Computerarchitecture - laboratory 3

Design Document

Radio-controlled Clock with DCF77 Interface

created by

Coskunyürek, Enes 764552

Kandemir, Tolgahan 761469

on May 26, 2021

at the university of applied sciences

Esslingen

Contents

1 Laboratory Introduction

- Implementing a complex application
- Mixed programming in C and assembler

In the second lab we developed a clock. The clock was driven by a timer, which triggered an interrupt every 10ms. A software counter in the ISR create a 1s tick, from which minutes and hours were derived. Unfortunately, this clock loses its time value, when turned off, and thus must be set manually when turned on. Additionally, because the quarts oscillator's precision is limited, the time will drift slightly, when the clock programs runs a long time. Another drawback is that the clock must be manually set when time changes from standard time (MEZ) to daylight savings time (MESZ) and vice versa.

In lab 3 setting the clock shall be done automatically by using the DCF77 radio signal, which transmits time and date information. In the Corona Edition of this lab the DCF77 radio signal will be simulated.

2 Introduction to the format of the DCF77 Signal

DCF77 is a low frequency radio transmitter, operated by the PTB. The transmitter is located near Frankfurt. Its signal can be received in most parts of Europe. The signal uses a digital amplitude modulation, which can be received by a simple antenna and decoded with a low pass filter an a comparator. In our lab we use an antenna mounted at the windows of the room and a decoder, which outputs a pulse signal, which contains the time and date information in serial BCD code. In the Corona Edition this signal is simulated as close as possible to the real antenna signal. Decoding this information must be done by the software.

When idle, the digital DCF77 signal is H (High). Every second the pulse goes to L (Low) (seconds impulses). Each L impulse contains 1bit of the time and date information. To mark the begin of a new minute, the last seconds (bit 59) before the begin of a minute (bit 0) is missing (minutes mark).

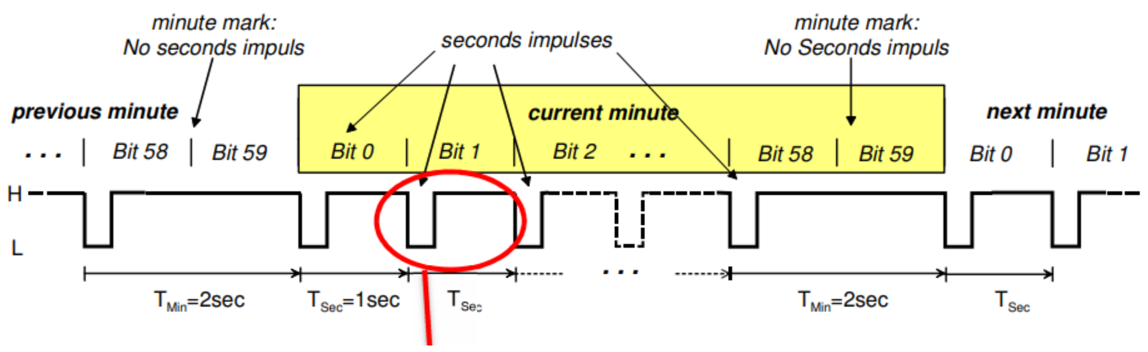


Figure 1: *Overview of the second impulses and minute marks*

the bits of the time and date information are coded via the length of the L-phase of the seconds impulses. A short L-impulse of approximately 100ms codes a '0' bit, a longer L-impulse of approx. 200ms stands for a '1' bit.

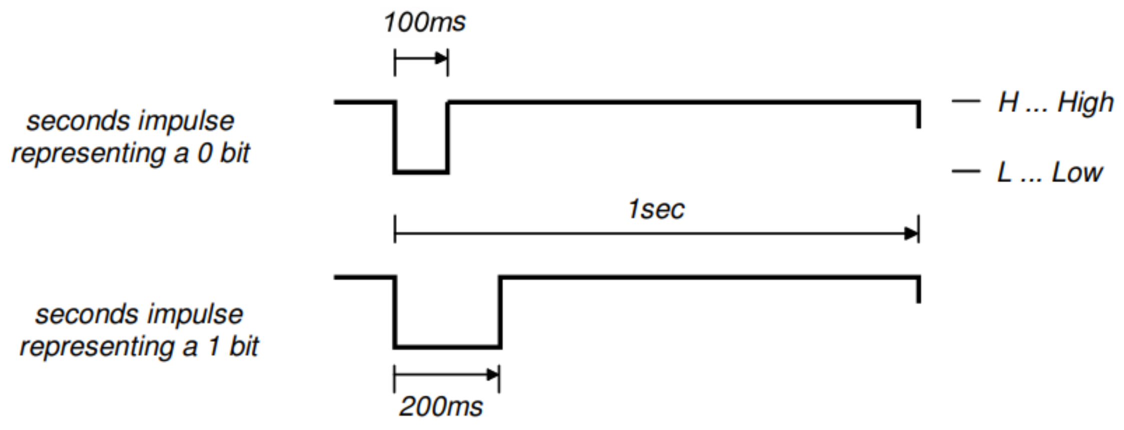


Figure 2: *Coding of 0 and 1 in second impulses*

The 59 data bits transmitted each minute contain the current date and time plus additional control information:

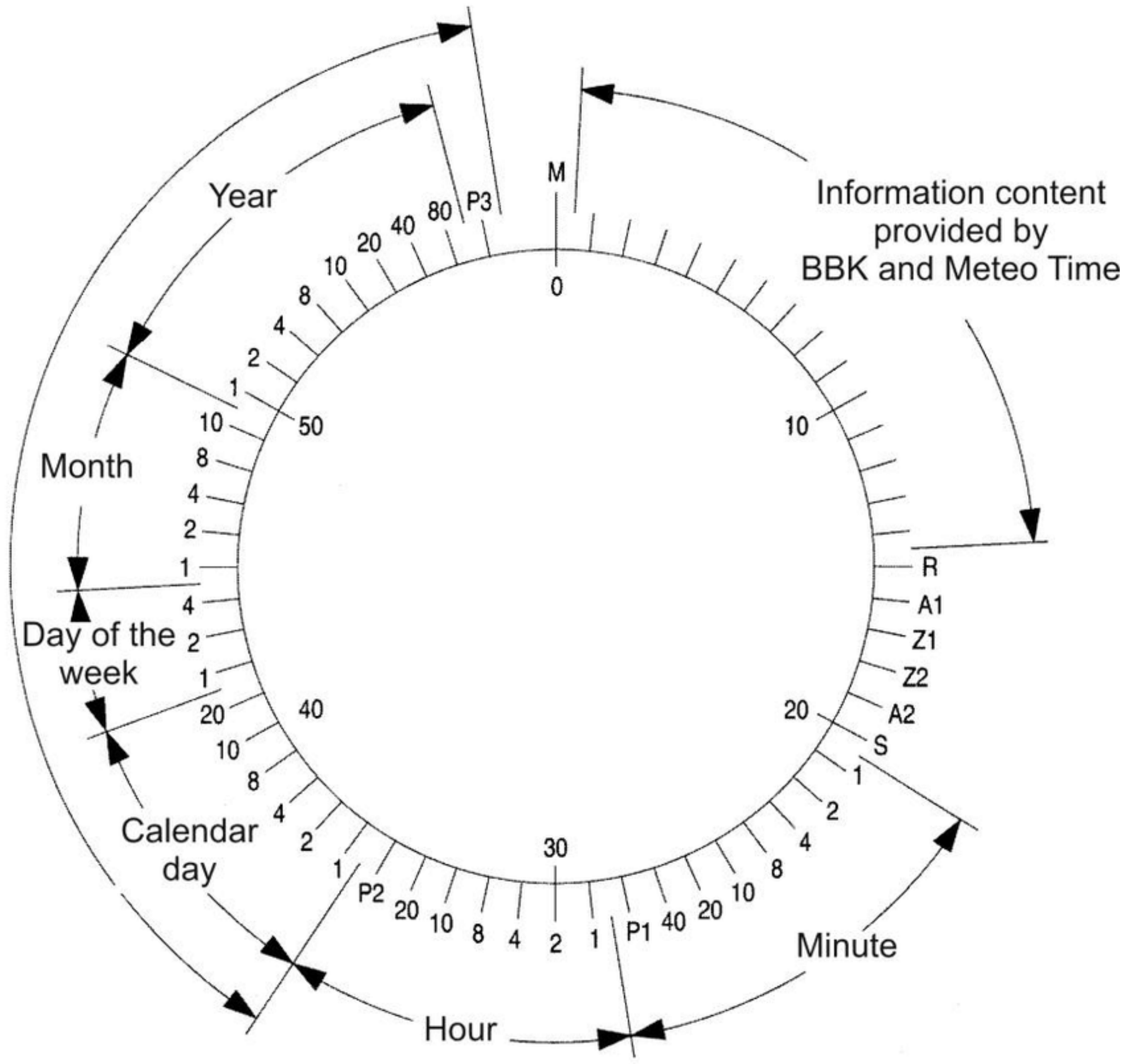


Figure 3: *Overview of the coding scheme of the DCF77 time code frame*

3 Requirements

The requirements for the radio-controlled clock are:

- The first line of the Dragon12s LCD display shall show the time in the following format:
  hours : minutes : seconds
  The second line shall display the date in the following format:
  day . month . year
- Because of the DCF77 transmission fails sometimes (bda radio signal is massice steel-concrete

buildings, transmitter maintenance etc.), the basis concept of l1b 2's clock is still used, i.e. the clock is driven by the timer clock. The DCF77 radio information fails, the clock shall continue to run based on the timer clock.

- The timer clock shall be used to toggle the LED on [port B.0 once per second.
- The LED on port B.1 shall be turned on, when the DCF77 signal is Low and turned off, when the DCF77 signal is High. Thus, when the DCF77 signal is received, this LED will also toggle once per second.
- LED on port B.3 shall be turned on, when a complete and correct DCF77 date and time information has been decoded and turned off at once, when no or wrong data has beed received. Check, if hours are in 0 ... 23, minutes and seconds in the 0 ... 59, days in the 1 ... 32 and months in the 1 ... 12 range. Parity checks are required, When an error is detected, your program shall turn on LED on port B.2, until valid data is received again.
- The DCF77 pulse signal on the real Dragon12 board is connected to port H.0 on the Dragon12 board. As the real DCF77 antenna signal may include spikes and the signal edges may jitter, the signal should be polled every 10ms rather than using port H's interrupt.

Event	*Signal edge*	*Purpose and condition*
NODCF77EVENT	No	Signal did not change since last polling
VALIDSECOND	Negative	Valid second impulse, if T_{Pulse}=1s ± 100ms. The tolerance window is required, as the DCF77 signals period and/or the timer interrupt period may jitter.
VALIDMINUTE	Negative	Valid minute mark, if T_{Pulse}=2s ± 100ms.
VALIDZERO	Positive	Valid 0 bit, if T_{Low}=100ms ± 30ms
VALIDONE	Positive	Valid 1 bit, if T_{Low}=200ms ± 30ms
INVALID	Negative Positive	If T_{Pulse} is outside of the tolerance window. If T_{Low} is outside of the tolerance window.

