MAY & JUNE 2025 | ELEKTORMAGAZINE.COM

9.95 EURO
8.29 GBP, 10.75 USD

FREE thanks to MOUSER ELECTRONICS

elektor MEDIA & LEARNING DIGITAL BONUS EDITION

537B

SINCE 1961

FOCUS ON

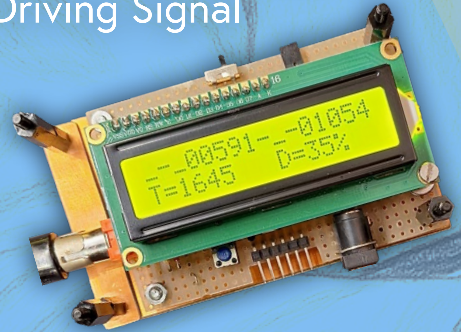
Test & Measurement

Oscilloscope/ Multimeter/ Generator

Fnirsi 2C53T with Two Channels and 50-MHz Bandwidth

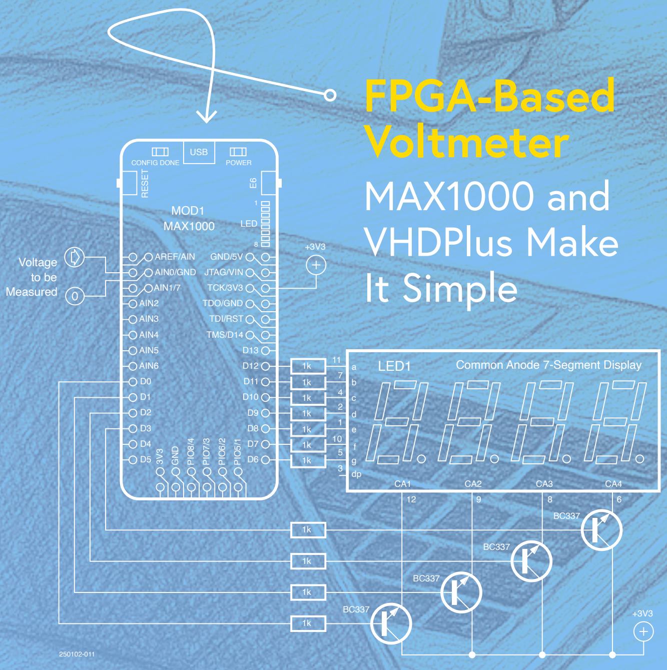
PWM Measurement with a PIC

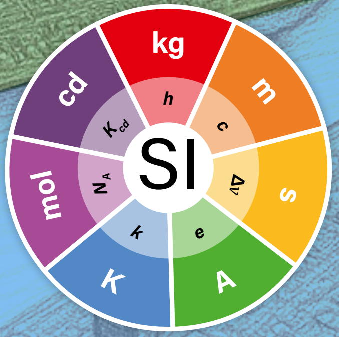
Get an Accurate On/Off Ratio Value from a Driving Signal



FPGA-Based Voltmeter

MAX1000 and VHDPlus Make It Simple





Infographics

Test & Measurement

CONTENTS

EDITORIAL

C. J. Abate
Content Director, Elektor

Test and Measure at Your Workbench

Test and measurement solutions are crucial for debugging, validating, and optimizing electronic designs. In this Bonus Edition, we highlight practical and accessible tools that engineers, students, and makers can bring straight to the workbench.

This issue features a smart approach to signal analysis with “PWM Measurement with a PIC.” Read the article to learn how to use a PIC16F628 to measure PWM pulse durations — capturing high/low times and full-period data with precision.

We also dive into the fascinating world of FPGA programming. In “FPGA-Based Voltmeter,” you’ll explore how to build a voltmeter using the VHDPlus environment and the MAX1000 FPGA board. It is a great project for anyone interested in bridging theory and hands-on digital design.

Looking to consolidate your tools? Take a look at the 2C53T. We offer insights about this compact 3-in-1 instrument that functions as an oscilloscope, multimeter, and function generator.

This Bonus Edition is all about empowering your electronics workflow. Whether you are fine-tuning signals, debugging circuits, or just exploring new test tools, we’ve got you covered.

Enjoy the read, and share your builds on the Elektor Labs platform!

The Team

International Editor-in-Chief: Jens Nickel | **Content Director:** C. J. Abate | **International Editorial Staff:** **Hans Adams,** Asma Adhimi, Roberto Armani, Eric Bogers, Jan Buiting, Rolf Gerstendorf (RG), Ton Giesberts, Saad Imtiaz, Alina Neacsu, Dr. Thomas Scherer, Jean-Francois Simon, Clemens Valens, Brian Tristam Williams | **Regular Contributors:** David Ashton, Stuart Cording, Tam Hanna, Ilse Joostens, Prof. Dr. Martin Ossmann, Alfred Rosenkränzer | **Graphic Design & Prepress:** Harmen Heida, Sylvia Sopamena, Patrick Wielders | **Publisher:** Erik Jansen | **Technical questions:** editor@elektor.com

COLOPHON

Elektor May/June 2025 Bonus Edition

Elektor Magazine is published 8 times a year by
Elektor International Media b.v.
PO Box 11, 6114 ZG Susteren, The Netherlands

Phone: +31 46 4389444

elektor.com | elektormagazine.com

For all your questions
service@elektor.com

Become a Member
elektormagazine.com/membership

Advertising & Sponsoring
Büsra Kas
Tel. +49 (0)241 95509178
busra.kas@elektor.com
elektormagazine.com/advertising

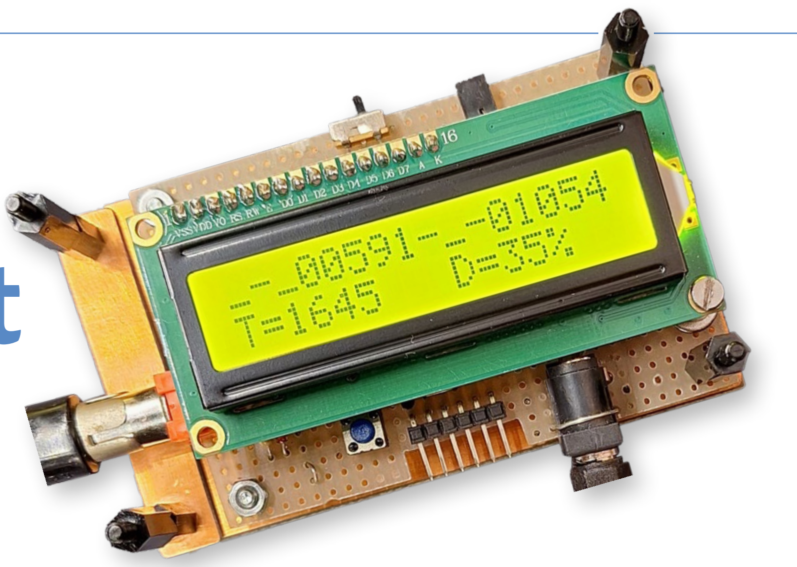
International Editor-in-Chief
Jens Nickel

Content Director
C. J. Abate

Publisher
Erik Jansen

PWM Measurement with a PIC

Get an Accurate ON/OFF Ratio Value from a Driving Signal



By Giovanni Carrera (Italy)

Although programming an Arduino is simpler than a PIC, the latter was more precise and accurate in calculating the timing of a PWM signal, due to its architecture. The challenges to be faced are certainly harder, but at last they will enrich the experience of those who enjoy coding. The proposed instrument can simultaneously measure high and low pulse durations and also measure the period, all with the resolution of one microsecond for pulse durations from 10 µs to 65,535 µs for PWM signals with frequencies from 7.63 Hz to 50 kHz.

The technique of varying the pulse duration over time is called PWM (Pulse Width Modulation), a modulation that originated in telecommunications but has also found use in the digital-analog world to vary the average voltage or current whose value depends on the ratio of the duration of the positive pulse to the entire period, a ratio that is called *duty cycle*.

PWM is used to regulate voltage in switching power supplies, for motor control, for servo actuators, and even for the output of some sensors. For these reasons, most modern microcontrollers have one or more PWM outputs. Some transducers, such as, for example, acoustic distance sensors, have a digital pulse output whose duration is proportional to the amplitude of the signal. The purpose of this project is not to generate PWM signals, but to measure their pulse duration with good accuracy.

In this article, I will describe a version that uses the PIC16F628 to measure the durations of the upper and the lower parts of the pulses, as well as the period, as visible in the introductory photo. If you also want to calculate the duty cycle as a percentage, you will need a PIC16F**648** because the project requires division. The program only needs a few additional instructions, but the code exceeds 2 K words, and the F628 has just 2 K of flash memory. Source programs and compiled HEX files are provided for both versions [1]. The rest of the hardware remains exactly the same.

The PIC Microcontroller

Why this choice? In my case, I had already done something like this several years ago, when I mainly used PICs in my projects, then switched to developing with Arduino IDE and MicroPython.

In PIC microcontrollers, the internal clock is one-fourth the frequency of the quartz oscillator, and all instructions are executed in one clock, except for program branches. With a 16 MHz crystal, the internal clock is 4 MHz, so the instruction execution time is only 0.25 µs.

The PIC16F628 is a Microchip microcontroller with RISC (Reduced Instruction Set Computer) architecture that works on 8-bit data, but its 35 instructions each occupy a 14-bit word; in our case the flash memory is 2K words. This PIC, among its many I/O functions, also has event capture on the CCP1 pin (RB3), at which it stores the contents of Timer1 in two registers, CCPR1L and CCPR1H. Something similar is also there for the Arduino UNO's ATmega328P, but the choice of a PIC leads to a much simpler, more compact system with much lower power consumption.

Of course, those used to working with Arduino will face greater difficulties in designing anything with PICs. It's not as easy to find pre-made boards or the many libraries available for Arduino to manage the devices used. In my opinion, working with Arduino only doesn't allow delving very deeply or even knowing exactly what one is doing at all.

Many Arduino users do nothing more than replicate project connections, now made with Fritzing figures, and with pictures of the components

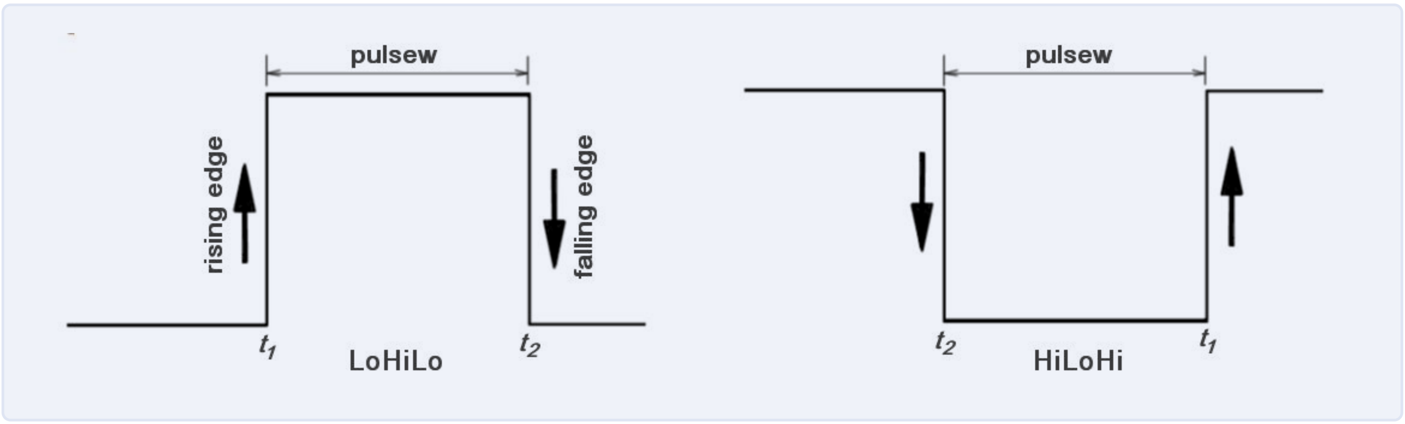


Figure 1: Two different capture modes are available, for the "LoHiLo" (left) and "HiLoHi" (right) types of pulses.

arranged on breadboards, and do not know how it all works (even though it often does not work at all because of uncertain or incorrect connections).

