Computer Architecture Lab 3

Basel & Elias

June 25, 2023

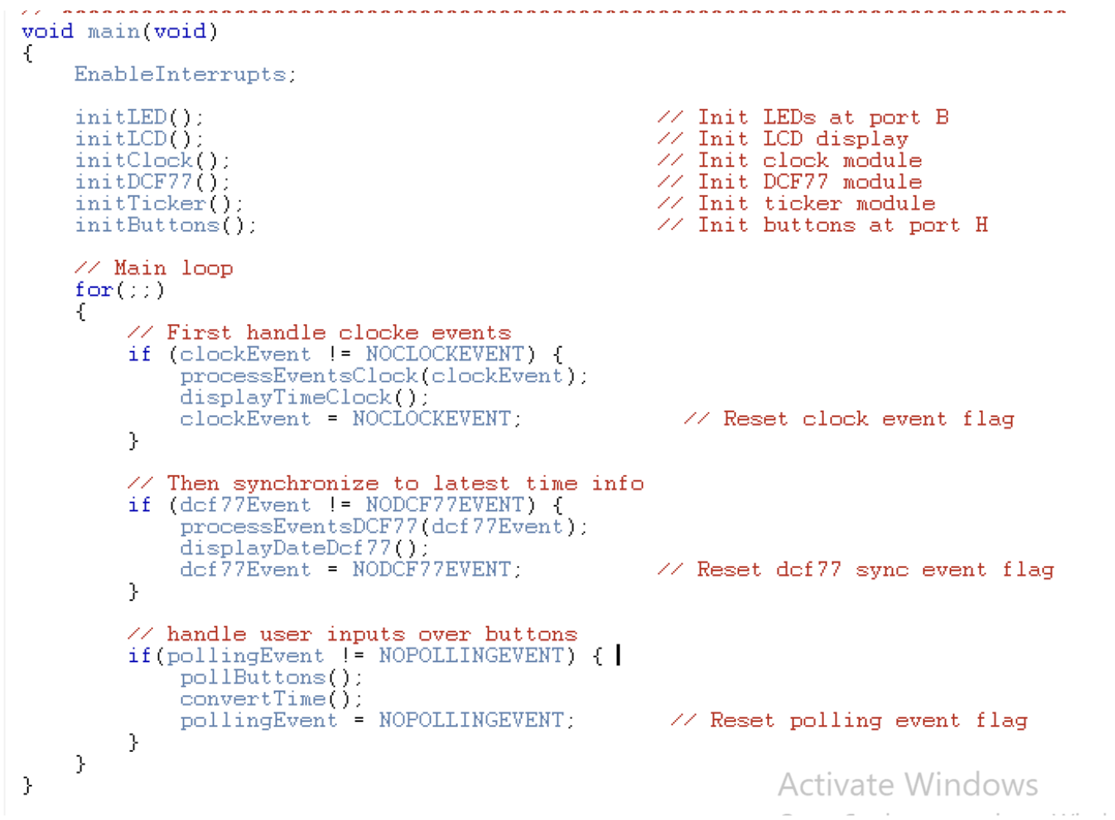
Abstract

In our lab3 we tried to read the dcf77 signal, extract and decode the information to obtain the current date and display the date on dragon12 Board using C language.

1 Modules

In this section we will try to explain each module and its functions and how they work.

