Practical 3: I2C and PWM

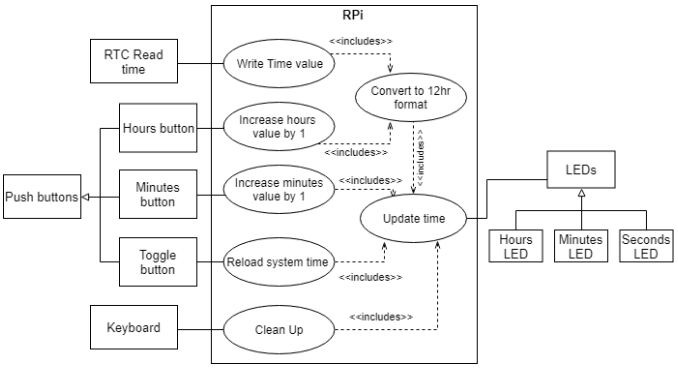
Ronak Mehta[†] and Vikyle Naidoo[‡]
EEE3096S Class of 2019
University of Cape Town
South Africa
[†]MHTRON001 [‡]NDXVIK005

***Abstract*—This practical was aimed at understanding I2C (Inter- Integrated Circuit) and PWM (Pulse Width Modulation) using the Wiring Pi configuration**

I. Introduction

It is often hard to hold the system time correctly in a Raspberry Pi since it does not have an RTC (Real Time Clock). This practical was hence designed to interface with the RTC using I2C and set the time using buttons and interrupts. It created a modified version of a binary clock which gave out the time in hours, minutes and seconds in terms of a binary counter represented by a number of LEDs. It also used buttons to fetch and write back time values to the RTC using interrupts and debouncing.

II. UML USE-CASE DIAGRAM



III. I2C COMMUNICATION USING WIRING PI

The I2C bus consists of two bidirectional open drain lines which are pulled up with resistors. These lines are Serial Data Line (SDA) and Serial Clock Line (SCL). Wiring Pi includes a library which makes it easier to use Raspberry Pis on-board I2C interface.

A. Initialization:

- To use the Wiring Pi GPIO library, *#include wiringPi.h* header file must be included in the program.
- To use the I2C library in Wiring Pi, *#include wiringPiI2C.h* header file must be included in the program.
- To initialize the I2C system with a given device identifier, this command is used: *int wiringPiI2CSetup(int devId);*

B. Send Data:

- To send data to other devices, it is required to use the Write command. A simple device write will send data to the device without accessing any internal registers. The command for this is:
  int wiringPiI2CWrite (int fd, int data);
- To send data into a desired device register, this command is used instead:
  int wiringPiI2CWriteReg8 (int fd, int reg, int data);
  The above command writes an 8-bit data value into the desired register.

C. Receive Data:

- To receive data from other devices, it is required to use the Read command. A simple device read will receive data from a device without using any registers. The command for this is:
  int wiringPiI2CRead (int fd);
- To receive data from a desired device register, this command is used instead:
  int wiringPiI2CReadReg8 (int fd, int reg);
  The above command reads an 8-bit data value from the desired register.

```
#include <wiringPi.h>                    //To use Wiring Pi GPIO library
#include <wiringPiI2C.h>                 // To use I2C library

int RTC, MM, mins;
const char RTCMIN = 0x01;                //Address of RTC minutes register
const char RTCAddr = 0x6f;               //Hardware address of the slave

RTC = wiringPiI2CSetup(RTCAddr);         //Initialize and set up RTC
wiringPiI2CWriteReg8(RTC, RTCMIN, MM);   //Writes 8-bit MM value into RTCMIN register
mins = wiringPiI2CReadReg8(RTC, RTCMIN); //Reads 8-bit value from RTCMIN register
```

Fig. 1: Some commands used for this practical from the BinClock.c file indicating the initialization, sending and receiving of data

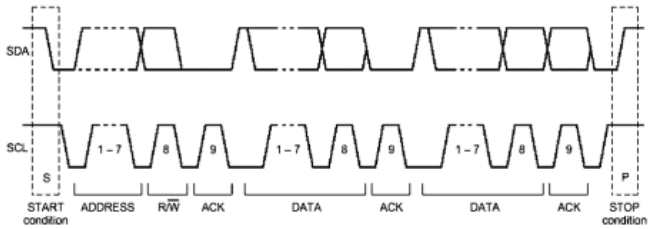


Fig. 2: General timing diagram for reading/writing data [1]