The compiler used for PICs is less user-friendly than the Arduino IDE and the libraries available are much scarcer, for example there is no `pulsein()` function to measure the duration of a pulse. For these reasons, the designer must go deeper into the operation of micro-controllers, as I also did to do this project, where I had to study the datasheet [2] to understand the operation of timers and CCP capture.

The mikroPascal compiler, which I used to develop the programs with the PICs, is free only for programs whose code is smaller than 2K words (1 word = 14 bits), which, in this case, was not exceeded also because this is the flash size of the PIC used while the RAM is only 224 bytes. With these limits, not much can be done: Using, for example, a float division easily exceeds the memory limit. But compilation is much faster than with Arduino, and certainly the code produced is also more compact and optimized.

The program was written in Pascal, using a powerful cross-compiler on Windows PCs: the mikroPascal PRO for PIC, version 7.6.0. A full version can be downloaded at the MIKROE website [3].

In this first version of the program, the code is shorter, so you can compile it without problems. Those more familiar with C or Basic can download these compilers and translate the program into the new language. If no changes are made to the program, you do not even need to download the compiler, but just need a PIC programmer using the already compiled hex file, *PWMmeter.hex*.

A further version of the program that calculates the duty cycle has also been developed, but, due to floating-point division, it requires more flash memory and a PIC16F648. In this case, if you have not purchased the compiler license, you must program the chip with the HEX file provided with this project.

Both of these microcontrollers are pin-to-pin compatible.

Measurement of Pulse Duration

Some microcontrollers with a built-in BASIC interpreter, such as Stamp, Picaxe or BasicX, use a special routine (`PULSIN`) to measure pulse duration, but cannot measure short pulses because of their relative slowness. Arduino boards use the `pulsein(`*pin*`,` *level*`,` *timeout*`)` function to measure the duration of a pulse, high or low level, with microsecond resolution. For longer times there is the `pulseInLong()` function, which uses the interrupt and measures times from 10 µs to 3 minutes. Both functions return an `unsigned long`.

The program uses two interrupts: Timer0 for sampling time while the second interrupt source is generated on CCP (capture/compare/PWM) capture events on the rising and falling edge of the pulse and the respective times are counted in Timer1.

Timing With Timer0

Timer0 generates an interrupt that is used to sample measurements and display them on the LCD every 50 events, corresponding to about 0.5 seconds. Timer0 is an 8-bit TMR0 counter, preceded by an 8-bit programmable divider called a prescaler.

The program selects the internal clock, which has a frequency equal to one quarter of the crystal frequency: 16 MHz /4 = 4 MHz. If we set the prescaler with the highest value (1:256), Timer0 will have as input a frequency of f1 = 4 MHz / 256 = 15.625 kHz, with a period of 64 µs; loading 100 into the counter will result in an interrupt every 156 (256-100) pulses of frequency f1, corresponding to a period of 156 × 64 = 9984 µs (about 100 Hz).

The Timer0 interrupt is generated when the timer/counter in the TMR0 register goes into overflow from 0xFF to 0x00, which sets the T0IF bit. This interrupt generates the clock that is used to sample the measure-ments and present them on the LCD, every 50 events, corresponding to about 0.5 s.

Event Capture With Timer1

The program, after the rising edge on pin RB3, changes the capture mode for the falling edge, and vice versa. The pulses can be low → high → low level, which we will call LoHiLo, or, inverted, HiLoHi, as shown in **Figure 1**.

For LoHiLo type pulses the program sets the `CCP1CON := $05` register with capture on rising edge. When this event occurs the interrupt routine saves the contents of the `CCPR1L` and `CCPR1H` regis-ter pair which will then be transferred to t1. Then the `CCP1CON := $04`, setting the capture on falling edge, at whose event the register pair `CCPR1L` and `CCPR1H` will be read again, which will then be trans-ferred to t2, as seen in Figure 1. The `pulsew` time is derived from the t2-t1 difference.

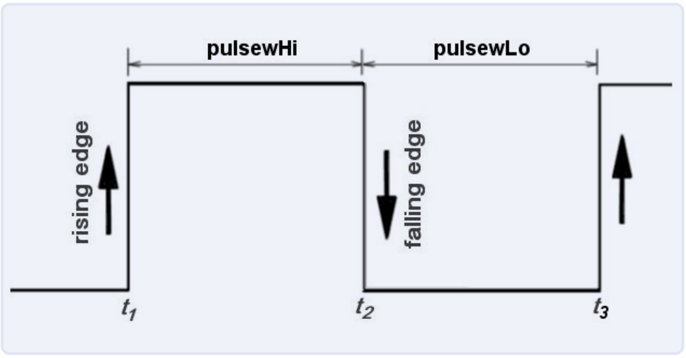


Figure 2: Acquisition of t_1, t_2, and t_3 timings to perform the needed calculations.

For HiLoHi-type pulses, things are similar, but you start with the falling edge (t2) and end with the rising edge (t1), in which case the `pulsew` time = t1–t2. Timer1 has a 16-bit counter, preceded by a 2-bit prescaler.

The internal clock of 4 MHz is also used here. With the prescaler = 1:1 there is a resolution is 0.25 µs, which is excessive if we consider a normal crystal, so the 1:4 ratio with a resolution of 1 µs was used. Therefore, the system measures pulses of duration between about 10 µs up to 65,535 µs. Below 10 µs, it does not work, because there are several instructions in the interrupt routine, and this requires some execution time. If you want to measure longer times, with a resolution of 2 µs, you have to use the 1:8 ratio. In this case, the times should be multiplied by two after converting the variables to long integers.

The bespoke new, bigger version of the original program can measure the two durations at the same time; therefore, you can calculate the PWM period and also the duty cycle. In this case, the program selects the capture for the rising edge, stores the time in t1, raises the `firstcre` flag to distinguish this edge from the last one, then sets for the falling edge and puts the time in t2, and sets for the next rising edge which it will put in t3, as seen in **Figure 2**. At the third edge, the interrupt routine sets the `datok` flag to indicate that times t1, t2 and t3 have been measured.

Therefore, it calculates:

pulsewHi = t2–t1, pulsewLo = t3–t2, period = pulsewHi + pulsewLo

Duty cycle requires operations with `float` variables (`real` in Pascal) that exceed the flash memory limits of the 16F628, but we can do them with a calculator:

Period = T = pulsewHi + pulsewLo
Duty cycle: D = pulsewHi / T × 100 [%]

Comparison With Arduino pulseIn()

For comparison, just for a LoHiLo pulse, I also wrote a small program for the Arduino Uno:

```
// program pulseintest
#define pulsein 4

void setup() {
  Serial.begin(115200);
  pinMode(pulsein, INPUT);
}
```

```
void loop() {
 unsigned long duration = pulseIn(pulsein, HIGH);
  Serial.print("Pulse High [us] = ");
  Serial.println(duration);
  delay(500);
}
```

To check the times, I used my own crystal-based device with programmable divider and square wave output as a reference; in **Table 1** are the results compared with that of my PIC system.

Table 1: Measurements.

ref. [µs]	PIC [µs]	Arduino [µs]	Err.% PIC	Err.% Arduino
5		5…6		0…20
50	50	50…51	0	0…2
500	500	494…500	0	−0.8…0
5,000	5,000	4,966	0	−0.68
50,000	49,997	49,662	−0.006	−0.676

As can be seen, the PIC system is very accurate.

Component List

Resistors
R1 = 10 kΩ, ±5%, ¼ W
R2 = 100 Ω, ±5%, ¼ W
R3 = 100 Ω, ±5%, ½ W
Rp1 = 22 kΩ, trimmer

Capacitors (ceramic)
C1, C2 = 22 pF, 50 V
C3, C4, C5, C6 = 100 nF, 25 V

Semiconductors
U1 = PIC16F628A or PIC16F648A microcontroller
U2 = LM7805 5 V regulator
D1, D2 = 1N4148 diode

Miscellaneous
Display = LCD 2×16
X1 = 16 MHz crystal
SW1 = push button (RESET)

The Circuit Schematic

Figure 3 shows the schematic system's schematic. Using an LM7805 (a 78L05 will also do), the system should be powered as an Arduino UNO, that is, with a voltage from 7 V (optimum value) to 12 V, but it consumes, without backlight (W2 off), less than 10 mA, of which a current of about 3 mA is drawn by the regulator. A much simpler system is to use three common 1.5 V batteries. In this case, what might be more critical is the LCD, which has a rated supply voltage of 5 V, but similar devices usually run on 4.5 V as well.

As a display, I used a common two-line 16-character LCD with an HD44780-compatible parallel interface. Jumper W1 is used to avoid power supply conflicts and must be removed if you program the chip

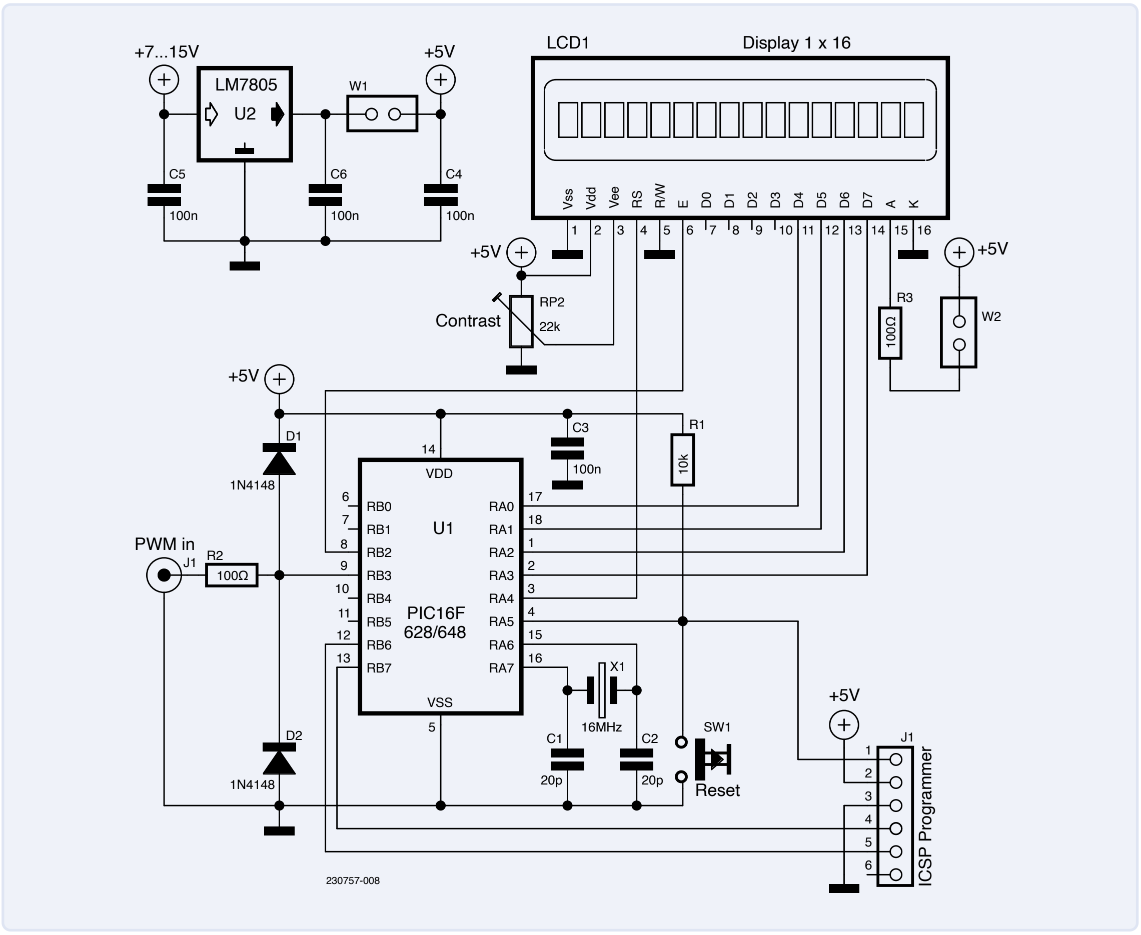


Figure 3: The project's schematic diagram.

with the PICKIT. Jumper (or switch) W2 is used to turn off the backlight and save power.