1.1 main.c



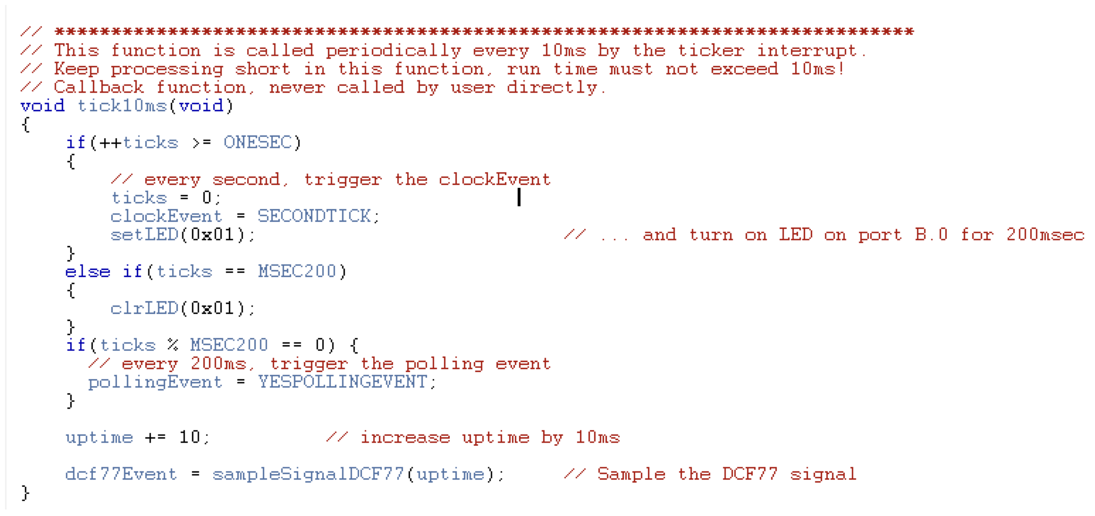
```c
void main(void)
{
    EnableInterrupts;

    initLED();                                  // Init LEDs at port B
    initLCD();                                  // Init LCD display
    initClock();                                // Init clock module
    initDCF77();                                // Init DCF77 module
    initTicker();                               // Init ticker module
    initButtons();                              // Init buttons at port H

    // Main loop
    for(;;)
    {
        // First handle clocke events
        if (clockEvent != NOCLOCKEVENT) {
            processEventsClock(clockEvent);
            displayTimeClock();
            clockEvent = NOCLOCKEVENT;              // Reset clock event flag
        }

        // Then synchronize to latest time info
        if (dcf77Event != NODCF77EVENT) {
            processEventsDCF77(dcf77Event);
            displayDateDcf77();
            dcf77Event = NODCF77EVENT;            // Reset dcf77 sync event flag
        }

        // handle user inputs over buttons
        if(pollingEvent != NOPOLLINGEVENT) {
            pollButtons();
            convertTime();
            pollingEvent = NOPOLLINGEVENT;         // Reset polling event flag
        }
    }
}
```

Figure 1: main.c file source code

As shown in Figure 4. In our main.c module, we do initialization for the hardware peripherals and then we do an endless loop for the ”polling”.

1.2 clock.c

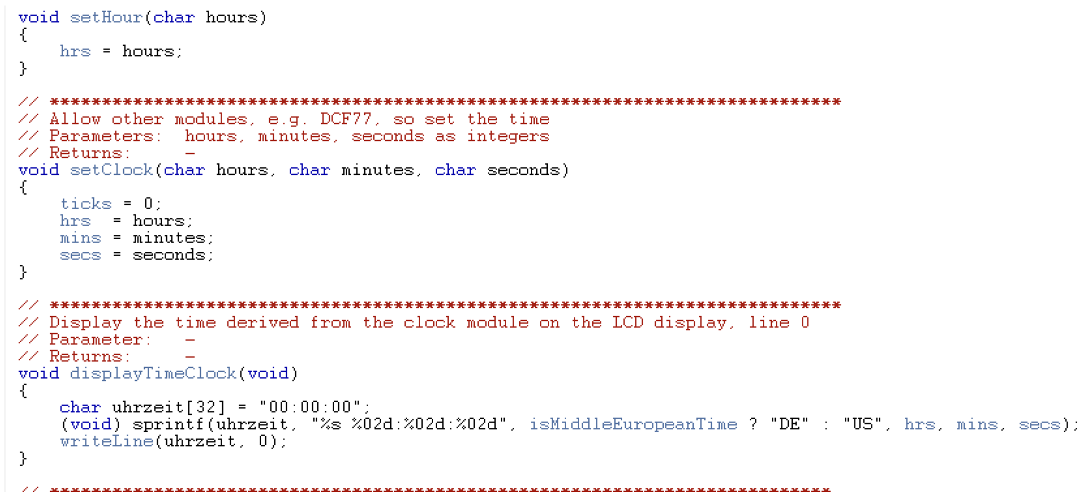
In Figure 2, We Sample our ”dcf77” signal each 10ms with ”sampleSignalDCF77” function (which will be explained in great details later).



```
// *********************************************************************************
// This function is called periodically every 10ms by the ticker interrupt.
// Keep processing short in this function, run time must not exceed 10ms!
// Callback function, never called by user directly.
void tick10ms(void)
{
    if(++ticks >= ONESEC)
    {
        // every second, trigger the clockEvent
        ticks = 0;
        clockEvent = SECONDTICK;
        setLED(0x01);                           // ... and turn on LED on port B.0 for 200msec
    }
    else if(ticks == MSEC200)
    {
        clrLED(0x01);
    }
    if(ticks % MSEC200 == 0) {
      // every 200ms, trigger the polling event
      pollingEvent = YESPOLLINGEVENT;
    }

    uptime += 10;           // increase uptime by 10ms

    dcf77Event = sampleSignalDCF77(uptime);     // Sample the DCF77 signal
}
```