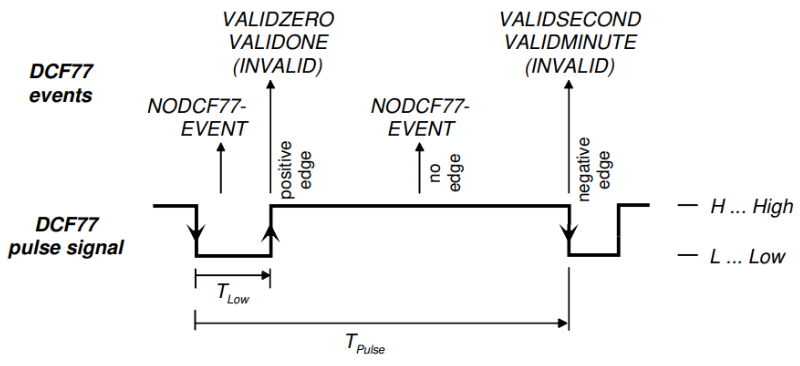


Figure 4: *DCF77 events*

4 Overview of the clock program

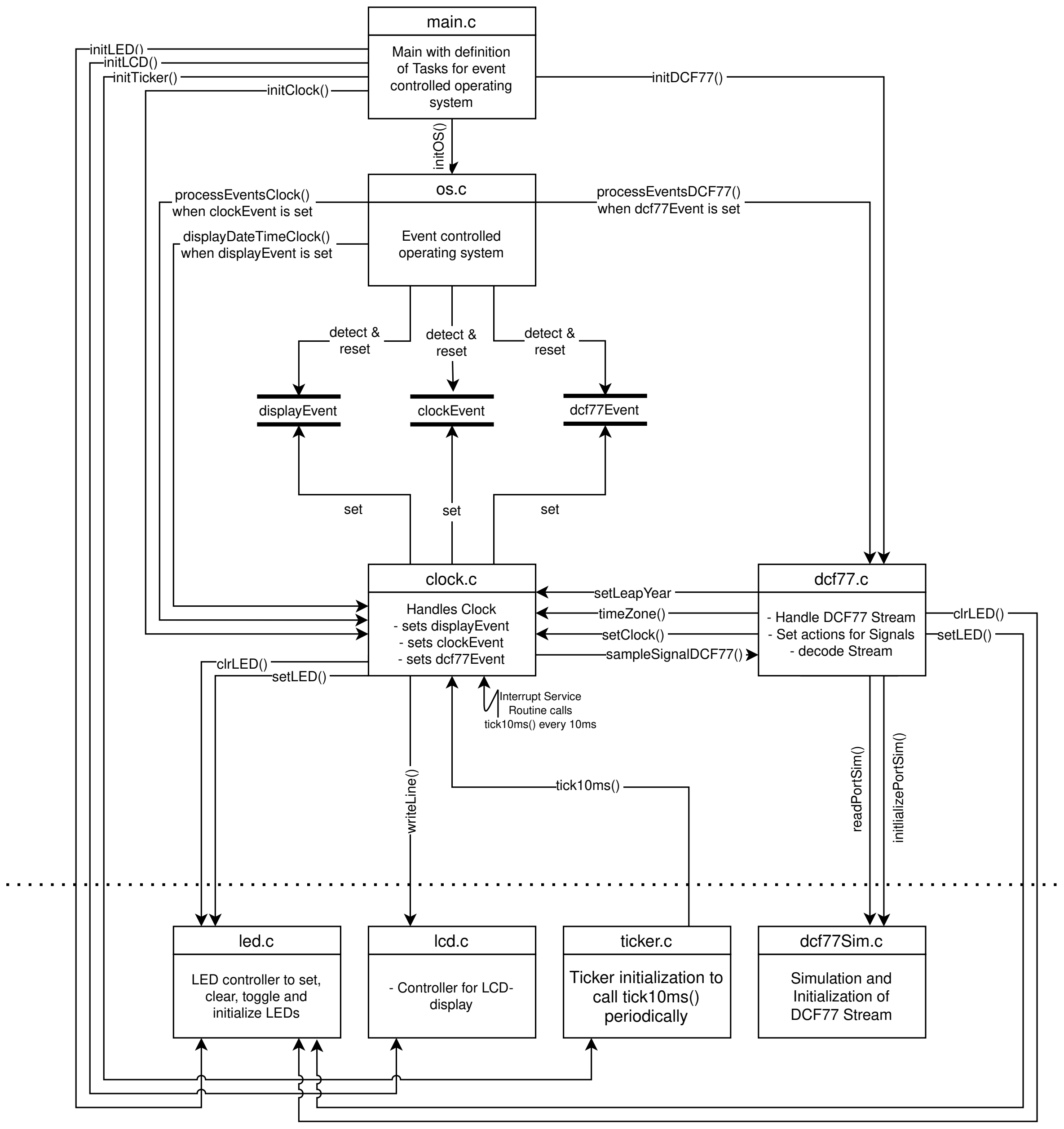

main.c
Main with definition of Tasks for event controlled operating system
initLED()
initLCD()
initTicker()
initClock()
initDCF77()
initOS()
os.c
Event controlled operating system
processEventsClock() when clockEvent is set
displayDateTimeClock() when displayEvent is set
processEventsDCF77() when dcf77Event is set
detect & reset
detect & reset
detect & reset
displayEvent
clockEvent
dcf77Event
set
set
set
clock.c
Handles Clock
- sets displayEvent
- sets clockEvent
- sets dcf77Event
dcf77.c
- Handle DCF77 Stream
- Set actions for Signals
- decode Stream
setLeapYear
timeZone()
setClock()
sampleSignalDCF77()
clrLED()
setLED()
clrLED()
setLED()
Interrupt Service Routine calls tick10ms() every 10ms
writeLine()
tick10ms()
readPortSim()
initializePortSim()
led.c
LED controller to set, clear, toggle and initialize LEDs
lcd.c
- Controller for LCD-display
ticker.c
Ticker initialization to call tick10ms() periodically
dcf77Sim.c
Simulation and Initialization of DCF77 Stream

Figure 5: *overview*

5 Variables

5.1 Global Variables

location	name	description
clock.c	CLOCKEVENT clockEvent	Event for os to trigger processEventsClock()
clock.c	DISPLAYEVENT displayEvent	Event fo os to trigger displayDateTimeClock()
clock.c	char *weekdays	Pointer to reference days of the week
clock.c	char *descZone	Pointer to reference the Timezone of the clock
clock.c	int zone	Used to differentiate in which zone the clock is set to
clock.c	int weekDecoder	Used to map weekdays into String representing weekdays
clock.c	int maxDayOfMonths[]	Used to differentiate the maximum days of a month
dcf77.c	int tLowCounter	Counter to validate a low period
dcf77.c	int minuteCounter	Counter variable to indivate the minute period
dcf77.c	int secondCounter	Counter to validate a second period
dcf77.c	position	Referrer to index the bit-sequence
dcf77.c	invalid	Variable to indicate a invalid bit-sequence
dcf77.c	bits[59]	Array to store the BCD-bit-sequence

5.2 static Variables

location	name	description
clock.c	char days	represenation of the date
clock.c	char months	*
clock.c	char years	*
clock.c	char hrs	representation of the time
clock.c	char mins	*
clock.c	char secs	*
clock.c	char uptime	used to store the cpu time
clock.c	char ticks	Counter variable for ticker to indicate 200 ms
dcf77.c	int minutes	Variable to store the minutes of bit-sequence
dcf77.c	int hours	Variable to store the hours of bit-sequence
dcf77.c	int day	Variable to store the day of bit-sequence
dcf77.c	int month	Variable to store the month of bit-sequence
dcf77.c	int year	Variable to store the year of bit-sequence
dcf77.c	int weekDecoder	Variable to store the weekDecoder of bit-sequence
dcf77.c	int lastSignal	Variable to store the lastSignal of bit-sequence

6 Module specific flowchart diagrams

6.1 Main (Main.c)

6.1.1 void main()

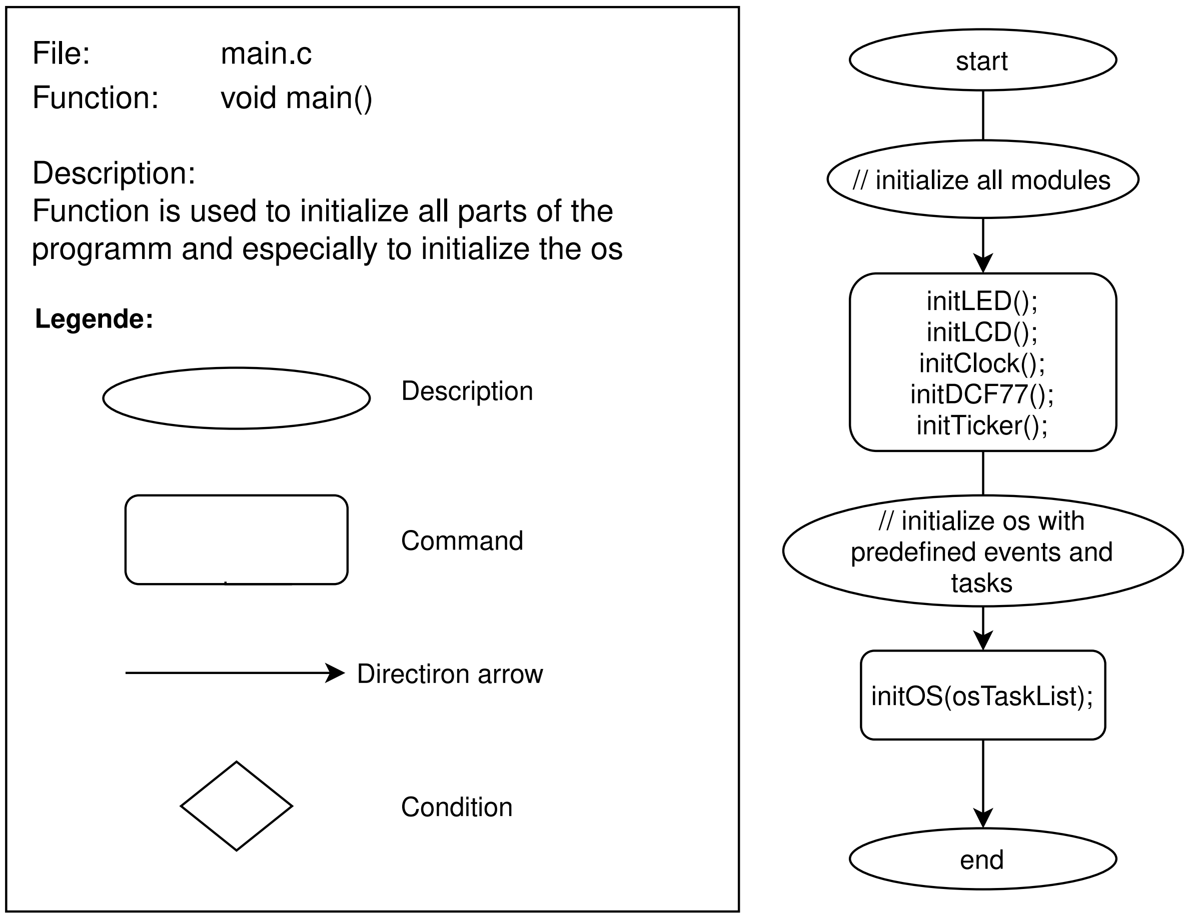


Figure 6: *Flowchart diagramm of function main()*

The main is used as the starting point for the whole programm. First of all, all the components e.g. LED, LCD, clock, DCF77 and ticker need to be initlialized. After all we need to initialize the *operating system.* The initialization function of the os takes a array which consists of a combinations of a function and a associated event. Whenever the event is set, the associated function will be called and the associated event will be reset.