The Prototype

Figure 4 shows the arrangement of components on the perforated matrix board used to make the prototype. The LCD has been removed to show the components mounted underneath.

On the top, a small switch (W2) is seen — it turns off the display backlighting, while on the left, an RCA connector used for PWM input is mounted, and, below, present is a 6-pin connector for chip programming, which is compatible with the PICkit in-circuit programmer. If you have a PIC programmer with ZIF sockets, this connector is not required, but you need to remove the PIC from its socket to program it. Next to it, still at the bottom, you can see the power connector. The board size can be further reduced, as it was recycled from a previous project. The 5-pins female strip connector on the right is not in use.

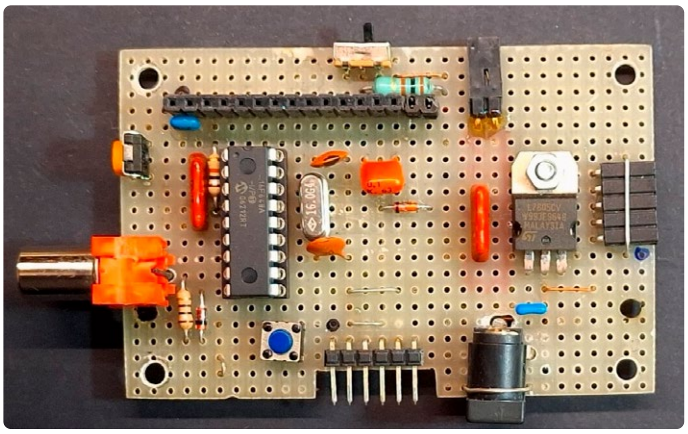


Figure 4: The finished prototype, made on a breadboard salvaged from a previous project. The 5-pin female strip connector on the right is not in use.

The Program

Most of the program has been described above. The outputs on the LCD are more basic than Arduino's *LiquidCrystal* library functions, and the messages must be strings or arrays of characters, so it is necessary to convert the variables to strings and remove any leading spaces.

The most important work is done by the interrupt routine. In the first part, it checks whether the interrupt is caused by Timer0 overflow, in which case `INTCON`, `T0IF` = 1, raises the `int0flg` flag, which is used for timing. It then reloads the number 100 into the counter so that it triggers an interrupt after 156 pulses and resets the interrupt `INTCON` and `T0IF` flags to 0.

The second part, which is more complex, is to handle the capture of the rising and falling edges and save the contents of the two capture registers, `CCPR1L` and `CCPR1H`, to variables `t1`, `t2` and `t3`, corresponding to the first rising edge, the falling edge, and the second rising edge. When all these three times have been captured, the `datok` flag is set, which will be used by the program to calculate the durations and period.

The times, expressed in µs, relating to the PWM's upper part, indicated with `'_-_'` and the lower part, indicated with `'-_-'`, are printed on the first line. The PWM period, i.e. the sum of the two times, is printed in the second line.

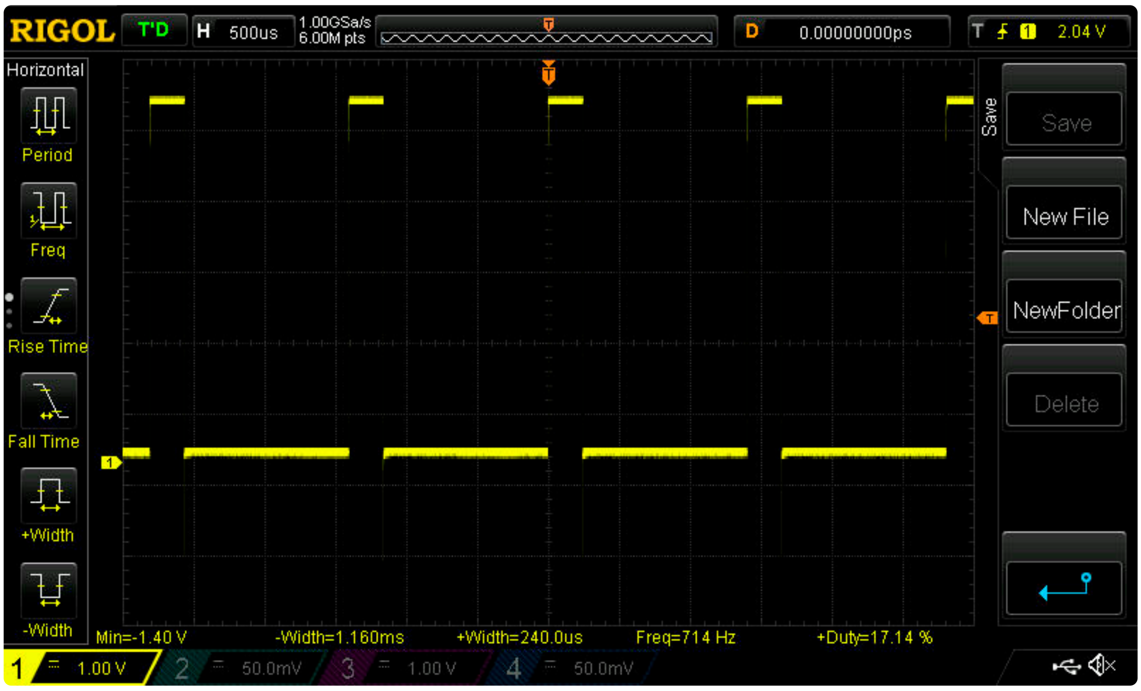


Figure 5: Scope screenshot of the measurement shown in Figure 6.

In the absence of a signal, the display does not show anything. You can see the complete code in **Listing 1**. This program occupies 1,236 words of flash memory (65%) and 74 bytes of RAM (37%), and compilation was done in 187 ms.

Program Version With Duty Cycle Print Option

In this case, you need a PIC16F648A, which is similar to the F628, but has 4K words of flash memory. As already mentioned, to compile this version you need to purchase the program license, which lifts the 2K-word limit. But, the already compiled hex file for the PIC16F648 is included in the package for this article and can be uploaded to the PIC straight away.

The program prints `pulsewHi` [μs], `pulsewLo` [μs], `period` [μs] and `dutyc` [%] on the display.

Unfortunately, the floating-point division gave me memory problems in the compilation, the reason being that even the PIC16F648 is not very suitable. I overcame the inconvenience by converting the percent duty cycle to an integer, losing the decimal part.

Comparative Tests

Figures 5 and **6** show a comparison between measurements taken with an oscilloscope and those of the PIC16F648 system. In this case, a TTL pulse generator was used.

As you can see, the measurements are very similar.

PIC16F648 PWMmeter Program Listing

The program is largely similar to the previous one, with the exception of the `LCDprint` routine.

Instructions for calculating and printing the duty cycle have been added. The modified routine is shown in **Listing 2**.
The modified code occupies 2,266 words of flash memory (55%) and 92 bytes of RAM (62%), and compilation was done in 47 ms.

230757-01

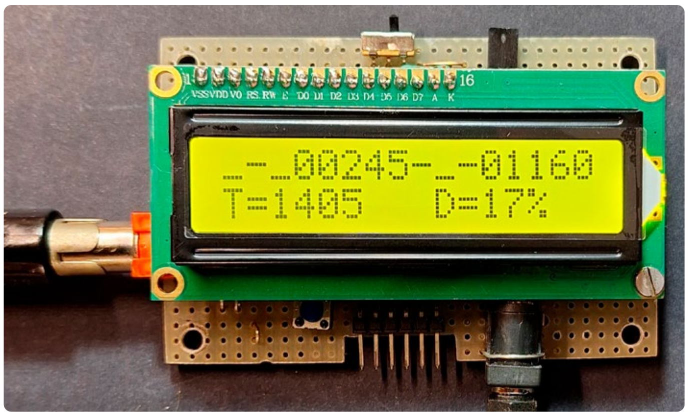


Figure 6: Readouts of the "High" pulse duration (245 μs), "Low" pulse (1,160 μs), total period T (1,405 μs), and the duty cycle (17%).

Listing 2: PIC16F648 PWMmeter (section).

```
procedure LCDprint; // print measures on LCD
begin
  Lcd_Cmd(_LCD_CLEAR); // lcd clear
  WordToStrWithZeros(pulsewHi, texdH);// microseconds
  ltrim(texdH); // trims the leading spaces
  WordToStrWithZeros(pulsewLo, texdL);
  ltrim(texdL); // trims the leading spaces
  Lcd_Out(1, 1,'_-_' + texdH + '-_-' + texdL);
  LongWordToStr(period, texPer);
  ltrim(texPer); // trims the leading spaces
  //FloatToStr_FixLen(dutyc, texd, 5);
  Lcd_Out(2, 1,'T=' + texPer);
  dutyc:= real(pulsewHi)/real(period)*100.0;
  IntToStr(integer(dutyc), texd);
  ltrim(texd);
  Lcd_Out(2, 10,'D=' + texd + '%');
  LCDout:= false;
end;
```

About the Author

Giovanni Carrera holds a degree in Electronic Engineering. As a university professor in the Faculty of Naval Engineering in Genoa, Italy, he taught numerous courses, such as naval automation and the simulation of ship propulsion systems. Carrera started working in the late 1970s with the 6502 CPU and then moved on to other processors. Today, he enjoys the designing and developing analog and digital electronic circuits, many of which he has written about on his blogs (ArduPicLab and GnssRtkLab) and in various magazines.

Questions or Comments?

Do you have technical questions or comments about this article? Please contact the Elektor editorial staff at editor@elektor.com.

Related Products

- **Tam Hanna, *Microcontroller Basics with PIC*, Elektor, 2020**
  www.elektor.com/19188
- **Microchip MPLAB PICkit 5 In-Circuit Debugger/ Programmer**
  www.elektor.com/20665

WEB LINKS

[1] Software download for this article: https://www.elektormagazine.com/labs/elektor-articles-software-downloads
[2] PIC16F627A/628A/648A Data Sheet: https://ww1.microchip.com/downloads/en/DeviceDoc/40044G.pdf
[3] MIKROE website: https://mikroe.com/mikropascal-pic

Listing 1: PIC16F628 PWMmeter.

```
program PWMmeter;
// PIC16F628 measures PWM (resolution= 1 usec)
// display the value on LCD every 0.5 sec
// timer#0 for interrupt every 9.984 msec (approx 100Hz)
// the LCD display has  2 rows x 16
// by G. Carrera 23/12/2023
// I/O configuration:
// PortB.3 = pulse input   (in)
// PortA.0..3,= LCD data (4 bit mode) (out)
// PortB.2 = LCD E (out)
// PortA.4 = LCD RS (out)
// xtal = 16.00 MHz
var
  t1,t2,t3: word;
  int0flg,datok,LCDout,firstcre: boolean;
  i,t1l,t1h,t2l,t2h,t3l,t3h: byte;
  pulsewLo,pulsewHi: word;
  texdL: string[5];
  texdH: string[5];
  period: longint;
  texPer: string[10];
  dutyc: real;
  texd: string[6];
```

