Embedded System Design

Professor Ning

March 3, 2025

Digital Watch Design using 8051 and Stopwatch

Lab 1

By: Sean Balbale

and

Poojaa Prakash Babu

Introduction

The *8051* is a microcontroller developed by Intel which will serve a vital role in the making of our embedded system. In this lab, we are building an embedded system using the *8051* microcontrollers along with a demultiplexer, 2K *EEPROM*, pushbuttons, a hex inverter, binary-coded decimal (BCD), and a latch to display a clock counting up on a 7-segment LED display.

The lab consists of two components. The first part involves wiring the embedded system according to a proper schematic diagram, followed by writing an Assembly program to implement the functionality. The program will implement interrupts and the concept of timer overflow to execute the necessary functions needed for this lab. The program will make the clock count up and reset when the pushbutton is pressed.

In the second part, additional resistors, capacitors, and pushbuttons will be incorporated to implement three scenarios:

1. When the pushbutton is pressed once, the clock will count up (first part of the lab).
2. When pressed twice, the system will function as a stopwatch, allowing it to run.
3. When pressed a third time, the stopwatch will pause, displaying the time at which it was stopped.

Another pushbutton will serve as a reset for both the clock and the stopwatch.

This lab report will cover both sections of the lab, discussing the hardware and software design aspects in detail.

Problem Statement

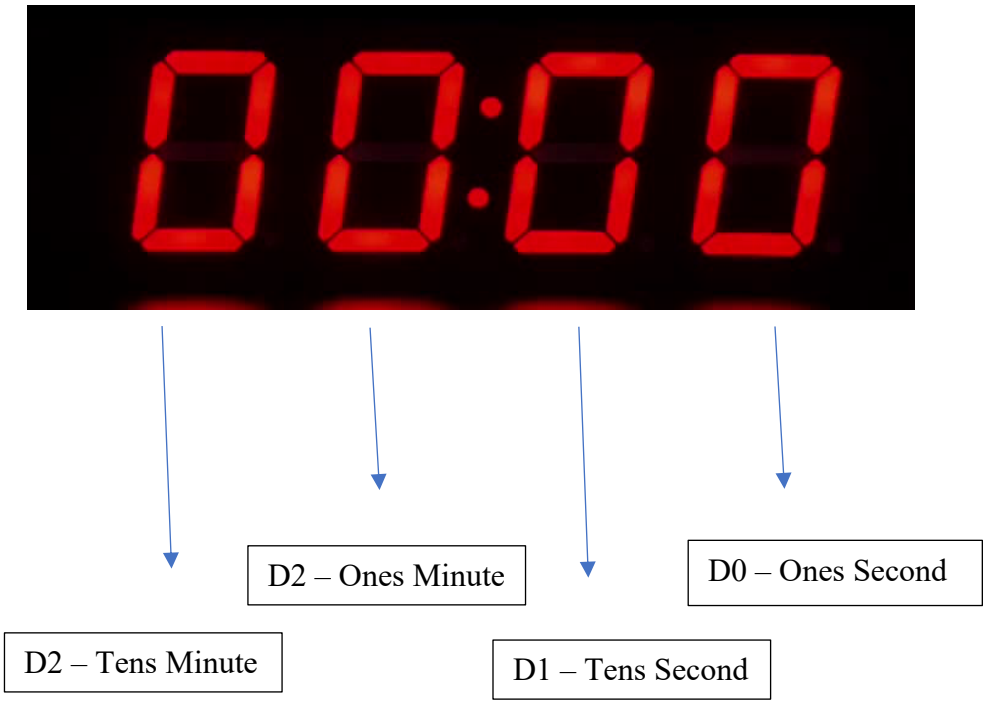
Implement a clock counting up on the display:

The embedded system, once fully wired and flashed with the Assembly code on the *EEPROM*, will display four digits on the 7-segment LED display. The rightmost digit represents seconds

(ones place), the second digit represents tens of seconds, the third digit represents minutes (ones place), and the fourth digit represents tens of minutes, following the format (00:00).

The logic follows a sequential counting pattern. When the ones place of the seconds counter increments from 0 to 9, it rolls back to 0 while increasing the tens place of the seconds counter by 1. This pattern continues until the display reaches 00:59, at which point it resets to 01:00, as there are 60 seconds in a minute. The minutes counter follows a similar process, counting up to 09:59 before transitioning to 10:00, incrementing the tens place of the minutes counter. The highest possible time display is 59:59, as this system does not include an hour digit framework.

Figure 1. Digit and their reference to value.



The details above apply to Part 1 of the lab, where the clock counts up. However, in Part 2, which introduces the stopwatch functionality for starting and stopping, the same digit layout logic from Part 1 is maintained. Additional iterations in the code are implemented to distinguish between the clock and stopwatch displays.

