Digital Clock

Marco Losso

December 16, 2024

Disclaimer

The code provided in this document has been simplified for comprehension purposes. It is not functional as is and represents more of a logical implementation than a syntactic one. Therefore, modifications may be needed to make it executable and compliant with the actual project specifications.

1 Introduction

The project aims to develop a digital clock displaying hours, minutes, and seconds, adjustable via a rotary encoder. The system utilizes various clock frequencies to ensure high precision in timekeeping. Additionally, optional features have been implemented to enhance user experience, such as the option to choose between 12-hour and 24-hour formats and the addition of a stopwatch with "short click" and "long click" detection. This report describes the project development process, the challenges faced, and the solutions adopted.

2 Hardware Configuration in Platform Designer

The project started with a pre-configured base in Platform Designer of Quartus, which provided pre-configured modules for timers at different frequencies, interfacing with external GPIO for the rotary encoder, and modules for controlling the 7-segment displays. The initial hardware configuration is shown in Figure 1.

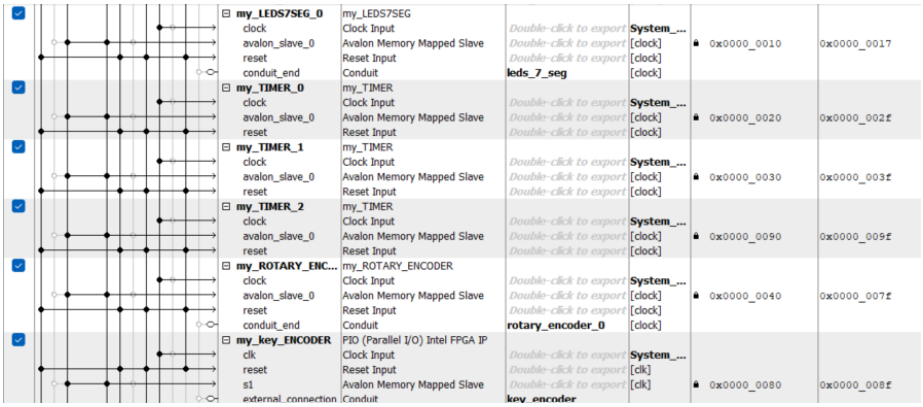

Figure 1: Excerpt of Platform Designer Structure

In Figure 1, various modules are connected:

- **my_LEDS7SEG_0**: This module controls the 6 7-segment displays. It is connected to the clock, Avalon slave, reset, and conduits.
- **my_TIMER_0**: A timer module operating at 1Hz, connected to the clock, Avalon slave, and reset.
- **my_TIMER_1**: Similar to my_TIMER_0, but operates at 2Hz.
- **my_TIMER_2**: Another timer module operating at 100Hz, connected to the clock, Avalon slave, and reset.
- **my_ROTARY_ENCODER_0**: Module for interfacing with a rotary encoder, connected to the clock, Avalon slave, reset, and conduit.
- **my_key_ENCODER**: Module for interfacing with the button on the rotary encoder, connected to the clock, reset, s1, and external connections.

The mentioned timers (my_TIMER_0, my_TIMER_1, and my_TIMER_2) are simple counters that write their value to a memory register and operate at frequencies of 1Hz, 2Hz, and 100Hz, respectively. These timers were used in the C program to handle precise timekeeping.