(Continues on the next page)

```
// Lcd module connections
var LCD_RS : sbit at RA4_bit;
var LCD_EN : sbit at RB2_bit;
var LCD_D4 : sbit at RA0_bit;
var LCD_D5 : sbit at RA1_bit;
var LCD_D6 : sbit at RA2_bit;
var LCD_D7 : sbit at RA3_bit;
var LCD_RS_Direction : sbit at TRISA4_bit;
var LCD_EN_Direction : sbit at TRISB2_bit;
var LCD_D4_Direction : sbit at TRISA0_bit;
var LCD_D5_Direction : sbit at TRISA1_bit;
var LCD_D6_Direction : sbit at TRISA2_bit;
var LCD_D7_Direction : sbit at TRISA3_bit;
// End Lcd module connections
procedure interrupt;
// read capture times between rising and falling edge or vice versa
begin
  if TestBit(INTCON, T0IF) then  // timer#0 interrupt
    begin
      ClearBit(INTCON, T0IF); // reset Timer#0 interrupt flag
      TMR0:= 100;
      SetBit(INTCON, T0IE); // Enables Timer#0 interrupt
      int0flg:= true;
    end;
  if TestBit(PIR1, CCP1IF) then // capture interrupt
    begin
      if  TestBit(CCP1CON, CCP1M0)  then // rising edge ($05)
        begin
          if firstcre then  // already captured the first rising edge t1
            begin
              t3l := CCPR1L;  //  load final time to t3
              t3h := CCPR1H;
              firstcre:= false;
              datok:= true;
            end
          else
            begin
              t1l := CCPR1L;  //  load initial time to t1
              t1h := CCPR1H;
              firstcre:= true; // the first rising edge is captured now
            end;
          CCP1CON := $04; // change to falling edge on CCP1
          ClearBit(PIR1, CCP1IF); //Clear CCP1 Int. flag (also due to changing)
        end
      else  // falling edge
        begin
          t2l := CCPR1L;  //  load intermediate time to t2
          t2h := CCPR1H;
          CCP1CON := $05; // change to rising edge on CCP1
          ClearBit(PIR1, CCP1IF); //Clear CCP1 Int. flag (also due to changing)
        end;
    end;
end;
procedure IOinit; // initialize I/O
begin
  TRISB:= %00111011;  // PORTB bit2 : output
  TRISA:= %10100000;  // PORTA bits 0..4 are outputs
  CMCON:= $07; // comparators off (RA0..RA4 usable as digital IO)
  Lcd_Init(); // Initialize LCD
  Lcd_Cmd(_LCD_CURSOR_OFF); // Turn off cursor
```

(Continues on the next page)

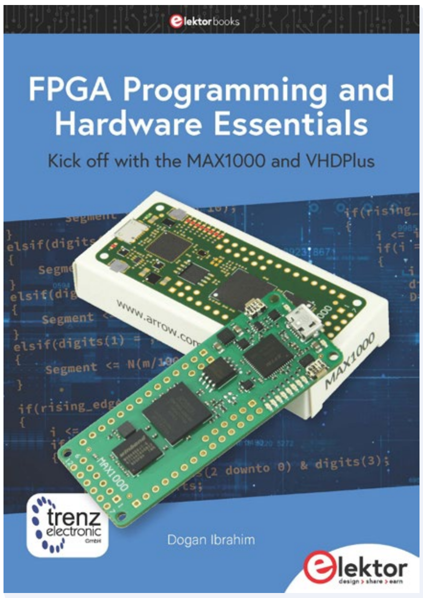
```
  CCP1CON := $05;  //  rising edge on CCP1
  T1CON := $21;   // 1:4 prescaler, int. clk, enable t1
  OPTION_REG:= $07; // set prescaler of Timer#0 to 256
  ClearBit(OPTION_REG,7); //  enable port B pull-ups
  SetBit(PIE1,CCP1IE); // Enables the CCP1 interrupt
  SetBit(INTCON, PEIE); // Enables peripheral interrupt
  TMR0:= 100; // T0 overflow every 156 counts
  SetBit(INTCON, T0IE); // Enables Timer#0 interrupt
  ClearBit(INTCON, T0IF); // reset Timer#0 interrupt flag
  SetBit(INTCON, GIE); // Enables global interrupt
  int0flg:= false;
  Lcd_Out(1, 1, 'PWMmeter-231223'); // Print text to LCD (row,column,text)
  Delay_ms(2000);
  Lcd_Cmd(_LCD_CLEAR); // lcd clear
  i:=0;
  firstcre:= false;
end;
procedure LCDprint; // print measures on LCD
begin
  Lcd_Cmd(_LCD_CLEAR); // lcd clear
  WordToStrWithZeros(pulsewHi, texdH);// microseconds
  ltrim(texdH); // trims the leading spaces
  WordToStrWithZeros(pulsewLo, texdL);
  ltrim(texdL); // trims the leading spaces
  Lcd_Out(1, 1,'_-_' + texdH + '-_-' + texdL);
  LongWordToStr(period, texPer);
  ltrim(texPer); // trims the leading spaces
  //FloatToStr_FixLen(dutyc, texd, 5);
  Lcd_Out(2, 1,'T=' + texPer);
  LCDout:= false;
end;
begin  // main program
  IOinit; // Initialize
  LCDout:= false;
  while true Do //  endless loop
    begin
      if int0flg then
        begin
          int0flg:= false;
          i:=i+1;
          if i= 50 then // about 0.5 seconds
            begin
              LCDout:= true;
              i:= 0;
            end;
        end;
      if datok then
        begin
          datok:= false;
          t1:= (t1h shl 8)+t1l;
          t2:= (t2h shl 8)+t2l;
          t3:= (t3h shl 8)+t3l;
          pulsewHi:= t2-t1;
          pulsewLo:= t3-t2;
          period:= longint(pulsewHi) + longint(pulsewLo);
          if LCDout then LCDprint;
        end;
    end;
end
```

FPGA-Based Voltmeter

MAX1000 and VHDPlus Make It Simple

By Dogan Ibrahim (United Kingdom)

Are you ready to master FPGA programming? Elektor's *FPGA Programming and Hardware Essentials* is the perfect way to dive into the world of field-programmable gate arrays. As a quick-start example, let's look at a voltmeter designed using the powerful VHDPlus programming environment and implemented on the MAX1000 FPGA.


elektor books
FPGA Programming and Hardware Essentials
Kick off with the MAX1000 and VHDPlus
trenz electronic
Dogan Ibrahim
elektor

Editor's Note. *This article is an excerpt from the Elektor book,* FPGA Programming and Hardware Essentials*. It was formatted and lightly edited to match* ElektorMag*'s conventions and page layout. The author and editor are happy to help with queries. Contact details are in the* **Questions or Comments?** *box.*

The FPGA is a single, configurable integrated circuit that can be programmed to handle a variety of tasks normally associated with individual chips. If you thought that home programming of an FPGA is difficult and in the realms of professionals, read on!

Meet the MAX1000 FPGA

The FPGA used in the book is essentially the MAX1000 dev board from Arrow Electronics (**Figure 1**). It is designed to help you get started using an FPGA device. Although the board is described in other chapters in the book, to kick off this article, its basic features should be mentioned right here:

- Intel MAX10 device
- Arrow USB Programmer 2
- 64 Mbit SDRAM
- 64 Mbit Flash memory
- 1× 12 MHz MEMS oscillator
- 1× optional MEMS oscillator of preferred frequency
- 8× red user LED
- 2× board indicator LED
- 2× user button

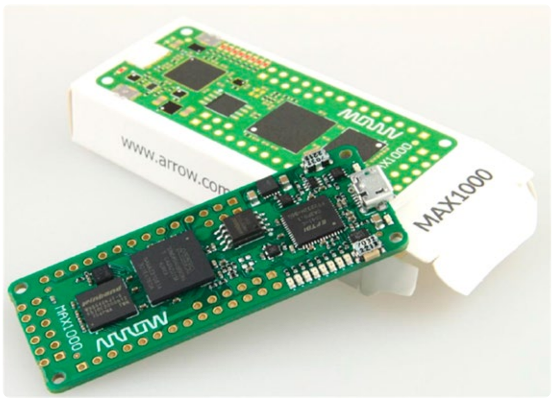


Figure 1: The MAX1000 FPGA.

- 1× 3-axis accelerometer
- 1× 12-pin Pmod header (just the holes)
- 1× user JTAG header (just the holes)
- 1× user I/O header (just the holes)
- Mini USB type-B connector

And for Software: VHDPlus IDE

Several software packages can be used to develop FPGA applications on the MAX1000 FPGA development board. In the book, you get to know both the basic features and the installation of the VHDPlus IDE for programming your MAX1000.

VHDPlus IDE is a superset of VHDL, which makes programming easier by extending its features and simplifying its syntax. VHDPlus is not a completely different language, but it extends the features of VHDL. So, everything you could do with VHDL is also possible with VHDPlus.

VHDPlus is a modern approach that makes FPGA programming faster and more friendly, especially for beginners. VHDPlus IDE supports the open CRUVI Standard. Simulation of a designed FPGA program can take a long time. The Simulation Assistant of VHDPlus IDE helps to simulate your programs and fix any errors quickly. VHDPlus IDE integrates key features from Quartus and is available on both Windows and Linux platforms. VHDPlus IDE also supports C++ support including a debugger.

Installing the VHDPlus IDE

The steps to download the VHDPlus IDE on your Windows computer are:

- Go to the VHDPlus website [1].
- Click on step 1 to download the *MAX 10 device support* file (**Figure 2**). The file will be downloaded to your *Downloads* folder. Save it in a folder.
- Scroll down to step 2, and download the **Quartus Prime Lite for Windows**. The file will be downloaded to your *Downloads* folder.
- Click on the file to install the Quartus Lite program.
- When you start the Quartus Lite program, you might get the following error: *you successfully installed the Quartus Prime software but did not install any devices. Do you want to launch the device installer to add devices?* Install the device support file .qdz as described below.
- Go to the Windows *Start* menu and find the program called *Device Installer* (Quartus Prime 18.1).

Updates Received from the Author
Since publishing the book and in response to reader feedback, we received two updates from the author mainly addressing the type of FPGA used.

1. Please refer to page 34, second bullet point:
A window will open (Figure 3.6) where you can select the development board you are using.
- This window is titled: *Connect and Compile.*
- On the top left side, you will see the text: "FPGA:" followed by a pull-down list.
- Select *MAX1000* from the list for the 8 kLE variant (the default option).
- Select *MAX1000 16k* from the list for the 16 kLE variant.

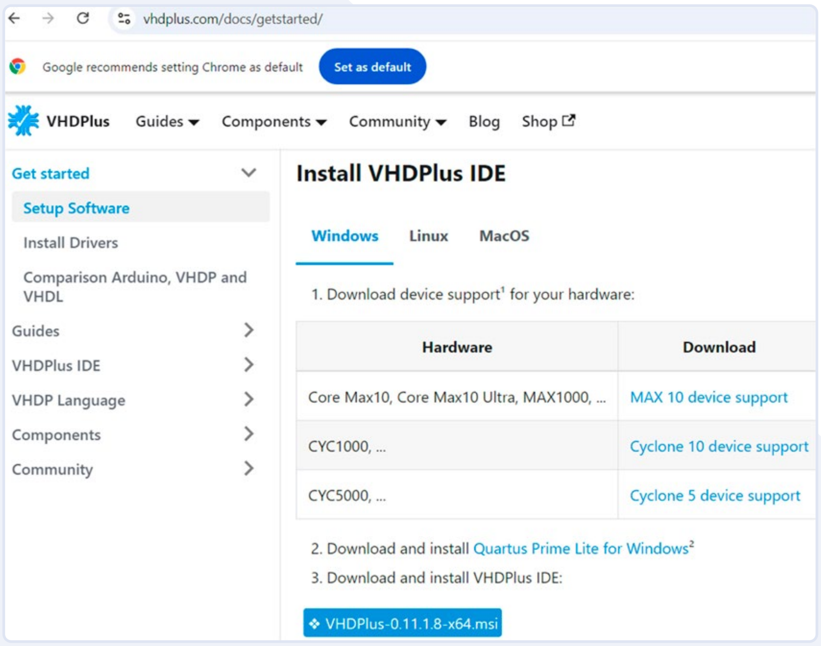
Important: Selecting the wrong variant will result in a JTAG error when attempting to download the compiled code to the FPGA.