6.2 Ticker (ticker.c)

6.2.1 void initTicker(void)

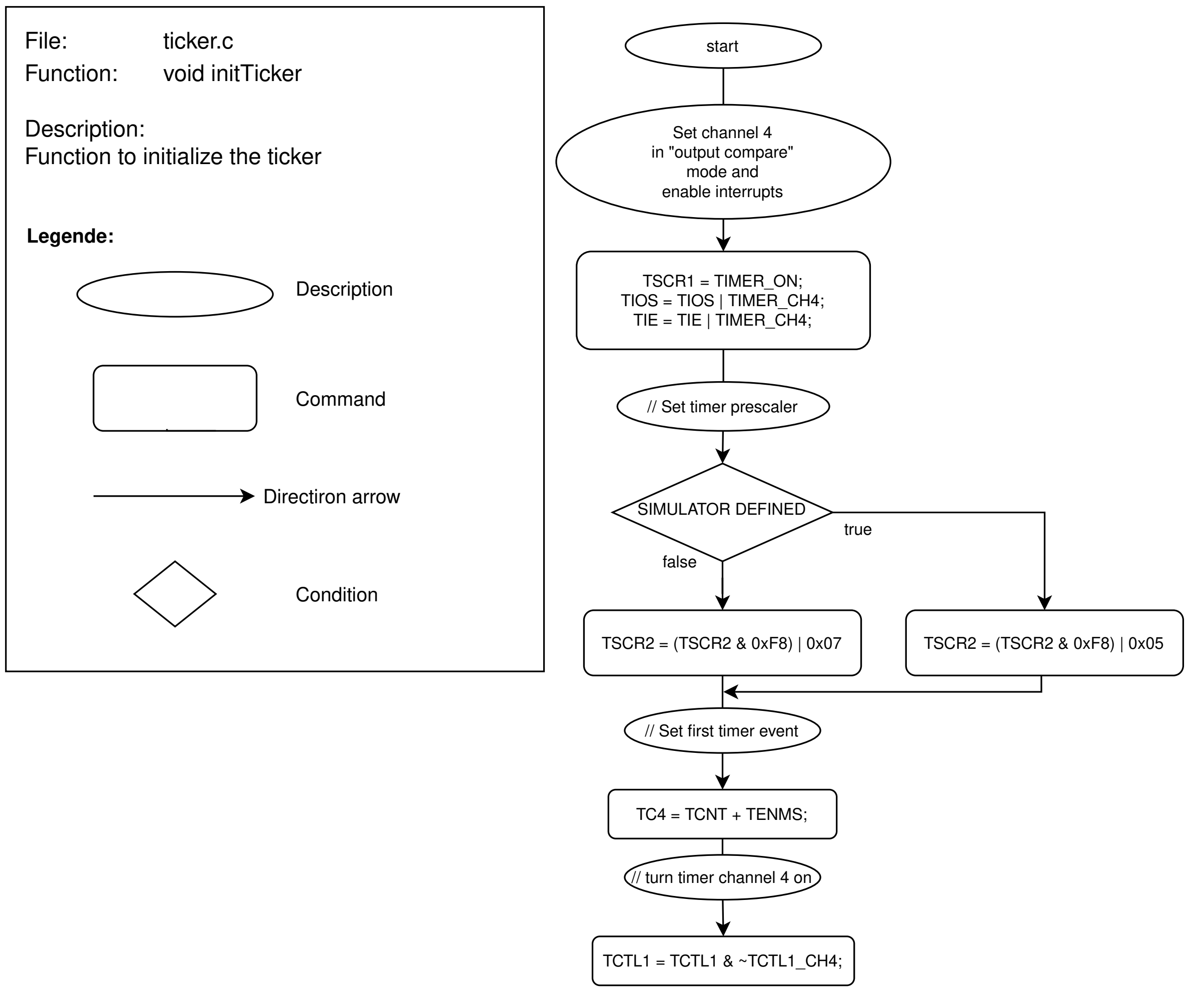


Figure 7: *Flowchart diagramm of function initTicker()*

This function was not written by the laboratory participants either, but was provided by professor Zimmermann. However, most of the commands are used to configure the ticker. We set the channel 4 to output compare mode, set the timer prescalar dependend on the definition of the simulator and to complete the process, we set the first timer event and turn timer channel 4 on.

6.2.2 void interrupt 12 isrECT4()

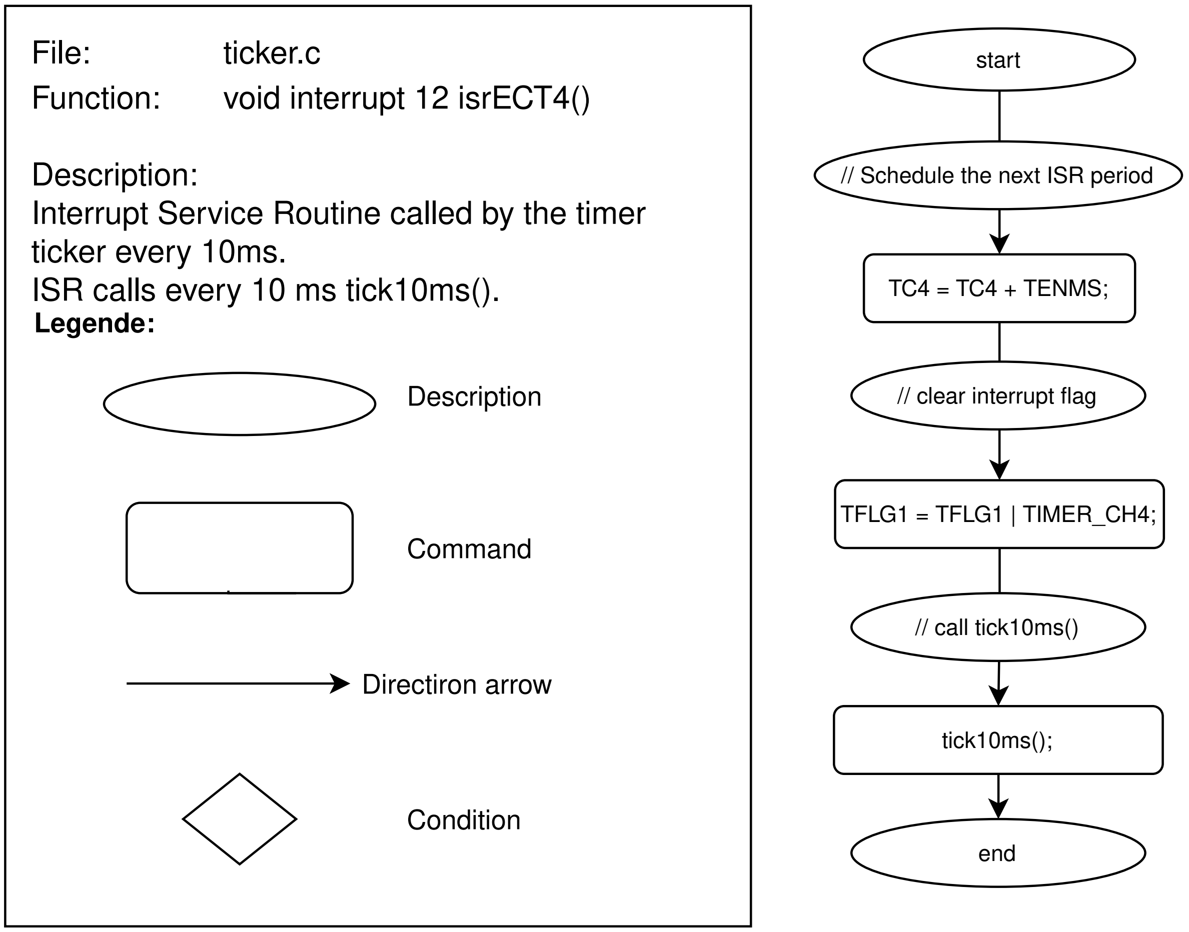

File: ticker.c
Function: void interrupt 12 isrECT4()
Description:
Interrupt Service Routine called by the timer ticker every 10ms.
ISR calls every 10 ms tick10ms().
Legende:
Description
Command
Directiron arrow
Condition
start
// Schedule the next ISR period
TC4 = TC4 + TENMS;
// clear interrupt flag
TFLG1 = TFLG1 | TIMER_CH4;
// call tick10ms()
tick10ms();
end

Figure 8: *Flowchart diagramm of ticker interrupt service routine.*

Again, this function was not written by the laboratory participants either, but was provided by professor Zimmermann. However, this ISR has a easy build up. We schedule the next interrupt by incrementing TC4 with 10 ms, clear the output flag and then call a function tick10ms().

6.3 Clock (clock.c)

6.3.1 void initClock()

