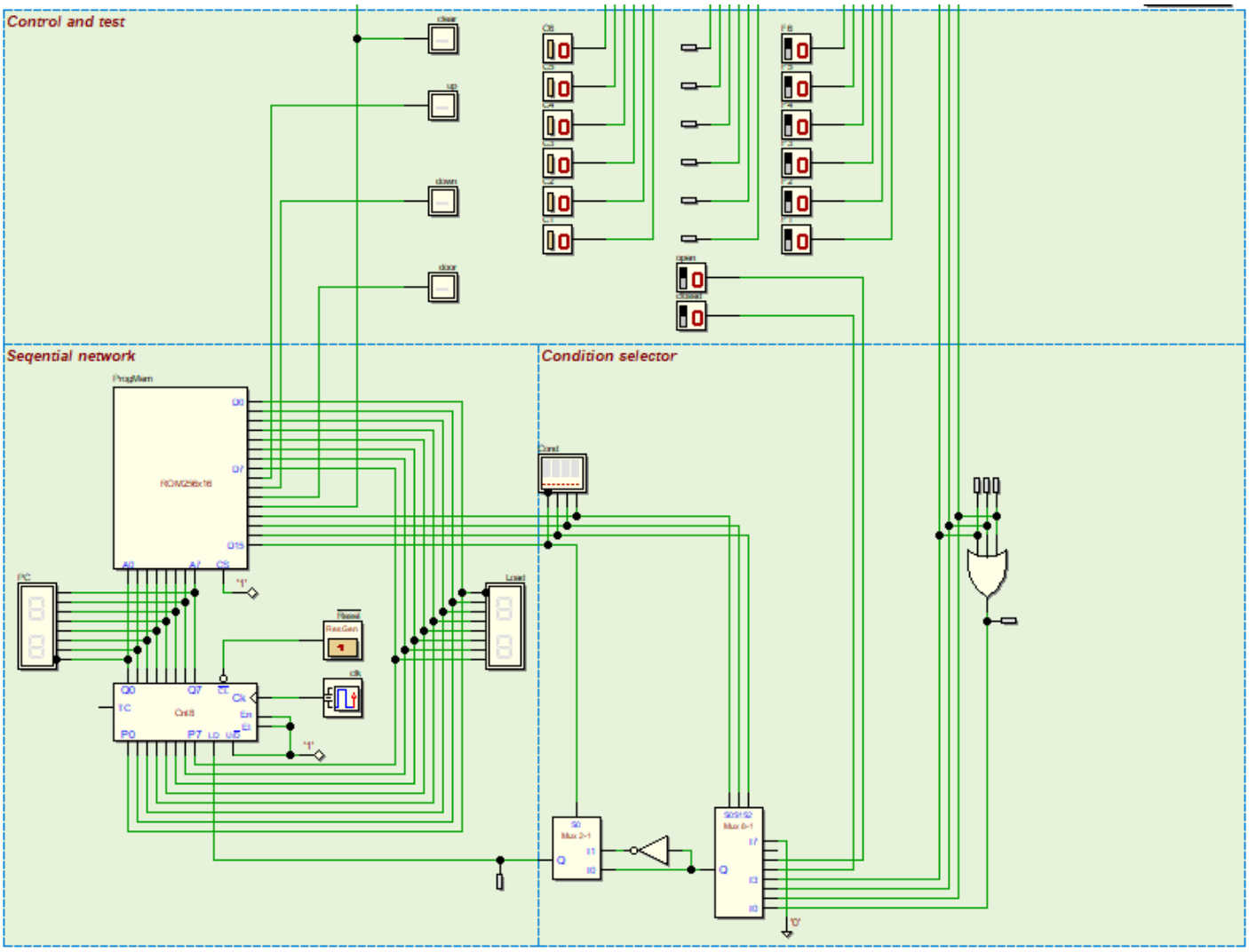
The system can be decomposed into the following parts:

- **Test and control**
  It displays and controls the inputs and outputs of the elevator controller device.
  In the schematic, some displays and switches are connected to represent those and make the device testing possible (there are just the switches and displays are visible).
- **Elevator controller**
  - **Sequential Network:**
    This is the core controller.
    It needs to contain a program memory [ProgMem] to store the instructions (the program).
    The memory needs to be addressed by a counter – Program Counter [PC].