Figure 2: tick10ms function

And now we have some utility function shown in Figure 3.



```
void setHour(char hours)
{
    hrs = hours;
}

// *****************************************************************************
// Allow other modules, e.g. DCF77, so set the time
// Parameters:  hours, minutes, seconds as integers
// Returns:     -
void setClock(char hours, char minutes, char seconds)
{
    ticks = 0;
    hrs   = hours;
    mins = minutes;
    secs = seconds;
}

// *****************************************************************************
// Display the time derived from the clock module on the LCD display, line 0
// Parameter:   -
// Returns:     -
void displayTimeClock(void)
{
    char uhrzeit[32] = "00:00:00";
    (void) sprintf(uhrzeit, "%s %02d:%02d:%02d", isMiddleEuropeanTime ? "DE" : "US", hrs, mins, secs);
    writeLine(uhrzeit, 0);
}

// *****************************************************************************
```

Figure 3: some utility Function in clock.c

1.3 dcf77.c

this module is the heart of our program, it sample the signal, decode it if its valid signal, do check errors and validate the signal.

To separate if we are work on Real Hardware or on the Simulator, we use this ifdef structure shown below.

```
#ifdef _HCS12_SERIALMON
    #define SIMULATOR 1
#else
    #define SIMULATOR 0
#endif
```

After this code we can use variable "SIMULATOR" if its set as 1 it means we are working on the simulator else we are working on the real hardware.

Now we initialize Our ports checking if we are working in the simulator or in the real hardware.

```
void initDCF77(void)
{   setClock((char) dcf77Hour, (char) dcf77Minute, 0);
    displayDateDcf77();

```

```c
    #ifdef SIMULATOR
      initializePortSim();
    #else
      initializePort();
    #endif
}
```

Now "displayDateDcf77" Function prints the line 1 on the LDC Display using global variables.

```c
// *************************
// Display the date derived from the DCF77 signal on the LCD display, line 1
// Parameter:   -
// Returns:     -
void displayDateDcf77(void)
{   char datum[32];
    char weekDayString[3];

    switch (convertedDayOfWeek) {
      case 1:
        (void) sprintf(weekDayString, "Mon");
        break;
      case 2:
        (void) sprintf(weekDayString, "Tue");
        break;
      case 3:
        (void) sprintf(weekDayString, "Wed");
        break;
      case 4:
        (void) sprintf(weekDayString, "Thu");
        break;
      case 5:
        (void) sprintf(weekDayString, "Fri");
        break;
      case 6:
        (void) sprintf(weekDayString, "Sat");
        break;
      case 7:
        (void) sprintf(weekDayString, "Sun");
        break;
      default:
        (void) sprintf(weekDayString, "err");
    }

    (void) sprintf(datum, "%s %02d.%02d.%04d",weekDayString, convertedDay,
    convertedMonth, convertedYear);
    writeLine(datum, 1);
}
```

In this function we read and sample a signal, we check the internals of valid 1 and valid 0 given in our lab3 homework document.

```c
// *************************
//  Read and evaluate DCF77 signal and detect events
//  Must be called by user every 10ms
//  Parameter:  Current CPU time base in milliseconds
//  Returns:    DCF77 event, i.e. second pulse, 0 or 1 data bit or minute marker
DCF77EVENT sampleSignalDCF77(unsigned int currentTime)
{
DCF77EVENT event = NODCF77EVENT;


    #ifdef SIMULATOR
      signal = readPortSim();
    #else
      signal = readPort();
    #endif

    if (!signal) {
      setLED(0x02);
    }
      else {
        clrLED(0x02);
      }// Output state of port S.2 on LED B.0

    // if change from 1 to 0
    if(prevSignal && !signal){
       timePulse = currentTime-fallTime;

       //range for valid sec
       if(timePulse > 900 && timePulse < 1100){
        event = VALIDSECOND;
        fallTime = currentTime;
       }
       //range for valid min
       else if(timePulse > 1900 && timePulse < 2100){
        event = VALIDMINUTE;
        fallTime = currentTime;
       }
    }else

    // if change from 0 to 1
    if(!prevSignal && signal){
      riseTime = currentTime;
      timeLow = riseTime-fallTime;
       //range for valid zero bit
      if(timeLow > 70 && timeLow < 130){
         event = VALIDZERO;

      }//range for valid one bit
      else if(timeLow > 170 && timeLow < 230){
         event = VALIDONE;
      }else{
         event = INVALID;
      }
    }
    prevSignal = signal;
    return event;
}
```