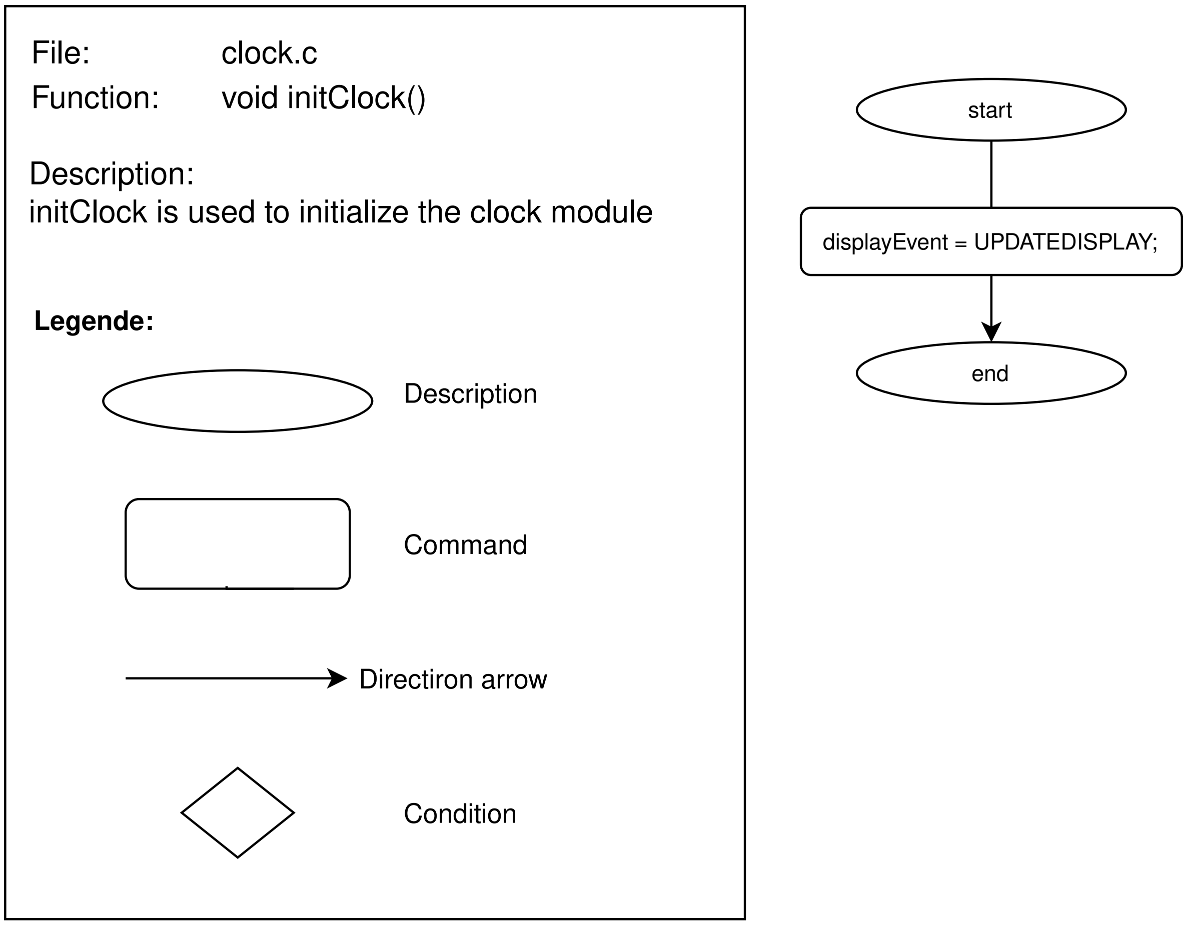
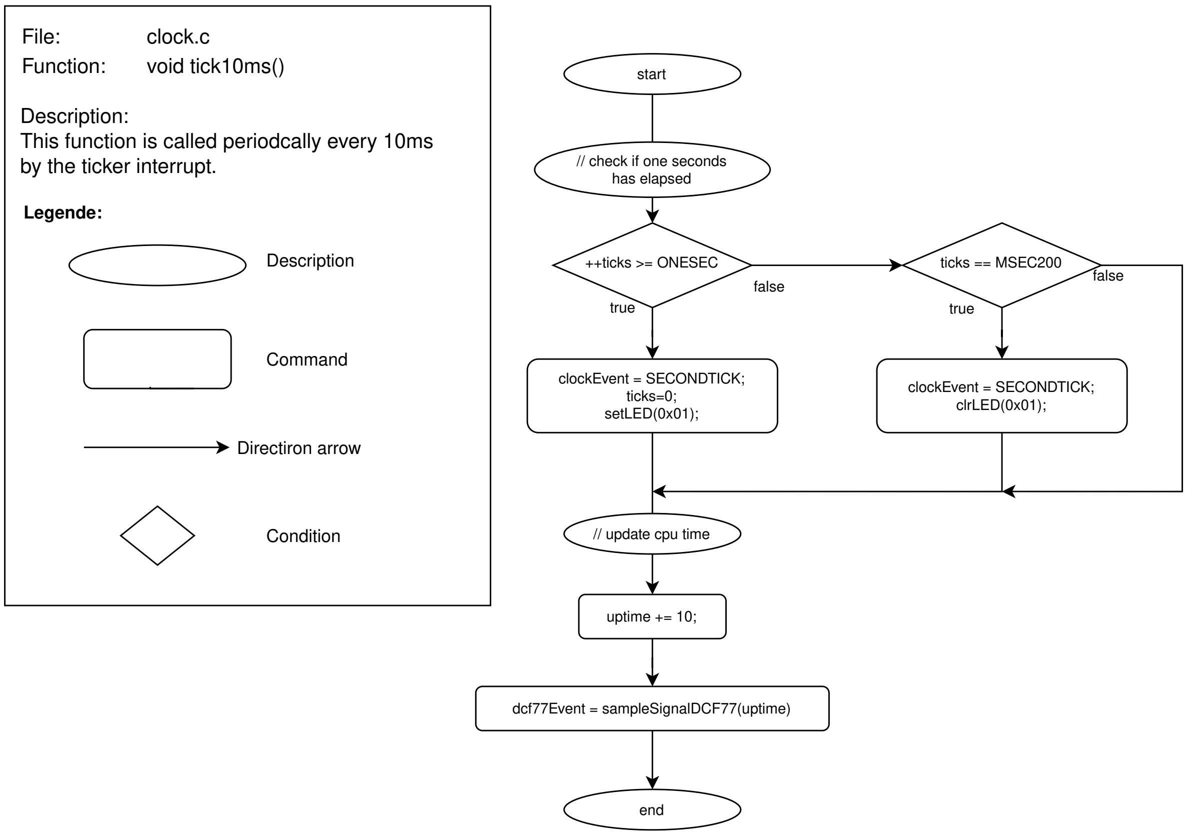

File: clock.c
Function: void initClock()
Description:
initClock is used to initialize the clock module
Legende:
Description
Command
Directiron arrow
Condition
start
displayEvent = UPDATEDISPLAY;
end

Figure 9: *Flowchart diagramm of function initClock*

This very complex function is used to initialize the clock module. It sets the displayEvent to *UPDATEDISPLAY*

6.3.2 void tick10ms()


File: clock.c
Function: void tick10ms()
Description:
Legende:
Description
Command
Directiron arrow
Condition
start
// check if one seconds has elapsed
++ticks >= ONESEC
true
false
ticks == MSEC200
true
false
clockEvent = SECONDTICK; ticks=0; setLED(0x01);
clockEvent = SECONDTICK; clrLED(0x01);
// update cpu time
uptime += 10;
dcf77Event = sampleSignalDCF77(uptime)
end

This function is called periodcally every 10ms by the ticker interrupt.

Figure 10: *Flowchart diagramm of function tick10ms()*

This function is called periodically every 10ms by the interrupt. In this function we increment the ticks of the ticker and check if one second has elapsed. If one second has elapsed, the clockEvent is updated, ticks are reset and the LED on PORT B.0 needs to be set. In case of ticks equals 200 ms, we clear the LED on PORT B.0.

After every call, we increment the uptime with 10ms and set the dcf77Event with the return value of function call *sampleSignalDCF77()*.

6.3.3 void processEventsClock(CLOCKEVENT event)

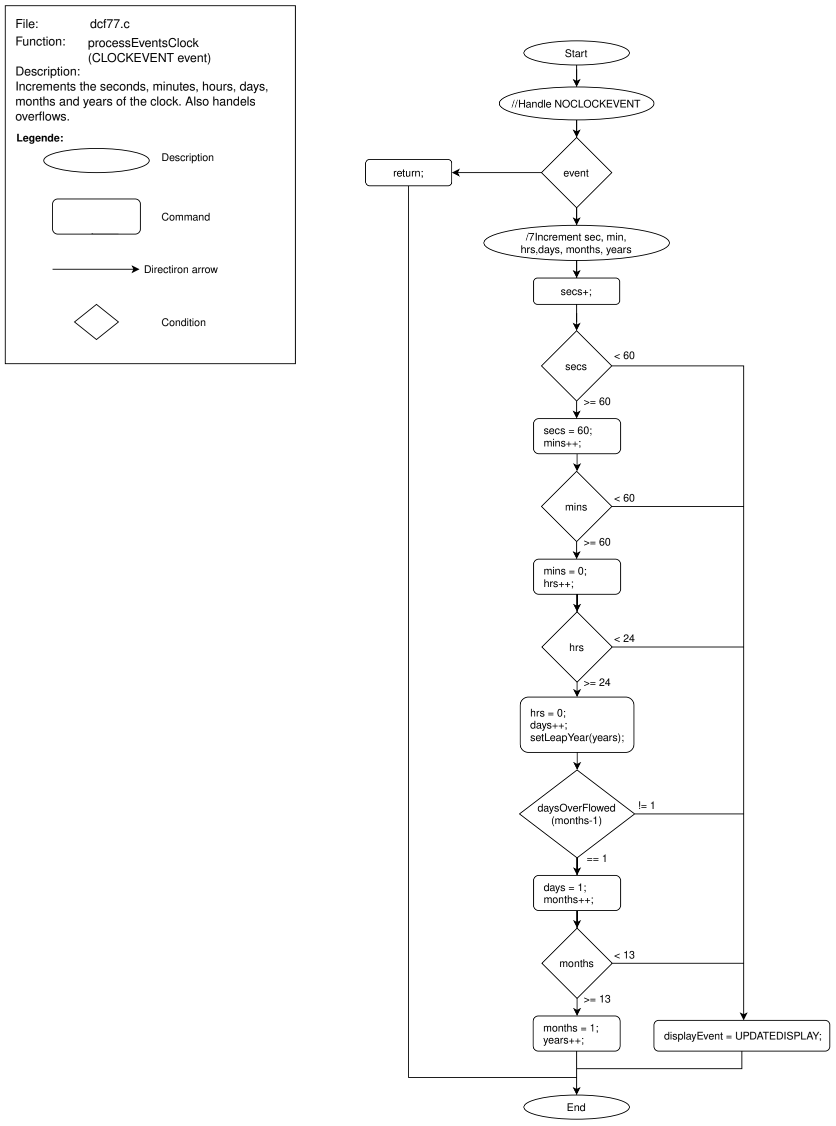

File: dcf77.c
Function: processEventsClock (CLOCKEVENT event)
Description: Increments the seconds, minutes, hours, days, months and years of the clock. Also handels overflows.
Legende:
Description
Command
Directiron arrow
Condition
Start
//Handle NOCLOCKEVENT
event
return;
/7Increment sec, min, hrs,days, months, years
secs+;
secs
< 60
>= 60
secs = 60; mins++;
mins
< 60
>= 60
mins = 0; hrs++;
hrs
< 24
>= 24
hrs = 0; days++; setLeapYear(years);
daysOverFlowed (months-1)
!= 1
== 1
days = 1; months++;
months
< 13
>= 13
months = 1; years++;
displayEvent = UPDATEDISPLAY;
End

Figure 11: *Flowchart diagramm of function processEventsClock()*

This function is used to process the clock itself. Whenever the event isn't *NOCLOCKEVENT* we increment the clock. For that we need to check if the seconds, minutes, hours, days, months and the year overflow. When this process finished, we update the *displayEvent*, so the operating system can call the print method to update the clock on the LCD display.

6.3.4 void setClock(...)

