# Bowling Score Keeper

## Overall Description

The objective is to provide display for current score of a bowling game. The system is implemented using embedded software running on an ARM Microcontroller Unit (MCU). Software development environment with required system libraries will be provided, as well as the hardware required to process and display data.

## Hardware Architecture

The hardware architecture of development system consists of following parts:

* LPC812 MCU with surface mounted device packaging (e.g. TSSOP20),
  + The LPC812 Max Board will be used as development platform,
* 7-segment display unit and its HT17K33 driver Integrated Circuit (IC),
  + This is interfaced to MCU with Inter-Integrated Circuit (I2C) bus
  + The 7-segment display unit has address 0x70
* An I2C compatible device returning the number of pins knocked down (“pins knocked device”),
  + This device cannot be made available.
  + The pin reading unit has address 0x90.
  + The I2C API has been tweaked by the instructors, so that it is possible to read the address 0x90. Obviously, you will get simulated data, but they are identical to real data.
* Required connections for:
  + USB connection for power, software deployment and debugging of development board,
  + I2C data and clocking,
  + Power and ground (GND) for I2C devices

This hardware architecture is shown in Figure 1. The finished product will be installed in the bowling machine case, using a larger display.

I2C clock wire (PIO0\_11)

LPC812 Max

USB cable

I2C bus

7-segment display (0x70)

Pins knocked device (0x90)

I2C data wire (PIO0\_10)

Figure 1. Hardware architecture

## Software Architecture

This chapter describes the software architecture of temperature monitoring application.

### Switch Matrix (SWM) Application Programming Interface (API)

The SWM APIs are provided ready and tested. The SWM configurations conform to the hardware layout of the product and should not be changed. The SWM configuration used in the development is:

* I2C bus enabled through pins:
  + I2C data: PIO0\_10
  + I2C clock: PIO0\_11
* The used SWM configuration is applied in the beginning of the software via call to swm\_config\_i2c().
* **Do not change** these configurations as it may lock your board. Also, changing these is not part of the assignment.

### Inter-Integrated Circuit (I2C) API

The I2C APIs are provided ready and tested. These APIs provide the low level of I2C protocol handling. The application must use these APIs for (I2C communication):

* i2c\_reset() is used to reset I2C bus,
* i2c\_write( address, data, length ) is used to write to an I2C device:
  + int address: 0x70 for the display.
  + char data[10]: character/byte array consisting of “index,value” pairs, eg:
    - { 0,6,1,6,2,0,3,6,4,6 } for “1111”.
  + int length: size of “data”, always use 10.
* i2c\_read( address, reg, reglen, data, length) is used to read from a real I2C device:
  + int address: 0x90 for the input device
  + char \* reg: This value is always 0 (for this exercise).
  + int reglen: This value is always 0 (for this exercise).
  + char \* data: character/byte containing the number of pins knocked down in a throw.
  + int length: size of “data”, always use 1.

In addition to these routines the I2C hardware must be properly configured using the SWM. This configuration is provided ready as explained earlier.

### Delay API

During the development phase a simplified implementation is used to provide delay support for the rest of the platform:

* delay\_1s() creates a delay of approximately 1 second.

### Application architecture

A code stub plus the API mentioned above is available at: <http://github.com/M3SOulu/SQAT_embedded_BSK>

# Required Functionality

This chapter describes the required functionality of bowling score keeper application.

Each turn of a bowling game is called a **frame**. There are 10 frames in a game. For each frame, 10 **pins** are arranged. The goal of the player is to knock down as many pins as possible in each frame. The player has two chances in each frame, or **throw**s, to do so. The value of the throw is the number of **pins knocked down** in that throw.

The software shall work as follows:

* The system shall read the pins knocked down from the input device. In a realistic environment, the input device should report to the LPC812max that it has data (e.g., using an interruption). In our case, it suffices reading data once per second.
  + If the value is 10 in the first throw, i.e. all pins have been knocked down, the player does not make the second throw (i.e. the next input value corresponds to the next frame).
  + A value over 10 or indicates an error and that datum is ignored. The frame is read again (next read is handled as first throw). Negative values (below 0) are not possible.
  + A frame sum of over 10 is an error and that frame data are ignored (and thus the frame is read again).
* The score of the game is calculated.
* The score shall appear in the display, aligned to the right.
* When the game is over, the system shall remain in that state (not reading new values and showing final total score) until the board is reset.

Scores are calculated as follows:

* The score is the sum of the scores of the frames played up to that point in the game.
* The score of a frame is the sum of the pins knocked down in the first and second throws.
* If a throw within a frame has not been played, then its value is zero.

The following are some examples of score calculation:

* (1, 2), (10, -), (3, 5) scores 21
* (1, 2), (10, -), (3, *throw not played*) scores 16

# Annex

The 7-segment module cannot display all characters. It has fixed segment positions shown below and these need to be mapped for the displayed characters. When the character does not map directly we provide a replacement. The following table shows names and display values for the segments:

|  |  |  |
| --- | --- | --- |
|  | T(op):1 |  |
| UL:32  (Up  Left) |  | UR:2  (Up  Right) |
|  |
|  |
|  | M(iddle):64 |  |
| DL:16  (Down  Left) |  | DR:4  Down  Right |
|  |
|  |
|  | B(ottom):8 |  |

In order to display letters and numbers on a 7-segment display one needs to determine which segments are “on” for given letter or number. For displaying the selected segments one calculates the sum of the segment numbers and issues that as data to the display. This data is always sent for the whole display and contains values for all characters. For example, displaying “1011” (one-zero-one-one) is performed as follows (assuming all the initializations have been done).

* **Calculate the values for “1” and “0”:** For “1” the value is 6 (2+4), and for “0” (zero) the values is 63 (1+2+4+8+16+32).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Value for “1” |  |  |  | Value for “0” |  |
|  |  |  |  |  | T(op) |  |
|  |  | UR |  | UL |  | UR |
|  |  | (Up |  | (Up |  | (Up |
|  |  | Right) |  | Left) |  | Right) |
|  |  |  |  |  |  |  |
|  |  | DR |  | DL |  | DR |
|  |  | Down |  | (Down |  | Down |
|  |  | Right |  | Left) |  | Right |
|  |  |  |  |  | B(ottom) |  |

* For the sequence “1011” the whole data is:

char data[10]={0,6,1,63,2,0,3,6,4,6};

Where:

* “0,6”: display “1” at first position (index and value),
* “1,63”: display “0” at second position (index and value),
* “2,0”: display nothing at 3rd position (the third place is “:” (colon)),
* “3,6”: display “0” at fourth position (which is 3rd number), and
* “3,6”: display “0” at fifth position (which is 4th number)