
HW #7: Requirements

Consider a digital alarm clock. This is a cheap, minimalist alarm clock with ONLY the inputs, outputs, and internal values indicated. The goal is to create a set of requirements for this simple alarm clock.

FAIR WARNING: you will be building upon this design later, so take the time to do it right, and make sure you save your answer in a place you can access it later when you need it. Be sure to use the defined terms below in your responses:

- Internal memory values (read/write):
    - **TIME**: current time in hours and minutes (no seconds); automagically increments according to time of day, rolling over at midnight. In other words, your system uses this as the actual source of the current time, and it's a hardware counter that you can read whenever you need to.
    - **ATIME**: alarm time in hours and minutes (no seconds); this is just an ordinary variable memory location. You can write a value to it and read from it.
    - **STATE**: a set of values (as many as you like) that tracks the state of the alarm using a state machine. (When code is written this would end being an enum, but for requirements it's just a list of state names.)
    - We strongly prefer that you do NOT use additional variables beyond these. You should keep track of logical flag values by changing which STATE you're in.
- Outputs (write only; **can NOT be read** so you can't use these as storage locations):
    - **DISPLAY**: a numeric display that can display either TIME or ALARM as you prefer (you'll want at least some states to set this value to determine which is displayed)
    - **ARMED**: an indicator showing whether the alarm is currently activated ("on" means alarm will sound; "off" means alarm function is de-activated). On many real alarm clocks this is a little alarm symbol or an LED indicator.
    - **BUZZER**: a noise maker that, when turned on, makes the alarm noise. Bzzzzzzzzzz.
- Input switches (read only; can NOT be written):

    - **SW-1**: a single on/off momentary push-button switch that can be used however you want. We strongly suggest you give it a more meaningful name.
    - **SW-2**: a second single on/off momentary push-button switch that can be used however you want. We strongly suggest you give it a more meaningful name.

The alarm design has to account for at least the following use cases:

- U-1: set time of day
- U-2: show time of day
- U-3: sound alarm
- U-4: deactivate alarm
- U-5: set alarm armed state (active/inactive)
- U-6: show alarm state
- U-7: set alarm time
- U-8: show alarm time
- U-9: housekeeping (optional) use this for any requirement that doesn't directly trace to another use case
- These are engineering type product requirements. Product marketing requirements would be simpler, such as "User shall be able to set the alarm time" ... which gets us back to the use cases.

7-1: Create a list of HIGH LEVEL PRODUCT requirements for the alarm clock. This are user-facing functions that the alarm clock must perform to be an alarm clock. Use the identified use cases above as a reminder list. (Usually the high level requirements come first, but we wanted to fix the use cases for this exercise to prevent too much random dispersion of student work.) For each high level requirement, state which use cases it covers. Make sure each use case is covered by at least one requirement. An example that you can use as a starting point is: "PR-1. Clock shall display the correct time of day when not performing other functions. (U-1, U-2)"

Here are some attributes of High Level Product requirements:

- Describes what the product does (its capabilities), but now how the internal software system works (not its design). In other words, it is from the user's point of view, not the software's point of view.

- Is specifically forbidden from referencing the STATE variable, which is the central part of the design (but not visible to the user)
- Can refer to MODES which are general operating modes but are not the same as the STATE variable. In particular a mode is a state in the user's head, not the systems head. For an alarm clock relevant modes might be: display time, set time, set alarm, alarm active. (But you can use others.) You can constrain requirements by which mode they are in, and you can talk about mode transitions. Yes, these map to internal system states, but not necessarily one to one. You should include a list of system modes and descriptions with your requirements for reference.
- Covers all use cases, with at least one requirement per use case
- Can have sub-requirements
- Here some examples to get you started (but you don't have to use these if you don't want to):

    - R-5a. The user shall be able to set the alarm time using SW1 and SW2.
        - R-5b. Pressing SW1 when in normal mode shall transition the system into set alarm mode
        - R-5c. Pressing SW2 when in alarm mode shall advance alarm time at one minute of alarm time per second that the button is held down
        - R-5d. Pressing SW2 and holding it for more than 5 seconds shall .... Rationale: speeds up large changes in alarm time.

RUBRIC:

- One slide (or perhaps two slides) with high level requirements

Supplemental Reading:

- Explanation of finite state machines: https://www.youtube.com/watch?v=vhiiia1_hC4
- Carl Weigers has a YouTube playlist of 35 videos on software requirements
- The EARS papers have an interesting way of looking at different types of requirements:
    - Mavin, "Listen, then use EARS," IEEE Software, March 2012 (pp. 17-18) https://www.researchgate.net/publication/220092379_Listen_then_use_EARS | http://doi.org/10.1109/MS.2012.36

    - Mavin et al., "Easy approach to requirements syntax (EARS)," RE '09, 2009. https://www.researchgate.net/publication/224079416_Easy_approach_to_requirements_syntax_EARS | https://doi.org/10.1109/RE.2009.9
- Exploring Requirements: Quality Before Design; Donald C. Gause and Gerald M. Weinberg; Hardcover 299 pages (1989); Dorset House Publishing Co., Inc.; ISBN: 0-932633-13-7
