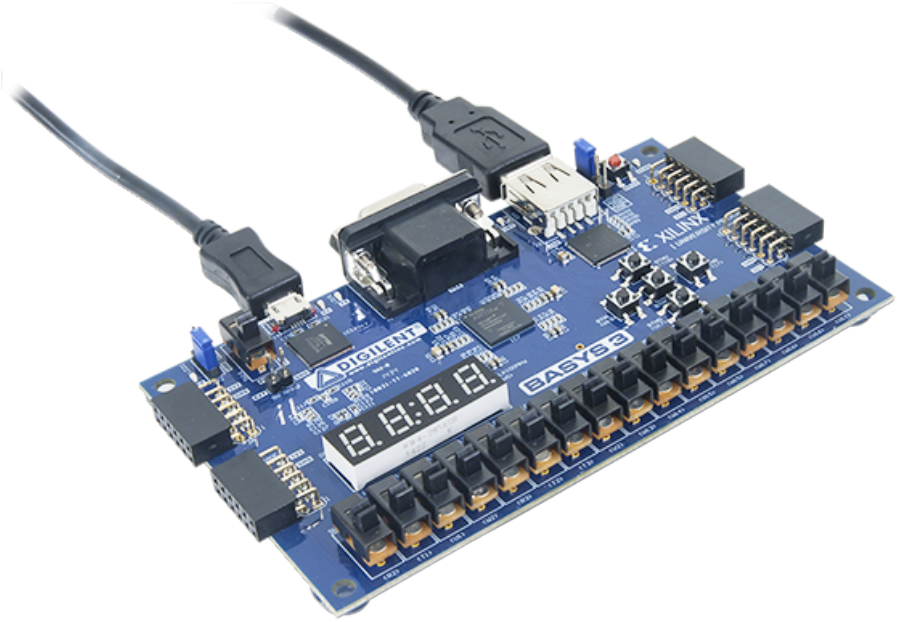


BS3 Clock - Instruction Manual

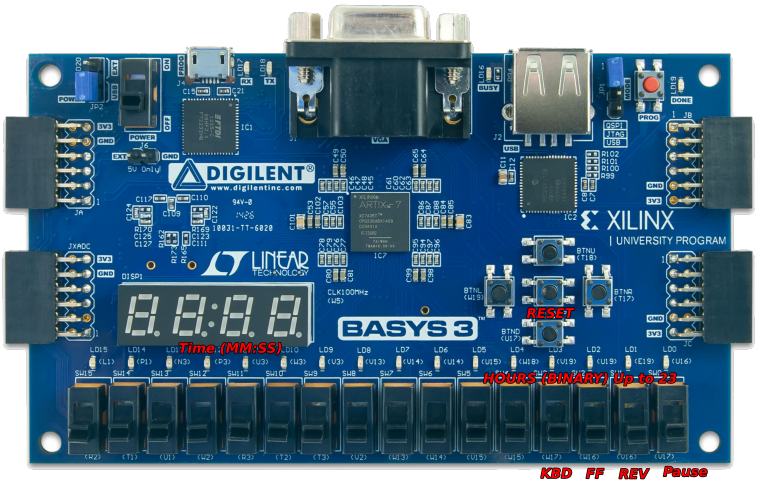
CONTENTS

For questions or contact, please visit **www.robinttw.com**

1 Introduction

The Basys-3 OTTER device is engineered to function as a digital clock, displaying time in a 24-hour format on a 7-segment LED display. This programmable device is designed for educational purposes, illustrating fundamental digital design concepts while providing practical utility. The clock's primary interface includes a 7-segment display for minutes and seconds, and LEDs to indicate hours.

Device Functionality and Specifications



- **Display Format:** Time is displayed in MM:SS format on the 7-segment display, with hours indicated by LEDs. This design choice facilitates easy reading and understanding of the current time.
- **Control Features:** The device includes switches for pausing, reversing, fast forwarding time, and setting the time manually via keyboard input. These controls introduce interactivity, allowing users to manipulate time display as needed.
- **Operating Range:** The clock operates on a 24-hour cycle, automatically resetting at midnight.
- **Precision and Accuracy:** The device checks and updates the time display every second, ensuring that the timekeeping is accurate and reliable.
- **Input and Output Values:** The range of input values is from 00:00 to 23:59 for time setting, with the output displayed accordingly on the 7-segment LED and LEDs for hours.

User Interaction and Experience

The Basys-3 OTTER digital clock is engineered to provide high precision and user-friendly interaction, underpinned by the 100 MHz operational clock speed of the Basys-3 FPGA board. This high-speed clock ensures the device's timekeeping is accurate and the interface is responsive, catering to the needs of users who require precision and efficiency.

- **Timekeeping Accuracy:** At 100 MHz, the device can precisely measure and update time, ensuring that the clock maintains accurate time to the second. This precision is crucial for reliable time tracking and display. Regular time is set to 6,000,000 cycles and the fastforward is only at 1,000,000. Regular time is slightly faster than actual time, but it is modifiable.
- **Sensor Measurements and Precision:** While the device primarily functions as a clock without traditional sensors, the precision of its timekeeping can be considered analogous to sensor accuracy in other devices. The high-frequency clock allows for fine-grained control and measurement, ensuring that the displayed time is consistently accurate.
- **Input Response Time:** The device checks and responds to inputs (pause, reverse time, fast forward, and time set) swiftly, thanks to the 100 MHz clock. This speed ensures that user commands are processed almost instantaneously, allowing for dynamic and interactive time control.

The Basys-3 OTTER digital clock's capabilities and user interaction features meet the functional requirements for accurate and interactive timekeeping. It is designed to be user-friendly, with switch-based controls for time manipulation and a keyboard for direct time input. This setup not only makes the device intuitive to use but also offers a hands-on experience in understanding how digital clocks function. The interaction mechanism is straightforward: toggling switches to pause, reverse, fast forward, or set time.

2 Operating Instructions

This section provides a comprehensive guide on how to operate the Basys-3 OTTER digital clock. The instructions are designed for users who are not familiar with the Basys3 board, assuming the device is preprogrammed and ready to use upon power-up.

Starting the Clock

1. **Power On:** Connect the Basys-3 board to a power source. The clock starts automatically, displaying the time in 24-hour format on the 7-segment display.

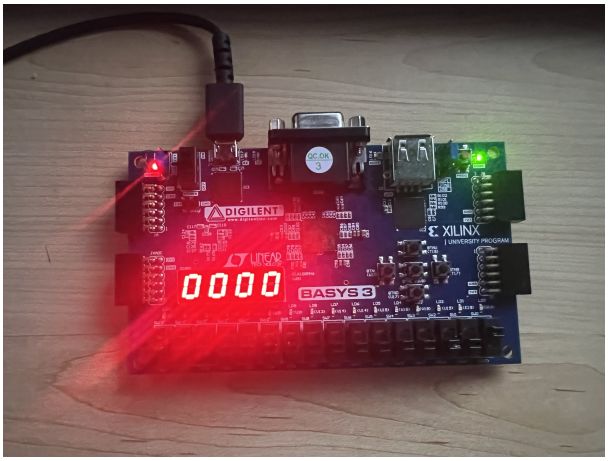


Figure 1: Powering on the Basys-3 OTTER clock.