3 Verilog HDL Code Modifications in the Top-Level Entity

After configuring the hardware in Platform Designer, some modifications were made to the Verilog HDL code in the top-level entity to connect the ARM processor with external peripherals. These modifications include linking the 7-segment displays and the rotary encoder. The added Verilog code is as follows:

```
module Top_Level_Entity
...
// External Connection
.leds_7_seg_out_dispa          (HEX0),  // leds_7_seg.out_dispa
.leds_7_seg_out_dispb          (HEX1),  //           .out_dispb
.leds_7_seg_out_dispc          (HEX2),  //           .out_dispc
.leds_7_seg_out_dispd          (HEX3),  //           .out_dispd
.leds_7_seg_out_dispe          (HEX4),  //           .out_dispe
.leds_7_seg_out_dispf          (HEX5),  //           .out_dispf

.rot_encoder_data1 (GPIO1GPIO[0]),  // rotary_encoder_0.data1
.rot_encoder_data1_reg   (),        //              .data1_reg
.rot_encoder_data2 (GPIO1GPIO[1]),  //                  .data2
.rot_encoder_data2_reg   (),        //              .data2_reg

.key_encoder_export     (GPIO1GPIO[2])   // key_encoder.export

endmodule
```

3.1 Description of Modifications

The modifications to the Verilog HDL code concern the external connections for controlling the 7-segment displays and interfacing with the rotary encoder:

- **7-segment Display**:
  - **leds_7_seg_out_dispa** - Connects the 7-segment display to pin HEX0.
  - **leds_7_seg_out_dispb** - Connects the 7-segment display to pin HEX1.
  - **leds_7_seg_out_dispc** - Connects the 7-segment display to pin HEX2.
  - **leds_7_seg_out_dispd** - Connects the 7-segment display to pin HEX3.
  - **leds_7_seg_out_dispe** - Connects the 7-segment display to pin HEX4.
  - **leds_7_seg_out_dispf** - Connects the 7-segment display to pin HEX5.
- **Rotary Encoder**:
  - **rotary_encoder_0_data1** - Connects to pin GPIO1GPIO[0] to read the encoder data.
  - **rotary_encoder_0_data2** - Connects to pin GPIO1GPIO[1] to read the encoder data.
- **Key Encoder**:
  - **key_encoder_export** - Connects to pin GPIO1GPIO[2] to read the encoder button data.

These connections allow the ARM processor to interface with the 7-segment displays and the rotary encoder, enabling real-time control and data visualization.

4 Modifications to the Verilog Module for the 7-Segment Display

To control the 7-segment displays, a Verilog module was used to encode the various values to be displayed. In the module coding, the output "a" corresponds to the situation where all display LEDs are off, while the output "b" indicates the display of the letter "h". The updated Verilog code for the module is as follows:

```
begin
    case(iDIG)
        4'h1: oSEG = 7'b1111001;    // ---t----
        4'h2: oSEG = 7'b0100100;    // |     |
        4'h3: oSEG = 7'b0110000;    // lt    rt
        4'h4: oSEG = 7'b0011001;    // |     |
        4'h5: oSEG = 7'b0010010;    // ---m----
        4'h6: oSEG = 7'b0000010;    // |     |
        4'h7: oSEG = 7'b1111000;    // lb    rb
        4'h8: oSEG = 7'b0000000;    // |     |
        4'h9: oSEG = 7'b0011000;    // ---b----
        4'ha: oSEG = 7'b1111111;    // => Tutti i LED spenti
        4'hb: oSEG = 7'b0001011;    // => Visualizzazione "h"
    endcase
end
```

4.1 Description of Modifications

In the code above, the lines **4'ha** and **4'hb** have been modified to handle the situation where all display LEDs need to be off and to display the letter "H":

- **4'ha: oSEG = 7'b1111111;** - This line of code sets all the bits of the oSEG output to 1, thereby turning off all the display LEDs.
- **4'hb: oSEG = 7'b0001011;** - This line of code sets the oSEG output to display the letter "H" on the 7-segment display.

