Michał WIDLOK*

ICSP PROGRAMMER CONTROLLED WITH REAL-TIME OPERATION SYSTEM FROM PC PARALLEL PORT

SUMMARY

Many of new programmable ICs like micro-controllers, PLDs and other devices feature some kind of ICSP (In Circuit Serial Programming) standard. Such devices can be programmed, reprogrammed or checked while soldered in the application circuit, using only a few lines and a special connector. This technology is very useful for the designer, however each manufacturer uses it's own ICSP standard and dedicated programming device. This means that sometimes even a single application need a few different programmers and connectors. To overcome this problems a single very universal ICSP programmer was build.
The article presents software driver and hardware implementation of the proposed programmer. To reduce design time very simple hardware was chosen – programmer does not have any "build in" intelligence and places all control tasks on the PC and software. Block schematic and a description of the modules and their connections is presented. To enable fast data transmission rate and provide good time resolution a hard real-time operation systems is needed. The software driver is written under QNX – very fast, reliable and well known OS. Description and general algorithm of the software driver and library is given. While ICSP programming is the main task, programmer also can be used as a digital I/O device for a PC. During about 2 years of use many communication protocols was implemented, not only for ICSP programming but also for debugging, control or testing of other applications. Concurrently author is working on RTAI Linux driver for the programmer to enable using it under that platform.

Keywords*: ICSP programmer, real-time, software driver, QNX*

PROGRAMATOR ICSP STEROWANY PRZEZ SYSTEM CZASU RZECZYWISTEGO Z RÓWNOLEGŁEGO PORTU KOMPUTERA PC

Wiele nowych programowalnych układów scalonych jak mikrokontrolery, układy PLD i inne posiada mechanizm ICSP (In Circuit Serial Programming – programowanie szeregowe wewnątrz układu). Układy te mogą być programowane i testowane, gdy już są wlutowane w docelowy obwód. Przeważnie potrzeba do tego kilku linii i specjalnej wtyczki. Ta technologia jest bardzo użyteczna dla projektanta, ale każdy producent posiada swój standard ICSP i dedykowany programator. To oznacza, że niekiedy nawet pojedyncza aplikacja wymaga kilku różnych programatorów i łączówek. Aby temu zaradzić, zaprojektowany i zbudowany został uniwersalny programator ICSP.
W artykule przedstawiono programowy sterownik i sprzętowy układ programatora. Aby skrócić czas projektowania urządzenia, zdecydowano się na prosty układ sprzętowy – programator nie posiada własnej „inteligencji", a wszystkie zadania sterowania muszą być zrealizowane przez program. Został przedstawiony schemat blokowy i opis wszystkich bloków oraz ich połączeń. Aby umożliwić odpowiednio szybką transmisję danych i dobrą rozdzielczość czasową, wymagany jest system operacyjny czasu rzeczywistego. Sterownik został napisany pod QNX – bardzo szybki, pewny i znany system operacyjny. W artykule zostały zamieszczony opis i algorytmy działania sterownika oraz biblioteki funkcji użytkownika. Mimo iż programowanie ICSP było głównym zadaniem, programator może również zostać użyty jako cyfrowe urządzenie wejścia/wyjścia dla komputera PC. Przez prawie dwa lata użytkowania wiele protokołów komunikacji zostało zaimplementowanych nie tylko dla programowania ICSP, ale też dla testowania i kontroli różnych aplikacji. Obecnie autor pracuje nad sterownikiem do programatora dla systemu Linux.

Słowa kluczowe*: programator ICSP, czas rzeczywisty, sterownik programowy, QNX*

1. INTRODUCTION

Most of the new programmable integrated circuits have some form of In-Circuit Serial Programming (ICSP) interface. This enables designer and manufacturer to quickly test new firmware for the application without any hardware changes. Typically during application design, a special service connector is provided to program and test micro-controllers and other programmable devices on the board. With such connector the application can be programmed, tested or reprogrammed even directly at the customer's home. However there is no single standard for ICSP interface. Almost every manufacturer has it's own specification for ICSP capable devices and provides dedicated programming hardware. Frequently even the same manufacturer has a few different programmers for different chip families. This is acceptable for mass production, but rather problematic for designing and prototyping. The main problem is

* Studia doktoranckie, EAIiE, Katedra Autom. Napędu i Urządzeń Przemysłowych

that such dedicated programmers are suitable only for single task – programming or testing. This means that designer have to have a lot of different devices, that often can not be connected to the PC at the same time, and that need a lot of different driver software. Sometimes these drivers uses incompatible input files format that creates further complication.