2. **Initial Display:** Upon startup, the 7-segment display will show the minutes and seconds as 0000. All the LEDs are off.

Turning Off and Resetting

1. **Power Off:** Turn the power switch on the basys3 board by the microusb cable.
2. **Reset:** Press the central button (BTNC) to reset the time back to 0.

Interacting with the Clock

3. **Pause Time:** To pause the clock, toggle Switch 1. The time display freezes, allowing you to note the current time.

4. **Reverse Time:** Activate Switch 2 to enable the reverse time feature, which decrements the time display for reviewing past times or setting a specific time in the past. Can reverse into the previous day.

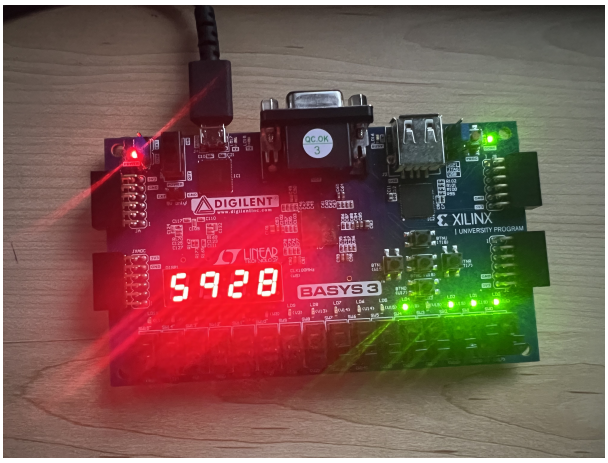


Figure 2: Reversing time with Switch 2.

5. **Fast Forward Time:** Using Switch 3, the clock can fast forward, allowing you to quickly advance to a future time.

6. **Set Time:** To manually set the clock, use Switch 4 to activate keyboard input mode. Enter the time using the keyboard connected to the Basys-3 board. Here, the time of 2042 is inputted.

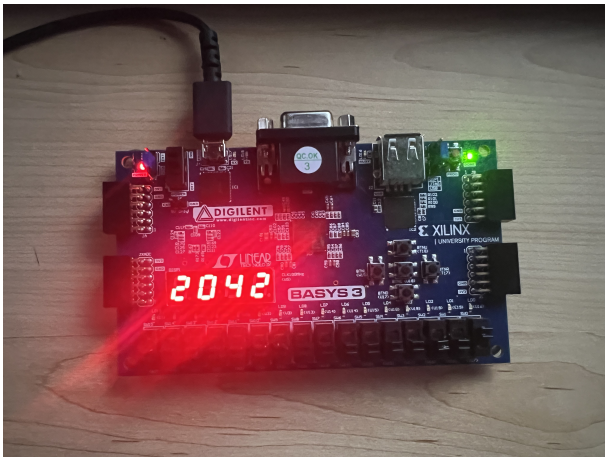


Figure 3: Setting time manually with keyboard input.

Reading the Display

- **Understanding the Display:** The clock display is segmented into parts - the LEDs indicate the hours, while the 7-segment display shows minutes and seconds. Familiarize yourself with this layout to read the time accurately. The LEDs are in binary thus the highest hour is 10111.

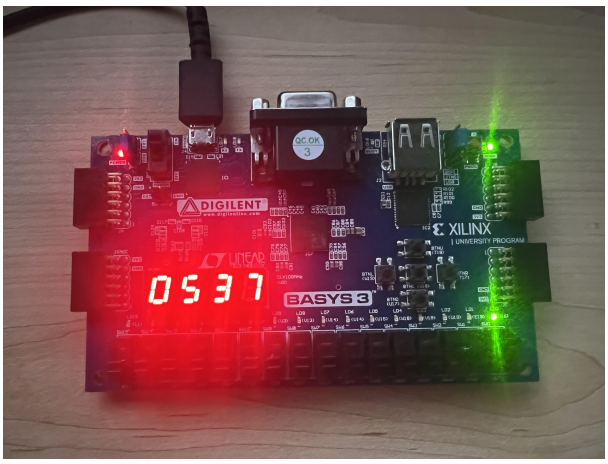


Figure 4: 1 LED is On at 0537 representing the time 01:05:37

3 Software Design

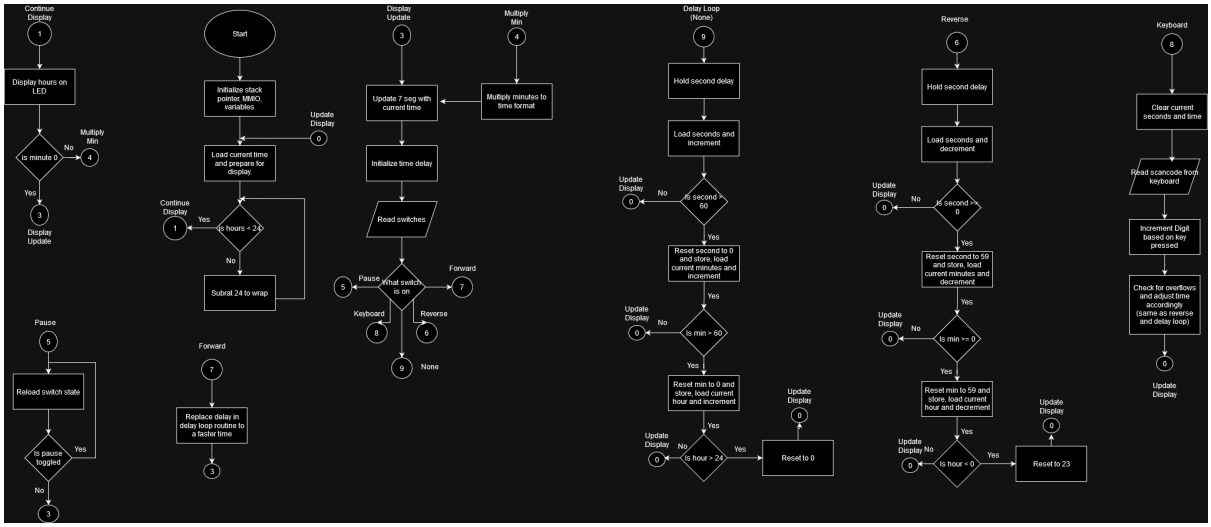


Figure 5: Complete Flowchart

This section provides an overview of the software design for the Basys3 Clock project. It covers the main program flow, as well as the various subroutines and their functionalities. The program flow is represented by the complete flowchart shown in Figure 11. Click on the image to see the entire flowchart.

