AWAC02

Documentation

v0.3.0

1 Overview

The documentation is meant as a guide for the developer. Additional aid is found in the source header files where most functions are documented.

2 Source Files

In this section an overview of the project files are given. The files are listed in order of relevance.

File(s)	Description
`main.c`	The main file. Contains the clock-mode state-machine.
`constants.h`	Global constants for defining number of displays, enabling logging etc.
`menu.h`, `menu.c`	The state machine for the menu.
`utilities.h`, `utilities.c`	High level utilities. Mostly for applying and showing settings.
`log.h`, `log.c`	Logging functions. Everything here will be unavailable if the LOG constant is undefined.
`rtc.h`, `rtc.c`	High level functions to set and show the external RTC clock.
`display.h`, `display.c`	Low level functions to interface with the Qwiic Alphanumeric Display
`user_alarms.h`, `user_alarms.c`	High level functions to find and modify alarms set by the user.
`helpers.h`, `helpers.c`	Simple helper-functions.
`external_interrupts.h`, `external_interrupts.S`	Register and extract the external interrupts (button presses and RTC alarms).
`time.h`, `time.S`	Start and use the hardware times.
`twi.h`, `twi.S`	Interface to the TWI (I2C) hardware. Read and write to external devices.
`eeprom.h`, `eeprom.S`	Read and write to the EEPROM (non-volatile storage).
`flash.h`, `flash.S`	Read constant data from flash. For example an ASCII conversion table.
`io.h`, `io.S`	Direct control of pins.
`usart.h`, `usart.S`	Send data through USART. This is used to implement printf in ‘log.c‘
`math.S`	Low level math for assembly routines.

3 Schematic

In figure 1 the circuit is shown. The value of R1-R7, C1 and C5-C7 is only approximate. You can try different values. In addition to these components two Sparkfun Qwiic Alphanumeric Displays are used. One of the displays must be modified to disable the pull-up resistors and change the slave address to `0x71`.

