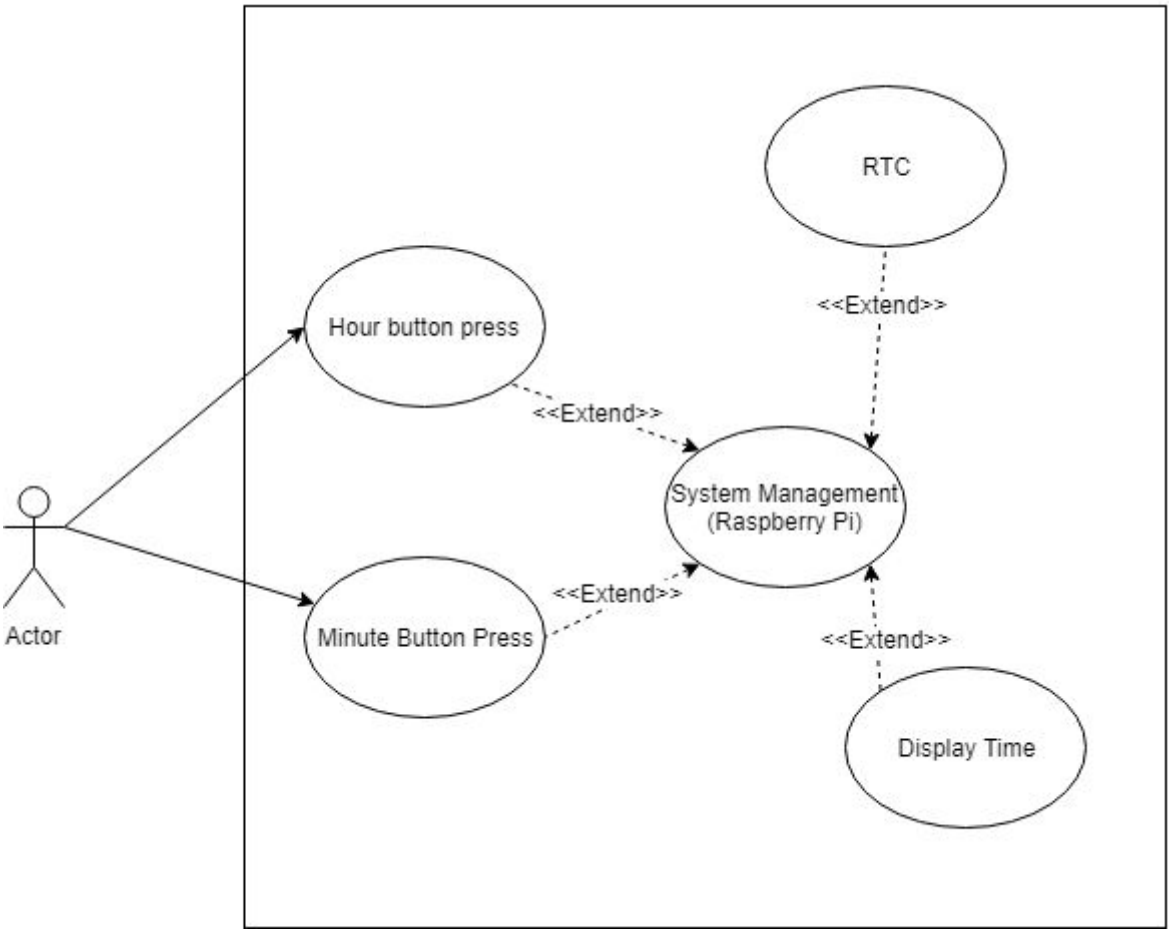
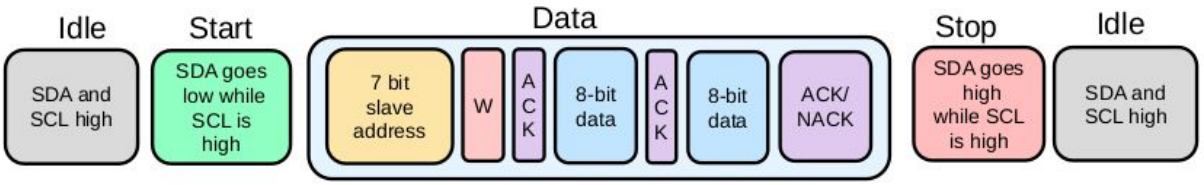
Prac 3 Memo