To overcome this problems and to test real-time capabilities of current PCs the new universal ICSP programmer was designed and build. In the begging, idea was to build single device that can program many different micro-controllers (MicroChip's PIC mostly) and other programmable ICs. However, during designing of driver software it was clear that it can be made to support many different communication standards like SPI, I2C, JTAG or RS232. It was also possible to create a simple digital recorder with moderate sampling rate. Such characteristic makes possible to use the programmer as an application debugger or tester, making it a very universal and handy device. Author uses it for about 2 years with very good results, for programming, testing and debugging his designs. This paper presents the construction and software implementation of the programmer and some test that was carried over.

2. BASIC ICSP REQUIREMENTS

ICSP usually does not need very high transmission rates however, time relation or signal sequences are often very strictly defined. Typically the start-up sequence or programming pulses times and delays are specified within 10–50 µs. Following this is guidelines is very important because it is quite easy to destroy programmed device in milliseconds. This is especially true for EEPROM or EPROM ICs because they are sensitive to programming pulses. FLASH ICs are much more robust. This means that every universal programming device should be able to give microseconds accuracy and should be very reliable. As was mentioned earlier, data transmission is not critical but it is also desirable to reduce programming time. If FLASH memory of single modern micro-controller is about 100 kB then programmer have to transmit this amount of data plus any overhead like setting up programming pulses and so on. To reduce this time to about 10–15 seconds (value acceptable for a designer waiting for it) transmission speed should be at least 8–10 kB/s. This means that the clock should be at about 80 kHz however the more the better. ICSP devices usually allows much faster transfer rate.

If the programmer is also supposed to service as application debugger or digital recorder then it is clear that the time resolution should be 2–5µs and clock frequency above 100 kHz is required. It might be seemed not very hard to reach but we have take into account that different and sometimes not standard protocols have to be realized. Because of this it is impossible to use typical dedicated ICs to do the task. Also there should be possibility to configure lines as inputs or outputs on the fly, and to provide high power/high voltage outputs. Very desirable is also adjustable power supply and possibly a simple hardware. Finally (after a few tries) all this goals were met.

3. CHOOSING OF HARDWARE AND SOFTWARE PLATFORMS

3.1. Programmer hardware

To cut costs and design time very simple hardware platform was chosen. It is clear that this will cause very high load on control PC's processor especially during high frequency transmitting or receiving. The block diagram of the programmer is shown on Figure 1. The programmer has 3 configurable input or output lines with TTL signal levels, 3 output/high impedance TTL lines, 2 high voltage outputs (regulated 1.2–15 V, 200 mA) and regulated 1.2–7 V power supply. Voltage regulator is short-circuit protected and has 1 A current capability.