Specific Design Goal

Implementing a Clock and Stopwatch in an Embedded System

The objective of this project is to develop a wired embedded system capable of displaying digits that count up in a clock format while also implementing a stopwatch. This is achieved through the integration of hardware components and assembly code. Our design goal is structured into four key parts:

1. **Hardware Design and Component Integration**
   The first step is understanding all the components on the board and correctly wiring them according to the schematic. This involves integrating the *8051* microcontroller, *EEPROM*, demultiplexer, BCD system, pushbuttons, and other essential hardware to ensure proper display functionality. The demultiplexer helps with multiplexing, allowing fewer pins to control multiple 7-segment displays, while the 2K *EEPROM* ensures data retention when power is turned off. The pushbuttons serve as input controls for resetting and managing the clock.
2. **Assembly Code Implementation Using Timer Overflow**
   The second focus is writing assembly code that operates based on a timer overflow. This ensures the digits increment and roll over correctly as they transition from D0 to D3, maintaining an accurate clock counting mechanism. The *8051* microcontroller outputs Binary Coded Decimal (BCD) values, which are then fed into a BCD to 7-segment decoder, ensuring the correct digits are displayed.
3. **Software and Hardware Integration for Real-Time Execution**
   The third goal is the seamless coupling of software and hardware. This involves flashing the assembly code onto the *EEPROM*, managing external interrupts, and ensuring that digit increments occur correctly. The system must run at the same speed as an actual clock, which serves as a verification method. Proper interrupt handling is critical to ensuring smooth operation without delays or inconsistencies.
4. **Stopwatch Implementation with External Interrupts**
   The final goal is adding a stopwatch function that operates independently without

interfering with the clock. The stopwatch must be able to start, stop, and reset as needed. Both the clock and stopwatch require precise external interrupt handling, ensuring they function correctly without overlapping. The stopwatch should not disrupt the clock's counting process, and both systems should operate smoothly within the embedded system.

Design Solution

Hardware Design Strategy

The hardware for the 7-segment display system is wired in a structured manner to ensure efficient operation. The *8051* microcontroller serves as the central unit, processing inputs from pushbuttons and controlling the display output. It sends BCD values to a BCD to 7-segment decoder (*7447*), which converts them into corresponding signals for the 7-segment display. To manage multiple digits, a demultiplexer (*74HC138*) is used, allowing the microcontroller to select which digit is active at any given time. A latch (*74HC573*) is incorporated to hold display data, ensuring stability and preventing flickering. The hex inverter (*74HC04*) is used where necessary to invert logic signals, particularly for components that require active-low inputs. The system also includes a 2K *EEPROM* for storing persistent data such as clock settings, which allows the *8051* microcontrollers to retrieve previously saved values upon power-up. Pushbuttons are connected to provide user input for starting, stopping, and resetting the clock or stopwatch, with debouncing handled either in hardware or software. All these components work together to drive the 7-segment LED display, ensuring that the time is displayed accurately and updated in real-time based on user interactions.

Functionality	**Chips Implemented**
Control	2816 EEPROM
Control	74573 D-type Latch
Control	8051 Microcontroller
Display	7447 BCD to 7 Segment Display
Display	74138 3x8 Demultiplexer
Display	7404 Hex Inverter

Display	LED 7-Segmenent Display

Table 1. Hardware chips required for system Design.

Type	**Components**
Hardware	12 MHz Crystal
Hardware	Capacitors
Hardware	Resistors
Hardware	Push Button
Software	WinDraft
Software	Keil µVision

Table 1. Remaining components needed to build the Embedded system in hardware and software.