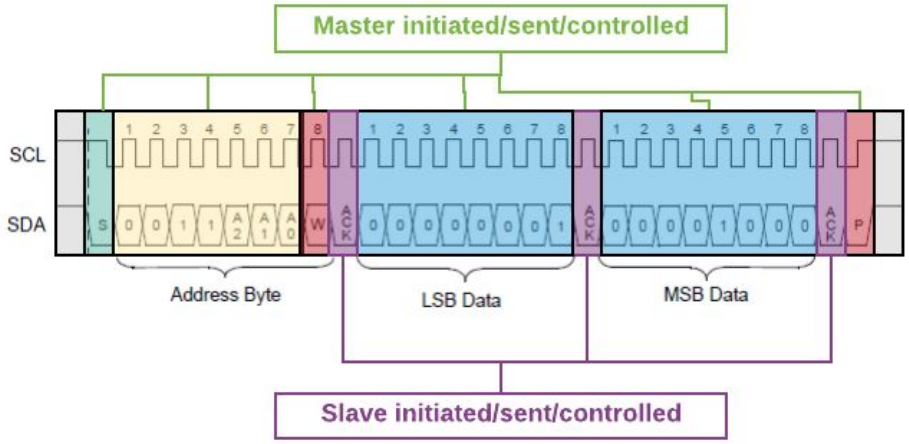
1. **Introduction (3)**
   Introduction should introduce the practical and talk about the components used. An example:
   “Prac 3 serves to use I2C (1) to communicate between a Raspberry Pi (1) and an RTC module(1) , namely the MCP7940M. The application is written in C++, using a provided template as a base. (1) WiringPi is used as module for implementation (1).”
   There’s really no “correct” answer here, it should just provide context to the rest of the write up.
2. **UML Use Case Diagram (5)**
   Should include components (actor, buttons, RTC, Raspberry Pi, display time) (0.5*5=2.5)
   The actor interacts with the buttons. (0.5)
   The raspberry Pi is the base use case, and everything else is an extension. (0.5*4=2)
   There are of course different ways to represent this system. Try be lenient with marks. Here’s an example. If I’ve made mistakes here, please let me know.

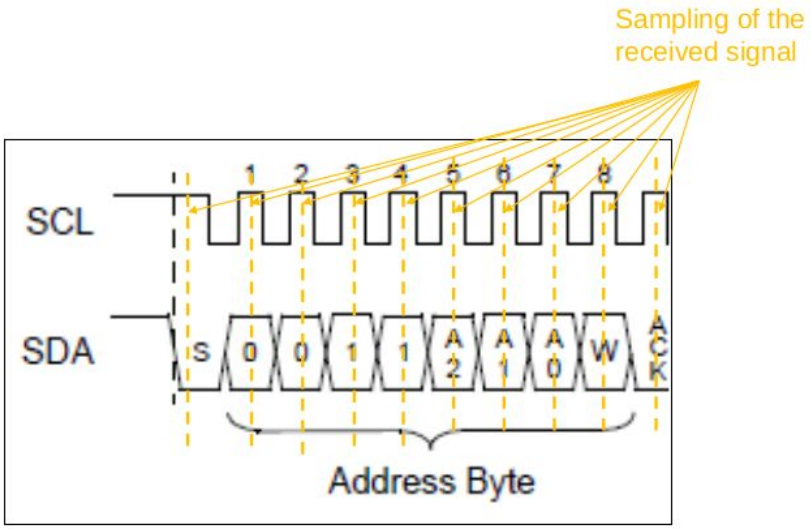


3. **I2C communications(4 + 6 =10)**

a. Init - should at least include the code for init of i2c (2 - one for address). They can just have the function definition for 1 mark.
   rtc = wiringPiI2CSetup (0x6f);
b. TX - rtc, register and value can be any variables. They can also just have the function definition
   wiringPiI2CWriteReg8 (rtc, register, value) ;
c. RX - as per TX
   value = wiringPiI2CReadReg8(rtc, register);
d. Timing diagram should include address, W, ack , data, ack, data, ack, on the SCL and SDA lines. 1 mark for address (+1 for 0x6f), 1 mark for data, 1 mark for ack, 2 marks for correct SDA/SCL in idle.







4. **Interrupts and debouncing (4)**

Only need to make 2 points on each. Can also mention how they're implemented in their code for 1 mark. Eg:

- Interrupts are used to respond immediately/asynchronously to an input, as opposed to polling which can be processor intensive. Interrupts can be triggered by, for example, a button press or timer. (2)
- Debouncing is used to prevent noise in a button press/signal from unintentionally triggering the interrupt. Software can be in hardware or software form. Hardware interrupts usually make use of a capacitor and resistor, whereas software debouncing can be implemented using a short "wait" in the service routine (not always good practice) or checking the system time of the last time the button was pressed, and only executing the ISR if enough time (usually a few mS) has passed. (2)

5. **Circuit diagram (6)**
    - Correct SDA and SCL connections between RPi and RTC (1)
    - RTC circuit is the same as what's given in the data sheet (2)
    - Buttons connected correctly (1)
    - LEDs have resistors (1)
    - Neatness (2)