2. The 8 kLE version is identified by the marking *10M08* on the large chip on the board, while the 16 kLE version is marked *10M16*.

- The *Device Installer* will prompt you for the folder where you saved the .qdz file. Select that folder.
- The program will search for all .qdz files. Choose the device to install. The .qdz file is just an installer package until you install it.
- Click step 3 to download VHDPlus. The file will be downloaded to your *Downloads* folder.
- Click the file to install the VHDPlus.

Figure 2: Download the MAX 10 device support file.

▼


vhdplus.com/docs/getstarted/

Google recommends setting Chrome as default | Set as default

VHDPlus | Guides | Components | Community | Blog | Shop

Get started
- Setup Software
- Install Drivers
- Comparison Arduino, VHDP and VHDL

Guides
VHDPlus IDE
VHDP Language
Components
Community

Install VHDPlus IDE

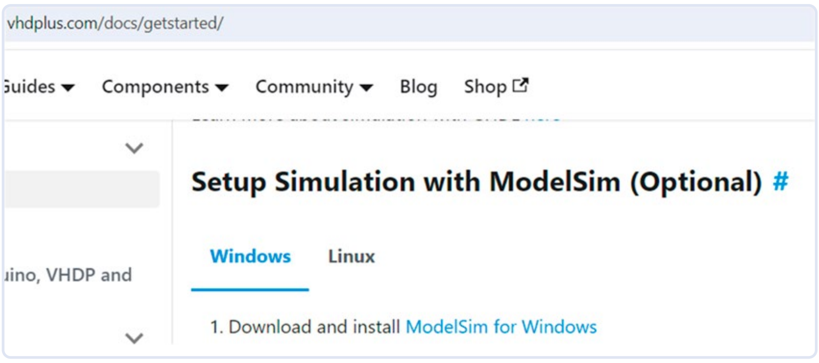
Windows | Linux | MacOS

1. Download device support¹ for your hardware:

Hardware	Download
Core Max10, Core Max10 Ultra, MAX1000, ...	MAX 10 device support
CYC1000, ...	Cyclone 10 device support
CYC5000, ...	Cyclone 5 device support

2. Download and install Quartus Prime Lite for Windows²
3. Download and install VHDPlus IDE:

VHDPlus-0.11.1.8-x64.msi


vhdplus.com/docs/getstarted/

Setup Simulation with ModelSim (Optional) #

1. Download and install ModelSim for Windows

Figure 3: Download the ModelSim simulator.

You need to install the drivers for your programmer to program your FPGA. Follow these steps:

- Download the *Arrow USB Programmer* driver for your operating system from [2].
- Unzip the downloaded file and run the installer to complete the installation.

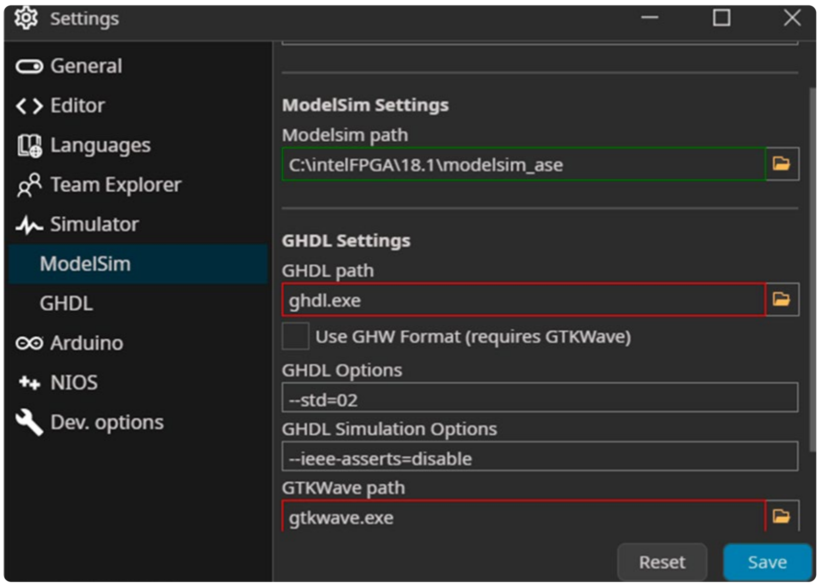
For direct compiling and programming with VHDPlus IDE, a connection to Quartus is needed. If Quartus is installed in the default directory, no further steps are required. Otherwise, adjust the Quartus path in VHDPlus IDE by going to *Extras Settings General*. Once Quartus is detected, the boundary will turn green.

Setting Up ModelSim Simulation

To install the ModelSim simulator, follow these steps:

- Visit the VHDPlus website [1].
- Scroll down and click to download *ModelSim for Windows* (**Figure 3**). The file will be saved in your *Downloads* folder.
- Install *ModelSim* by double-clicking the downloaded file.
- In VHDPlus IDE, specify the path to the *modelsim_ase* folder under *Extras Settings Simulator* (**Figure 4**).
- Run the simulation by right-clicking on a VHDL file.

Figure 4: Specify the simulator path in VHDPlus IDE.



ModelSim Settings
Modelsim path
C:\intelFPGA\18.1\modelsim_ase

GHDL Settings
GHDL path
ghdl.exe
Use GHW Format (requires GTKWave)
GHDL Options
--std=02
GHDL Simulation Options
--ieee-asserts=disable
GTKWave path
gtkwave.exe

Analog-to-Digital Converter (ADC)

ADCs are important devices in most microcontroller-based applications. This is because most physical sensors give analog output voltages. For example, an analog temperature sensor may give an output voltage that's directly proportional to the sensed temperature. So let's look at the MAX1000's FPGA analog inputs and develop a project using these inputs.

MAX1000 FPGA has nine analog inputs named AIN and A0 through A7. Each analog input is 12-bits wide, corresponding to 4096 steps. With the default +3.3 V reference voltage, the resolution of an ADC input is 3.3 V / 4096 = 0.8 mV. Therefore, analog input voltages can be sensed with 0.8 mV resolution. For example, if the input voltage is 0.8 mV, the output of the ADC will be binary *0000 00000001*. If the analog input voltage is 0.16 mV, the ADC output will be *0000 00000010*. Similarly, if the analog input voltage is 2.4 mV, the ADC output will be *0000 00000011* and so on.

The analog inputs are in the form of channels. **Table 1** shows the channel numbers of each analog input.

Project Voltmeter

Hardware

As a very first project, let's build an utterly simple voltmeter circuit that measures the applied voltage at one of the ADC inputs and displays it on a 7-segment

Table 1: MAX1000 ADC channels.

Analog pin	Channel
AIN0	1
AIN1	6
AIN2	5
AIN3	7
AIN4	3
AIN5	0
AIN6	2
AIN7	4
AIN	8

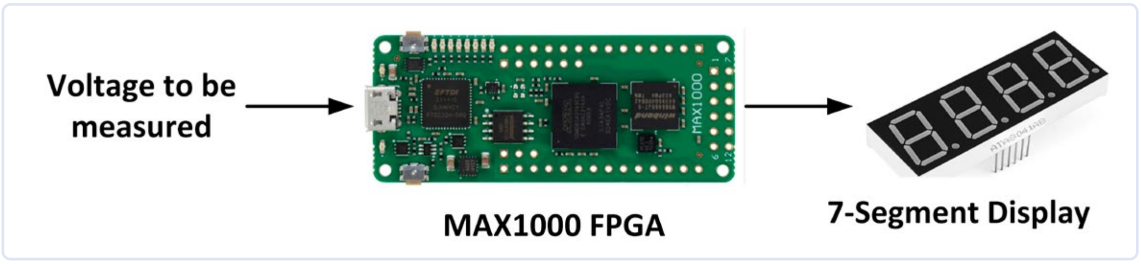


Figure 5: Block diagram of the Voltmeter project.

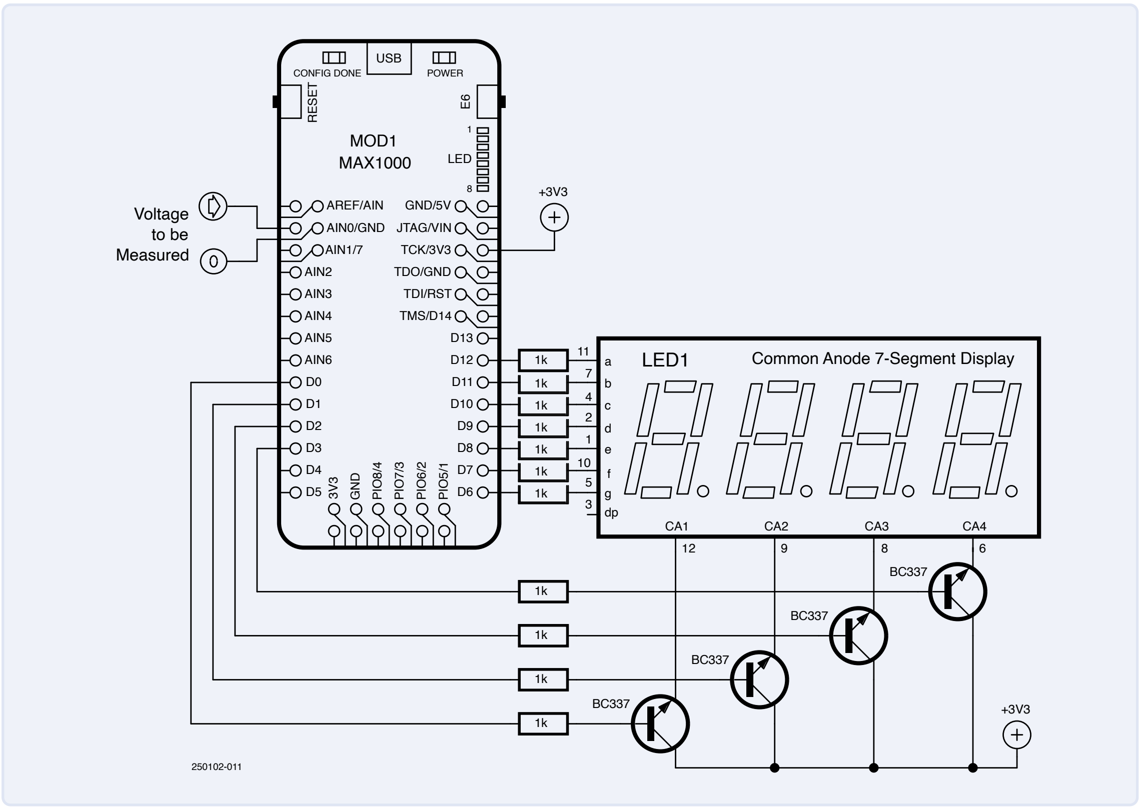


Figure 6: Circuit diagram (schematic) of the FPGA-based Voltmeter project.

display in millivolts. The maximum input voltage is +3.3 V (3300 mV). **Figure 5** shows the project block diagram pictorially. The circuit diagram is shown in **Figure 6**, where the voltage to be measured is applied to ADC input AIN0 (channel 1).