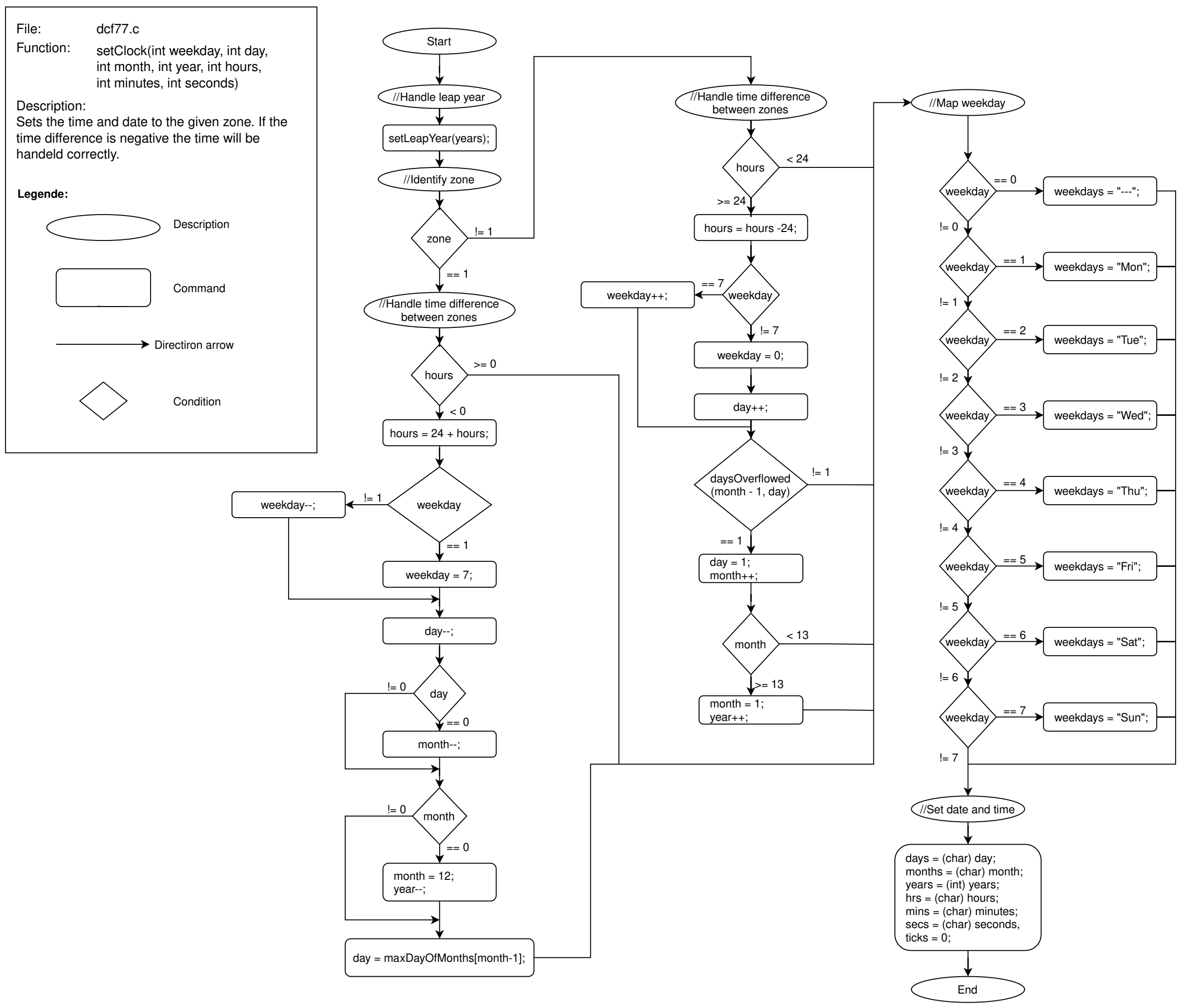


Figure 12: *Flowchart diagramm of function setClock()*

First of all, this diagramm can be a little bit confusing. However, we'll go through them step by step. First of all, we set set the leapYear by calling the function *setLeapYear*. Now we have two Cases:

- Case one occurs, when zone equals to 1. In this case, we check, if we need to handle an underflow of the hours. It this is needed, we subtract the underflow from the actual hours and check, if we need to handle a underflow of the weekdays. If this is necessary, we set the weekday to 7, otherwise, we just decrement the weekday regularly. Now we need to check, if we need to handle an underflow of the day. For that we decrement the days and then check, if the days equals 0. In that case, we decrement the months and therefore check if we need to handle an underflow too. After handling an underflow of the years, we can set our days to the maximum value of the month before and leave the function, by updating the global variables with the new calculated values.
- The other case occurs, if the zone eqals to 0. This case pretty much equals the first case with

the exception that we now handle overflows: If the hours have been overflowed, we subtract the overflow from 24 and check if weekday has been overflowed. In that case of an overflow, we reset the days to 0, otherwise we just increment the weekday with 1. After that we increment the days of a month and check, if the days have reached the maximum value of the associated month. In that case, we reset the day to 1 and check if the months have been overflowed. In case of an overflow, we reset the months to 1 and increment the years. After that we leave the function by configuring the global variables with the new specified time and adjust the printed weekday.

6.3.5 void displayDateTimeClock(DISPLAYEVENT event)

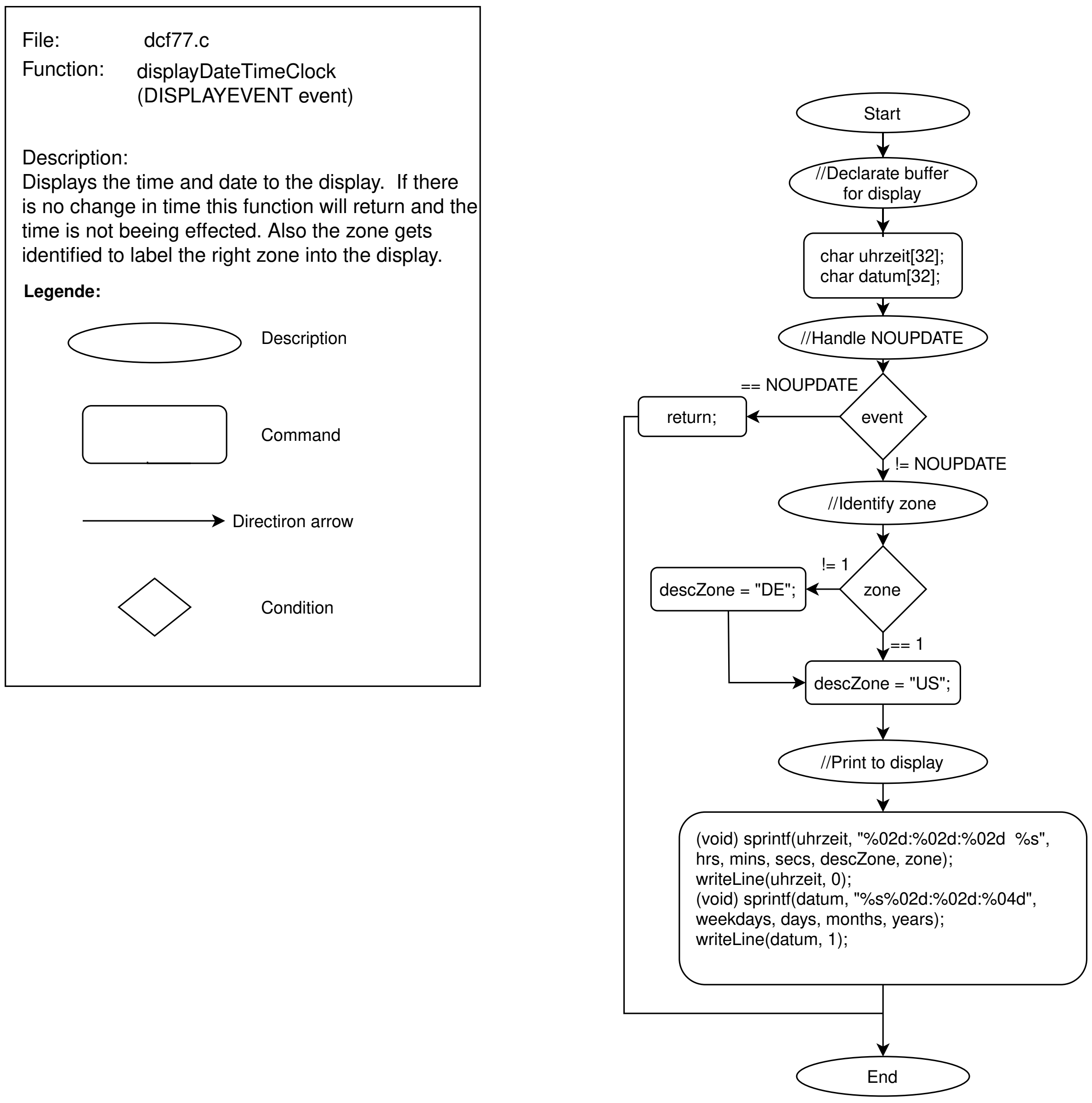

File: dcf77.c
Function: displayDateTimeClock (DISPLAYEVENT event)
Description:
Displays the time and date to the display. If there is no change in time this function will return and the time is not beeing effected. Also the zone gets identified to label the right zone into the display.
Legende:
Description
Command
Directiron arrow
Condition
Start
//Declarate buffer for display
char uhrzeit[32]; char datum[32];
//Handle NOUPDATE
event
== NOUPDATE
return;
!= NOUPDATE
//Identify zone
zone
!= 1
descZone = "DE";
== 1
descZone = "US";
//Print to display
(void) sprintf(uhrzeit, "%02d:%02d:%02d %s", hrs, mins, secs, descZone, zone); writeLine(uhrzeit, 0); (void) sprintf(datum, "%s%02d:%02d:%04d", weekdays, days, months, years); writeLine(datum, 1);
End

Figure 13: *Flowchart diagramm of function displayDateTimeClock*

This function is used to fill the buffers of line 0 and line 1. For that, we write the time and the date into the buffers. After that we can write them onto the LCD display.

6.3.6 void timeZone()

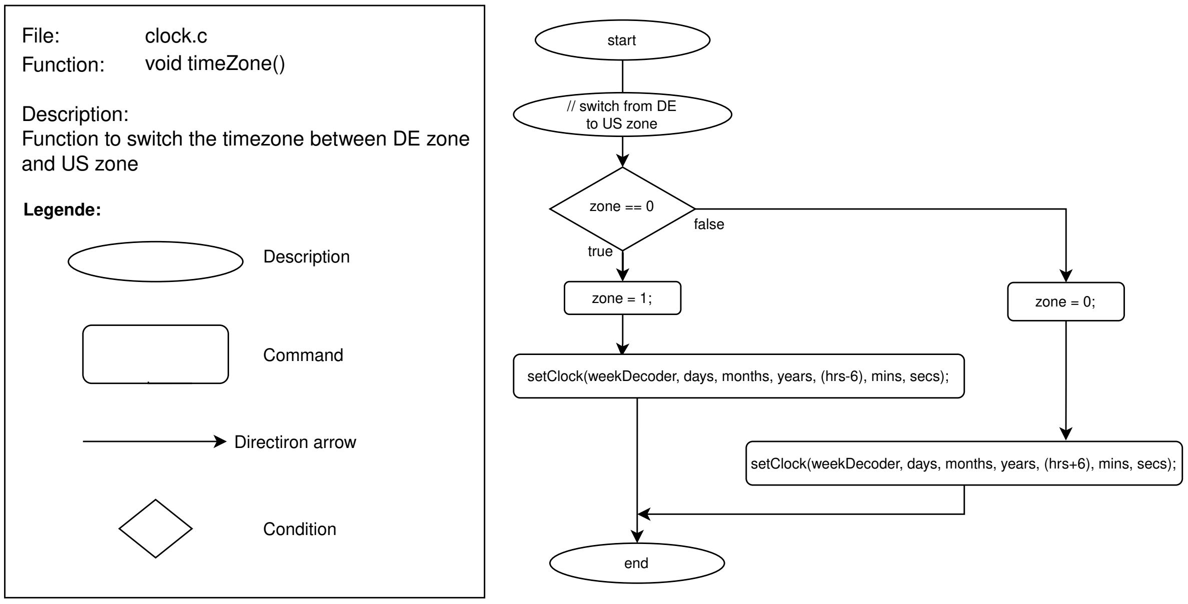

File: clock.c
Function: void timeZone()
Description:
Function to switch the timezone between DE zone and US zone
Legende:
Description
Command
Directiron arrow
Condition
start
// switch from DE to US zone
zone == 0
false
true
zone = 1;
zone = 0;
setClock(weekDecoder, days, months, years, (hrs-6), mins, secs);
setClock(weekDecoder, days, months, years, (hrs+6), mins, secs);
end

Figure 14: *Flowchart diagramm of function timezone*

This function is used to change change the timeZone of the clock. Note that in each case we need to add or subtract 6 hours from the displayed time. When the time is changed we need change our variable zone too.

