Low-level-programming Project Documentation: Binary Clock + USART

*Team Members - Group 7*: Böhler, Anton; Broer, Jan; Dietrich, Alexander

# Goal of the Project

The task given is the implementation of a binary clock with USART output. It requires counting hours, minutes and seconds, and outputs to I/O-pins as well as USART through console.   
A binary clock should work as follows: it requires a minimum of 5 or respectively 6 bits to display the 24 hours, 60 minutes, and 60 seconds. The seconds count should be incremented once a second, and overflow at 60 seconds to increment the minutes by one, while being reset to zero seconds. Minutes to hours works analog to this.

Additionally, to be able to set the clock to a specific state, the hours and minutes should be able to be manipulated. By pressing one of five buttons, the time can be increased or decreased by hours or minutes, as well as being reset to zero.

*Theoretical schematic of the circuit*

A picture containing diagram

Description automatically generated

*Console Output*

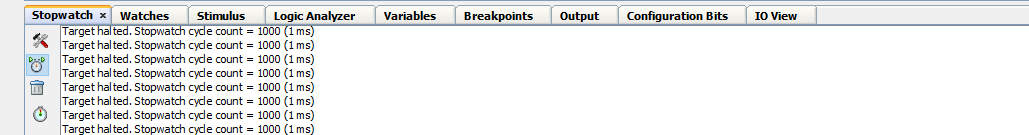
00001101:00100101:00000000  
00001101:00100101:00000001  
00001101:00100101:00000010  
00001101:00100101:00000011

# Solution

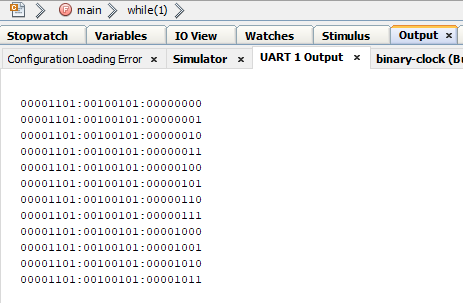
|  |  |
| --- | --- |
| Initialization I/O At first, pins are initialized as inputs or outputs. I/O Port A is a special case because all its ports are analog by default and can be set as digital. Therefor they need to be configured as digital pins. | // make PORTA digital (default analog)  ADCON1bits.PCFG = 0b1111;  // define clock outputs  TRISA = 0;  TRISB = 0;  TRISD = 0;  // define reset button as input  TRISCbits.RC0 = 1;  // define edit buttons as input  TRISCbits.RC1 = 1;//m-  TRISCbits.RC2 = 1;//m+  TRISCbits.RC3 = 1;//h-  TRISCbits.RC4 = 1;//h+ |
| Initialization USART Initializing the USART requires some flags to be set. In addition to that, the subroutine *putch* needs to be  (re-)defined. After that the *printf* subroutine can be called. | // initialize uart (used for printing)  TRISCbits.RC7 = 1;  TRISCbits.RC6 = 1;  TXSTAbits.TXEN = 1;  RCSTAbits.SPEN = 1;    printf("\n\n"); |
| Timer Interrupts Clock measuring is based on timer-1 count, using interrupts to increment milliseconds. Because of this, interrupts need to be activated. The use of the internal clock allows for an elegant solution without having to use additional external hardware.  To make the timer-loop accurate, a manual offset is added to account for the processing delay. | INTCON = 0b11000000; // enable global interrupts  PIE1bits.TMR1IE = 1; // enable timer-1 interrupt  T1CONbits.TMR1ON = 1;     // setup timer  void \_\_interrupt(high\_priority)  timer\_overflow\_interrupt(void){      // check for timer-1 interrupt flag      if(TMR1IF){          // reset timer-1          TMR1IF = 0;          TMR1 = TIMER\_1\_MAX\_VALUE - ONE\_MILISECOND + TIMER\_1\_CORRECTION;  ...  }  return;  } |
| Time-format 24-60-60 Counting seconds, minutes and hours is done using multiple variables. At each interrupt the milliseconds counter is incremented. On overflowing the respective next variable is incremented. | // increase time (handles overflows too)  milisecond++;  if(milisecond >= 1000){  milisecond = 0;      second++;  if(second >= 60){      second = 0;          minute++;          if(minute >= 60){          minute = 0;              hour = (hour + 1) % 24;          }      }  ...  } |
| Output There are two ways to view the clock output. On the one hand, there are output-pins (connected to LEDs in real world use) that indicate the binary state of hour, minute and second. On the other hand, there is an output using USART to transmit the time to a terminal. | // initialize output String  char binary[] = "00000000";    // edit String (dezimal to binary conversion)  for(int i=0;i<8;i++){      if(num % 2 == 1){ binary[7 - i] = '1'; }      else{ binary[7 - i] = '0' }      num /= 2;  }  // output to console  printf(binary);  // output clock on pins every second  LATA = (unsigned char)second;  LATB = (unsigned char)minute;  LATD = (unsigned char)hour; |
| Control-Buttons As a nice-to-have, while the micro controller is waiting for the timer-1 interrupt, it does polling on several ports. If an input at one of the buttons is detected, the corresponding action is triggered. For instance, pressing the button connected to RC1 results in a timer-stop and decrement of the minute-variable.  Each variable-change is displayed instantly.  When releasing the button, timer-1 gets reset and resumes counting. | // polling for buttons  while(1){      if(PORTCbits.RC0){          // reset clock          milisecond = 0;          second = 0;          minute = 0;          hour = 0;      }else if(PORTCbits.RC1){          // decrement minute          T1CONbits.TMR1ON = 0;          while(PORTCbits.RC1){              minute = (minute + 59) % 60;              output();              \_\_delay\_ms(BUTTON\_DELAY);          }          TMR1 = TIMER\_1\_MAX\_VALUE - ONE\_MILISECOND + TIMER\_1\_CORRECTION;          T1CONbits.TMR1ON = 1;      }  ...  } |