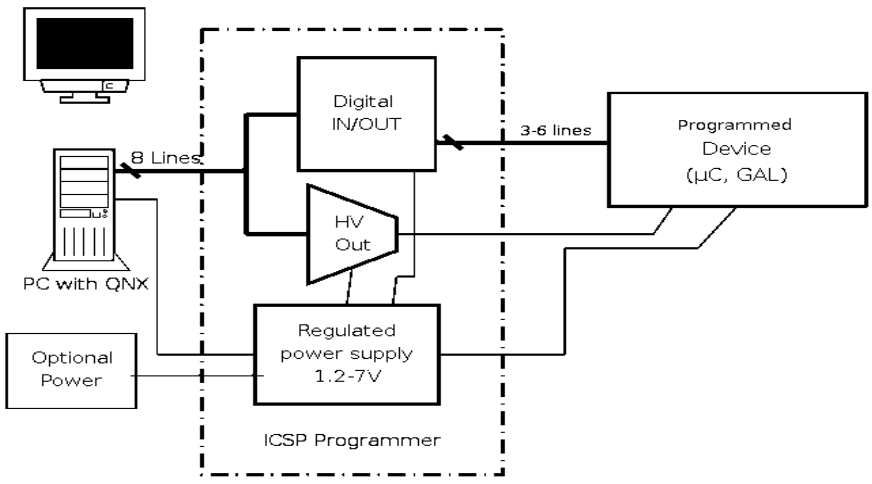

Fig. 1. ICSP Programmer block diagram

The programmer consists of digital Input/Output block, high voltage lines amplifier and regulated power supply. I/O block is build over single programmable IC – Lattice GAL22V10. This device is fast, has high power TTL outputs, and is very easily to configure it for given tasks. As high power output block an ICL7667 dual amplifier was used to convert TTL signals from PC to 15 V/200 mA outputs. An application power supply block consists an integrated voltage regulator with short circuit and over temperature protection. All of the blocks are connected to PC standard parallel port (Centronics).

This port was chosen because:

– most PCs and Laptops have it,
– it is very easy to configure and program it,
– it does not need any special hardware to work,
– it is quite robust,
– cables or other hardware is not critical for proper operation.

The programmer uses 20 pins connector and dedicated cable as a target application connection. Programmer needs its own power supply (8–15 V, 1.5 A maximum) for operation. It was build in that way to enable powering it directly from control PC (its 12 V power line). However if accurate high voltages are needed it is necessary to use dedicated stabilized power supply, that also was build.

3.2. Control software

Programmer locates all control tasks on PC. Such implementation is very universal and almost all interface standards can be realized with appropriate driver software. However it is clear that to meet needed performance a high quality real-time operation system and driver is essential. It is important that soft real-time is not enough because deadlines and jitter have to kept below certain values all the time. Software failure can lead to destruction of the programmed device or meaningless received data. To meet these goals QNX was chosen as the operating system. It is very fast, reliable, easy to configure and it was well know by the author. It is not recommended to write real-time control programs under Windows even if it is well know for most users and very popular, because of hard real time demands during programming procedure realization Not many computer have QNX installed, so using the programmer out of the laboratory or without dedicated notebook is problematic. Concurrently work is in progress to port all the software to RTAI Linux, that will achieve similar performance but the driver is not ready yet.

4. SOFTWARE DRIVER IMPLEMENTATION DETAILS

4.1. General overview

Control software was written in C – very popular and widely known language. The driver is implemented as a library of functions rather then stand alone program. Such configuration was chosen because it gives the best control of the programmer's hardware, it is very universal and it is easy to add new protocols or functions without any changes to the existing programs. Functions was grouped in the modules by category.

Table 1
Library modules and functions

Module	Functions names	Description
icsp_prog_hw	*SSPinit(), PROGinit(), LEDblink(), PROGtest()*	Hardware and programmer specific functions. Mostly used in other modules
icsp_prog_date	*SetDir(), SetHV(), SetData(), GetData()*	Direct I/O access functions. Also allow set up direction of programmable lines
icsp_prog_RO	*WriteRO(), ReadRO(), SetREN()*	Registered (output only) lines direct access
icsp_prog_tsc	*Delayus(), waitus(), ticus(), tocus(), Initus()*	Provide TSC counter access and time measuring. All arguments are in micro seconds
icsp_hex_com, icsp_hex_I32	*Ld_Hex_F(), Wr_Hex_F(), CompHEXI32(), ScanHexI32(), PrintHexI32()*	Allows loading and analyzing files in IntelHEX format. Such files are commonly used in programming tools for micro controllers
icsp_JEDEC	*Ld_Jedec_F(), Wr_Jedec_F(), CompJEDEC(), DispJEDEC()*	As above, but for JEDEC files. This format if used for JTAG devices
icsp_prog_sclk	*SendSClock(), SendSData(), ReceiveSDate()*	Data synchronous transmission with additional clock output. PIC17C756 requires such clock
icsp_prog_serc	*SendClock(), SendData(), ReceiveData()*	Synchronous (SPI) data transmission functions – Master mode
icsp_prog_sers	*SendDataSl(), ReceiveDataSl()*	Synchronous (SPI) data transmission functions – Slave mode
icsp_prog_asyn	*SendAsynClock(), SendAsynData(), ReceiveAsynData()*	Asynchronous data transmission functions. Uses RS232 format
icsp_MI2C	*SendI2CData(), ReceiveI2CData()*	I2C transmission – single master mode
icsp_JTAG	*ReserJTAG(), GoJTAG_xxxx(), SendJTAG(), ReceiveJTAG()*	JTAG interface communication functions. GoJATG_xx functions family set the JTAG "state machine" to given state
icsp_scope	*SampCH02(), SampTrig()*	Digital recorder functions. Allows additional recording control like trigger or delay