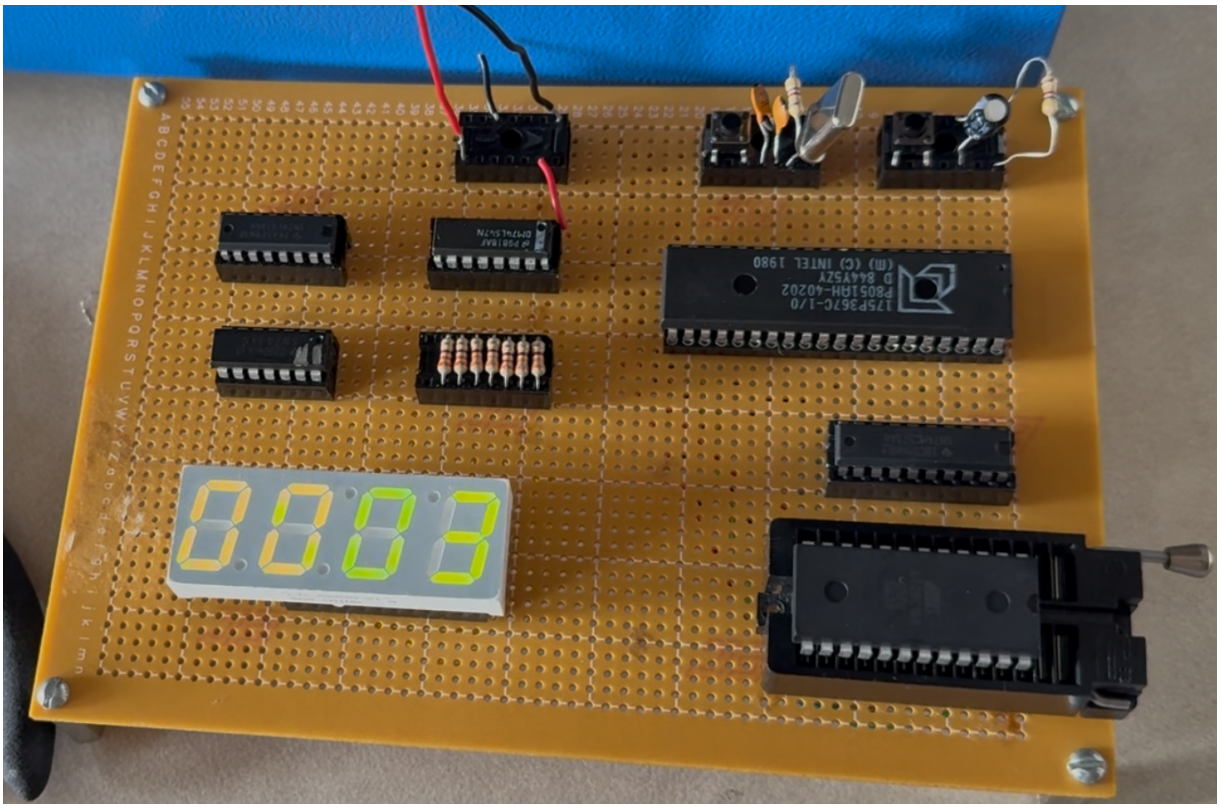


Figure 2. The embedded system of the clock and stopwatch on the 7-segmeent Display.

To effectively implement the *8051* microcontrollers, one must understand its hardware architecture to correctly address memory locations and allocate logic functions accordingly. The

8051 microcontroller consists of several key components, including address buses, data buses, internal and external memory, special function registers (SFRs), and interrupt service routines (ISRs).

Memory Architecture of the 8051

1. Address Bus (16-bit)
    - The *8051* has a 16-bit address bus, allowing it to access up to 64 KB of external memory.
    - It supports both internal and external memory for data and code storage.
2. Data Bus (8-bit)
    - The 8-bit data bus is used to transfer data between internal/external memory and the CPU.
3. Internal Memory (256 Bytes of RAM)
    - The first 128 bytes (00h to 7Fh) of RAM are divided into three sections:
        - 00h to 1Fh: Register banks (4 banks, each with 8 registers).
        - 20h to 2Fh: Bit-addressable RAM (can be accessed bit by bit).
        - 30h to 7Fh: General-purpose RAM.
    - The next 128 bytes (80h to FFh) consist of Special Function Registers (SFRs), which control:
        - Timers
        - Serial communication
        - Interrupts
        - I/O ports
4. External Memory
    - External Data Memory (RAM - 64 KB)
        - Read/Write memory, used for data storage.
    - External Code Memory (ROM - 64 KB)
        - Typically Read-Only during normal operation.
        - It can be programmed under special conditions (Flash RAM type or PROM).

Interrupt System in 8051

The *8051* microcontroller has five interrupt sources, each mapped to a fixed memory address for their Interrupt Service Routines (ISRs). These include:

1. External Interrupt 0 (EX0)
2. Timer 0 Overflow (TF0)
3. External Interrupt 1 (EX1)
4. Timer 1 Overflow (TF1)
5. Serial Communication Interrupt (RI/TI)

These interrupts allow the microcontroller to handle external signals, timer overflows, and serial communication events efficiently.

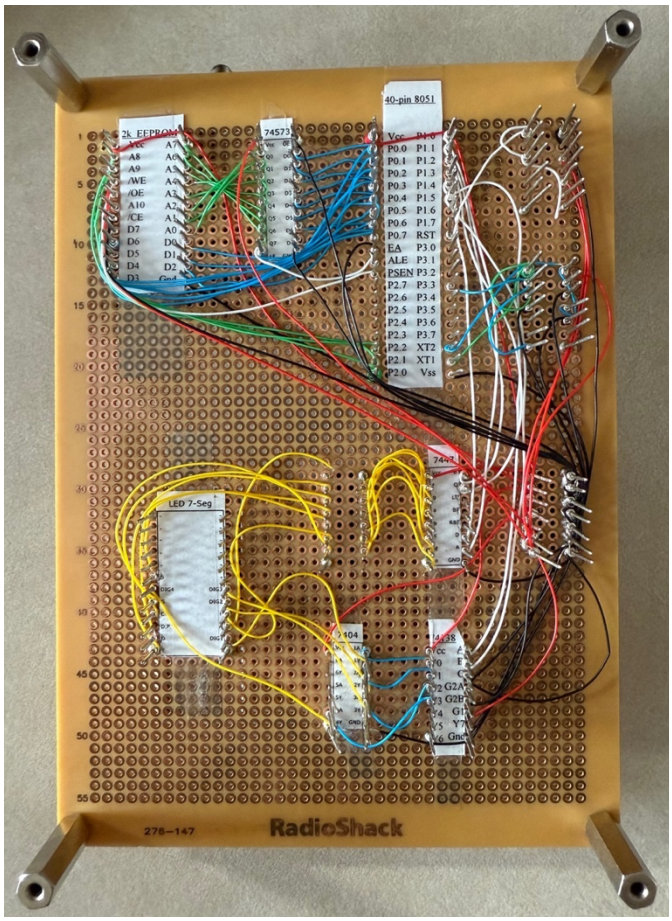


Figure 3. The wiring of the chips and components in the back of the embedded system.