6.3.7 int daysOverflowed(int month, int day)

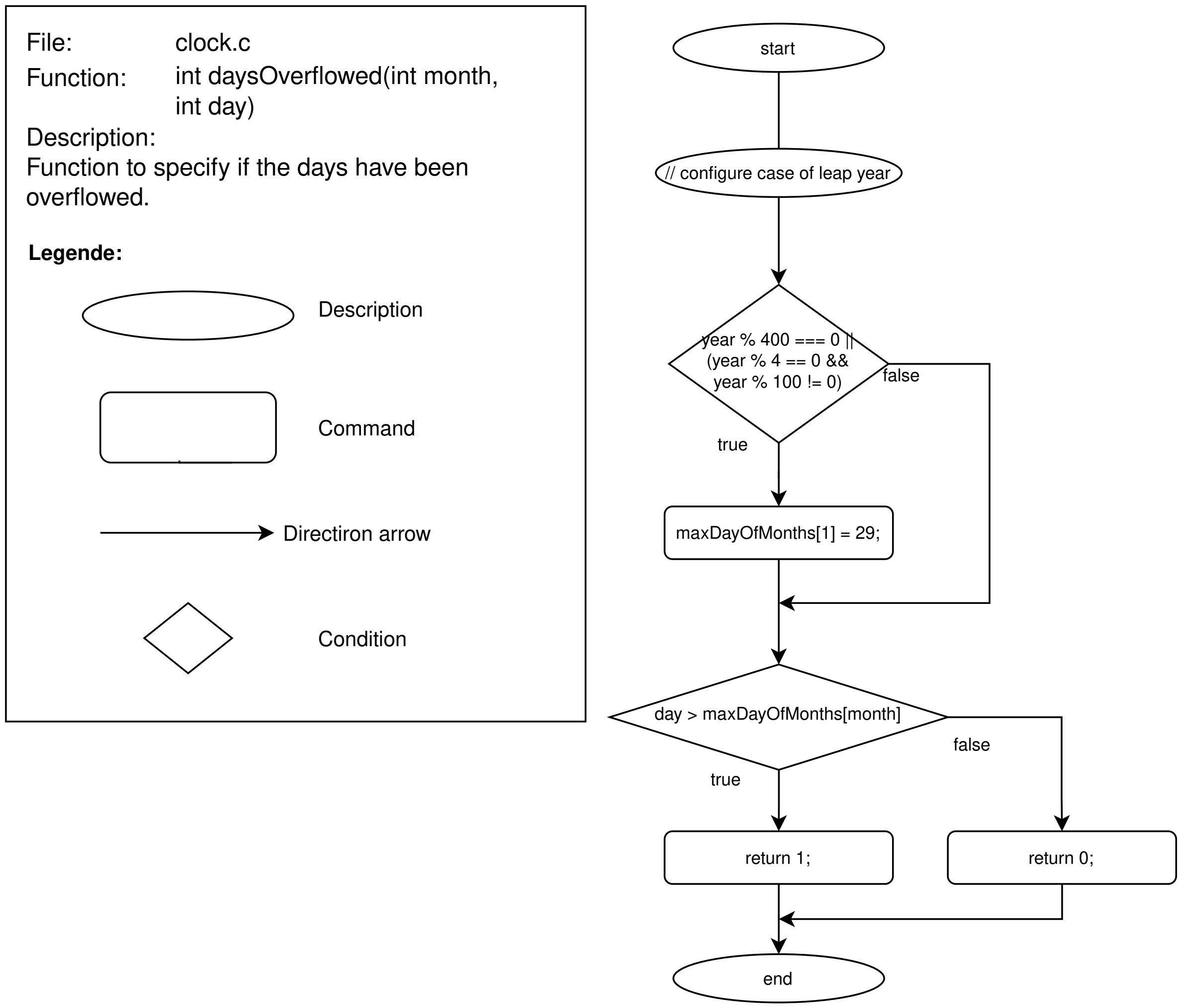

File: clock.c
Function: int daysOverflowed(int month, int day)
Description:
Function to specify if the days have been overflowed.
Legende:
Description
Command
Directiron arrow
Condition
start
// configure case of leap year
year % 400 === 0 || (year % 4 == 0 && year % 100 != 0)
false
true
maxDayOfMonths[1] = 29;
day > maxDayOfMonths[month]
false
true
return 1;
return 0;
end

Figure 15: *Flowchart diagramm of function daysOverflowed*

This function is used to specify if the days have been overflowed or not. Before we check the overflow, we need to configure the case of an leap year. If the day and month combination indicates an overflow, we return 1 (true), if not we return 0 (false).

6.3.8 void setLeapYear(int year)

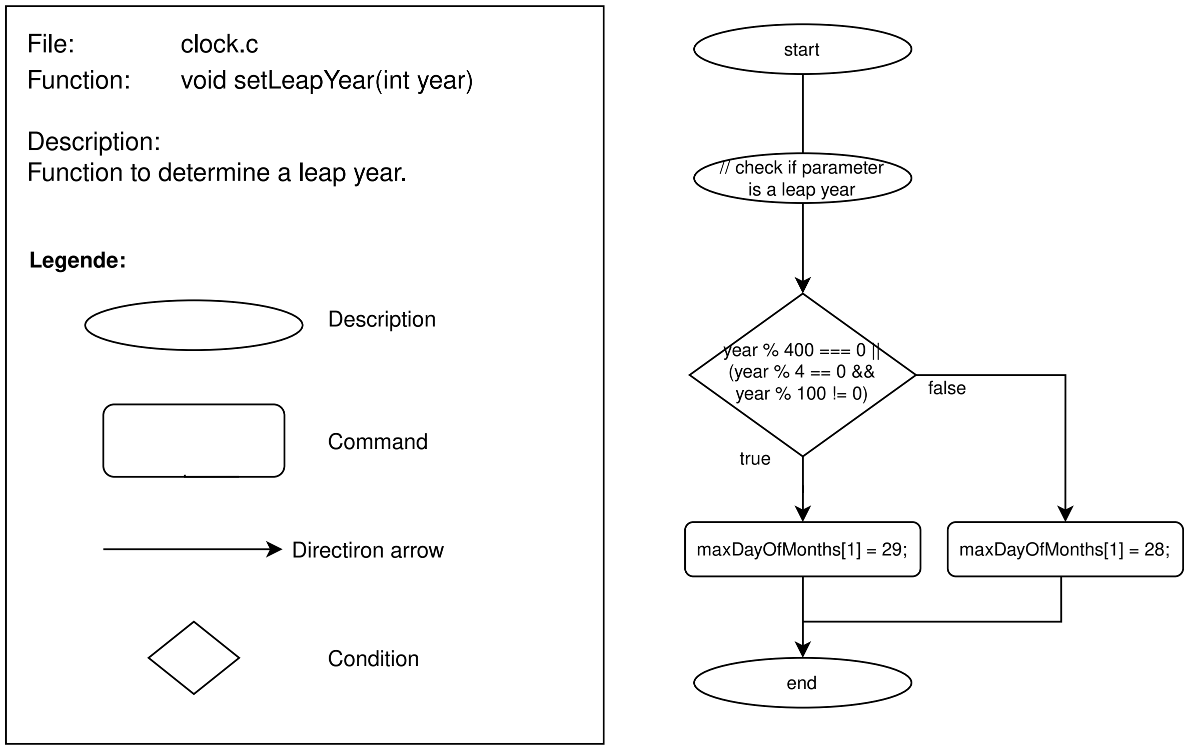

File: clock.c
Function: void setLeapYear(int year)
Description:
Function to determine a leap year.
Legende:
Description
Command
Directiron arrow
Condition
start
// check if parameter is a leap year
year % 400 === 0 || (year % 4 == 0 && year % 100 != 0)
false
true
maxDayOfMonths[1] = 29;
maxDayOfMonths[1] = 28;
end

Figure 16: *Flowchart diagramm of function setLeapYear*

This function is used to set leap years. In case of a leap year, we set the last day of february to 29 in the global variable *maxDayOfMonth[]*.

6.4 DCF77 (dcf77.c)

6.4.1 void initDCF77(void)

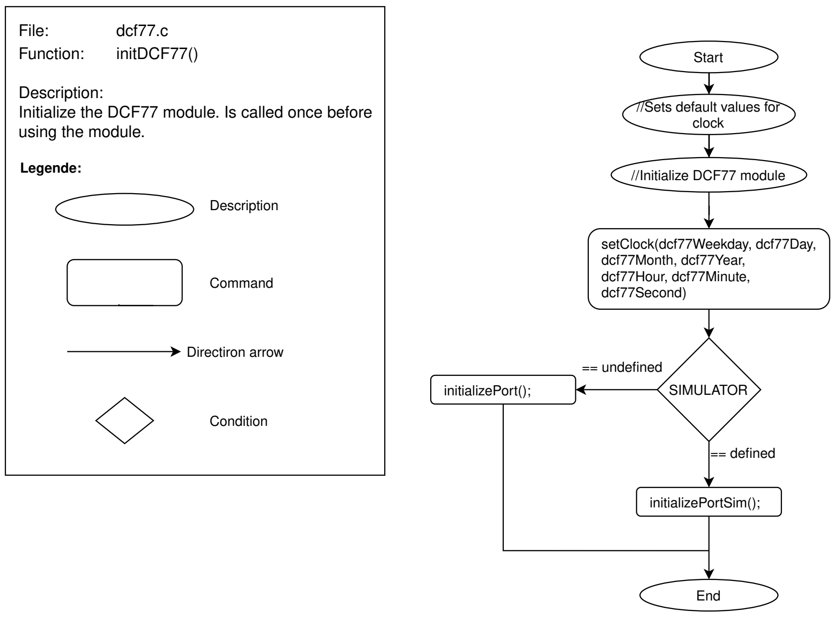

File: dcf77.c
Function: initDCF77()
Description:
Initialize the DCF77 module. Is called once before using the module.
Legende:
Description
Command
Directrion arrow
Condition
Start
//Sets default values for clock
//Initialize DCF77 module
setClock(dcf77Weekday, dcf77Day, dcf77Month, dcf77Year, dcf77Hour, dcf77Minute, dcf77Second)
SIMULATOR
== undefined
initializePort();
== defined
initializePortSim();
End

Figure 17: *Flowchart diagramm of function initDCF7*

Sets default values for the clock. Also initializes the DCF77 module.