Program Initialization

The program's initialization phase sets up the necessary environment and variables for a 24-hour clock on a Basys-3 board with a 7-segment display. The detailed explanation of the code is as follows:

```
li sp, STACK  # Initialize stack pointer
li s0, MMIO   # Set the base address for MMIO operations
la t1, SEC    # Load the address of the seconds counter
la t2, MIN    # Load the address of the minutes counter
la t0, HOUR   # Load the address of the hours counter
li t3, 0      # Set the initial time counters to 0
sw t3, 0(t1)  # Initialize seconds to 0
sw t3, 0(t2)  # Initialize minutes to 0
sw t3, 0(t0)  # Initialize hours to 0
```

- `li sp, STACK`: Sets the stack pointer to the start of the stack, ensuring that the program has space for function calls and local variables.
- `li s0, MMIO`: Initializes the base address for MMIO, which is used for interfacing with the hardware components such as the 7-segment display and LEDs.
- `la t1, SEC`, `la t2, MIN`, and `la t0, HOUR`: These instructions load the addresses of the second, minute, and hour counters into temporary registers, preparing them for manipulation.
- `li t3, 0`: Sets a temporary register to 0, which is used to initialize the time counters.
- `sw t3, 0(t1)`, `sw t3, 0(t2)`, and `sw t3, 0(t0)`: These instructions store the initial value of 0 into the memory locations of the seconds, minutes, and hours counters, effectively setting the clock to 00:00:00.

This section of the program lays the groundwork for the clock's operation, initializing the hardware interfaces and time-tracking variables.

Main Display Update Loop

The main display update loop of the program is responsible for managing the time display on the 7-segment display and the LEDs representing the hours. The process is detailed as follows:

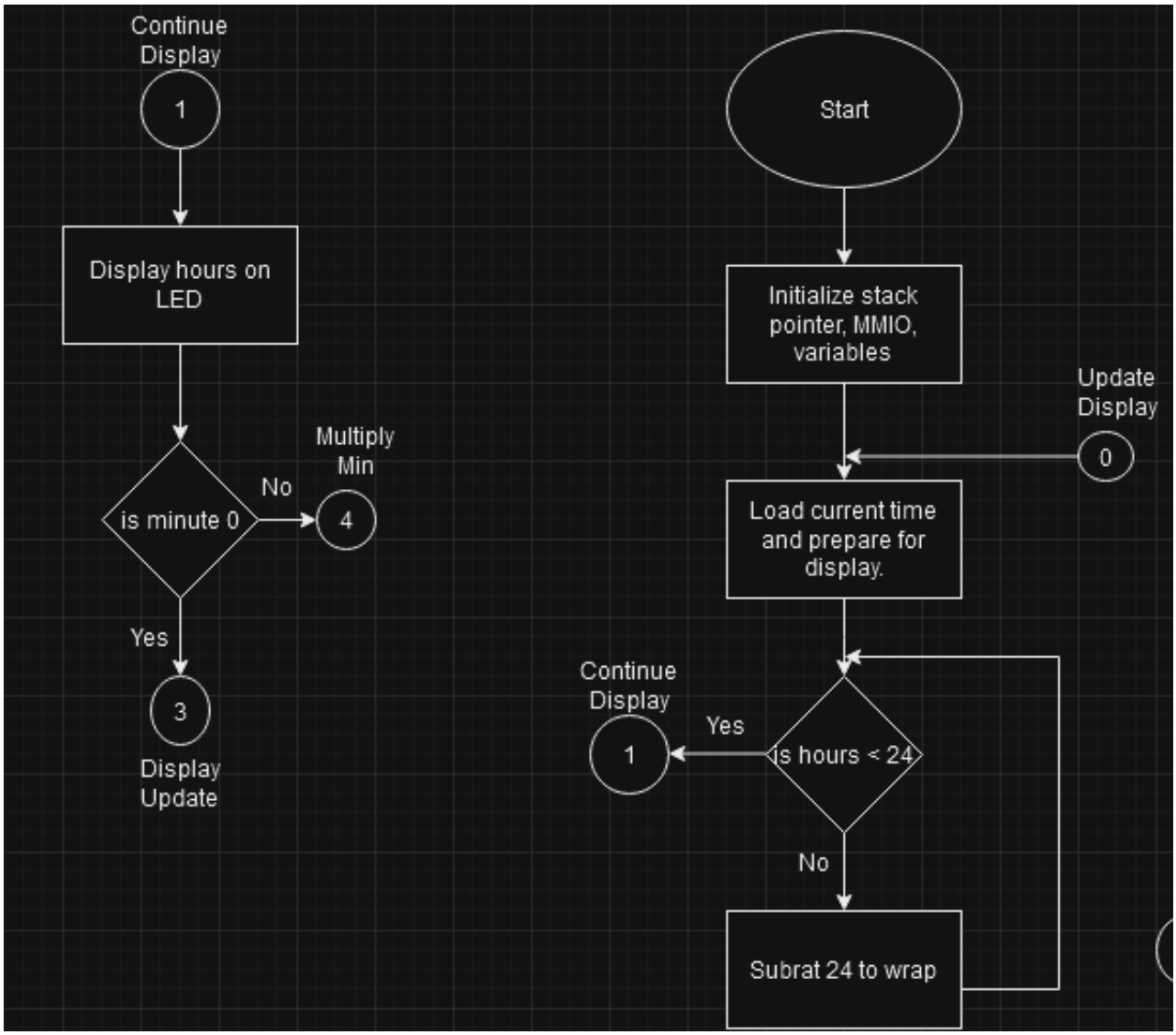


Figure 6: Logic of Display Loop

```
UPDATE_DISPLAY:
    lw t3, 0(t1) # Load seconds
    lw t4, 0(t2) # Load minutes
    lw t5, 0(t0) # Load hours
```

- `lw t3, 0(t1)`, `lw t4, 0(t2)`, `lw t5, 0(t0)`: Load the current seconds, minutes, and hours into registers `t3`, `t4`, and `t5` respectively.

```
    # Check and wrap hours to 24-hour format
    li t6, 24
    blt t5, t6, CONTINUE_DISPLAY # If hours < 24, proceed to display
    addi t5, t5, -24 # Subtract 24 from hours to wrap around
    j HOUR_MOD_24
```

- Hours are checked against 24 and wrapped around if necessary, ensuring the clock stays within the 24-hour format.

```
CONTINUE_DISPLAY:
    sw t5, LEDS(s0) # Display hours on LED display, now within 0-23 range
    beqz t4, DISPLAY_SECONDS # If minutes are 0, display only seconds
    li t6, 100 # Set the multiplier for minutes
```

- Hours are displayed on LEDs, and the program decides whether to display only seconds or include minutes based on the current minute value.

```
MULTIPLY_MINUTES:
    add t5, t5, t4 # Add minutes to t5
    addi t6, t6, -1 # Decrement multiplier
    bnez t6, MULTIPLY_MINUTES # Loop until multiplier is 0
```

- This loop multiplies the minutes and prepares the total time in minutes and seconds for display.

```
    add a0, t5, t3 # Add the multiplied minutes and seconds for display
    j DISPLAY_UPDATE
```

- Accumulates the total time in seconds for the display.

```
DISPLAY_SECONDS:
    mv a0, t3 # Move seconds to a0 for display
```

- If only seconds need to be displayed, this part is executed.

```
DISPLAY_UPDATE:
    sw a0, SEV_SEG(s0) # Display time on 7-segment display
```

- Finally, the time is displayed on the 7-segment display, showing either just seconds or the combined minutes and seconds.