Table 1 shows most important functions and categories.

The table lists only most important functions. Whole library consist of 17 modules 16 header files and over 60 functions, plus some "inline" functions, that are not accessible for the user. As shown in the table many common communication standards and file formats are implement already. Functions also can be used to directly control programmer's lines. All of the functions can be used directly in any QNX program. It can be seen that by using the library it is very easy to load files in different formats and then send or receive application data. All parameters like speed, polarity, protocol or format can be changed on the fly in very short time – very useful for debugging.

4.2. Time and frequency measurement system

As was stated before time resolution and accuracy of micro seconds is needed to meet the ICSP programming specifications of some chips. Such short response time and low jitter is quite difficult to achieve even for hard real-time operation systems. QNX (and RTAI Linux) running on modern PCs however can meet such requirements. For QNX the easiest way was to disable all interrupts during critical time. To accurately measure time TSC (Time Stamp Counter) – 64 bits counter presented on every new PC processor was used. This counter is incremented at every clock cycle, so it is enough to know the frequency of the processor to measure the time. For testing 1.7 GHz Pentium 4 was used, however Pentium running on Intel main-board is not the best choice for real-time use. Most of this boards have SMI (System Management Interrupt) "feature" – an interrupt that is called by system controller every few seconds. This is completely invisible for the software and OS, so it is impossible to block it in typical way. This can cause occasional jitter of several hundreds of micro seconds. It is sometimes possible to disable SMI by writing to main-board registers, but it is system depended and usually not documented anywhere. Unfortunately only that PC was available at that time. With 1.7 GHz clock it is theoretically possible to measure about 7 ns, but realistically this value is much longer i.e. about 0.7 µs. This is cause mostly by long time needed to access I/O ports. The other problem is how long time can be measured in that way. TCS is 64 bits in nature, but to speed up operations typically only 32 bits are used, how-ever functions for full 64 bits access are also provided. To give a good overview of possible measurement range for 32 and 64 bits counter the maximum available time is computed:

– for 32 bits

$$T_{max} = \frac{1}{1.7\ \text{GHz}} \cdot 2^{32} = \frac{4294967296}{1700000000} \approx 2.5\ \text{seconds};$$

– for 64 bits

$$T_{max} = \frac{1}{1.7\ \text{GHz}} \cdot 2^{64} \approx 1.08 * 10^{10}\ \text{seconds} \approx$$

$$\approx 125590.58\ \text{days} \approx 344\ \text{years}.$$

It is clear that even 32 bits counter should be sufficient for most of the time, but if one use 64 bits counter the measurement range is many times longer then the lifetime of typical PC.

4.3. Flowchart of typical program

Flowchart of programming typical ICSP device is shown on Figure 2 below.