6.4.2 DCF77EVENT sampleSignalDCF77(int currentTime)

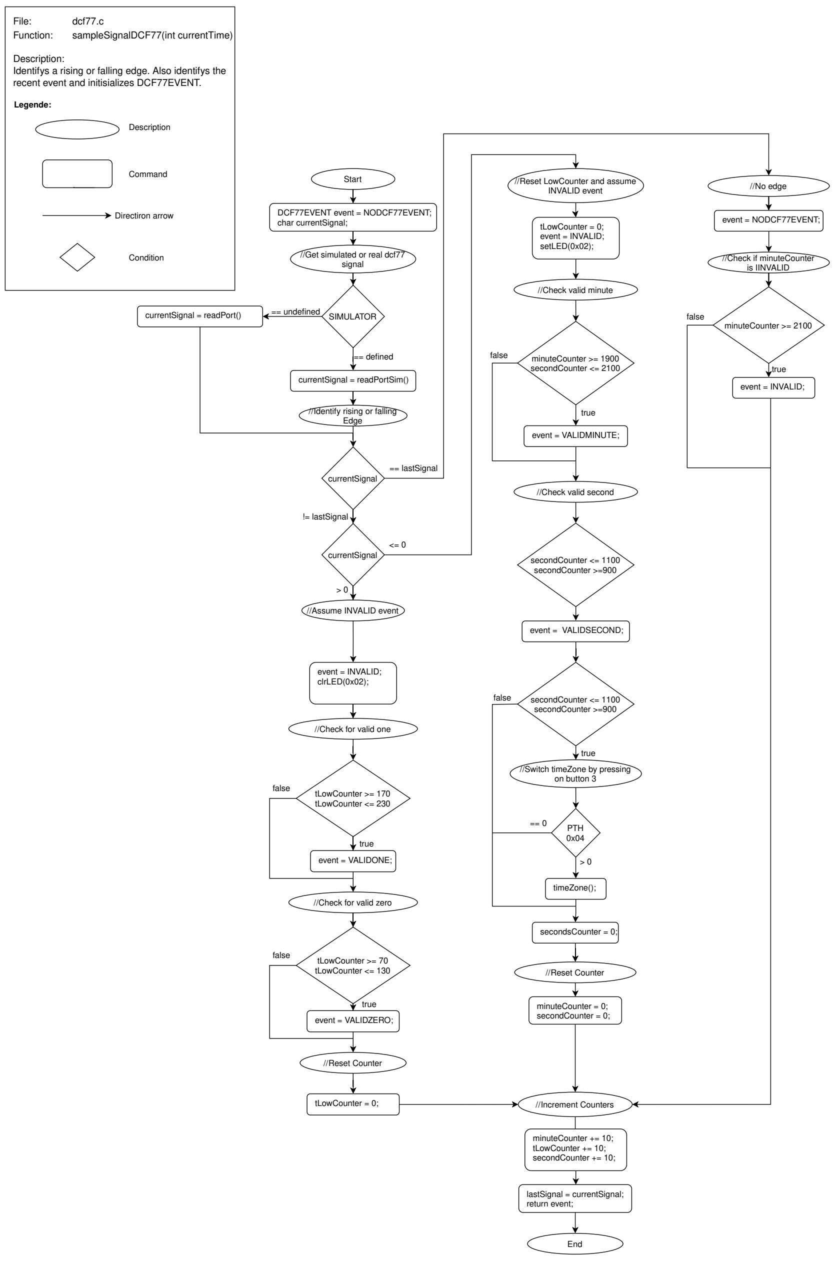

File: dcf77.c
Function: sampleSignalDCF77(int currentTime)
Description:
Identifys a rising or falling edge. Also identifys the recent event and initisializes DCF77EVENT.
Legende:
Description
Command
Directiron arrow
Condition
Start
DCF77EVENT event = NODCF77EVENT; char currentSignal;
//Get simulated or real dcf77 signal
SIMULATOR
== undefined
currentSignal = readPort()
== defined
currentSignal = readPortSim()
//Identify rising or falling Edge
currentSignal
== lastSignal
!= lastSignal
currentSignal
<= 0
> 0
//Assume INVALID event
event = INVALID; clrLED(0x02);
//Check for valid one
tLowCounter >= 170 tLowCounter <= 230
false
true
event = VALIDONE;
//Check for valid zero
tLowCounter >= 70 tLowCounter <= 130
false
true
event = VALIDZERO;
//Reset Counter
tLowCounter = 0;
//Reset LowCounter and assume INVALID event
tLowCounter = 0; event = INVALID; setLED(0x02);
//Check valid minute
minuteCounter >= 1900 secondCounter <= 2100
false
true
event = VALIDMINUTE;
//Check valid second
secondCounter <= 1100 secondCounter >=900
event = VALIDSECOND;
secondCounter <= 1100 secondCounter >=900
false
true
//Switch timeZone by pressing on button 3
PTH 0x04
== 0
> 0
timeZone();
secondsCounter = 0;
//Reset Counter
minuteCounter = 0; secondCounter = 0;
//No edge
event = NODCF77EVENT;
//Check if minuteCounter is IINVALID
minuteCounter >= 2100
false
true
event = INVALID;
//Increment Counters
minuteCounter += 10; tLowCounter += 10; secondCounter += 10;
lastSignal = currentSignal; return event;
End

Figure 18: *Flowchart diagramm of function sampleSignalDCF77*

This function is used to specify the events on the actual signal. This seems a little be overwhelming but we will got through it step by step. First of all, we call the function readPortSim(), that returns the actual value of the signal. For what we will look are falling and rising edges:
The first step is to check if the currentSignal has changed to the last received signal. In this is the case, we will specify if we have rising or falling edges:

- In case of rising edges we assume that the event is invalid. Now we clear the LED on PORT B.1 and check if we have a valid one signal or a valid zero signal. A one signal is valid, if the signal stays low for 170 to 230 ms. In that case, we set the event variable to VALIDONE. A zero signal is valid, if the signal stays low for 70 to 130 ms. In that case, we set the event variable to VALIDZERO. After we have validated a rising edge we need to reset the lowCounter.

- In case of falling edges we assume that the event is invalid, too. For that, we set the LED on PORT B.1 on. Here we need to check, if we have a valid minute or a valid second. A valid minute events needs to be set, if the signal stays high for 1900 to 2100 ms. A valid second instead needs to be stay high for 900 to 1100 ms. Especially in case of a valid second, we check, if the button 3 on PORTH has been pressed and therefore switch the timeZone from DE to US and vice versa.

In case of the signal hasn't been changed from the previous one, we signal that no event has been occured. After that we increment all counters for the next pass and set the lastSignal to the currentSignal.

6.4.3 void processEventsDCF77(DCF77EVENT event)

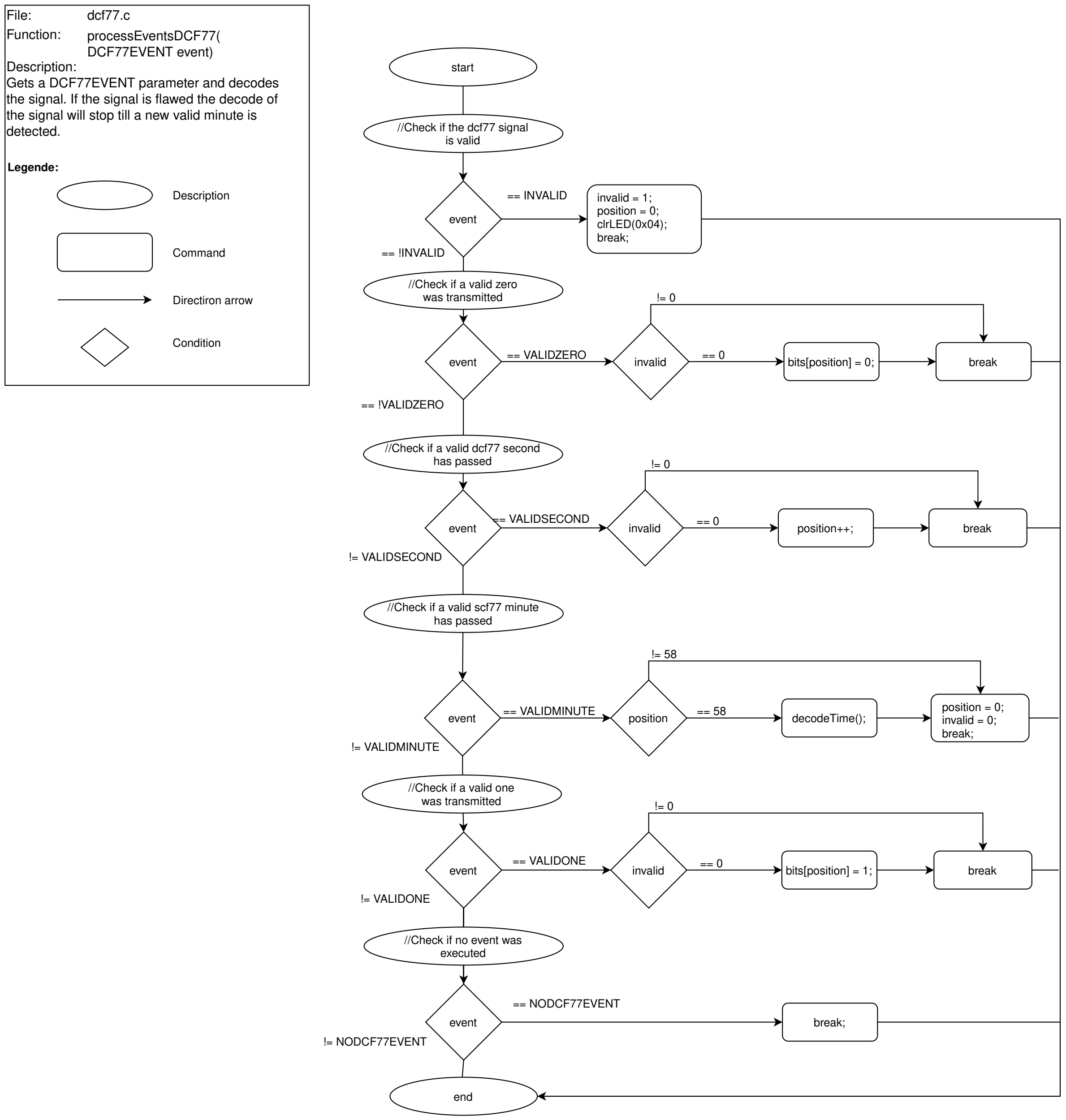

File: dcf77.c
Function: processEventsDCF77(DCF77EVENT event)
Description:
Gets a DCF77EVENT parameter and decodes the signal. If the signal is flawed the decode of the signal will stop till a new valid minute is detected.
Legende:
Description
Command
Directiron arrow
Condition
start
//Check if the dcf77 signal is valid
event
== INVALID
invalid = 1; position = 0; clrLED(0x04); break;
== !INVALID
//Check if a valid zero was transmitted
event
== VALIDZERO
invalid
== 0
!= 0
bits[position] = 0;
break
== !VALIDZERO
//Check if a valid dcf77 second has passed
event
== VALIDSECOND
invalid
== 0
!= 0
position++;
break
!= VALIDSECOND
//Check if a valid scf77 minute has passed
event
== VALIDMINUTE
position
== 58
!= 58
decodeTime();
position = 0; invalid = 0; break;
!= VALIDMINUTE
//Check if a valid one was transmitted
event
== VALIDONE
invalid
== 0
!= 0
bits[position] = 1;
break
!= VALIDONE
//Check if no event was executed
event
== NODCF77EVENT
break;
!= NODCF77EVENT
end

Figure 19: *Flowchart diagramm of function processEventsDCF77*

This function handels the current event and actually decodes the signal into 1s and 0s which are stored in an array called "bits".