5 Implementation of the C Code

5.1 Memory Address Definitions

The first part of the code defines the memory addresses for the various peripherals used in the project. These addresses are necessary to interface with the rotary encoder, the 7-segment display, the timers, and the encoder button.

```
// Memory address definitions
#define ENCODER_ADDR 0xff200040
#define DISPLAY_ADDR 0xff200010
#define TIMER_ADDR 0xff200020
#define BUTTON_ADDR 0xff200080
#define TIMER_LAMP_ADDR 0xff200030
#define TIMER_CRON 0xff200090
```

5.2 Clock Implementation

One of the main features of the project is the implementation of the clock. The function that updates the clock is responsible for counting seconds, minutes, and hours, and for displaying the data on the 7-segment displays.

```
if (old_click != new_click) {
// Se il valore del clk a 1 Hz e' cambiato, si aggiorna l'ora
    sec++;
    if (sec >= 10) {
        sec = 0;
        decisec++;
        if (decisec >= 6) {
            decisec = 0;
            min++;
            if (min >= 10) {
                min = 0;
                decimin++;
                if (decimin >= 6) {
                    decimin = 0;
                    ore++;

                    if (ore >= 10) {
                        ore = 0;
                        deciore++;
                    }
                }
            }
        }
    }
```

```
    // Verifica del caso speciale per ore >= 24 o 12
    if (formato_ora == 24) {
        if (deciore >= 2 && ore >= 4) {
            deciore = 0;
            ore = 0;
        }
    } else { // Formato 12 ore
        if (deciore >= 1 && ore >= 2) {
            // Gestisce il passaggio da 11:59:59 a 12:00:00
            ore = 0;
            deciore = 0;
        }
    }
}

if (deciore != 0) {
// Discriminazione del caso in cui le ore sono maggiori di 10
// in modo da tenere spento il display piu' significativo
    output_display = 0xAA000000 | (deciore << 20) | (ore << 16)
                                | (decimin << 12) | (min << 8)
                                | (decisec << 4) | sec;
} else {
    output_display = 0xAAA00000 | (deciore << 20) | (ore << 16)
                                | (decimin << 12) | (min << 8)
                                | (decisec << 4) | sec;
}
```

The **aggiornamento_orologio** function updates the time and manages the hour format (12-hour or 24-hour). If the value of the register managed by the 1 Hz timer changes, the seconds are incremented, which in turn increment the minutes and hours as needed. The display is updated accordingly. This function is crucial for maintaining accurate timekeeping and properly updating the 7-segment displays. It also shows how the two time formats (12-hour or 24-hour) are handled.

5.3 Time Adjustment via Rotary Encoder and Implementation of Blinking

To allow the user to adjust the digital clock's time using the rotary encoder, a mechanism was implemented to detect the button press and the encoder rotation. Additionally, during the adjustment, the display of the digit being adjusted blinks to indicate to the user which element is being modified.

5.3.1 State Management in the Main

In the **main**, the following code manages the system state to switch between different adjustment modes (seconds, minutes, hours) and detects the encoder

button press:

```
if (pulsante_precedente == 1 && pulsante_corrente == 0) {
// Rileva pressione del pulsante
    if (visualizzazione == 0){
        stato = (stato + 1) % 5; // aggiornamento stato
    }
}

// aggiornamento pulsante
pulsante_precedente = pulsante_corrente;

if(stato == 1){
// se lo stato e' 1 si passa alla selezione del
//formato dell'ora prima di poterla modificare
    formato_ora = seleziona_formato_ora(formato_ora);
} else if(stato != 0 && stato != 1){
// se lo stato e' diverso da 0 o da 1 si passa alla modifica
// delle variabili
    modifica_orologio(se, min, ore, formato_ora);
}
```