This loop continuously updates the display with the current time, handling the logic for wrapping hours and calculating the time to be shown on the 7-segment display.

Main Control Loop

The main control loop of the clock program manages time incrementation and user interactions through switch inputs. Below is the detailed breakdown of the code:

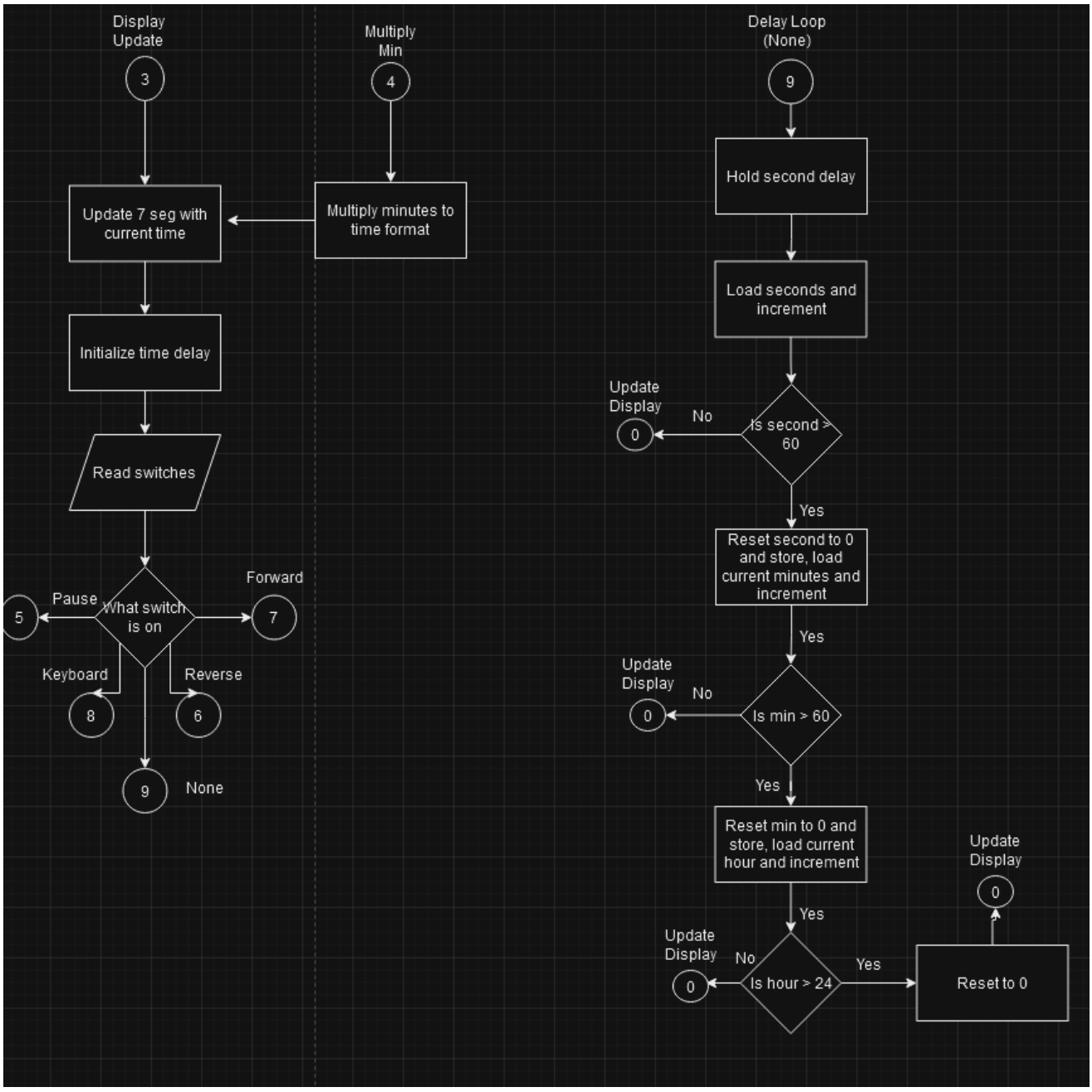


Figure 7: Logic of Main Loop

```
MAIN_LOOP:
    li t3, ONE_SEC # Load loop counter for 1-second delay
```

- The loop starts by initializing a delay counter with a constant representing one second, used for maintaining real-time progression.

```
# Read switch states and act accordingly
lw a1, SWITCHES_AD(s0)   # Load the switch states
andi a2, a1, 0x01        # Isolate the first switch state
bnez a2, PAUSE_LOOP      # If the first switch is on, pause the clock
```

- Switch inputs are read and used to control the clock's behavior, such as pausing the time.

```
# 1-second delay loop
DELAY_LOOP:
addi t3, t3, -1 # Decrement delay counter
bnez t3, DELAY_LOOP # Loop until counter reaches 0
```

- A delay loop simulates a one-second delay, essential for real-time clock operation.

```
# Time incrementation logic
lw t3, 0(t1) # Load current seconds
addi t3, t3, 1 # Increment seconds
sw t3, 0(t1) # Store updated seconds
```

- Seconds are incremented and stored, advancing the clock's time.

```
# Wrapping seconds and incrementing minutes
li t6, 60
blt t3, t6, UPDATE_DISPLAY # If seconds < 60, go to update display
li t3, 0
sw t3, 0(t1) # Reset seconds to 0
```

- The program checks if seconds reach 60 to wrap around and increment minutes.

```
# Incrementing hours and wrapping around 24 hours
li t5, 0
sw t5, 0(t0) # Reset hours to 0
j UPDATE_DISPLAY
```

- Hours are incremented and reset after reaching 24, maintaining the 24-hour format.

The control loop effectively manages the progression of time, updating seconds, minutes, and hours while handling user inputs through switches for pausing, reversing, fast-forwarding time, and setting specific time values.