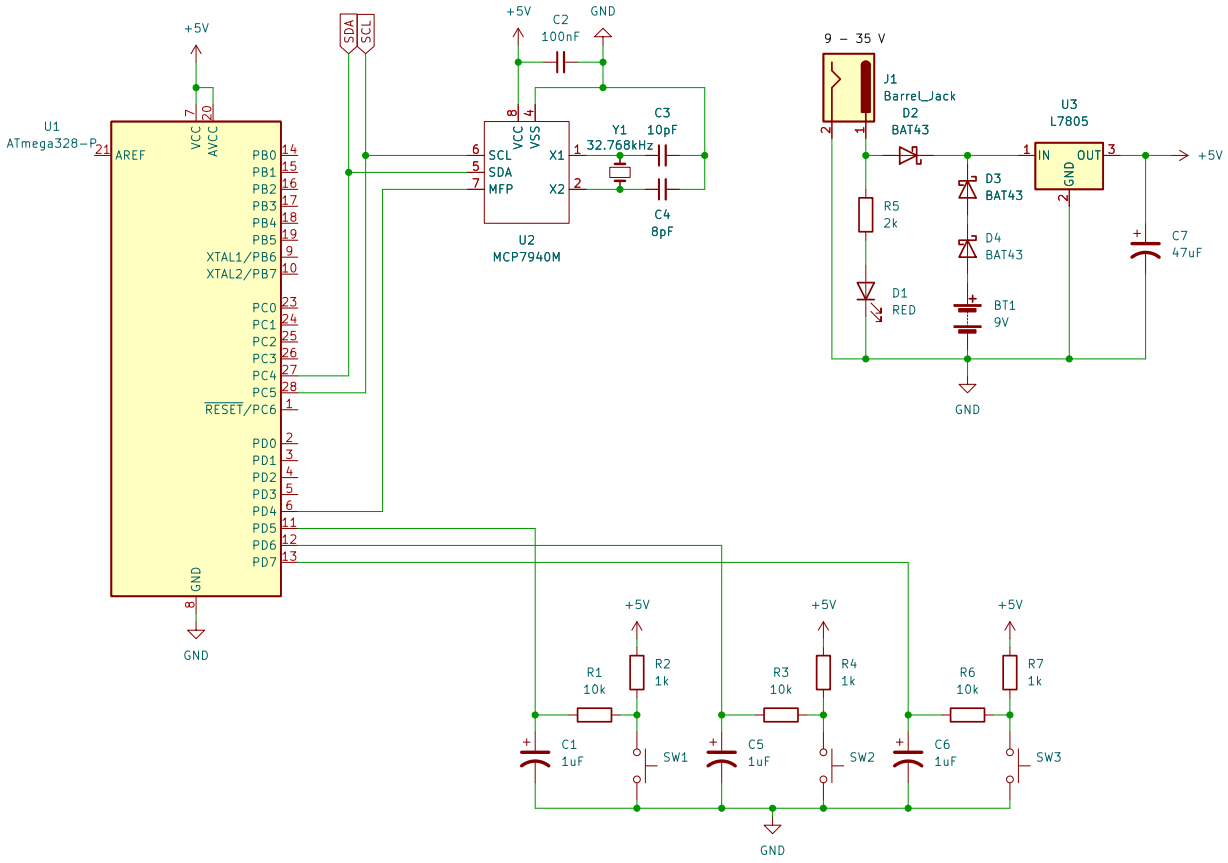

Figure 1: The circuit (excluding displays).

4 State Machine

The state machine for clock mode is in `main.c` and the state machine for the menu mode is in `menu.c`. They are implemented using the `goto` statement which can lead to hard to understand code. To give the programmer a better overview of the states and make development easier some states are documented here and shown in flow-charts. The program starts in the `enter_clock_mode` state. The clock mode state machine is illustrated in Figure 2.

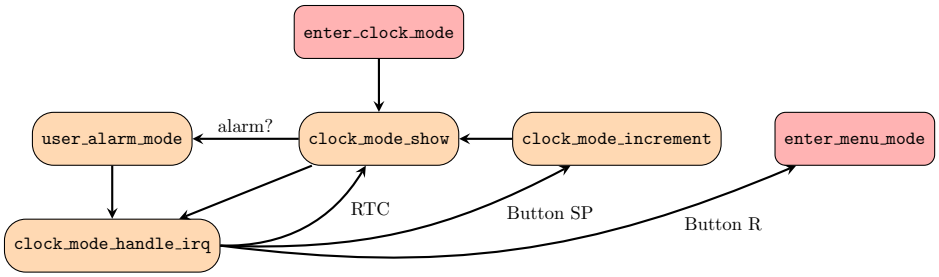

Figure 2: The clock mode state machine

The menu mode state machine is a bit more complicated. A functional overview is shown in Figure 4. There are three basic state types in the menu. *Menu* states, *choose* states and *set* states. The menu states are very simple. Each show some text on the display, then wait for a button interrupt. If the `L` or `R` button is pressed the state is changed to the previous or next menu state. If the `SP` button is pressed it is changed to a choose or set state.

A choose state is used when there is multiple options for the setting that is edited. For example when changing time. A choose state first shows the current value of that setting, sets an RTC alarm for when it is changed and waits for an interrupt. If the interrupt is from the RTC the new value is shown. If it is from the `L` or `R` buttons the state changes to the previous or next choose state. If it is from the `SP` button the state changes to the corresponding set state.

A set state works a lot like a choose state. A RTC interrupt changes the value shown and `SP` interrupt changes state (go back to the choose state). But a `L` or `R` button interrupt de- or increments the value instead.

Figure 3 shows the program flow between a choose and a set state in detail. The actual implementation varies slightly between types.