Software

We have to use the *ADC* library in order to use the ADC inputs. The steps are as follows:

- Open the IDE and give a name to your program (e.g., *Voltmeter*).
- Click on *Library Explorer* in the bottom left corner of the screen
- Select the following (**Figure 7**): *Intel_IP Interface MAX10_ADC*
- Right-click on *MAX10_ADC* and then select *Add to active project*. The ADC library has now been

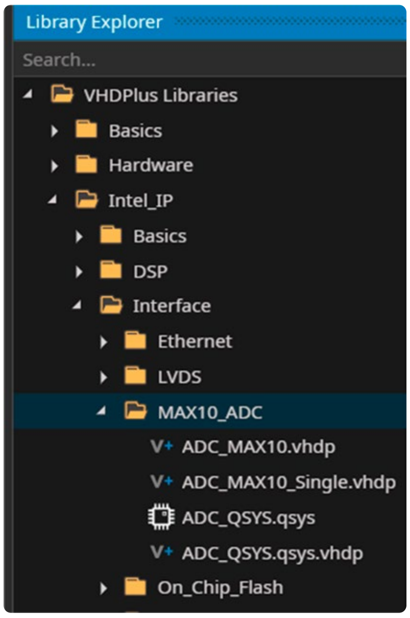
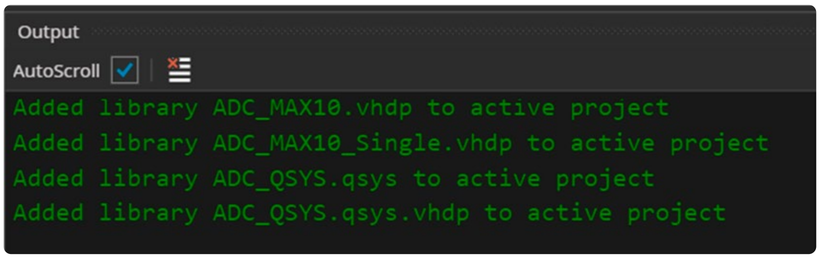


Figure 7: Add the ADC library to your project.



```
Output
AutoScroll
Added library ADC_MAX10.vhdp to active project
Added library ADC_MAX10_Single.vhdp to active project
Added library ADC_QSYS.qsys to active project
Added library ADC_QSYS.qsys.vhdp to active project
```

Figure 8: Confirmation that the ADC library has been added to the project.

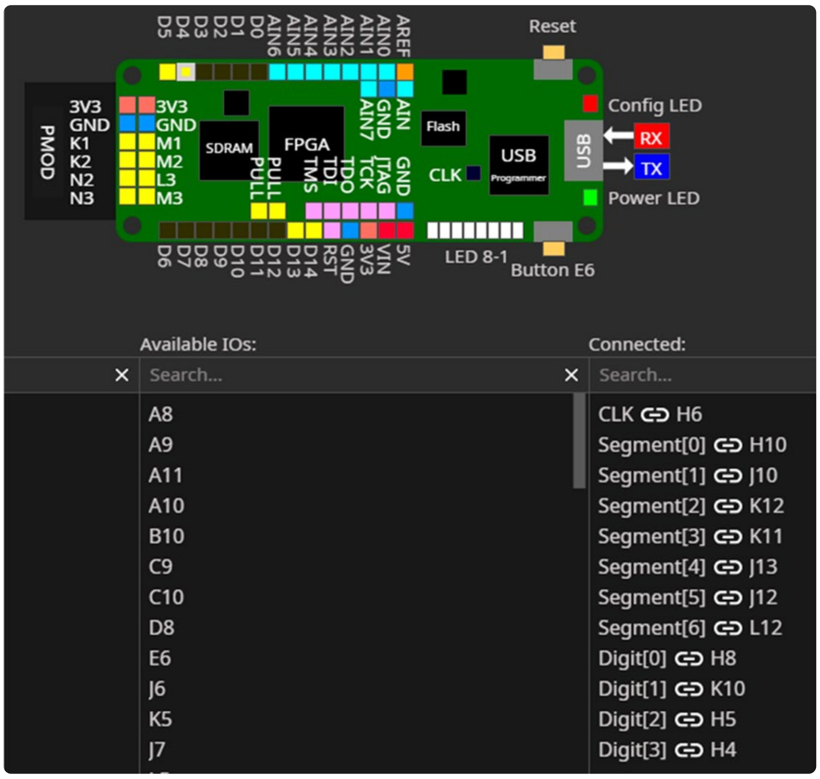
added to our project. You should see a message at the bottom of the screen saying that the *ADC_MAX10 ADC* library has been added to the active project (**Figure 8**).

- You can now start writing your code using the *ADC_MAX10_Single* Component from the *ADC* library.

Figure 9 shows the connection diagram, which is basically the 7-segment display connection discussed in more detail in another chapter in the book.

The program listing (Program: *Voltmeter*) is shown in **Listing 1**. The 7-segment LED part of the program is the same as we have seen in Chapter 5. Additionally, we have defined signals `temp`, `ADCdata`, and `ADCchannels`. Signal `temp` runs from 0 to 3300 which is the maximum voltage output from the ADC. `ADCdata` is the raw ADC data which can take a value between 0 and 4095 where 4095 corresponds to binary *1111 11111111*, `ADCchannels` is the channel number and it can take values 0 to 8.

A new component with the name `ADC_MAX10_Single` is defined in the program which is used to access the single ADC module in the ADC library. This component has two variables in the library: channel number (`Channel`) and the channel data (`Data`).



```
Available IOs:      Connected:
A8                  CLK ⇔ H6
A9                  Segment[0] ⇔ H10
A11                 Segment[1] ⇔ J10
A10                 Segment[2] ⇔ K12
B10                 Segment[3] ⇔ K11
C9                  Segment[4] ⇔ J13
C10                 Segment[5] ⇔ J12
D8                  Segment[6] ⇔ L12
E6                  Digit[0] ⇔ H8
J6                  Digit[1] ⇔ K10
K5                  Digit[2] ⇔ H5
J7                  Digit[3] ⇔ H4
```

Figure 9: FPGA connection diagram.

Listing 1: The VHDPlus code that goes into making the Voltmeter run on an FPGA.

```
Main
(
   Segment: OUT STD_LOGIC_VECTOR(6 downto 0);
   Digit: OUT STD_LOGIC_VECTOR(3 downto 0);
)

{
     TYPE MyArray is an array (0 to 9) of STD_LOGIC_VECTOR(1 to 7);
     SIGNAL N: MyArray;
     SIGNAL m: integer range 0 to 9999 := 0;
     SIGNAL temp: integer range 0 to 3300 := 0;
     SIGNAL digits: STD_LOGIC_VECTOR(3 downto 0) := "0001";
     SIGNAL i: integer range 0 to 36000 := 0;
```

```
    SIGNAL ADCdata: natural range 0 to 4095 := 0;
    SIGNAL ADCchannels: natural range 0 to 8 := 0;

    NewComponent ADC_MAX10_Single
    (
       Channel => ADCchannels,
       Data => ADCdata
    );

    N(0) <= "1000000";
    N(1) <= "1111001";
    N(2) <= "0100100";
    N(3) <= "0110000";
    N(4) <= "0011001";
    N(5) <= "0010010";
    N(6) <= "0000010";
    N(7) <= "1111000";
    N(8) <= "0000000";
    N(9) <= "0010000";

 Process()
 {
     if(digits(0) = '1')
     {
        Segment <= N(m mod 10);
     }
     elsif(digits(3) = '1')
     {
        Segment <= N(m/10 mod 10);
     }
     elsif(digits(2) = '1')
     {
        Segment <= N(m/100 mod 10);
     }
     elsif(digits(1) = '1')
     {
        Segment <= N(m/1000 mod 10);
     }
     if(rising_edge(CLK))
     {
      i <= i + 1;
      if(i = 36000)
      {
           i <= 0;
           digits <= digits(2 downto 0) & digits(3);
           Digit <= digits;
      }
      ADCchannels <= 1;            -- Read channel 1 (AIN0)
      temp <= ADCdata;             -- As raw data
      m <= temp * 3300 / 4095;     -- in millivolts
     }
 }
}
```

Inside the rising edge of the clock, channel 1 is selected which corresponds to ADC input AIN0. Raw ADC data is coped to signal `temp`. This value is then multiplied by 3300 and divided by 4095 so that the raw reading is converted into physical millivolts. Copying this value to `m` displays the ADC reading, i.e., the voltmeter reading on the 7-segment display.

Let's Test It!

Finally, **Figure 10** shows the project as it was built by the author for the book publication. Though limited in capability, this little voltmeter is a great introduction into the fascinating world of FPGA programming with low-cost and easily accessible hardware and software. What's your next project?

250102-01

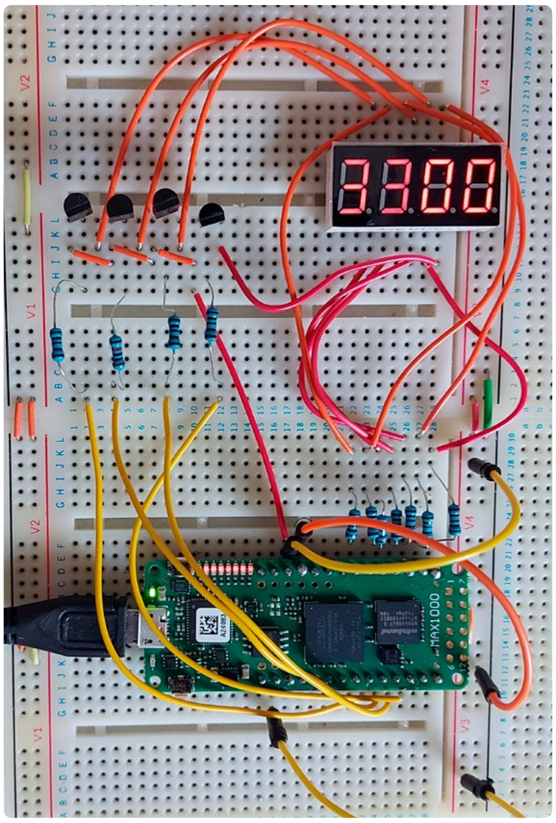


Figure 10: The Voltmeter project constructed on a breadboard.

Questions or Comments?

Do you have any questions or comments related to this article? Email the author at d.ibrahim@btinternet.com or Elektor at editor@elektor.com.

About the Author

Prof. Dr. Dogan Ibrahim has a BSc degree in electronic engineering, an MSc degree in automatic control engineering, and a PhD degree in digital signal processing. Dogan has worked in many industrial organizations before he returned to academic life. Prof. Ibrahim is the author of over 70 technical books and has published over 200 technical articles on microcontrollers, microprocessors, and related fields. He is a Chartered electrical engineer and a Fellow of the Institution of the Engineering Technology. He is a certified Arduino professional.

WEB LINKS

[1] VHDPlus website: https://vhdplus.com/docs/getstarted/#install-vhdplus-ide
[2] Arrow USB Programmer driver, Trenz Electronic: https://tinyurl.com/Arrow-USB-Programmer

Elektor Webinars
An Elektor Webinar in Cooperation with DownStream Technologies
DownStream
elektor e-zine
Your dose of electronics
Available Now: Elektor Mag, Guest-Edited by Espressif
Out Now!
We are delighted to welcome Espressif as Elektor Mag's 2023 guest editor. The issue is available now!
more...
Download white paper
elektor MAG
design › share › earn
50+ Circuits & Projects
132 Pages!
Circuit Special 2023
Design Ideas, Tips and Tricks

Elektor infographic

Test and Measurement

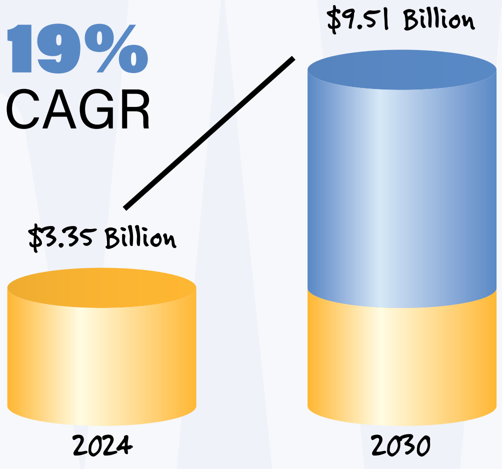
EV Battery Testing: The Key Growth Driver

In the electric vehicle (EV) test equipment market, there is a rising demand for specialized testing tools, such as battery testers and thermal management systems, that assess critical aspects of EVs [1]. The Battery Electric Vehicle (BEV) segment dominates, driving growth in EV test equipment due to its unique testing needs. With around 40 million EVs on the road in 2023 [2], BEVs require extensive battery, drivetrain, and charging interface tests to ensure performance, efficiency, and safety. Advancements in battery technologies, including solid-state and high-capacity lithium-ion chemistries, require specialized protocols to validate their efficiency.