Time Control Features

The clock program includes functionalities to pause, reverse, and fast-forward time, controlled by user inputs via switches. The implementation of these features is as follows:

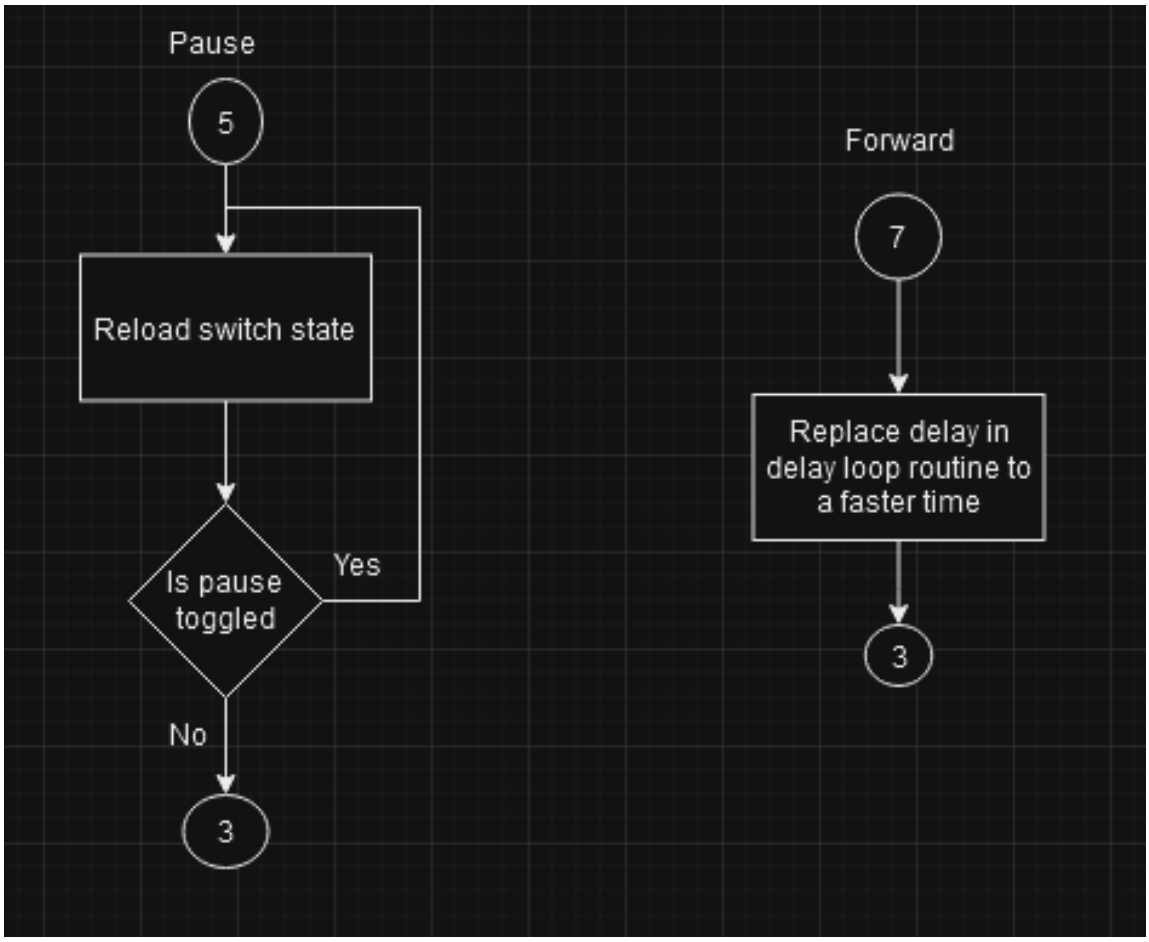


Figure 8: Logic of Pause and Fast Forward

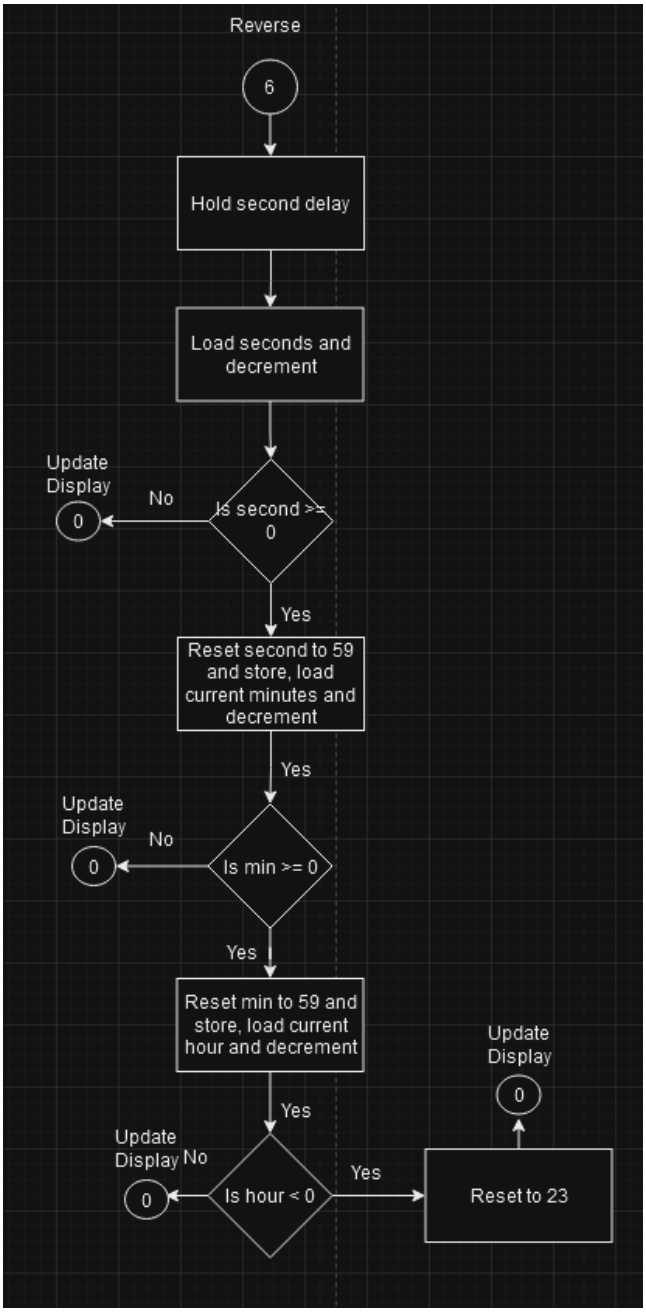


Figure 9: Logic of Reverse

```
PAUSE_LOOP:
    lw a1, SWITCHES_AD(s0)  # Reload the switch states
    andi a1, a1, 0x01       # Check the first switch state again
    bnez a1, PAUSE_LOOP     # Stay in pause loop if switch is still on
    j MAIN_LOOP             # Return to main loop when switch is off
```

- PAUSE_LOOP keeps the clock in a paused state as long as the first switch remains active, effectively halting time progression.

```
REVERSE_TIME:
    li t3, 1000000 # Load loop counter

REVERSE_DELAY_LOOP:
    addi t3, t3, -1 # Decrement delay counter
    bnez t3, REVERSE_DELAY_LOOP # Continue looping until counter is 0
```

- REVERSE_TIME initiates a reverse time functionality, decrementing the current time at a controlled rate.

```
    # Decrementing time logic
    lw t3, 0(t1) # Load the current seconds
    addi t3, t3, -1 # Decrement the seconds
    bgez t3, SKIP_SEC_RESET # If seconds >= 0, skip resetting seconds
```

- Seconds are decremented, and if they fall below zero, they reset to 59, indicating a wrap-around to the previous minute.

```
    # Handling minutes and hours during time reversal
    lw t4, 0(t2) # Load the current minutes
    addi t4, t4, -1 # Decrement minutes
    bgez t4, SKIP_MIN_RESET # If minutes >= 0, skip resetting minutes
```

- Similarly, minutes are decremented, and if necessary, they reset to 59 while hours decrement, maintaining the reverse time flow.

```
FAST_FORWARD:
    li t3, 1000000 # Load a faster loop counter for fast forward
    j DELAY_LOOP
```

- FAST_FORWARD accelerates the passage of time by reducing the duration of the delay loop, effectively speeding up the clock.