Here we try to evaluate the DCF77 Signal,

```
// **************************
// Process the DCF77 events
// Contains the DCF77 state machine
// Parameter:    Result of sampleSignalDCF77 as parameter
// Returns:      -
void processEventsDCF77(DCF77EVENT event)
{
   if(event==VALIDSECOND){
     validSecondsCtr++;
   }else if(event==VALIDMINUTE){

    displaySuccessfulSync(); //display leds as required
    displaySuccessfulDecoding();
    translateData();

     if(dataInvalid() || parityInvalid()){//check paritys
        event = INVALID;
     }else{
        convertTime(); // if we have error-less data, display LCD
        setClock(convertedHour,dcf77Minute,validSecondsCtr);
     }

     validSecondsCtr = 0;
     dataLSB = 0;
     dataMSB = 0;
   }else if(event==VALIDONE){
     if(validSecondsCtr<32){//if smaller that 32 we are in the LS Byte, data is writte
     LSB -> MSB
        dataLSB = dataLSB | ((long)1 << validSecondsCtr); // add 1 bit to the LSB
     }else{
        dataMSB = dataMSB | ((long)1 << (validSecondsCtr-32)); // add 1 bit to the MSB
     }

```

```c
  }else if(event==VALIDZERO){

  }

  if(event==INVALID){
    clrLED(0x08); // led displays as required
    setLED(0x04);
  }
}
```

These Function are Used to do some Error Checking.

```c
// *************************
// checks if the data recieved has invalid values
// Parameter:   -
// Returns:     boolean of check result
char dataInvalid(void){

    if (dcf77Minute <= 59 && dcf77Hour <= 23 && dcf77Day <= 31 || dcf77Day >= 1 &&
    dcf77Month <= 12 && dcf77Month >= 1 && dcf77DayOfWeek <= 7 && dcf77DayOfWeek >= 1){
    return 1;
    }
    return 0;

}

// *************************
// checks if the data recieved has invalid parity bits
// Parameter:   -
// Returns:     boolean of check result
char parityInvalid(){
    return (parity1%2 || parity2%2 || parity3%2);
}
```

This function helps us translating the data from data frame meaning data, Year,month,day,minute.. All have specific internal and need to get extraced differently.

```c
void translateData(){
    char numArray[] = {1,2,4,8,10,20,40,80};

    parity1 = 0;
    parity2 = 0;
    parity3 = 0;

    dcf77Minute = 0;
    dcf77Hour = 0;
    dcf77Day = 0;
    dcf77DayOfWeek = 0;
    dcf77Month = 0;
    dcf77Year = 2000;
    for(i = 0;i<8;i++){
      // translating year
      dcf77Year+= ((dataMSB>> (i+18)) & 1)*numArray[i];
      // translating parity bit 1
      parity1+= ((dataLSB>> (i+21)) & 1);

      if(i<7){
        // translating minute
        dcf77Minute+= ((dataLSB>> (i+21)) & 1)*numArray[i];
        // translating parity bit 2
        if(i<3){
          parity2+= ((dataLSB>> (i+29)) & 1);
        }else{
          parity2+= ((dataMSB>> (i-3)) & 1);
        }
      }
      if(i<6){
        // translating day
        dcf77Day+= ((dataMSB>> (i+4)) & 1)*numArray[i];
        // translating hour
        if(i<3){
          dcf77Hour+= ((dataLSB>> (i+29)) & 1)*numArray[i];
        }else{
          dcf77Hour+= ((dataMSB>> (i-3)) & 1)*numArray[i];
        }

      }
      if(i<5){
        // translating month
        dcf77Month+= ((dataMSB>> (i+13)) & 1)*numArray[i];
      }
      if(i<3){
        // translating weekday
        dcf77DayOfWeek+= ((dataMSB>> (i+10)) & 1)*numArray[i];
      }
    }

    // translating parity bit 3
    for(i = 4;i<26;i++){
       parity3 += (dataMSB>> i) & 1;
    }
}
```