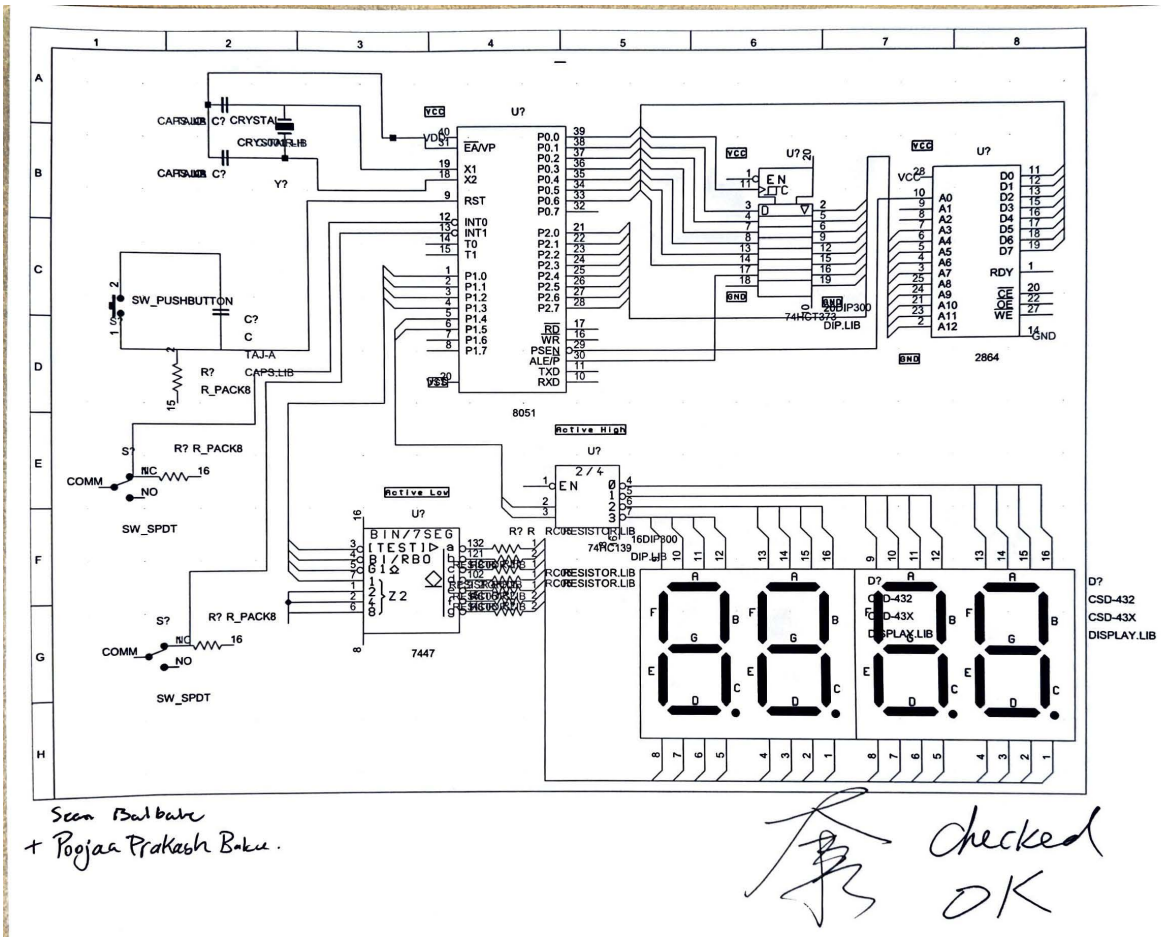


Figure 4. The wiring diagram for the embedded system.

Implementation of Stopwatch in Part 2 of Lab 1

To enable the embedded system in Part 2 of the lab to handle both a clock and a stopwatch, we needed to implement an additional push button alongside the original reset button in the schematic. This allowed us to toggle between three modes: 0 = clock, 1 = run stopwatch, and 2 = stop stopwatch.

To achieve this, we incorporated a push button with a resistor and configured it as a falling-edge trigger, meaning the logic transitions from high to low. To properly integrate this functionality, we designed a small circuit that connects to port 3.3 of the *8051* microcontroller, as this port is responsible for enabling external interrupt 1 in the hardware design, ensuring compatibility with the software implementation.

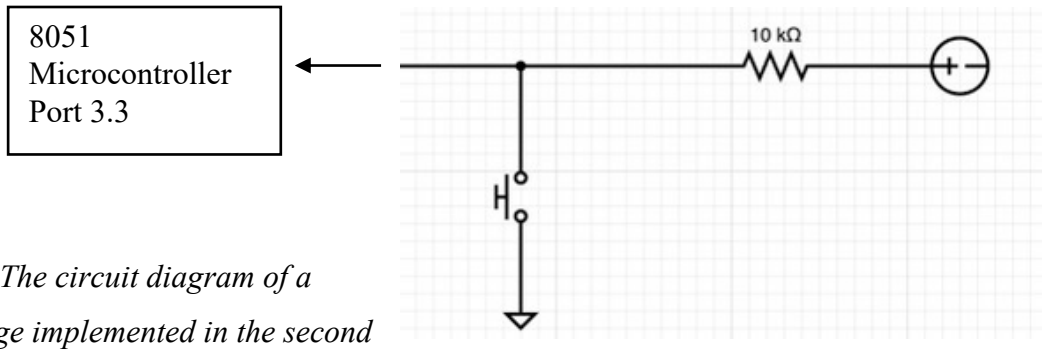


Figure 5. The circuit diagram of a falling edge implemented in the second push button.