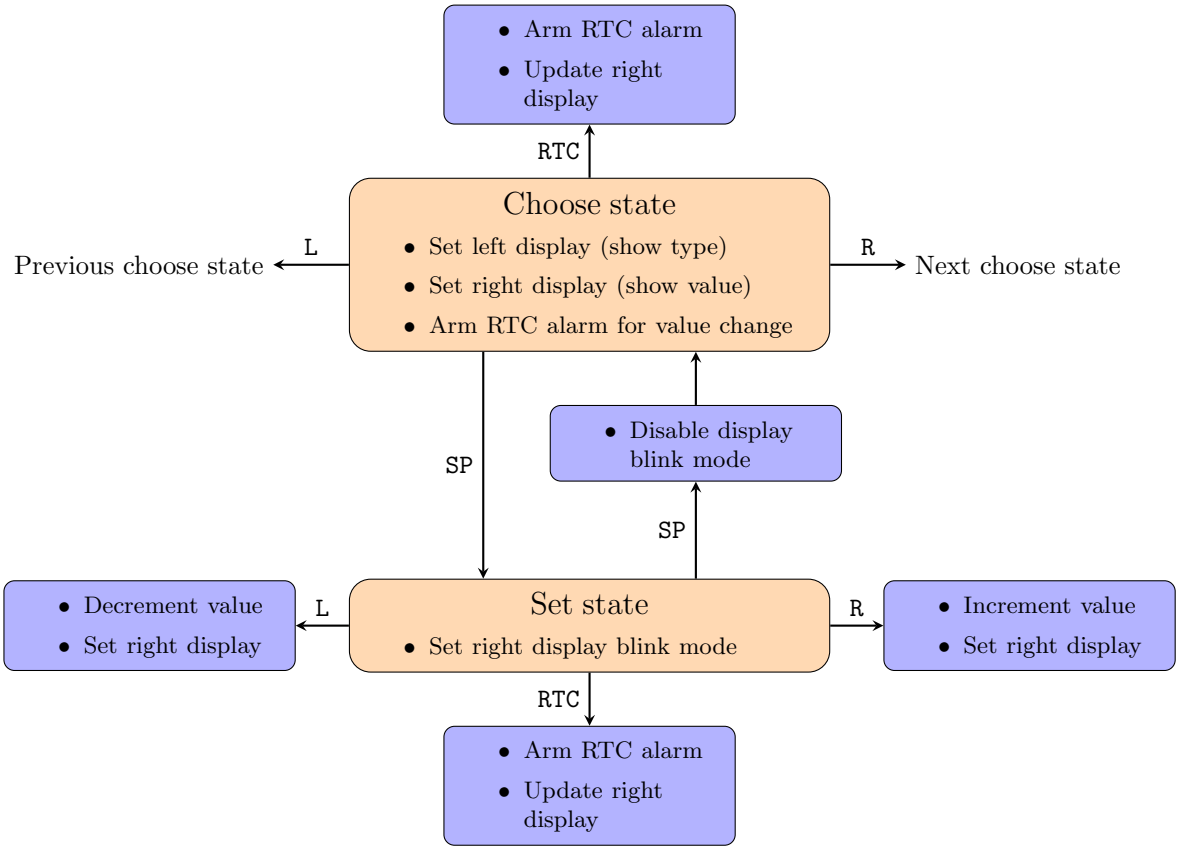

Figure 3: A choose and set state combo

5 EEPROM

The ATmega328p has 1024 bytes non-volatile EEPROM storage. This space is used to store user-made settings such as display brightness and alarms. This means that this information will be available even after a power loss. The current time will however be lost.

Low level EEPROM access is implemented in `eeprom.h` and `eeprom.S`.

5.1 User Alarms

User alarms is the alarms set by the user with the time at which the buzzer should sound. Not to be confused with RTC alarms which is interrupts triggered by the RTC. User alarms are implemented in `user_alarms.h` and `user_alarms.c`.

The user alarms are stored as an array with the element size of 2 bytes. Each element contains the day of the week, hour, minute and status of the alarm. The element content is described by Table 1. In addition to the elements, the array length n is also stored in EEPROM. Note that the array size is $2n$ bytes.

