Radio Clock with DCF77

by Nicolas L. F. September 2011

Abstract

Since the 1980s radio clocks have been popular, and in this article Nicolas guides us through the creation of his own radio clock using the DCF77 time signal. The article explains the signals and the program used to interpret them.

Requirements

Software:

- C compiler or Flowcode v3 or v4 PIC

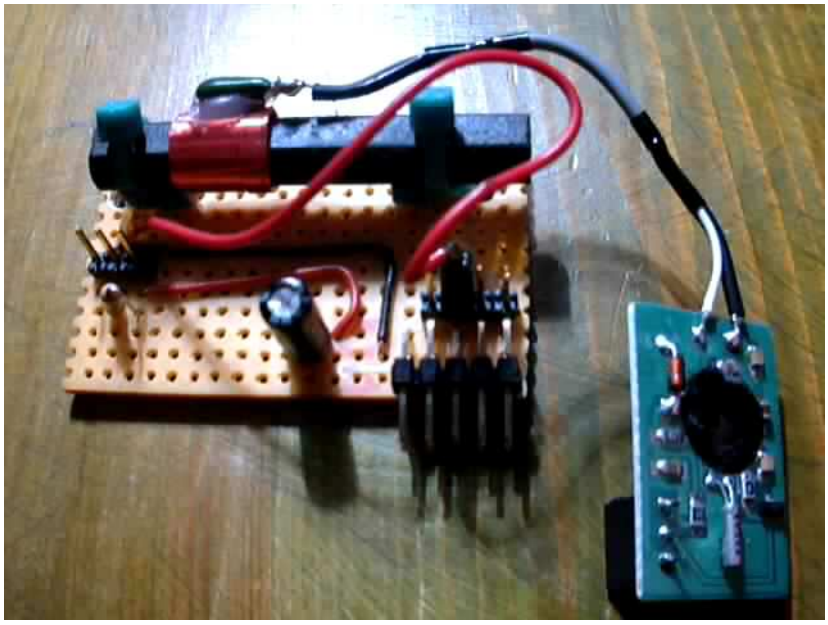
Hardware:

- DCF77 Receiver
- EB006 Multiprogrammer board
- EB008 Quad 7-Segment Display

Introduction

About a year ago I started a project to program a little radio clock (atomic clock) using a DCF77 receiver with a 16F877A PIC microcontroller and a 4Mhz oscillator. The finished project would show the hours and minutes on the quad 7-segment display. I programmed this in C using Matrix E-blocks, however there is also a Flowcode program that accompanies this article using an LCD display rather than the 7-segment display..

The Receiver



The DCF77 receiver receives an AM signal with a carrier frequency of about 77.5kHz. This signal comes from a German radio transmitter based in Mainflingen. This longwave transmitter pushes this signal out up to 2000km away from Frankfurt, allowing the signal to be picked up by the majority of Europe.

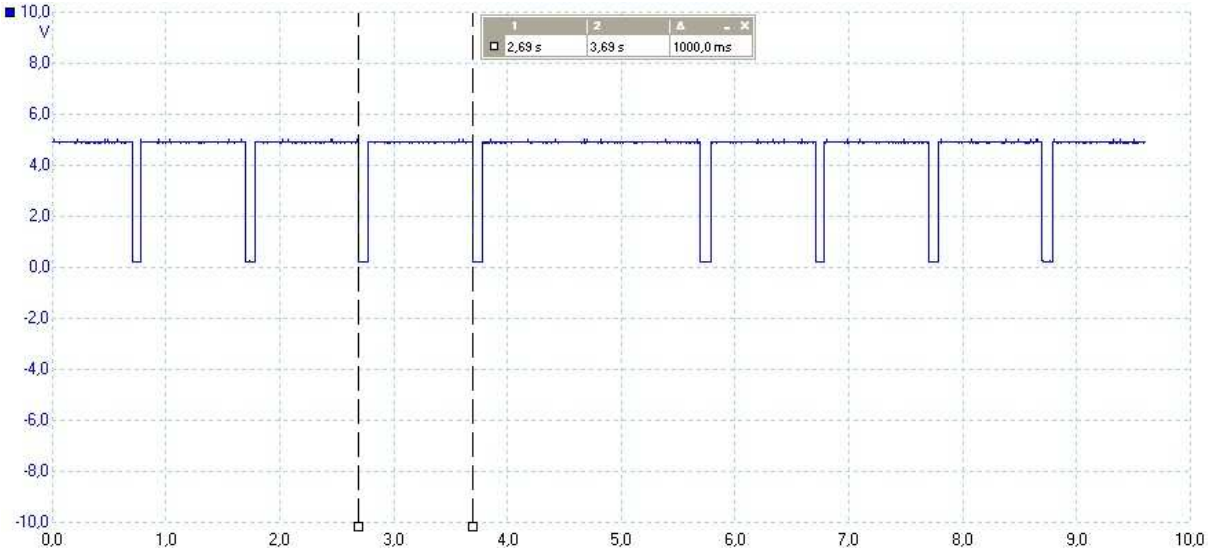
You can find out more about the transmitter and the DCF77 time signal by using internet search engines or encyclopedias.