The performance testing segment is expected to grow the fastest by 2033. Companies such as TÜV SÜD are leading this sector with expertise in lithium-ion batteries, offering services such as cyclic and calendar aging tests, rapid charging profiles, and electrochemical impedance spectroscopy to ensure optimal battery performance [3].

Key Players
- TÜV SÜD
- Bureau Veritas
- SGS Societe Generale De Surveillance SA
- TÜV Rheinland AG Group
- Dekra
- Applus+

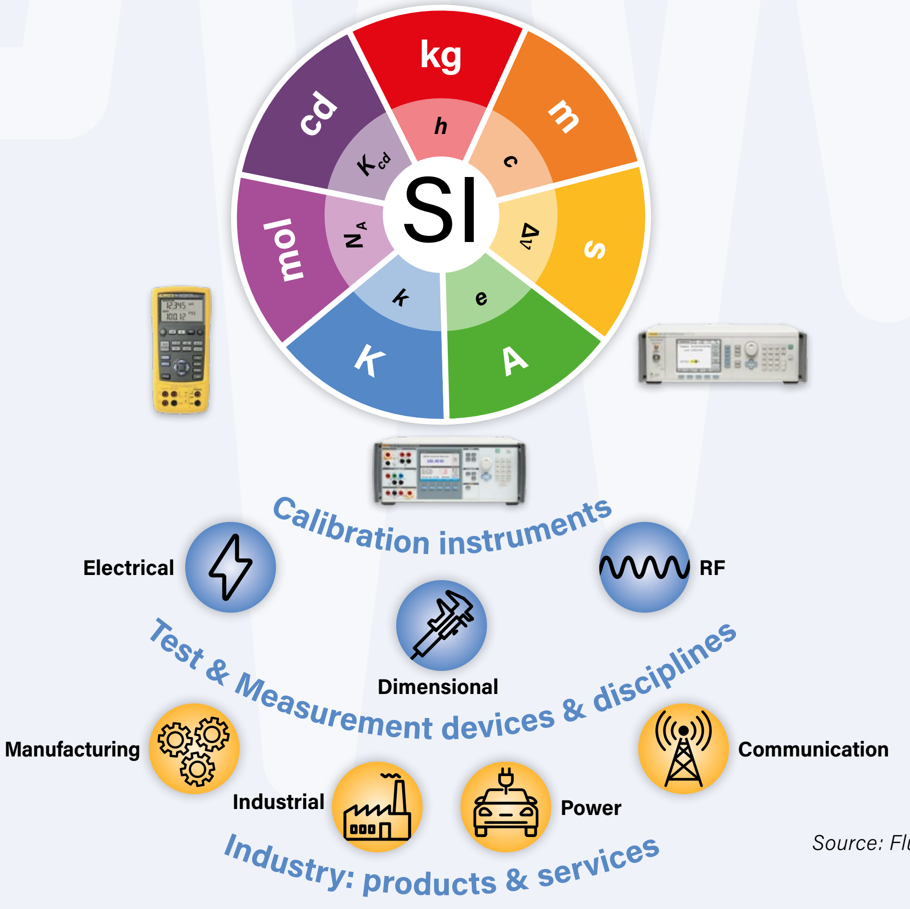
EV Battery Testing Market



Source: Research and Markets [3]

The Calibration Universe

As precision manufacturing tightens its quality standards, the calibration services market is on the rise, and it is expected to grow at a CAGR (Compound annual growth rate) of 4.5% during 2025 to 2033 [4]. Metrology ensures accuracy in both pre- and post-production, maintaining compliance with tight tolerances. However, evolving calibration standards add complexity, challenging service providers to stay ahead [5]. At the same time, the rise of self-calibrating devices and software solutions could shake up traditional services, pushing providers to adapt.



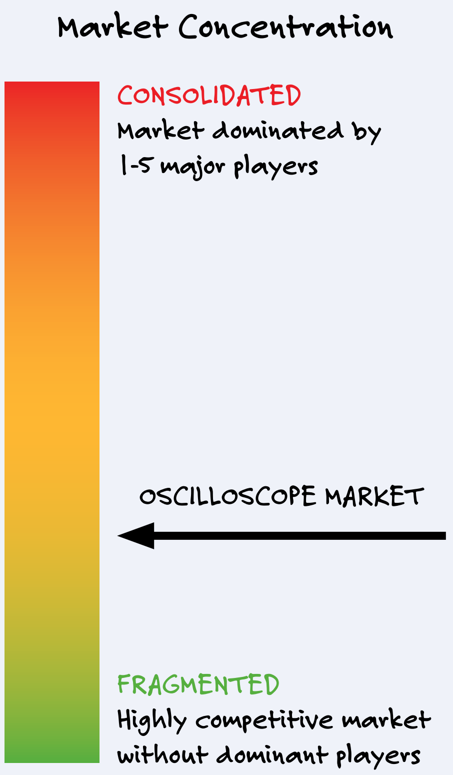
Source: Fluke [6]

Oscilloscopes Stay in the Lead

The General Purpose Test Equipment (GPTE) market is experiencing steady growth, driven by increasing demand for precision measurement tools across various industries. According to Technavio [7], the GPTE market is expected to expand by $2.06 billion from 2023 to 2028. Holding the largest market share within GPTE, the oscilloscope market alone is projected to grow from $3.74 billion in 2025 to $5.49 billion by 2030, at a CAGR of 7.99% [8].

Oscilloscope Market Leaders

- Tektronix
- Keysight Technologies
- Rohde & Schwarz
- Teledyne LeCroy
- Yokogawa Test & Measurement Corporation



Source: Mordor Intelligence [8]

250108-01

WEB LINKS

[1] Towards Automotive, “Electric Vehicle Test Equipment Market Size, Trends and Developments,” September 2024: https://tinyurl.com/EV-market
[2] IEA, “Trends in Electric Cars,” 2024: https://www.iea.org/reports/global-ev-outlook-2024/trends-in-electric-cars
[3] Research and Markets, “EV Battery Testing Market,” December 2024: https://www.researchandmarkets.com/reports/6036229
[4] IMARC Group, “Calibration Services Market,” 2024: https://www.imarcgroup.com/calibration-services-market
[5] MarketsandMarkets, “Calibration Services Market,” 2023: https://tinyurl.com/mam-calibration-market
[6] Fluke, “Why is Calibration Important?”: https://www.fluke.com/en-us/learn/blog/calibration/why-is-calibration-important
[7] Technavio, “General Purpose Test Equipment (GPTE) Market,” Aug 2024: https://tinyurl.com/technavio-GPTE-market
[8] Mordor Intelligence, “Oscilloscope Market,” 2024: https://www.mordorintelligence.com/industry-reports/oscilloscope-market

Oscilloscope/ Multimeter/ Generator

Fnirsi 2C53T with Two Channels and 50-MHz Bandwidth

By Harry Baggen (Netherlands)

There seems to be no end to the constant flow of new products released by the Chinese manufacturer Fnirsi. This time we will look at the 2C53T, a compact 3-in-1 measuring instrument that combines an oscilloscope, a multimeter, and a function generator. It looks exactly like the 2C23T, which I examined about a year ago. So, what's new?

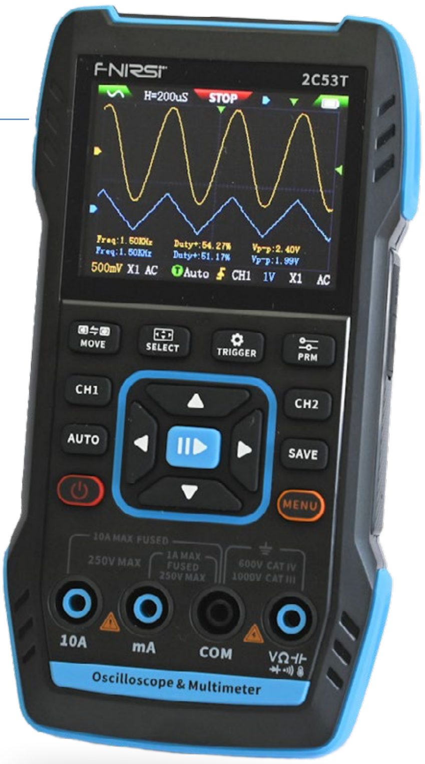


Figure 1: The Fnirsi 2C53T is compact and replaces three separate measuring devices.

The Fnirsi 2C53T [1] is a compact 3-in-1 measuring instrument that combines an oscilloscope, a multimeter, and a function generator (**Figure 1**). There seems to be no end to the constant flow of new products released by the Chinese manufacturer Fnirsi. Sometimes these are improved versions of previously released devices, and that is the case with the 2C53T reviewed here. On the outside, the device looks exactly like the 2C23T, which I examined about a year ago [2].

Refreshing Your Memory

As I already noted, the case remains exactly the same, with only slight changes to the labeling on some of the push buttons. For those unfamiliar with its predecessor, here are the main physical characteristics of both the 2C23T and the Fnirsi 2C53T. The unit has a fairly small case measuring 17 × 9 × 3.5 cm, featuring a 2.8-inch color LCD that provides a reasonably bright image with good contrast. The case has blue rubber-like material on the corners and feels quite sturdy. Power is supplied by a built-in 3 Ah lithium battery, which lasts for about six hours of use.

On the front, there are four banana sockets for the multimeter, and at the top of the device, there are three BNC sockets: two for the (two-channel) oscilloscope and one for the function generator. The device is operated via 15 pushbuttons. A USB-C connector on the side allows for charging the internal lithium battery and connecting to a PC for firmware updates and screenshot downloads.

Many Accessories Included

The 2C53T now comes in a handy storage case that also has space for the included set of multimeter test leads, two 100-MHz oscilloscope probes (the 2C23T only came with one), a BNC cable with crocodile clips for the generator, a USB-C cable, and a small manual (**Figure 2**). As with most Fnirsi devices, everything is neatly finished and well-packaged. Even the included storage case is sturdy and practically designed.

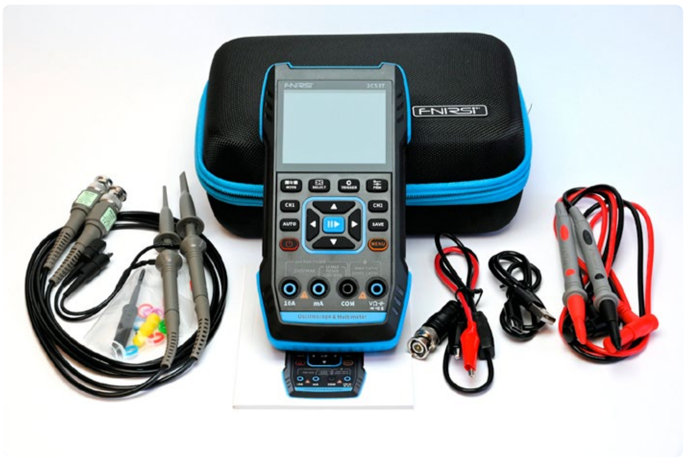


Figure 2: Together with the 2C53T, you get two oscilloscope probes and a handy storage case.

Differences Between the Fnirsi 2C53T and 2C23T

With the 2C53T [1], the specifications and capabilities of the oscilloscope function have been significantly improved and expanded. The input bandwidth has increased from 10 MHz to 50 MHz, and the sampling rate has been boosted from 50 to 250 Msamples/s (**Figure 3**). That's exceptional for a measuring device in this price range! Additionally, the oscilloscope now includes an X-Y mode and offers eight mathematical functions to choose from. There's also an FFT analysis, though it's not particularly useful for a device of this type.