The array is not sorted. That means that the entire array needs to be traversed to look for a match. However, the main performance consern is writes as one write takes a very long time (around 3.3 ms). In Table 2 we see that deletes are particulary problematic. Fortunatly deletes should be uncommon. Finding

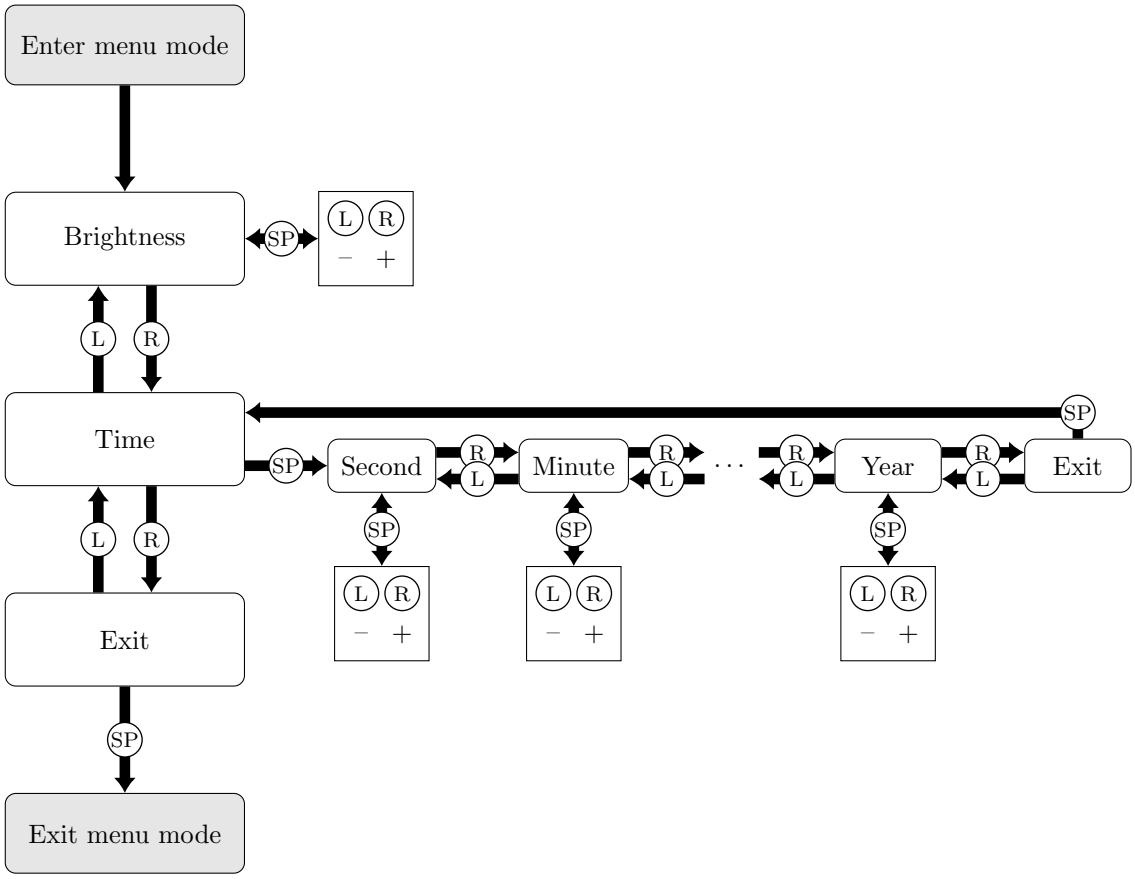

Figure 4: An overview of the menu

is the most common operation and it does not require any writes.

Data	Range	Bits	Byte	Notes
D.o.t.w.	0-7	7-5	high	0 is all weekdays
Hour	0-23	4-0	high	-
Minute	0-59	7-2	low	-
Status	0-1	1	low	0 is off, 1 is on

Table 1: An user alarm

Operation	Performance	Writes
Add item	$\mathcal{O}(1)$	3
Find item	$\mathcal{O}(n)$	0
Delete item	$\mathcal{O}(n)$	$\leq 2n-2$
Modify item	$\mathcal{O}(1)$	2

Table 2: User alarm performance analysis