Software Design Strategy

Part one assembly code breakdown

This lab report is divided into two main sections, corresponding to the first and second parts of the lab. The first portion focuses on implementing a stopwatch that displays four digits on a 7-segment display, counting up from 00:00 in seconds, tens of seconds, minutes, and tens of minutes. Pressing the reset button returns the clock to 00:00.

The code initializes time registers and counters by setting up registers to store time values for each digit of the display:

```
MOV     R0, #00h      ; 1-second digit
MOV     R1, #00h      ; 10-seconds digit
MOV     R2, #00h      ; 1-minute digit
MOV     R3, #00h      ; 10-minutes digit
MOV     R4, #00h      ; Display position
MOV     R5, #00h      ; Counter for 1 second (200 * 5ms = 1000ms)
```

The R5 register acts as a counter to track milliseconds, ensuring that 200 cycles of 5ms correspond to 1 second. Timer 0 is configured in 16-bit mode to generate an interrupt every 5 milliseconds, with preloaded values in TH0 and TL0 to maintain accurate timing:

```
MOV     TMOD, #01h     ; Timer 0 in mode 1 (16-bit timer)
```

```
MOV     TH0, #0ECh     ; Load high byte for 5ms interval
MOV     TL0, #078h     ; Load low byte for 5ms interval
```

The stopwatch depends on precise time intervals to function correctly. The Interrupt Service Routine (ISR) for Timer 0 is triggered every 5ms when the timer overflows:

```
Timer0_ISR:
   PUSH    ACC
   PUSH    PSW
   CLR     TR0
   MOV     TH0, #0ECh
   MOV     TL0, #08Bh
   CLR     TF0
   SETB    TR0
```

Key steps include preserving register values by pushing the accumulator and program status word onto the stack, reloading the timer values to maintain a steady 5ms interval, and restarting the timer after resetting the overflow flag (TF0). Adjustments for clock cycle delays are made to ensure precision.

The time counter update occurs when the R5 register increments with each 5ms interrupt:

```
INC     R5           ; Increment ms counter
MOV     A, R5
CJNE    A, #0C8h, Display    ; 200 * 5ms = 1 second
MOV     R5, #00h       ; Reset ms counter
```

When R5 reaches 200, indicating that 1 second has elapsed, the ISR resets it to 0 and increments the seconds register. The ones-second counter (R0) increments every second:

```
INC     R0           ; Increment ones-second
MOV     A, R0
CJNE    A, #0Ah, Display    ; If not 10
MOV     R0, #00h       ; Reset ones-second
```

When R0 reaches 10, it resets to 0, and the tens-place counter (R1) increments. Since there are 60 seconds in a minute, R1 increments until 6, then resets, and R2 (ones-minute) increments. R3 follows the same pattern for tens of minutes.

The display update process cycles through four positions, updating each digit sequentially. R4 keeps track of the current digit being updated:

```
Display:
   MOV     A, R4
   CJNE    A, #00h, Try_Pos1
   MOV     A, R3          ; Get 10 minutes
   SJMP    Output_Digit
```

The ORL (OR Logical) instruction is used for display formatting, modifying bits in a register without affecting others:

```
Try_Pos1:
   CJNE    A, #01h, Try_Pos2
   MOV     A, R2          ; Get 1 minute
   ORL     A, #10h
   SJMP    Output_Digit
```

This ensures that the correct segment is lit up to reflect the corresponding digit value. ORL A, #10h indicates the 1-minute digit, ORL A, #20h indicates the 10-seconds digit, and ORL A, #30h indicates the 1-second digit.

This code successfully implements a stopwatch with four-digit display functionality, utilizing Timer 0 interrupts for accurate time delays, incrementing registers to track time, and bitwise operations (ORL) to manage digit selection for the display.

Part two assembly code breakdown

This code implements a dual-function device that acts both as a clock and a stopwatch using a 7-segment display. The program is divided into several key sections:

Interrupt Vectors and Startup

The code begins with several ORG directives to locate different pieces of code in memory.

At address 0000h, the program jumps to the MAIN routine.

At 0013h, the external interrupt 1 (INT1) vector is set to jump to EXT1_ISR.

At 000Bh, the Timer 0 vector is directed to Timer0_ISR.

The main code starts at address 0100h.

System Initialization and Register Setup

Stack Pointer & Timer Mode:

In the MAIN routine, the stack pointer is initialized (MOV SP, #30h) and Timer 0 is configured in mode 1 (a 16-bit timer) with the instruction MOV TMOD, #01h.

Stopwatch Registers (Bank 0):

The code then clears the bank select bits (using CLR RS0 and CLR RS1) to ensure that the registers in Bank 0 are used for the stopwatch. The following registers are initialized:

R0: 1/100 second digit

R1: 1/10 second digit

R2: seconds digit

R3: 10 seconds digit

R4: Display position counter

R5: A counter to track 5ms intervals

Clock Registers (Bank 1):

By setting RS0 (SETB RS0), the code switches to Bank 1 where the clock digits are stored:

R0: seconds ones

R1: seconds tens

R2: minutes ones

R3: minutes tens

Operating Mode:

Back in Bank 0 (CLR RS0), the memory location at address 20h is set to 00h, which represents the default mode (with 0 for Clock, 1 for Run/Stopwatch, and 2 for Stop).

Interrupt Setup and Timer Configuration

External Interrupt 1:

The code configures External Interrupt 1 to be falling edge–triggered (SETB IT1) and enables it (SETB EX1).