INVALID event:
Sets the global variable invalid to 1 and the global variable position to 0. So we ensure that the current transmitted minute will be rejected. Also clearing the LED on Port (0x04).

VALIDZERO event:
Sets a 0 in the bits array with the index of position which represent the current second. Only if the global variable invalid isn't set to 0 this function will set a valid 0.

VALIDONE event:
Sets a 1 in the bits array with the index of position which represent the current second. Only if the global variable invalid isn't set to 0 this function will set a valid 1.

VALIDSECOND event:
Increments position which represent one second of the DCF77 signal. This will only be done when the variable ivalid is 0.

VALIDMINUTE event:
Calls the method decodeDateTime() which updates the clock with the transmitted time of the DCF77 signal. Only if the position equals 58 this function will be called because when the position reaches 58 we can ensure that a full and valid DCF77 signal was transmitted. Also the variable position and invalid are reseted to 0 so we can start to receive a new valid minute with the help of the DCF77 signal.

NODCF77EVENT event:
When the last event and the current event are the same this event will be thrown and nothing will happen.

6.4.4 void decodeDateTime()

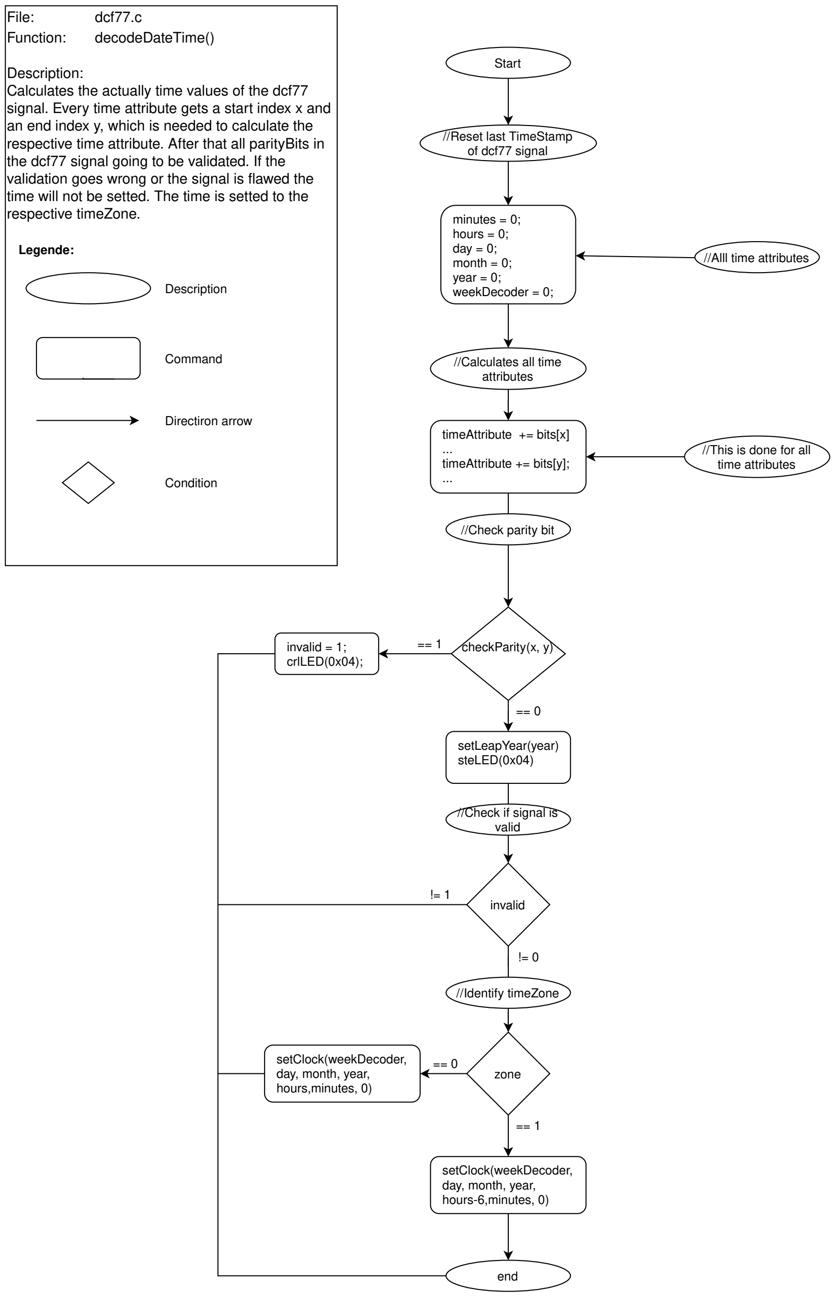
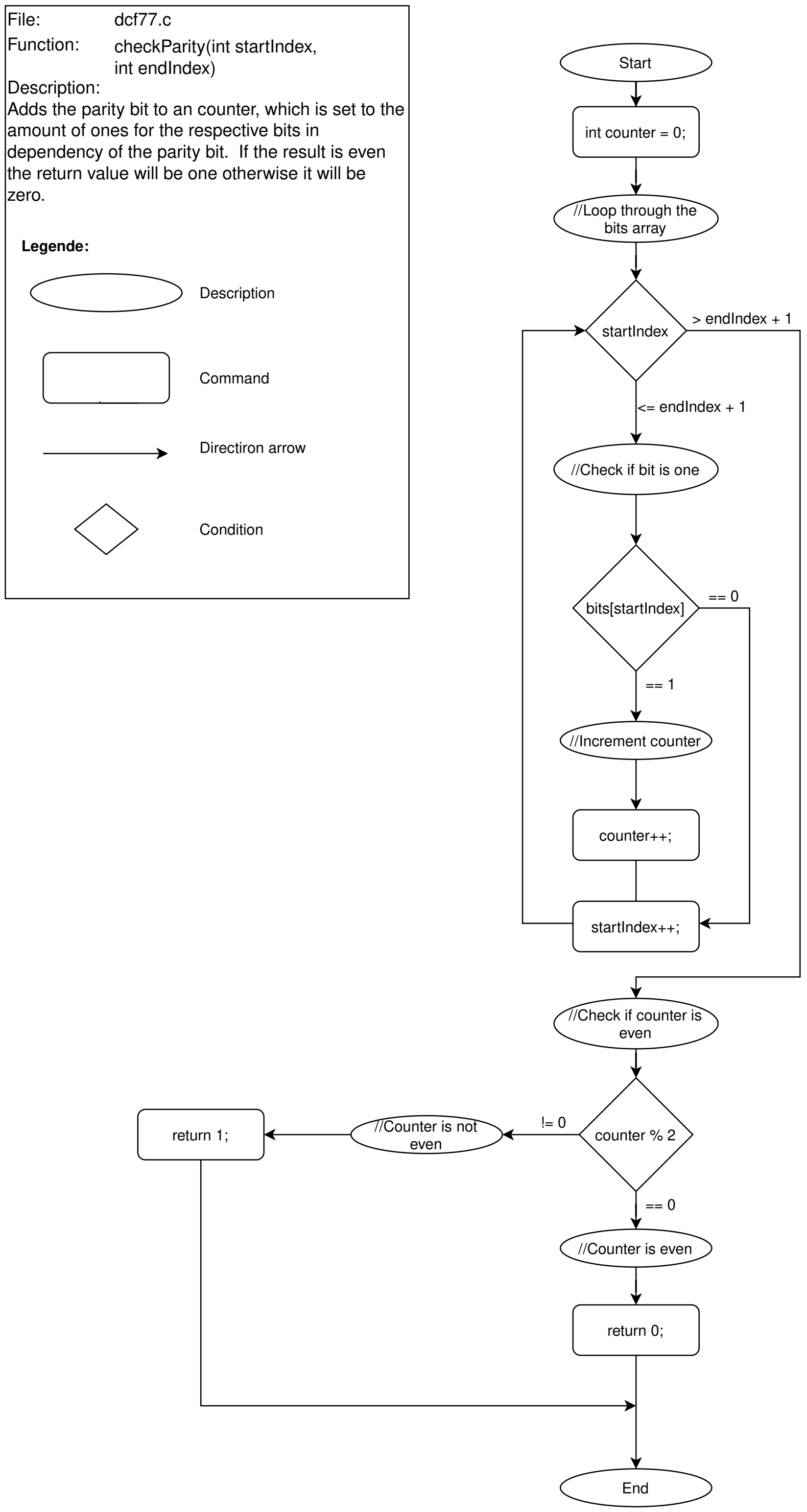

File: dcf77.c
Function: decodeDateTime()
Description:
Calculates the actually time values of the dcf77 signal. Every time attribute gets a start index x and an end index y, which is needed to calculate the respective time attribute. After that all parityBits in the dcf77 signal going to be validated. If the validation goes wrong or the signal is flawed the time will not be setted. The time is setted to the respective timeZone.
Legende:
Description
Command
Directiron arrow
Condition
Start
//Reset last TimeStamp of dcf77 signal
minutes = 0; hours = 0; day = 0; month = 0; year = 0; weekDecoder = 0;
//Alll time attributes
//Calculates all time attributes
timeAttribute += bits[x] ... timeAttribute += bits[y]; ...
//This is done for all time attributes
//Check parity bit
checkParity(x, y)
== 1
invalid = 1; crlLED(0x04);
== 0
setLeapYear(year) steLED(0x04)
//Check if signal is valid
invalid
!= 1
!= 0
//Identify timeZone
zone
== 0
setClock(weekDecoder, day, month, year, hours,minutes, 0)
== 1
setClock(weekDecoder, day, month, year, hours-6,minutes, 0)
end

Figure 20: *Flowchart diagramm of function decodeDateTime*

This function is used to decode the BCD bit stream. The position of the minutes, hours, day, month, year and weekDay have a fixed position in the stream. Index 21-27 is used for the minutes, 29-34 is used for the hours, 36-41 is used for the day of month, 45-49 is used for the month, 50-57 is used for the last two digits of the year. Additionally we need to add 2000 to the BCD encoded year. After that we check the even parity bits at position 28 for the minutes, 35 for the hours and bit at position 58 for the date. In case of a valid date and time, we set the LED on PORT B.2 to on and call the function setClock(), to update the date and time of the internal clock. In case of an invalid date and/or time, we clear the LED on Port B.2.

6.4.5 int checkParity(int startIndex, int endIndex)



File: dcf77.c
Function: checkParity(int startIndex, int endIndex)
Description:
Adds the parity bit to an counter, which is set to the amount of ones for the respective bits in dependency of the parity bit. If the result is even the return value will be one otherwise it will be zero.

Figure 21: *Flowchart diagramm of function checkParity*

Checks the parity bit to the given startIndex and endIndex. The counter is set to the amount of ones included in the given bit sequence including the parity bit. If the amount of ones is even this function will return zero otherwise it will return one.