These features enhance the functionality of the clock, allowing for interactive time control through the provided switches.

Time Setting via Keyboard

The program provides functionality to set the seconds manually through keyboard input, with subsequent overflow handling into minutes and hours. Here's how it works:

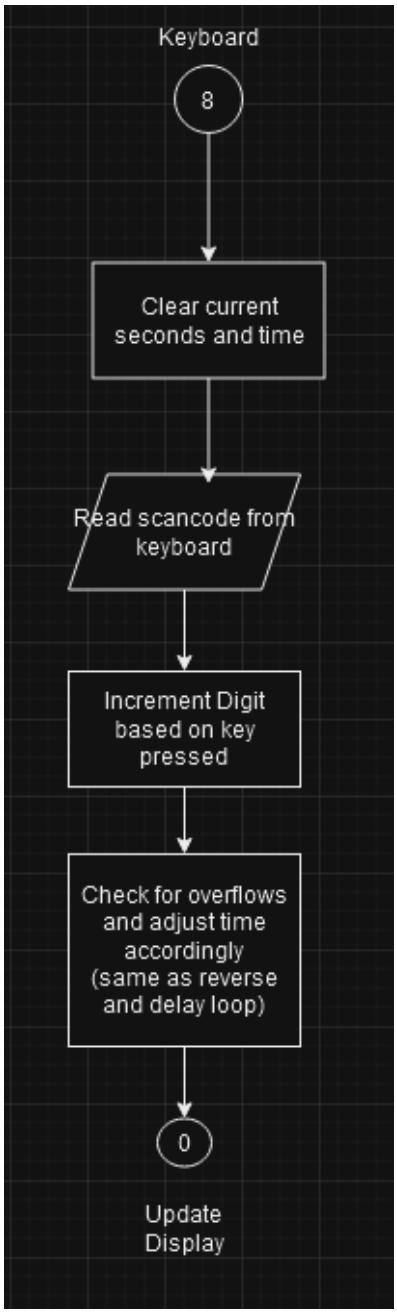


Figure 10: Logic of Keyboard Input

```
SET_SECONDS:
    li t4, 0 # Clear total seconds accumulator
    la t5, TEMP_SEC # Address of temporary seconds storage

SET_LOOP:
    lw t1, KEYBOARD(s0) # Read scancode from keyboard MMIO
    li t6, 0x08 # Mask for switch 4
    lw t2, SWITCHES_AD(s0) # Read switch state
    and t2, t2, t6
    beqz t2, FINISH_SET # Exit loop if switch 4 is off
```

- This code initializes the seconds setting process, allowing the user to input numbers via the keyboard. The loop continues until switch 4 is turned off.

```
    # Scancode-to-digit conversion logic
    li t6, 0x16 # Scancode for '1'
    beq t1, t6, ADD_ONE
    ...
    li t6, 0x45 # Scancode for '0'
    beq t1, t6, ADD_ZERO

    j SET_LOOP
```

- Keyboard scancodes are translated to numerical values, which are then used to accumulate the total seconds.

```
FINISH_SET:
    lw t3, 0(t5)          # Load the accumulated seconds from TEMP_SEC
    li t6, 60             # Constant for seconds-to-minutes conversion
```

- After setting the seconds, the program checks for overflow, potentially incrementing minutes and hours accordingly.

```
    # Overflow handling for seconds, minutes, and hours
    CHECK_SEC_OVERFLOW:
    blt t3, t6, DONE_SEC  # If seconds less than 60, done
    sub t3, t3, t6        # Subtract 60 from seconds
    addi t4, t4, 1        # Increment overflow minutes
    j CHECK_SEC_OVERFLOW

    ...

    CHECK_HOUR_OVERFLOW:
    blt t3, t6, DONE_HOUR # If hours less than 24, done
    sub t3, t3, t6        # Subtract 24 from hours
    j CHECK_HOUR_OVERFLOW
```

- Each time unit (seconds, minutes, hours) is checked for overflow, and the program adjusts the values to ensure proper time representation.

```
DONE_HOUR:
sw t3, 0(t5)          # Store the corrected hours back to HOUR

j UPDATE_DISPLAY      # Jump to update the display with the new time
```

- Finally, the corrected time is stored, and the display is updated to reflect the new time settings.