Timer 0 Configuration:

Timer 0 is preloaded with values (TH0 and TL0) to generate an interrupt every 5 milliseconds. Interrupts are enabled (SETB ET0 and SETB EA), and the timer is started (SETB TR0).

Main Loop

The main loop (MainLoop) is simply an infinite jump (SJMP MainLoop) because all timing and display updates are handled in the interrupt service routines (ISRs).

Timer 0 Interrupt Service Routine (Timer0_ISR)

Preservation of Registers:

At the beginning of the ISR, the accumulator (ACC) and the program status word (PSW) are pushed onto the stack to preserve their values.

Timer Reloading for a 5ms Interval:

The timer is stopped (CLR TR0), reloaded with preset values (e.g., MOV TH0, #0ECh and MOV TL0, #08Ch), and restarted after clearing the overflow flag (CLR TF0).

Time Base Management:

Clock Update:

The register R5 is incremented on every 5ms interrupt. When R5 reaches 200 (since 200 × 5ms = 1 second), it resets to 0 and the code switches to Bank 1 to update the clock digits.

Seconds (ones) are incremented, and if they reach 10, they reset and the tens-place counter is updated.

Similar logic is applied for minutes (ones and tens), with seconds tens resetting at 6 (60 seconds) and minutes tens resetting at 6 (60 minutes).

Stopwatch Update:

If the current mode (stored at memory location 20h) is set to Run (value 1), the stopwatch digits in Bank 0 are updated. The update is designed to occur every two 5ms intervals (i.e. every 10ms) by checking if the current interval count is even.

The digits are incremented in a cascading manner: 1/100 seconds, then 1/10 seconds, seconds, and finally 10 seconds, with appropriate resets after reaching their respective limits.

Display Update Logic:

After time counters are updated, the code selects which digit to display based on the current mode and display position (tracked by register R4).

For the clock mode, the code switches temporarily to Bank 1 to fetch the appropriate digit. It uses bitwise OR operations (ORL) to combine digit values with segment selection codes (e.g., #10h, #20h, #30h).

For the stopwatch, a similar approach is used, and the display position counter is incremented (and wrapped around when necessary) to ensure that all four digits are updated sequentially on the 7-segment display.

Finalizing the ISR:

Before returning from the interrupt, the saved PSW and ACC values are restored with POP instructions, and the routine returns with RETI.

External Interrupt Service Routine (EXT1_ISR)

Mode Switching:

The external interrupt (triggered on a falling edge) is used to cycle through the operating modes.

The routine reads the current mode from memory location 20h, increments it, and if the new mode value exceeds 2, it wraps back to 0.

This new mode is then saved back to the same memory location.

Like the Timer0_ISR, the EXT1_ISR preserves the ACC and PSW by pushing them onto the stack and restores them before executing RETI.

In essence, this assembly program sets up two key timekeeping functions—a clock (using registers from Bank 1) and a stopwatch (using registers from Bank 0). Timer 0 generates a 5ms interrupt that updates a counter; every 200 counts (1 second), the clock is updated, and at shorter intervals, the stopwatch is updated if in Run mode. The display is refreshed digit by digit by cycling through positions with the help of bitwise operations. Additionally, an external interrupt allows the user to cycle through modes (clock, run, stop), making this a versatile timekeeping application on an 8051 microcontroller.

This thorough approach ensures precise time delays, effective use of register banks for different functions, and efficient interrupt-driven updates to the display.

Implementation

Software Issues

The count-up function initially ran too fast because the inherent delays caused by clock cycles and machine cycles were not taken into account. To correct this, the timing calculations were adjusted, factoring in these cycles, and the time delay was modified from 078h to 08Bh to achieve the desired speed.

Another issue arose with the stopwatch function, which consistently started at 41 minutes. This indicated a problem with data retention in the registers. The root cause was that the clock values were stored in addresses like 30h, 31h, but they were not retaining the values correctly. To resolve this, two different register banks were initialized: Bank 1 for clock settings and Bank 0 for stopwatch settings. This change ensured that values were retained correctly, ultimately leading to a successful embedded system implementation.

Hardware Issues

After flashing the *EEPROM* and reinserting it into the *8051*-microcontroller board, the digital clock lights began flickering, which indicated loose wiring. To resolve this, all connections on the hardware board were thoroughly rewired and secured.

Another major issue arose when attempting to add a second external input for the stopwatch. The lack of space in the buses made it difficult to accommodate the additional push button. The initial wiring for the push buttons, capacitors, and resistors was too spread out, preventing any additional connections. To create space, some capacitors and resistors were relocated onto a single bus, and the wiring was adjusted to ensure proper integration. The final connections included one push button connected to Port P3.3 and the second push button connected to the Reset pin.

This refined implementation resolved both software and hardware challenges, ensuring the correct functionality of the system.

Results and Discussion

In this lab, we gained hands-on experience in wiring an embedded system, flashing an *EEPROM*, and writing Assembly language code. We successfully built an embedded system capable of displaying a clock, implementing a counting mechanism, and incorporating a reset function. Additionally, we added a stopwatch feature that could start, stop, and reset. To verify the accuracy of our system, we compared it to an actual clock and stopwatch, confirming that it maintained precise timekeeping.