The Following function converts the time when Timezone is changed.

```
// *************************
// converts the current time/date from recieved raw data into the respective format
// middle european time (DE) or eastern time (US)
// Parameter:   -
// Returns:     -
void convertTime(){
   convertedYear = dcf77Year;
   convertedMonth = dcf77Month;
   convertedDay = dcf77Day;
   convertedHour = dcf77Hour;
   convertedDayOfWeek = dcf77DayOfWeek;
   if(!isMiddleEuropeanTime){
      convertedHour -= 6;
      // manages the "underflow" of time if conversion yields negative day,month,hour values
      if(convertedHour < 0){
          decrementDay();//if hours are below 0 dec days
         if(convertedDay < 1){//if days below 1 dec month
            convertedMonth-=1;
            if(convertedMonth < 1){//if month below 1 decrement year
             decrementMonth();
            }
            convertedDay = daysInMonth(convertedMonth,convertedYear);
         }
      }
   }
   setHour(convertedHour);
}
```

The Following Given Functions are some small utility functions.

```
// *************************
// converts the current time/date from recieved raw data into the respective format
// middle european time (DE) or eastern time (US)
// Parameter:   -
// Returns:     -
void convertTime(){
   convertedYear = dcf77Year;
   convertedMonth = dcf77Month;
   convertedDay = dcf77Day;
   convertedHour = dcf77Hour;
   convertedDayOfWeek = dcf77DayOfWeek;
   if(!isMiddleEuropeanTime){
      convertedHour -= 6;
      // manages the "underflow" of time if conversion yields negative day,month,hour values
      if(convertedHour < 0){
          decrementDay();//if hours are below 0 dec days
         if(convertedDay < 1){//if days below 1 dec month
            convertedMonth-=1;
```

```
                if(convertedMonth < 1){//if month below 1 decrement year
                 decrementMonth();
                }
                convertedDay = daysInMonth(convertedMonth,convertedYear);
            }
        }
    }
    setHour(convertedHour);
}
```

1.4 buttons.c

In This module we read the buttons, they are only used to switch The TimeZone

```
void initButtons(){
   DDRH = 0x00;                          //Port H as inputs
}

void pollButtons(){
    #ifdef SIMULATOR
       if(PTH_PTH3){
         isMiddleEuropeanTime = !isMiddleEuropeanTime;
       }
    #else
       if(!PTH_PTH3){
         isMiddleEuropeanTime = !isMiddleEuropeanTime;
       }
    #endif
}
```

Note: Please note the modules that not documented are the ones that aren't changed at all. such as: "lcd.c, led.c, ticker.asm, dcf77Sim.c, ...);

2 Flowcharts

Please note that if you zoom in the picture you can read the flowcharts easily.

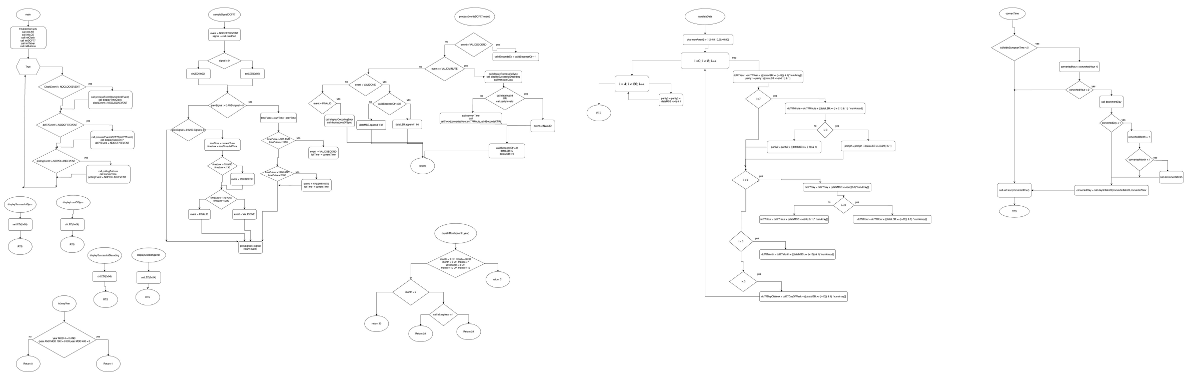


Figure 4: Flowcharts of main functions

References