5.3.2 Clock Adjustment Function

The **modifica_orologio** function is responsible for adjusting the clock variables (seconds, minutes, hours) and for making the display blink:

```
void modifica_orologio(int *clk_2Hz_ptr,
                       int *sec, int *min, int *ore,
                       int stato, int formato_ora) {

    if ((clk_2Hz_curr % 2) == 0){
    // Quando il valore nel registro e' pari i display
    // della variabile sotto modifica vengono spenti

    switch (stato) {

            case 2:
                // Spegne i display dei secondi
                output_display &= 0xFFFFFF00;
                // Lampeggia i display dei secondi
                output_display |= 0x000000AA;
                modifica_sec_min(sec);
                break;

            case 3:
                // Spegne i display dei minuti
                output_display &= 0xFFFF00FF;
```

```
                // Lampeggia i display dei minuti
                output_display |= 0x0000AA00;
                modifica_sec_min(min);
                break;

            case 4:
                // Spegne i display delle ore
                output_display &= 0xFF00FFFF;
                // Lampeggia i display delle ore
                output_display |= 0x00AA0000;
                modifica_ore(ore, formato_ora);
                break;
        }
    } else {
    // Permette la modifica delle variabili
    // mantenendo i display accesi
        switch (stato) {
            case 2:
                modifica_sec_min(sec);
                break;

            case 3:
                modifica_sec_min(min);
                break;

            case 4:
                modifica_ore(ore, formato_ora);
                break;
        }
    }
}
```

The **modifica_orologio** function uses the encoder values to adjust seconds, minutes, and hours. During the adjustment, the display of the digit being adjusted blinks, making it clear to the user which element is being modified.

5.4 Hour Format Adjustment

In the project, the user has the option to choose between two time display formats: the 12-hour format and the 24-hour format. Part of this implementation is visible in the state selection in the **main** (as described in the previous subsection) and in the standard clock update function. Below is the **seleziona_formato_ora** function, which allows choosing the hour format via the rotary encoder movement.

```
int seleziona_formato_ora(int formato_corrente) {

    // Aggiorna il display per mostrare il
    // formato selezionato
    if (formato_corrente == 24) {
    // Viene visualizzato h 24
        output_display = 0xAAABAA24;
    } else {
    // Viene visualizzato h 12
        output_display = 0xAAABAA12;
    }

    // Cambia formato ora tra 12 e 24 ad
    // ogni click dell'encoder
    if (encoder_reg_curr != encoder_reg_prev) {
        if (formato_corrente == 24) {
            formato_corrente = 12;
        } else {
            formato_corrente = 24;
        }
        encoder_reg_prev = encoder_reg_curr;
    }

    return formato_corrente;
}
```

The **seleziona_formato_ora** function detects the movements of the rotary encoder to change the hour format. With each click of the encoder, the hour format is switched between 12-hour and 24-hour. The display is updated accordingly to show the selected format.

This functionality adds flexibility to the system, allowing the user to choose their preferred hour format in a simple and intuitive way.

5.5 Implementation of the Stopwatch

The stopwatch was implemented to operate in parallel with the clock. It is possible to switch from the clock display to the stopwatch display using the rotary encoder. A short click on the encoder starts or stops the stopwatch, while a long click resets the stopwatch.

5.5.1 Display and Command Management of the Stopwatch

In the **main**, the logic to manage the display and commands of the stopwatch is as follows.

Below is the **aggiorna_display** function, which handles the display of the clock or stopwatch data on the 7-segment displays.

```
if(stato == 1){
    // se lo stato e' 1 si passa alla selezione
    // del formato dell'ora prima di poterla modificare
} else if(stato != 0 && stato != 1){
    // se lo stato e' diverso da 0 o da 1 si passa alla
    // modifica delle variabili
} else {
    // altrimenti si rimane nello stato di default
    if (encoder_reg_curr != encoder_reg_prev) {
    // Toggle visualizzazione tra orologio e cronometro
        visualizzazione = !visualizzazione;
    }

  display_o = aggiornamento_orologio(clk_1Hz, formato_ora);
  display_c = aggiorna_cronometro(clk_100Hz, visualizzazione);

  // Aggiorna il display in base alla visualizzazione corrente
  aggiorna_display(visualizzazione, display_o, display_c);
}

void aggiorna_display(int vis, int out_o, int out_c) {
    if (vis == 0) {
        out_o; // Visualizza l'orologio
    } else {
        out_c; // Visualizza il cronometro
    }
}
```