A critical takeaway from this lab was understanding how Interrupt Service Routines (ISRs) function in an embedded system. We explored when ISRs are assigned specific addresses, when bit addressing is necessary, and when we have flexibility in address selection. Additionally, timer overflow played a crucial role in structuring the logic of our code, determining the appropriate moments to trigger an interrupt and resume execution. These aspects were fundamental to ensuring smooth digit transitions in the clock display.

One of the biggest challenges we faced was stabilizing the 7-segment display output, which required careful attention to wiring. Loose connections caused erratic display behavior, reinforcing the importance of secure wiring, proper resistor selection, and effective register bank management. Additionally, timing precision was essential, as setting the correct TLO and THO values was critical for ensuring consistent clock performance. Incorrect time cycle calculations led to timing inaccuracies, requiring iterative debugging and adjustments to achieve reliable functionality.

Furthermore, we encountered challenges in synchronizing the stopwatch function with the clock display, as precise timing intervals were required to ensure smooth transitions between digits. The use of interrupts allowed us to control execution flow effectively, ensuring the clock and stopwatch operated without interfering with each other.

Overall, this lab provided a deeper understanding of embedded system design, ISR handling, timer-based logic, and debugging techniques, all of which are crucial for developing real-time embedded applications.

References

https://www.adafruit.com/product/3108

https://moodle.trincoll.edu/pluginfile.php/702733/mod_resource/content/1/8051%20tutorial.PDF

Appendix

Code 1:

```
ORG     0000h
AJMP    MAIN

ORG     000Bh
AJMP    Timer0_ISR

ORG     0030h
MAIN:       MOV     SP, #30h
MOV     TMOD, #01h

; Timer 0 setup for 5ms
MOV     TH0, #0ECh
MOV     TL0, #078h
```

```
; Initialize registers for time values
MOV     R0, #00h      ; 1 second digit
MOV     R1, #00h      ; 10 seconds digit
MOV     R2, #00h      ; 1 minute digit
MOV     R3, #00h      ; 10 minutes digit
MOV     R4, #00h      ; Display position
MOV     R5, #00h      ; Counter for 1 second (200 * 5ms = 1000ms)

; Enable interrupts
SETB    ET0
SETB    EA
SETB    TR0

MainLoop:   SJMP    MainLoop

;this loop takes aproximatly 70 machine cycles. Timer value might need to be adjusted to 0xECh, 0xBDh.
; default value is 0xECh, 0x77h
Timer0_ISR: PUSH    ACC
PUSH    PSW

; Reload timer
CLR     TR0
MOV     TH0, #0ECh
```

```
MOV     TL0, #08Bh
CLR     TF0
SETB    TR0

; Update time counter
INC     R5              ; Increment ms counter
MOV     A, R5
CJNE    A, #0C8h, Display    ; 200 * 5ms = 1 second
MOV     R5, #00h        ; Reset ms counter

; Update seconds
INC     R0              ; Increment ones second
MOV     A, R0
CJNE    A, #0Ah, Display    ; If not 10
MOV     R0, #00h        ; Reset ones second

; Update tens seconds
INC     R1              ; Increment tens seconds
MOV     A, R1
CJNE    A, #06h, Display    ; If not 60 seconds
MOV     R1, #00h        ; Reset tens seconds

; Update minutes
```

```
INC     R2              ; Increment minutes
MOV     A, R2
CJNE    A, #0Ah, Display    ; If not 10 minutes
MOV     R2, #00h        ; Reset minutes

; Update tens minutes
INC     R3              ; Increment tens minutes
MOV     A, R3
CJNE    A, #06h, Display    ; If not 60 minutes
MOV     R3, #00h        ; Reset tens minutes

Display:    ; Get current position and display digit
MOV     A, R4
CJNE    A, #00h, Try_Pos1
MOV     A, R3           ; Get 10 minutes
SJMP    Output_Digit

Try_Pos1:   CJNE    A, #01h, Try_Pos2
MOV     A, R2           ; Get 1 minute
ORL     A, #10h
SJMP    Output_Digit

Try_Pos2:   CJNE    A, #02h, Try_Pos3
```

```
MOV     A, R1          ; Get 10 seconds
ORL     A, #20h
SJMP    Output_Digit

Try_Pos3:   MOV     A, R0          ; Get 1 second
ORL     A, #30h

Output_Digit:
MOV     P1, A

; Update display position
MOV     A, R4
INC     A
CJNE    A, #04h, Save_Pos
MOV     A, #00h
Save_Pos:   MOV     R4, A

POP     PSW
POP     ACC
RETI

END
```