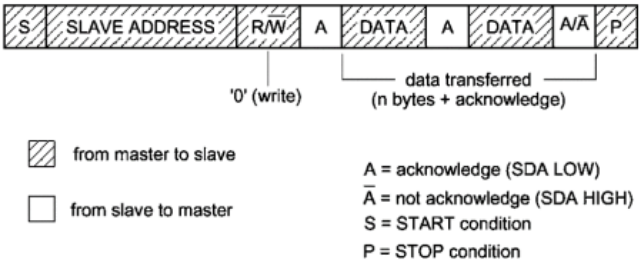


Fig. 3: Summary for Writing data[1]

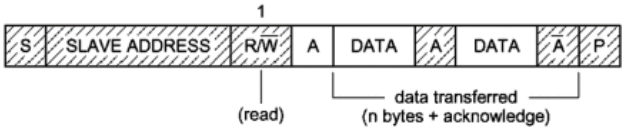


Fig. 4: Summary for Reading data[1]

IV. INTERRUPTS AND DEBOUNCING

An *interrupt* is a signal which alerts the microprocessor (RPi in our case) to a high priority condition indicating that an event needs immediate attention. There are two types of interrupts namely hardware and software interrupts. Whenever an interrupt occurs, the system completes the execution of the current instruction and starts the execution of an Interrupt handler.

Advantage: Since mostly all embedded systems have a real-time based application, it is important to call an interrupt and stop the already running instruction in case of a rise of an immediate situation which needs to be taken care of. For example, if one is working with a temperature sensor and monitoring its behavior, it is important to call for an interrupt if there is a sudden temperature rise or fall which might affect the purpose of the system. For this practical, a cleanup function for Wiring Pi was written that would be caught up and executed upon a keyboard interrupt.

Debouncing is a process which removes ripples of signal from a single button/key press and hence provides a clean transition to the output. There are two types of debouncing namely hardware and software debouncing. In this practical, in order to save space, it will be using software debouncing.

Advantage: Debouncing helps to ensure that only a single signal is accounted for, when an electrical contact has been established. For example, when pressing the push button to increase either the hour or minute value, it is expected to just increase by one when using debouncing. This can be ensured by calling for an interrupt which might say that when a difference between interruptTime and lastInterruptTime is faster than 200ms, then it can be assumed as a bounce and be ignored.

V. CIRCUIT DIAGRAM

Below is the schematic detailing the Raspberry Pi connections for the Binary clock. This practical also made use of buttons and LEDs as input and output peripherals.

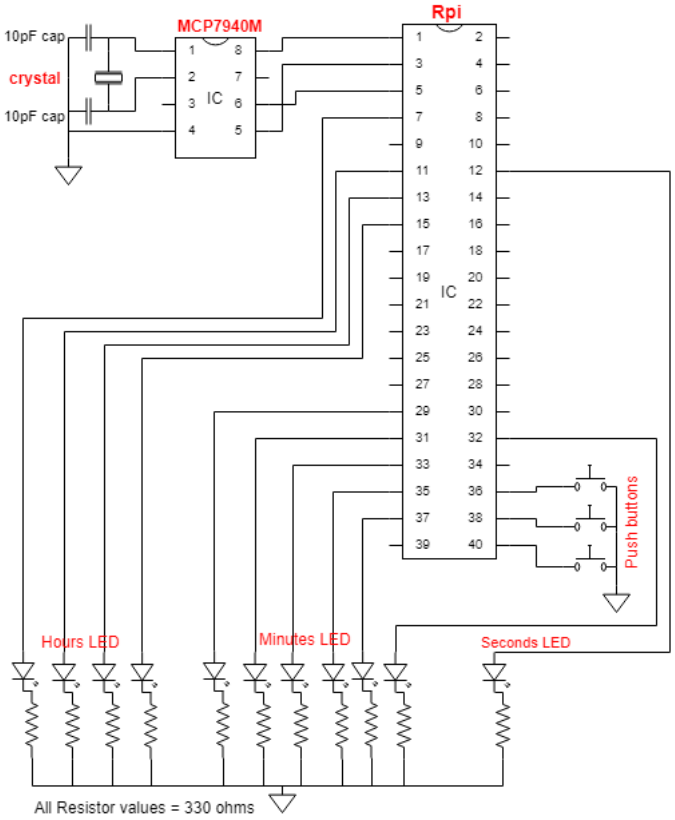


Fig. 5: Schematic detailing RPi connections for Binary Clock

REFERENCES

[1] maxEmbedded: I2C Basics
http://maxembedded.com/2014/02/inter-integrated-circuits-i2c-basics/

[2] Vula briefing
Prac3 Document

[3] Keegan Github
https://github.com/kcranky/EEE3096S