PC value is simply incremented if the selected condition is not active or loads a “jump” address if the selected condition is active. Condition can be selected and jump address can be specified in the instruction – details below.

- **Condition Selector:**
  By a value, the 7 different conditions can be selected, and inversion can be activated on it also.

Schematic



Control and test block

Features:

- Displays the states of the request bits comes from elevator controller device.
- Display the pending calls for every floor.
- Controls (emulates) the elevator position.
- Controls (emulates) the door position.

Control block stores the unserved calls for every floor. The bit for a floor is set, when internal or external button is pressed, and cleared when the elevator is on that floor and “clear” request bit is active.

That component also calculates a reduced set of the pending calls:

- Call pending above the elevator's level
- Call pending on the level of the elevator
- Call pending below the elevator's level

The elevator's actual level is detected by switches on every level. Switch is activated on a floor if the elevator reaches the position where it can be stopped to open the door. That way, it is possible to there is not any level detector switches are activated.

That module needs to be "replaced" by a module to make the testing of elevator controller possible.

Elevator controller

Initially the elevator is standing on the floor with door opened and shall wait for a call. If one of the internal or external buttons is pressed, that call should be "served" by the following sequence:

- Door needs to be closed,
- The elevator needs to be moved to the target floor and
- When it is reached, the door needs to be opened.
- When door is opened, the "pending call" signal needs to be cleared actively.

Input

Input ID	Name	Description
In1	Call pending below	Active, when there is any call pending below the elevator position.
In2	Call pending same	Active, when call pending on the same level as the elevator position.
In3	Call pending above	Active, when there is any call pending above the elevator position.
In4	Door closed	Active, when the door is closed and door lock activated.
In5	Door open	Active, when the door is fully open.

Call pending signals are valid only when one of the elevator position detectors is activated.

Between the end positions of the door, none of the corresponding signals are active.

Output

To control the elevator, the elevator controller module needs to provide the following outputs:

Output ID	Name	Description
Out0	Move up	Request to activate the motor to raise the elevator.
Out1	Move down	Request to activate the motor to lower the elevator.
Out2	Door target state	0: request to close the door – if not already closed 1: request to open the door – if not already open
Out3	Delete pending call	Request to clear the pending call for the actual position/floor of the elevator.

Sequential network block

This block contains, interprets and evaluates the program. Contains:

- Program Memory [ProgMem]: a 256-element array containing uint16_t values which are the instructions,
- Program Counter [PC]:

- Incremented if condition is inactive – turns-around when 0xFF is reached
- Set to “jump address” part of the instruction if condition is active.

- Schematic also contains some displays for the PC and jump address and some components to make PC working as described.

After startup, PC needs to be initialized to 0x00.

Instruction consists of the following bit-fields:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Condition				Requests				Jump address							
Invert	Select			Pending call reset	Door state	Move down	Move up								

Outputs of the device can be specified in the bits 8 ..11.

Condition selector block

That module provides the state of the selected condition by the index input parameter. The following conditions can be selected by the value of the field in the instruction:

Value	Selected condition	Description
0	In1 or In2 or In3	Active, when there is any pending call (calculated internally). *
1	In1	See input specification table. *
2	In2	See input specification table. *
3	In3	See input specification table. *
4	In4	See input specification table.
5	In5	See input specification table.
6	-	Not used.
7	Const 0	That condition is always inactive, independently of any other state.

* Can be active only if any of the elevator position detectors are active.

Value of the selected condition can be inverted by set “condition invert” bit to 1 in the instruction.