The DCF77 output is a TTL signal (either 5V or 0V). *(Note: I needed to add a 10k pull-up resistor).*

Each second we will receive a pulse, except for one. These pulses are required for us to know the time and date. Each pulse of 200ms denotes a binary ‘1’ and each pulse of 100ms denotes a binary ‘0’,

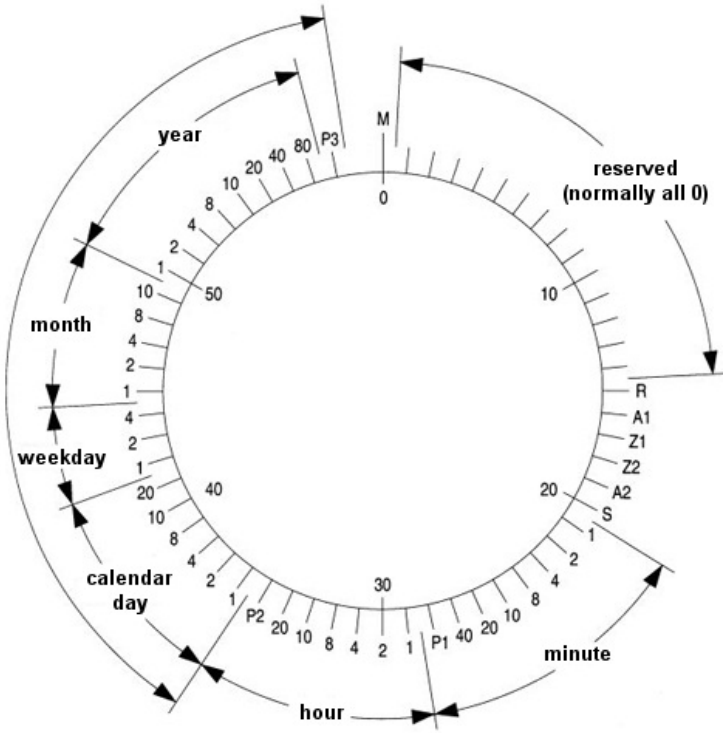
The 59th second does not have a pulse. It’s purpose is to allow synchronization with the next minute.

Here is a screen shot of the DCF77 receiver's output.



As you can see all pulses represent a '0' bit (100ms pulse) and you can also see the synchronisation pulse.

The following image will show you what those pulses represent.



In this program I have only decoded the hours and minutes from each pulse, however as you can see the full date is available to decode if you wished, and it would be a fairly simple task to modify the program to include the days, months, years.

One thing to point out though, the German radio station is GMT +1!

The Program

The program I originally created was done in C for the mikroC compiler, however a Flowcode program has been included, this uses an LCD display rather than a 7-seg display. I decided to use a PIC 16F877A because I'm used to testing programs with that microcontroller, but a simple PIC 16F84A or 16F88 (minimum PORTB and PORTA) would be enough to run this program.