4 Appendix

Clock Code

```

# This program will run a 24 clock on Basys-3 OTTER 7 Seg
# Format of MM:SS for 7 Seg
# LEDS are the hours
# Switch 1 - Pause time
# Switch 2 - Reverse time
# Switch 3 - Fast forward time
# Switch 4 - Keyboard Input
# written by R. Simpson

.eqv MMIO, 0x11000000 # MMIO address
.eqv SEV_SEG, 0x40 # 7 segment display offset
.eqv LEDS, 0x20 # LED offset for hours display
.eqv STACK, 0x10000 # Stack pointer initial value
.eqv ONE_SEC, 6000000 # Frequency for delay
.eqv SWITCHES_AD, 0x0  # Switches are at the base MMIO address
.eqv KEYBOARD, 0x100 # Keyboard MMIO address

.data
SEC: .word 0 # seconds
MIN: .word 0 # minutes initialized to 0
HOUR: .word 0 # hours initialized to 0
TEMP_SEC: .word 0 # Temporary storage for seconds input

.text
.global _start


# Program entry point
_start:
    li sp, STACK # Initialize stack pointer
    li s0, MMIO # Initialize MMIO base address
    la t1, SEC # Load the address of SEC
    la t2, MIN # Load the address of MIN
    la t0, HOUR # Load the address of HOUR
    li t3, 0 # Initialize seconds to 0
    sw t3, 0(t1) # Store initial seconds at SEC
    sw t3, 0(t2) # Store initial minutes as 0 at MIN
    sw t3, 0(t0) # Store initial hours as 0 at HOUR


# Main display update loop
UPDATE_DISPLAY:
```

```
45      lw t3, 0(t1) # Load seconds
46      lw t4, 0(t2) # Load minutes
47      lw t5, 0(t0) # Load hours
48
49      # Check and wrap hours to 24-hour format
50      li t6, 24
51
52  # Check and wrap hours to 24-hour format
53  HOUR_MOD_24:
54      blt t5, t6, CONTINUE_DISPLAY # If hours < 24, proceed to display
55      addi t5, t5, -24 # Effectively subtract 24 from hours to wrap around
56      j HOUR_MOD_24
57
58  CONTINUE_DISPLAY:
59      sw t5, LEDS(s0) # Display hours on LED display, now within 0-23 range
60      beqz t4, DISPLAY_SECONDS # If minutes are 0, display only seconds
61      li t6, 100 # Set the multiplier for minutes
62
63
64  # Minute multiplication loop
65  MULTIPLY_MINUTES:
66      add t5, t5, t4 # Add minutes to t5
67      addi t6, t6, -1 # Decrement multiplier
68      bnez t6, MULTIPLY_MINUTES # Loop until multiplier is 0
69
70      add a0, t5, t3 # Add the multiplied minutes and seconds for display
71      j DISPLAY_UPDATE
72
73  DISPLAY_SECONDS:
74      mv a0, t3 # Move seconds to a0 for display
75
76
77  DISPLAY_UPDATE:
78      sw a0, SEV_SEG(s0) # Display time on 7-segment display
79
80  MAIN_LOOP:
81      li t3, ONE_SEC # Load loop counter for 1-second delay
82
83      # Read switch states and act accordingly
84      lw a1, SWITCHES_AD(s0)    # Load the switch states
85      andi a2, a1, 0x01         # Isolate the first switch state
86      bnez a2, PAUSE_LOOP       # If the first switch is on, go to pause loop
87
88      andi a2, a1, 0x02             # Isolate the second switch state for reverse/decre
89      bnez a2, REVERSE_TIME         # If the second switch is on, go to reverse time
90
91      andi a2, a1, 0x04             # Isolate the third switch state for fast forward
92      bnez a2, FAST_FORWARD         # If the third switch is on, go to fast forward
```

```
93
94      andi a2, a1, 0x08         # Isolate the fourth switch state
95      bnez a2, SET_SECONDS      # If the fourth switch is on, go to set seconds
96
97  # 1-second delay loop
98  DELAY_LOOP:
99      addi t3, t3, -1 # Decrement delay counter
100     bnez t3, DELAY_LOOP # Continue looping until counter is 0
101
102     lw t3, 0(t1) # Load the current seconds
103     addi t3, t3, 1 # Increment the seconds
104     sw t3, 0(t1) # Store the updated seconds
105
106     li t6, 60 # Set limit for seconds
107     blt t3, t6, UPDATE_DISPLAY # If seconds < 60, go to update display
108
109     li t3, 0 # Reset seconds to 0
110     sw t3, 0(t1) # Store the reset seconds
111
112     lw t4, 0(t2) # Load the current minutes
113     addi t4, t4, 1 # Increment minutes
114     sw t4, 0(t2) # Store the updated minutes
115
116     li t6, 60 # Set limit for minutes
117     blt t4, t6, NO_RESET_MIN # If minutes < 60, skip resetting minutes
118
119     li t4, 0 # Reset minutes to 0
120     sw t4, 0(t2) # Store the reset minutes
121
122     lw t5, 0(t0) # Load the current hours
123     addi t5, t5, 1 # Increment hours
124     li t6, 24
125     blt t5, t6, STORE_HOUR # If hours < 24, store and proceed
126     li t5, 0 # Reset hours to 0
127
128 STORE_HOUR:
129     sw t5, 0(t0) # Store the updated hours
130     j UPDATE_DISPLAY
131
132 DECREMENT_HOUR:
133     lw t5, 0(t0) # Load the current hours
134     addi t5, t5, -1 # Decrement hours
135     bgez t5, STORE_HOUR # If hours >= 0, store and proceed
136     li t5, 23 # Set hours to 23
137
138     j STORE_HOUR
139
140 NO_RESET_MIN:
```

```
141     j UPDATE_DISPLAY # Update display and continue
142
143 STOP:
144     j STOP
145
146 PAUSE_LOOP:
147     lw a1, SWITCHES_AD(s0)  # Reload the switch states
148     andi a1, a1, 0x01       # Check the first switch state again
149     bnez a1, PAUSE_LOOP     # Stay in pause loop if switch is still on
150     j MAIN_LOOP             # Return to main loop when switch is off
151
152 REVERSE_TIME:
153     li t3, 1000000 # Load loop counter
154
155 REVERSE_DELAY_LOOP:
156     addi t3, t3, -1 # Decrement delay counter
157     bnez t3, REVERSE_DELAY_LOOP # Continue looping until counter is 0
158
159     lw t3, 0(t1) # Load the current seconds
160     addi t3, t3, -1 # Decrement the seconds
161     bgez t3, SKIP_SEC_RESET # If seconds >= 0, skip resetting seconds
162
163     li t3, 59 # Reset seconds to 59
164     sw t3, 0(t1) # Store the reset seconds
165
166     lw t4, 0(t2) # Load the current minutes
167     addi t4, t4, -1 # Decrement minutes
168     bgez t4, SKIP_MIN_RESET # If minutes >= 0, skip resetting minutes
169
170     li t4, 59 # Reset minutes to 59
171     sw t4, 0(t2) # Store the reset minutes
172
173     lw t5, 0(t0) # Load the current hours
174     addi t5, t5, -1 # Decrement hours
175     bgez t5, STORE_HOUR # If hours >= 0, store and continue
176
177     li t5, 23 # Reset hours to 23 if it goes below 0
178     j STORE_HOUR
179
180 SKIP_SEC_RESET:
181     sw t3, 0(t1) # Store the updated seconds
182     j UPDATE_DISPLAY
183
184 SKIP_MIN_RESET:
185     sw t4, 0(t2) # Store the updated minutes
186     j UPDATE_DISPLAY
187
188
```

```
189 FAST_FORWARD:
190     li t3, 1000000 # Load a faster loop counter for fast forward
191     j DELAY_LOOP
192 
193 SET_SECONDS:
194     li t4, 0 # Clear total seconds accumulator
195     la t5, TEMP_SEC # Address of temporary seconds storage
196 
197 SET_LOOP:
198     lw t1, KEYBOARD(s0) # Read scancode from keyboard MMIO
199     li t6, 0x08 # Mask for switch 4
200     lw t2, SWITCHES_AD(s0) # Read switch state
201     and t2, t2, t6
202     beqz t2, FINISH_SET # Exit loop if switch 4 is off
203 
204     # Scancode-to-digit conversion logic
205     li t6, 0x16 # Scancode for '1'
206     beq t1, t6, ADD_ONE
207     li t6, 0x1E # Scancode for '2'
208     beq t1, t6, ADD_TWO
209     li t6, 0x26 # Scancode for '3'
210     beq t1, t6, ADD_THREE
211     li t6, 0x25 # Scancode for '4'
212     beq t1, t6, ADD_FOUR
213     li t6, 0x2E # Scancode for '5'
214     beq t1, t6, ADD_FIVE
215     li t6, 0x36 # Scancode for '6'
216     beq t1, t6, ADD_SIX
217     li t6, 0x3D # Scancode for '7'
218     beq t1, t6, ADD_SEVEN
219     li t6, 0x3E # Scancode for '8'
220     beq t1, t6, ADD_EIGHT
221     li t6, 0x46 # Scancode for '9'
222     beq t1, t6, ADD_NINE
223     li t6, 0x45 # Scancode for '0'
224     beq t1, t6, ADD_ZERO
225 
226     j SET_LOOP
227 
228 ADD_ONE:
229     li t3, 1
230     add t4, t4, t3
231     j SET_LOOP
232 
233 ADD_TWO:
234     li t3, 2
235     add t4, t4, t3
236     j SET_LOOP
```

```
237
238 ADD_THREE:
239     li t3, 3
240     add t4, t4, t3
241     j SET_LOOP
242
243 ADD_FOUR:
244     li t3, 4
245     add t4, t4, t3
246     j SET_LOOP
247
248 ADD_FIVE:
249     li t3, 5
250     add t4, t4, t3
251     j SET_LOOP
252
253 ADD_SIX:
254     li t3, 6
255     add t4, t4, t3
256     j SET_LOOP
257
258 ADD_SEVEN:
259     li t3, 7
260     add t4, t4, t3
261     j SET_LOOP
262
263 ADD_EIGHT:
264     li t3, 8
265     add t4, t4, t3
266     j SET_LOOP
267
268 ADD_NINE:
269     li t3, 9
270     add t4, t4, t3
271     j SET_LOOP
272
273 ADD_ZERO:
274     li t3, 0
275     add t4, t4, t3
276     j SET_LOOP
277
278 FINISH_SET:
279     lw t3, 0(t5)          # Load the accumulated seconds from TEMP_SEC
280     li t6, 60             # Constant for seconds-to-minutes conversion
281
282     # Manual overflow calculation for seconds to minutes
283     li t4, 0              # t4 will hold the overflow minutes
284 CHECK_SEC_OVERFLOW:
```

```
285     blt t3, t6, DONE_SEC  # If seconds less than 60, go to DONE_SEC
286     sub t3, t3, t6        # Subtract 60 from seconds to handle overflow
287     addi t4, t4, 1        # Increment minutes overflow
288     j CHECK_SEC_OVERFLOW
289
290 DONE_SEC:
291     la t5, SEC            # Load address of SEC
292     sw t3, 0(t5)          # Store the corrected seconds back to SEC
293
294     # Manual overflow calculation for minutes
295     la t5, MIN            # Load address of MIN
296     lw t3, 0(t5)          # Load current minutes
297     add t3, t3, t4        # Add overflow minutes to current minutes
298 CHECK_MIN_OVERFLOW:
299     blt t3, t6, DONE_MIN  # If minutes less than 60, go to DONE_MIN
300     sub t3, t3, t6        # Subtract 60 from minutes to handle overflow
301     addi t4, t4, 1        # Increment hours overflow
302     j CHECK_MIN_OVERFLOW
303
304 DONE_MIN:
305     sw t3, 0(t5)          # Store the corrected minutes back to MIN
306
307     # Manual overflow calculation for hours
308     li t6, 24             # Constant for hours wrap-around
309     la t5, HOUR           # Load address of HOUR
310     lw t3, 0(t5)          # Load current hours
311     add t3, t3, t4        # Add overflow hours to current hours
312
313 CHECK_HOUR_OVERFLOW:
314     blt t3, t6, DONE_HOUR # If hours less than 24, go to DONE_HOUR
315     sub t3, t3, t6        # Subtract 24 from hours to handle overflow
316     j CHECK_HOUR_OVERFLOW
317
318 DONE_HOUR:
319     sw t3, 0(t5)          # Store the corrected hours back to HOUR
320
321     j UPDATE_DISPLAY      # Jump to update the display with the new time
322
323
324
```