# Results

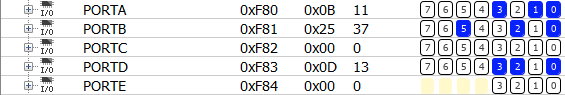
As a result, the console output displays the binary time each second. Simultaneously, the IO view shows the pin activity (LED state).

Stopwatch cycle count 1ms to check if the timing is correct

Text

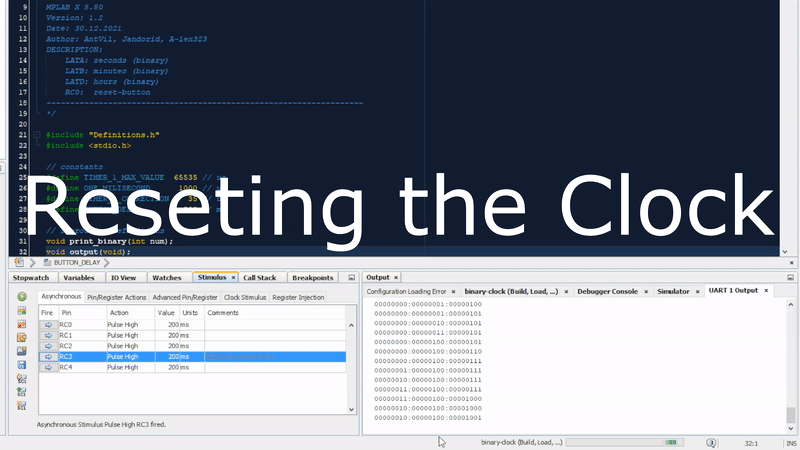
Description automatically generated with low confidenceConsole output at regular use Console output when the reset button is pressed

IO View of the Ports A (Seconds), B (Minutes) ,D (Hours)

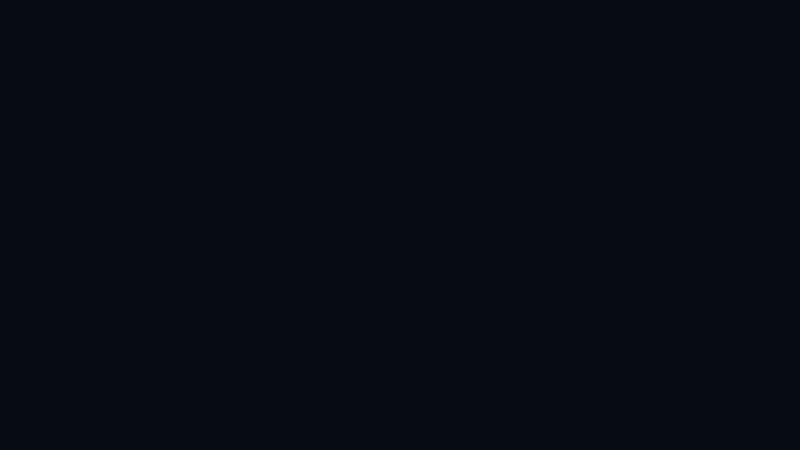


*Graphical user interface, application

Description automatically generatedMinutes increment after seconds overflow (11 1011 ≙ 59).*

**

*‘Button’ is pressed (RC0), this resets the clock.*

*Shows how an underflow is handled.*

**

*Pressing ‘buttons’ to increment / decrement hours.*

# Conclusion

The project was a lot of fun and each team member learned valuable skills while working on it. The outcome of the project was a success. The goal was reached within reasonable amount of time. The acquired knowledge from the ongoing lectures was utilized.

Working on this project deepened our understanding of pic programming and embedded systems in general.

# Literature

* <https://stackoverflow.com/questions/14564813/how-to-convert-an-integer-to-a-character-array-using-c>
* <https://stackoverflow.com/questions/15850042/xcode-warning-implicit-declaration-of-function-is-invalid-in-c99>
* PIC18F2420/2520/4420/4520 Data Sheet: <https://ww1.microchip.com/downloads/en/DeviceDoc/39631E.pdf>
* Prof. Gontean Lecture & Codesamples
* Tools used:
  + MPLAB X (IDE)
  + Fritzing (Electronic Design)
  + Github (Repository): <https://github.com/AntVil/binary-clock>
  + OBS (Screen Capture)
  + Vegas Pro 13 (Video Editing)