Defines, Variables and Declarations:

```
#define SETHIGH(port, bit) (PORT##port |= (1<<bit))
#define SETLOW(port, bit) (PORT##port &= ~(1<<bit))
#define GETBIT(port, bit) ((PORT##port & (1<<bit)) && 1)
#define INVERTBIT(port, bit) (GETBIT(port, bit) ? SETLOW(port, bit) : SETHIGH(port, bit))
#define SETSAMEBIT(port1, port2, bit1, bit2) (GETBIT(port1, bit1) ? SETHIGH(port2, bit2) :
SETLOW(port2, bit2))

typedef char u8;

u8 display = 0;
u8 number2print[4] = {0};
u8 DCF_bits[60] = {0};
u8 lastsecond = 0;
u8 second = 0;
u8 counter = 0;

void InitPorts(void);
void InitInterrupt(void);

u8 SegmentMask(u8 number);

void DCF_PrintTo7Seg(void);
void DCF_ReceiveBits(void);
void DCF_DecodeBits(void);

void interrupt(void);
```

I made a couple defines that allow me to set a bit of a port high or low or return a bit of a port. Those defines are really, really, really usefull !!

Next we see a couple of global variables:

- *display* : Used to select one of the 7 segment displays. 0 = left display, 3 = right display.
- *number2print[4]* : The display buffer. ex.: number2print[0] => number to print on display 0.
- *DCF_bits[60]* : Array that will keep each received bit.
- l*astsecond* : Will be used as a boolean. If true, we'll write the display buffer on the displays, else we'll search for the 59st second again.
- *second* : Increments with each pulse.
- *counter* : Used to delay interrupt.

After the global variables, you'll see the different functions we'll need to use:

- *InitPorts* : To initialise 16F877A ports
- *InitInterrupt* : To initialise interrupt
- *SegmentMask* : A BCD to 7 segment decoder, for Common Cathode 7 segment displays
- *DCF_PrintTo7Seg* : Will print the displaybuffer
- *DCF_Receivebits* : Will receive the 58 bits and detect the 59st second
- *DCF_Decodebits* : Will decode the hours and minutes
- *interrupt*

Functions Explained

***InitPorts**:*
ADCON1 is an 8 bit register (see datasheet). It is used to "tell" the microcontroller if port A/E are/is used as an analog or digital input. I initialized it with 0x06 (hex value). This means I set port A and E as digital I/O. Next will be clearing and setting each port.

Pin RA4 will be used to receive the DCF signal.

***InitInterrupt**:*
The OPTION_REG is an 8 bit register (see datasheet). It contains various control bits to configure Timer0/WDT prescaler, timer TMR0, external interrupt and pull-ups on PORTB.
I initialised it with 0x8F. This means I'll use the TMR0 with a prescaler of 1:256 on the external oscillator. So, the interrupt will be executed every ((4MHz / 4) / 256)Hz.

The INTCON is also an 8 bit register (see datasheet). It contains various enable and flag bits for TMR0 register overflow, PORTB change and external INT pin interrupts. I initialized it with 0xA0. This means I enable and will use the TMR0 interrupt.

SegmentMask:
It is simply a BCD to 7 segment decoder. If the given number is higher than 9 it will write an E (Error) on one or more displays.

DCF_PrintTo7Seg:
As said before, it will print the hours and minutes on the display. Here I'm using the SETHIGH define to enable one of the four displays. If the *lastsecond* variable is equal 1, we'll print a number of the display buffer on one of the displays. Else we'll print the received bits on display 0 and the seconds on display 2 and 3. In addition to that, I use the dotpoint of the 3rd (right) display to show you the received DCF signal. If the variable display is 4 it will be reset.

DCF_ReceiveBits:
This is one of the most important functions of the program! It enters a loop and it will stay in this loop while count is lower than 48.The *count* variable is used to detect the 59st second.

In the ‘while’ loop there's an ‘if’ statement.

If the DCF signal is high it will increment *count* and it will pause the program for 20ms. The maximum time that the DCF input is high when receiving a bit is 900ms (when receiving a 0 bit). Thus, if we manage to exceed 900ms, this means we have detected the 59th second. (48 x 20ms) = 960ms.