5.5.2 Stopwatch Update Function

The **aggiorna_cronometro** function manages the time counting for the stopwatch and detects the button press duration to implement the reset, in addition to the basic start and stop functionalities.

```
int aggiorna_cronometro (int *clk_100Hz, int vis) {

    if(vis == 1) {
    // rileva il pulsante solo se siamo
    // nella visualizzazione del cronometro
        pulsante_corrente = *button_ptr;
        if (pulsante_precedente == 1
                && pulsante_corrente == 0
                && vis == 1) {
            // Reset del tempo di pressione
            tempo_premuto = 0;
        }
    }
    if (pulsante_corrente == 0) {
        if (new_clk_100Hz != old_clk_100Hz) {
            // Incrementa il tempo di pressione
            tempo_premuto++;
        }
    } else if (pulsante_precedente == 0
                    && pulsante_corrente == 1) {
        // Rileva rilascio del pulsante
        if (tempo_premuto >= 100) {
            // Pulsante premuto per piu' di un secondo
            // (100 tick di 100Hz)
            *centesimi = 0;
            *secondi = 0;
            *minuti = 0;
            // Reset del cronometro
            *cronometro_attivo = 0;
        } else {
        // Pulsante premuto per meno di un secondo
            *cronometro_attivo = !(*cronometro_attivo);
        }
    }

    if (new_clk_100Hz != old_clk_100Hz
            && *cronometro_attivo) {
        // Se il valore del clk a 100 Hz e' cambiato e
        // il cronometro e' attivo, aggiorno il cronometro
        (*centesimi)++;
        if (*centesimi >= 100) {
            *centesimi = 0;
            (*secondi)++;
```

```
            if (*secondi >= 60) {
                    *secondi = 0;
                    (*minuti)++;
            }
        }
    }

    // Reset del cronometro se raggiunge 59:59:99
    if (*minuti >= 59) {
        *minuti = 0;
        *secondi = 0;
        *centesimi = 0;
    }

    // Costruzione output display
    output_display =

      (*minuti >= 10 ? *minuti / 10 * 0x100000 : 0xAAA00000)
    | (*minuti == 0 ? 0xAA0000 : *minuti % 10 * 0x010000)
    | (*minuti == 0 && *secondi < 10 ?
           0xAAA000 : *secondi / 10 * 0x001000)
    | (*secondi % 10 * 0x000100)
    | (*centesimi / 10 * 0x000010)
    | (*centesimi % 10 * 0x000001);

    // Restituisce il valore di output_display
    return output_display;
}
```

This function updates the stopwatch based on the ticks from the 100Hz timer, allowing for precise counting of hundredths of a second. It also handles the short and long button clicks to start/stop and reset the stopwatch.

In constructing the output variable for the 7-segment display, logical operators are used to ensure that the displays of the most significant digits remain off, providing a more aesthetically pleasing visualization.

6 Conclusions

The project has resulted in the creation of a digital clock complete with advanced features, such as time adjustment via a rotary encoder, blinking digits during modification, and the option to choose between 12-hour and 24-hour formats. Additionally, a stopwatch functioning in parallel with the clock has been integrated, offering the user the ability to start, stop, and reset the count with ease.

Using a pre-configured hardware base in Quartus's Platform Designer accelerated development, allowing focus on high-level functionalities and interfacing with external peripherals. Modifications to the Verilog HDL code and the implementation of the C program were crucial to ensure the system's correct functioning and provide an intuitive and efficient user interface.

The flexibility and precision achieved through the my_TIMERS blocks and the various display management functions demonstrate the effectiveness of the adopted approach. The optional features implemented, such as the discrimination between short and long clicks, further enrich the user experience, making the system versatile and adaptable to various needs.

In conclusion, this project represents a practical example of how to combine hardware and software to create complex and functional embedded systems, laying solid foundations for future expansions and improvements.