A so-called *Persistence* function has been added, allowing signals to "glow" for a certain period, similar to how old oscilloscopes with cathode ray tubes displayed waveforms. The number of measurement values that can be displayed on-screen has expanded to 14 per channel. Finally, all oscilloscope settings have been consolidated into a dedicated menu, making navigation much clearer.

Multimeter Updates

Not much has changed with the multimeter. The resolution has increased from 4 digits (9999) to 4½ digits (19999), and the display has been redesigned with a sleeker, clearer layout. Now, there is only one analog scale (**Figure 4**). The specifications and capabilities remain the same: a basic accuracy of 0.5%, the ability to measure DC and AC voltage and current, as well as resistance, capacitance, and temperature. A continuity tester and diode tester are also included.

As with most Fnirsi multimeters, the device automatically detects whether DC voltage, AC voltage, or resistance is applied to the inputs, but you can also manually switch to a specific function.

Changes to the Function Generator

Some modifications have been made to the built-in function generator. The number of available waveforms has expanded from six to 13. The output signal now has a maximum amplitude of 3 V peak-to-peak (previously 3.3 V). However, the downside is that the frequency range has been reduced to 50 kHz (compared to 1 MHz on the 2C23T). Well, you have to make trade-offs somewhere.

Getting Started with the Fnirsi 2C53T

If you've ever worked with a 2C23T, you'll feel right at home with the 2C53T. Aside from a few minor details, the operation remains the same. When you power on the device, a menu appears with various icons allowing you to select a measuring instrument or access the settings menu. It's also possible to configure the 2C53T to start directly with a specific measuring instrument.

The oscilloscope is the most important instrument in this device, primarily due to its large bandwidth and high sampling rate. Unfortunately, the screen has not been enlarged, so it still feels quite crowded, especially when multiple measurement readings are displayed. The oscilloscope responds quickly to signal changes, and operation has been somewhat improved with the addition of a dedicated settings menu. Adjusting the time base, sensitivity, trigger level, and signal position on the screen is now clearer: pressing the *Select* key allows you to choose which of these parameters can be adjusted using the cursor pad keys. The selected parameters are indicated at the top of the screen.

It's difficult to determine whether the Auto-Setup function has been updated, but I did get the impression that it works slightly faster than on its predecessor. The BNC connectors at the top are

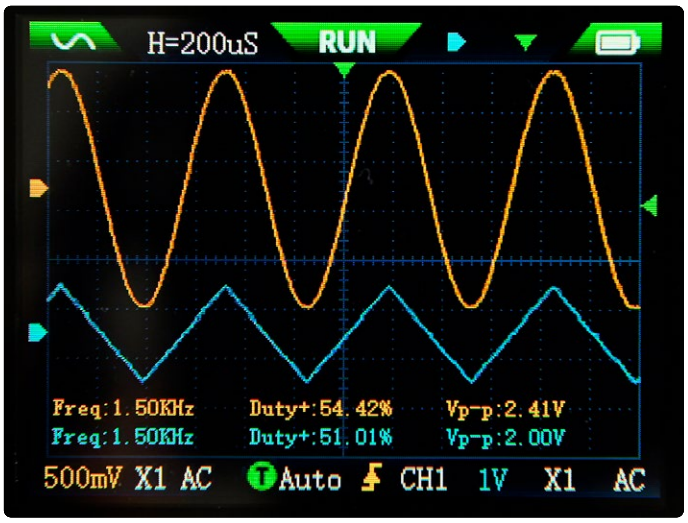


Figure 3: The oscilloscope is now usable up to 50 MHz and also seems to respond faster than the 2C23T.

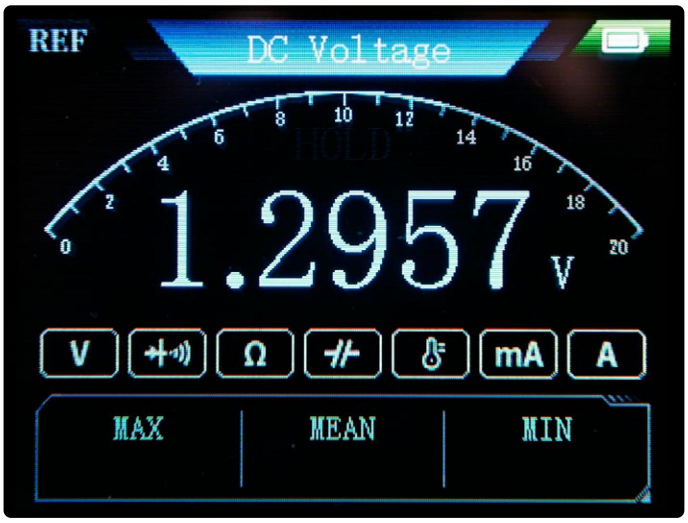


Figure 4: The display of the multimeter has a sleeker design, there is now only an analog scale and that is sufficient.

still positioned very close together, which means that probes with plastic covers won't fit. Fortunately, two compatible probes are included.

The oscilloscope's bandwidth does indeed extend to at least 50 MHz. However, the signal amplitude increases slightly from 10 MHz to 50 MHz, which also causes a slight variation in the measured values. This is likely due to a slightly nonlinear input stage. However, once you're aware of this, you can compensate for it accordingly.

Multimeter Performance of the Fnirsi 2C53T

Like the 2C23T, the multimeter of the Fnirsi 2C53T features auto-detection, but it does not function below 0.7 V. If needed, you can manually switch modes if you suspect the displayed value is incorrect or if a resistance value is shown instead of a voltage reading.

I find the multimeter's display an improvement over the 2C23T, as it now includes a single analog scale above the measured value. The minimum, maximum, and average measured values are displayed at the bottom. Once again, I tested the accuracy of the multimeter across various voltage, current, resistance, and capacitance values, and the results were identical to those of the 2C23T. In fact, the accuracy is at least twice as good as the stated specifications, which is very impressive!

That said, I do question the decision to increase the number of digits, as they are not particularly necessary given the 0.5% basic accuracy. However, this could be useful for comparative measurements. The meter includes two internal fuses for the two current ranges, which can be replaced by unscrewing the housing.

Finally, it's a bit disappointing that, as with its predecessor, the multimeter's four input sockets are spaced slightly less than the standard 19 mm, making it incompatible with some standard accessories. Fnirsi likely would have had to make the case a bit wider to accommodate the proper spacing.

Although the function generator now offers more waveform options, the signal quality has not improved. The sine wave clearly appears slightly flattened at the bottom (**Figure 5**). Zooming in reveals the relatively rough steps that make up the signal, making it suitable only for general testing where signal quality is not a priority. The frequency range limitation to 50 kHz is particularly noticeable, representing a significant reduction compared to the 2C23T.

The operation of the generator remains unchanged. The frequency can still be set precisely per Hz. The maximum output voltage is now 3 V peak-to-peak under no-load conditions. The output impedance has been adjusted and is now approximately 50 Ω.

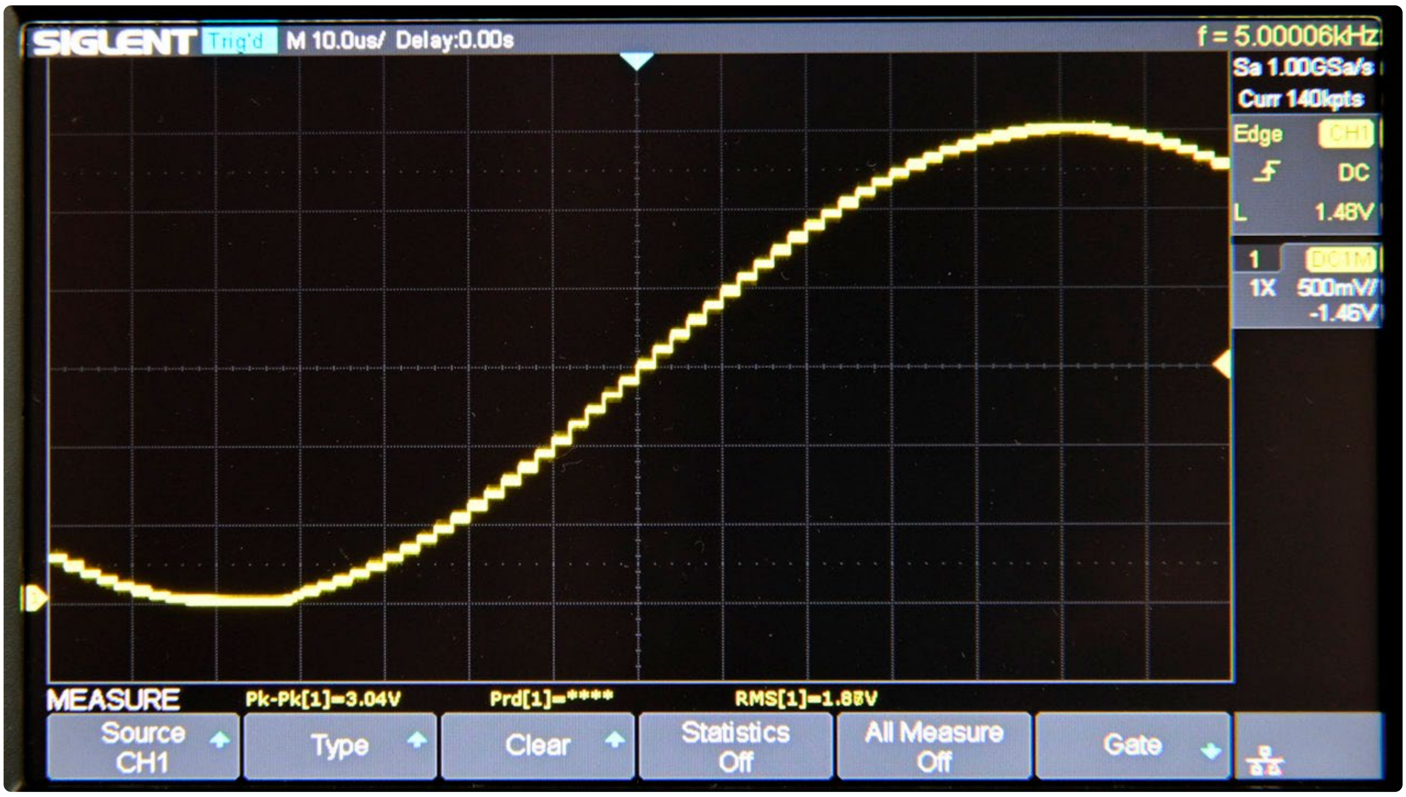


Figure 5: The signals from the function generator are rather coarse, you can clearly see the quantization steps.

However, keep in mind that the output signal is entirely positive relative to ground, as there is no output capacitor or symmetrical power supply.

A Good Deal

The new Fnirsi 2C53T [1] may cost a few euros more than its predecessor, the 2C23T, but it delivers significant improvements in the oscilloscope section. In particular, the 50-MHz bandwidth and 250-Msamples/s sampling rate are major upgrades.

Even though the multimeter remains largely unchanged and the function generator's frequency range is significantly reduced, I would still choose the 2C53T for its oscilloscope capabilities alone. Plus, at this price, you also get two oscilloscope probes and a convenient storage case — a nice bonus!

250137-01

WEB LINKS

[1] Fnirsi 2C53T: https://www.elektor.com/products/fnirsi-2c53t-upgrade-3-in-1-2-ch-oscilloscope-50-mhz-multimeter-signal-generator

[2] H. Baggen, "Fnirsi 2C23T Oscilloscope, Multimeter and Signal Generator," elektormagazine.com, April 2024: https://www.elektormagazine.com/review/fnirsi-2c23t-oscilloscope-multimeter-generator

You design. We deliver.

The Newest Products for Your Newest Designs®

MOUSER ELECTRONICS
Products
Manufacturers
Services & Tools
Technical Resources
Help
The Newest Products for Your Newest Designs®
Ensure your products are 100% authentic
Mouser was the first SAE AS6496 accredited distributor
mouser.com

mouser.com/new

MOUSER ELECTRONICS