Embedded Systems

Project Description Tea Timer

Prof. Dr. (Purdue Univ.) Jörg Friedrich

Table of Contents

1 Introduction

The Tea Timer functions as a little embedded application that helps a user to have his tea come out just right. As part of this project we will not build the hardware for this, but rather use the existing Dragon12 board in the lab. It provides all features we need for this application.

2 Requirements

The user can enter a time between 0 and 4 minutes using the potentiometer when the system is in "Preset" state. Precision should be five seconds. (e.g. 5s, 10s, 15s, etc.).

After the timer is started by the user the timer counts down the time and sounds a beeper when the entire time has elapsed. If the same switch is hit again, the timer stops.

If at any time the switch is pressed for more than 2 seconds, the system is reset to its initial state (preset state, preset can be changed).

If the switch is hit for less than 2 seconds while the system pauses, the countdown continues.

The preset value can only be changed when the system is in the initial preset state.

If the switch is hit while the system beeps, the beep shall stop and the system shall return into "Preset" state.

Input Elements

Switch SW5: start, stop, continue, stop beep, reset to initial state

AD converter AD7, 10 bit: seconds shall be AD value / 4.

Output Elements

Main output is on the LCD. Output shall be designed as follows.

Display while being set (initial state):

First line: "Preset: 03:00"
Second line: "Actual: --:--"

The preset changes while it is being set. Display during countdown

First line: "Preset: 03:00"
Second line: "Actual: 01:34"

The actual value changes during countdown. After expiration of the time interval the beeper shall sound five times for one second each, with a break of one second in between beeps.

3 Suggested Implementation

The suggested implementation consists of two state machines. One state machine takes care of switch input and debouncing. The other state machine implements the rest of the system. The general structure should be that of a foreground/background system.

4 Implementation Procedure

The implementation procedure consists of the following steps in that order:

1. Design using UML notation, directory layout

2. Software construction and implementation on the host (standard PC) using either Visual .NET or DevCpp development environment
3. Development of hardware related software functionality on the target
4. Integration of all software parts on the target

Design

Before any software is implemented a design is required. The design should consist of UML state charts and message sequence diagrams (optional).

The first level directory layout in which the implementation shall take place has to look like:

- Application (containing all software parts common to both the host and the target environment)
- HAL-Host (containing software that represents the hardware related functions for the host environment, like switch input, LCD driver and timer events)
- HAL-Target (containing software that exercises the target hardware, like the real switch drivers, LCD driver, etc.)
- IDE-Host (containing the project files for the host ide (e.g. Visual Studio or DevCpp) and the simulation input and output files)
- IDE-Target (containing the CodeWarrior project files and auxiliary files)

The main function shall be contained in a module called “main.c”. There must not be more than one “main.c”, and this “main.c” must be placed in the Application directory. There shall be no “define” to distinguish between host and target environment; this can be implemented using the IDEs by including alternate files (e.g. hidef.h containing an empty EnableInterrupts definition for the host environment).

Implementation on Host

After the design is completed and discussed with peers software can be written. However, we do not write any software for the target yet but rather write software on the development host in such a fashion that it is later on easily possible to transfer it to the target.

We therefore have to simulate switch inputs, timer events and outputs. This simulation environment shall be designed such that a simple exchange of small files shall suffice to move to the target platform. Here is an example.

We do not have an LCD device on the host. We could simulate it with two printf calls. However, on the target we may not want to use printf because we do not have a printf for the LCD there.

Instead, we create a little module that permits to clear the display and execute a writeln(). In case of the host, the writeln contains printf() statements, and in case of the host the writeln() executes the LCD driver code.

For the input signals we read them from an input file in case of the host, and from a driver in case of the target. We just have to make sure that the interface to the main application is the same for both cases.

The input files shall look like (the format is important):

```
+10 // system advances by 10 timer ticks
SW5 // switch 5 was pressed for one tick
AD:496  // AD converter outputs 496, should be 2 min, 5 s
+5 // another 5 ticks, absolute time here 16 ticks
AD:24  // AD converter outputs 24, should be 5 s
```

```
SW5 // switch 5, increment preset by one second
SW5+1000 // switch pressed for 1000 ticks (10 seconds)
…
```

We develop several input files that let us go through different scenarios. We run these input files on the host and check simulation output. Only when we have completed this step we move on.

The simulation shall generate an output file in the following format:

```
<tickcount> -- Preset: 3:00
<tickcount> -- Actual: -:--
…
<tickcount> -- BeeperOn
<tickcount> -- BeeperOff
…
```

The simulation output file shall contain the text in the exact format described above (watch for proper spacing), and shall generate output only when displayed values change. Only the line that changes (Preset or Actual) shall be written to the output. The tickcount variable represents the number of ticks from the start of simulation as an 8 digit right aligned integer with leading spaces. Beeper On and Beeper Off events shall be output as “BeeperOn” and “BeeperOff”.

Implementation on Target

On the target, we need to develop special driver software for the timer, the beeper, switch inputs, and the LCD. This software we have to develop and test on the target.

After we have tested this software, we take the application that we have already simulated on the host and port it to the target. This step should require only little time and there should be no logical errors any more in our application.

When are You Done?

Check off the following points before submitting your solution:

- The state chart design conforms to UML notation and properly and in sufficient detail describes what is going on (all states, all events)
- The directory layout is exactly as described above
- The simulation runs on the same computer where the target IDE for downloading the target software is installed (it can be your computer, but it has to be the same)
- There is no need to touch any files in the Application directory to switch between host and target environment
- The HAL-xx directories do not contain application logic, just driver software
- The C-code is of professional quality (no paths in include statements, no copy/paste style software parts, guards in include files, good names, etc.)
- The state, event, and action names from the design show up in the implementation
- The input and output file formats are exactly as described above
- Debouncing shall work properly
- The software requirements are properly implemented