Demo

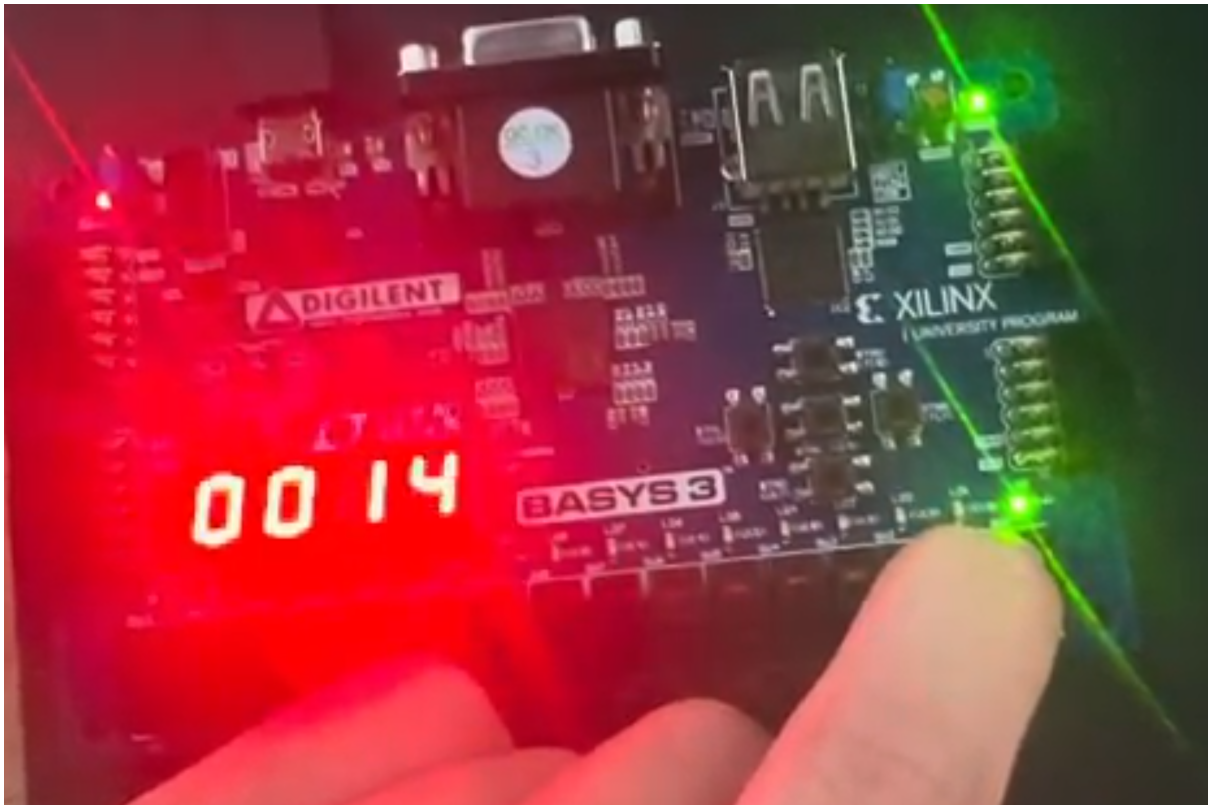


Figure 11: Demo Video (No Keyboard) **(Click to view)**