Code 2:

```
ORG     0000h
        AJMP    MAIN

        ORG     0013h         ; External Interrupt 1 vector
        AJMP    EXT1_ISR

        ORG     000Bh         ; Timer 0 vector
        AJMP    Timer0_ISR

        ORG     0100h
MAIN:       MOV     SP, #30h
        MOV     TMOD, #01h     ; Timer 0, mode 1 (16-bit)

        ; Initialize stopwatch registers (Bank 0)
        CLR     RS0           ; Select Bank 0
        CLR     RS1
        MOV     R0, #00h      ; 1/100 second digit
        MOV     R1, #00h      ; 1/10 second digit
        MOV     R2, #00h      ; seconds digit
        MOV     R3, #00h      ; 10 seconds digit
        MOV     R4, #00h      ; Display position
```

```
MOV     R5, #00h      ; Counter for 5ms intervals

; Initialize clock registers (Bank 1)
SETB    RS0           ; Select Bank 1
MOV     R0, #00h      ; seconds ones
MOV     R1, #00h      ; seconds tens
MOV     R2, #00h      ; minutes ones
MOV     R3, #00h      ; minutes tens

; Back to Bank 0
CLR     RS0
MOV     20h, #00h     ; Mode (0=Clock, 1=Run, 2=Stop)

; Setup External Interrupt 1
SETB    IT1           ; Falling edge triggered
SETB    EX1           ; Enable INT1

; Timer 0 setup for 5ms
MOV     TH0, #0ECh
MOV     TL0, #078h
SETB    ET0           ; Enable Timer 0
SETB    EA            ; Enable global interrupts
SETB    TR0           ; Start Timer 0
```

```
MainLoop:   SJMP    MainLoop      ; Everything handled by interrupts

Timer0_ISR: PUSH    ACC
            PUSH    PSW

            ; Reload timer for next 5ms
            CLR     TR0
            MOV     TH0, #0ECh
            MOV     TL0, #08Ch
            CLR     TF0
            SETB    TR0

            ; Update clock every 200 intervals (1 second)
            INC     R5
            MOV     A, R5
            CJNE    A, #0C8h, Update_Stopwatch  ; 200 * 5ms = 1 second
            MOV     R5, #00h

            ; Switch to Bank 1 for clock update
            SETB    RS0

            ; Update clock seconds ones
```

```
INC     R0
MOV     A, R0
CJNE    A, #0Ah, Switch_Bank_0
MOV     R0, #00h

; Update clock seconds tens
INC     R1
MOV     A, R1
CJNE    A, #06h, Switch_Bank_0
MOV     R1, #00h

; Update clock minutes ones
INC     R2
MOV     A, R2
CJNE    A, #0Ah, Switch_Bank_0
MOV     R2, #00h

; Update clock minutes tens
INC     R3
MOV     A, R3
CJNE    A, #06h, Switch_Bank_0
MOV     R3, #00h
```

```
Switch_Bank_0:
        CLR     RS0         ; Switch back to Bank 0

Update_Stopwatch:
        MOV     A, 20h      ; Check mode
        CJNE    A, #01h, Display_Update

        ; Update stopwatch every 2 intervals (10ms)
        MOV     A, R5
        ANL     A, #01h     ; Check if even number
        JNZ     Display_Update

        ; Update stopwatch digits (Bank 0)
        INC     R0          ; 1/100 seconds
        MOV     A, R0
        CJNE    A, #0Ah, Display_Update
        MOV     R0, #00h

        INC     R1          ; 1/10 seconds
        MOV     A, R1
        CJNE    A, #0Ah, Display_Update
        MOV     R1, #00h
```

```
        INC     R2          ; Seconds
        MOV     A, R2
        CJNE    A, #0Ah, Display_Update
        MOV     R2, #00h

        INC     R3          ; 10 seconds
        MOV     A, R3
        CJNE    A, #06h, Display_Update
        MOV     R3, #00h

Display_Update:
        ; Select display based on mode
        MOV     A, 20h
        JNZ     Show_Stopwatch

Show_Clock: MOV     A, R4
        SETB    RS0         ; Switch to Bank 1
        CJNE    A, #00h, Clock_Pos1
        MOV     A, R3       ; Minutes tens
        CLR     RS0         ; Back to Bank 0
        SJMP    Output_Digit

Clock_Pos1: CJNE    A, #01h, Clock_Pos2
```

```
        MOV     A, R2         ; Minutes ones
        CLR     RS0
        ORL     A, #10h
        SJMP    Output_Digit

Clock_Pos2: CJNE    A, #02h, Clock_Pos3
        MOV     A, R1         ; Seconds tens
        CLR     RS0
        ORL     A, #20h
        SJMP    Output_Digit

Clock_Pos3: MOV     A, R0         ; Seconds ones
        CLR     RS0
        ORL     A, #30h
        SJMP    Output_Digit

Show_Stopwatch:
        MOV     A, R4
        CJNE    A, #00h, Stop_Pos1
        MOV     A, R3         ; 10 seconds
        SJMP    Output_Digit

Stop_Pos1:  CJNE    A, #01h, Stop_Pos2
```

```
        MOV     A, R2         ; Seconds
        ORL     A, #10h
        SJMP    Output_Digit

Stop_Pos2:  CJNE    A, #02h, Stop_Pos3
        MOV     A, R1         ; 1/10 second
        ORL     A, #20h
        SJMP    Output_Digit

Stop_Pos3:  MOV     A, R0         ; 1/100 second
        ORL     A, #30h

Output_Digit:
        MOV     P1, A

        ; Update display position
        MOV     A, R4
        INC     A
        CJNE    A, #04h, Save_Pos
        MOV     A, #00h
Save_Pos:   MOV     R4, A

        POP     PSW
```

```
        POP     ACC
        RETI

EXT1_ISR:   PUSH    ACC
        PUSH    PSW

        MOV     A, 20h        ; Get current mode
        INC     A             ; Next mode
        CJNE    A, #03h, Save_Mode
        MOV     A, #00h       ; Wrap to mode 0
Save_Mode:  MOV     20h, A        ; Save new mode

        POP     PSW
        POP     ACC
        RETI

        END
```