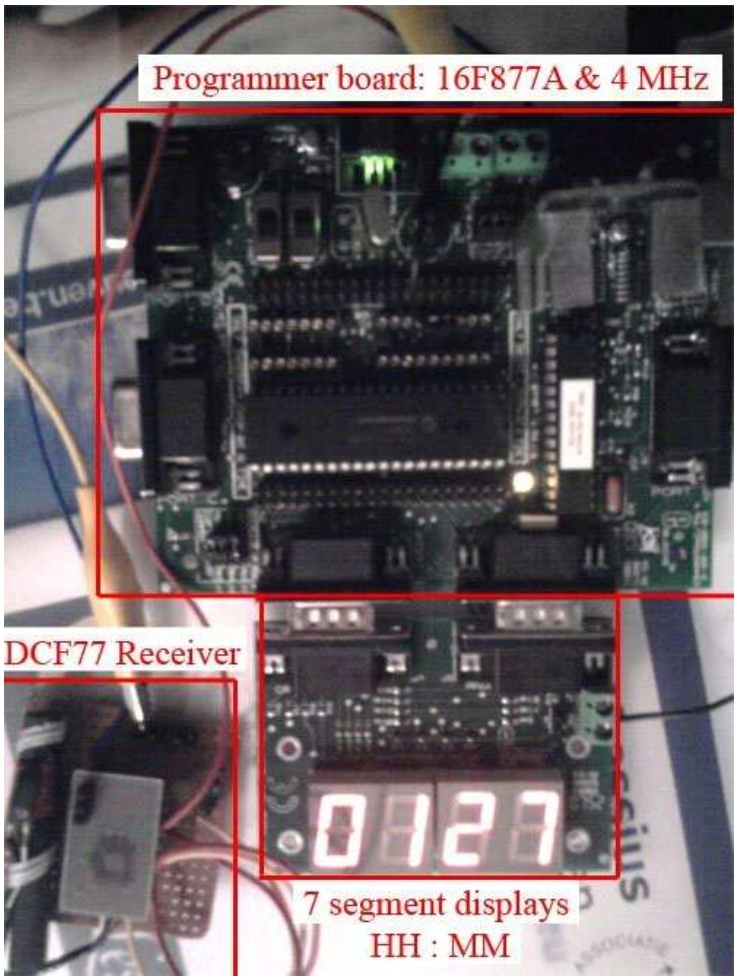
Else if the DCF signal is low, it'll pause the program for 120ms. If the signal is high again, this means we received a 0 bit. Else if the signal is still low, we received a 1 bit.

Here we need to reset the counter, because if we received a pulse, then this means this isn't the 59th second! It will then enter another loop and will stay there as long as the DCF signal is low. After that loop, it'll increment the variable *second*.

I also added a little if statement to avoid buffer overflow. After the (first) while loop we have another ‘if’ statement. This one is used to "tell" the microcontroller : "Ok, you can display the time" and of course, we reset the variable *second*.

DCF_DecodeBits:
This function is used to decode the received bits and write the right number in the right display buffer.

Interrupt:
To end the program, this function will print a digit on the display each time the counter variable equals 15. This is only used as delay.



Here is the finished project, the 16F887A chip mounted in an EB006 multiprogrammer, the EB008 segment display shows the time in an HH:MM format.

Voila,
This is what I did to receive, decode and display the time received from the German radio station!

I hope you have enjoyed this article and learned something new!

Nicolas L. F.

To see this clock in action please see the YouTube video linked in the "Further reading" section at the end of the article.

Image Sources:
Image 1 - From www.micro-examples.com
Image 2 - Screenshot from scope software
Image 3 - Wikipedia article on DFC77
Image 4 - Photo

Further reading

Below are some links to other resources and articles on related subjects, and technical documentation relating to the hardware used for this project…

YouTube Video:	http://www.youtube.com/watch?v=qNsx_SFmZ4o
Flowcode:	http://www.matrixmultimedia.com/flowcode.php
Eblocks:	http://www.matrixmultimedia.com/eblocks.php
Learning Centre:	http://www.matrixmultimedia.com/lc_index.php
User Forums:	http://www.matrixmultimedia.com/mmforums
Product Support:	http://www.matrixmultimedia.com/sup_menu.php