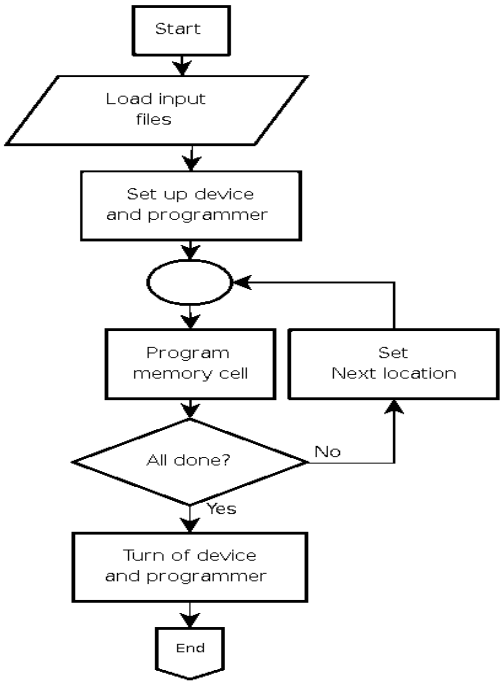

Fig. 2. Example ICSP device programming flowchart

Generally many devices can be programmed in the way shown on Figure 2. It can be seen that only device specific parts have to be realized by the application programmer. All other tasks can be done by using available library functions. One possible problem with such implementation is that every new micro-controller or any other programmed device must have it's own "driver" made of library functions and some device specific code. However (as author's experience shows) it is really easy to add new drivers. Typically it can be done simply by following manufacturer's programming specification manual for given device. If new one is similar to other, already implemented, all operation is reduced to changing programming pulses length, memory limits and transmission frequency.

5. CARRIED TESTS AND ACHIEVED RESULTS

To reduce risk of damaging programmed ICs (that sometimes was really expensive), a lot of test has been done on the programmer. Most important is to minimize probability of program crash or hard look-up. Such events can leave the programmer in totally unknown and possibly danger state. Any program or device error should instantly trigger the shut down of high voltage lines and switch all lines to 'input' or 'high impedance' state. This might screw-up program loaded to device, but will rather not destroy it. In fact

to give the user additional control of the programmer behavior all library routines call user specified "error function" in every unusual condition. That function should be set in the begging of the driver. If not specified a default function will be used that turn off all programmer's lines. This was found to be the most safe for programmed devices. Anyway during 2 years of use only one micro-controller was damaged, and this was happen not because of programmer fault but because of misunderstanding of programming specification by the author. Fortunately todays real-time operating systems like QNX or RTAI Linux are very robust and error caused by look-up of the OS was never noticed. User should however keep in mind the special nature of some computer main-boards or processors. For example it was not possible to catch SMI on Intel system on the scope connected to programmer outputs, but this not mean that it could not happen during programming. A special real time test programs are recommended to check how the PC and OS work with real time applications before any programming is done. For example RTAI Linux has very good calibration and testing utility included.

During testing a lot of different test were done. Most important results are presented below. All this tests were done with the programmer connected to 1.7 GHz Pentium 4 running QNX and were recorded by digital oscilloscope. It should be noted that this system suffers from SMI, but fortunately such event never show up during tests. Concurrently programmer is used with AMD notebook with very similar performance. Most important results are presented below.

5.1. Maximum frequency synchronous transmission

During this test arbitrary data was sent by the programmer output lines. Figure 3 presents data and clock lines waveforms. Synchronous transmission is much more sensitive for edge timing then for clock time or frequency.

As can be seen on the above figure jitter is very low (impossible to measure with used equipment) and data line is set always on the rising clock edge. Achieved frequency was 380 kHz, however typically (during normal programming) the frequency is lower – experimental value is about 200 kHz. This is caused mostly be overhead imposed by programmed device protocol, needed set up or programmed device limitations.

5.2. Asynchronous data transmission – jitter test

Asynchronous data transmission is quite sensitive to the frequency jitter, while transfer rate is less important. The most popular baud rate is 9600 bits/second used by many RS232C devices – printers, plotters, modems and so on. This test was done to check if jitter imposed by the programmer is not to high. As above arbitrary data was send and output was recorded on the scope. The clock signal is a special "helper" line not used by the applications but provided by the programmer.

Figure 4 shows the results. Frequency jitter was very low – it is impossible to see it on the figure, while measured value (using Octave – Linux version of Matlab) was about 0.7 μs. This value is many times lower then needed – RS232 can tolerate up to 10–15%.

5.3. Response time for external events

While receiving data (asynchronous especially) it is important that reaction time for transmitter edge is short. If this time would to long the receiver will not synchronize it's clock correctly and error can occur. To check programmer reaction time an rising edge was send to the input line and programmer was set to start receive data on that edge. The results are shown on Figure 5. Top waveform is the triggering line, wile bottom is the additional clock signal delivered by the programmer.

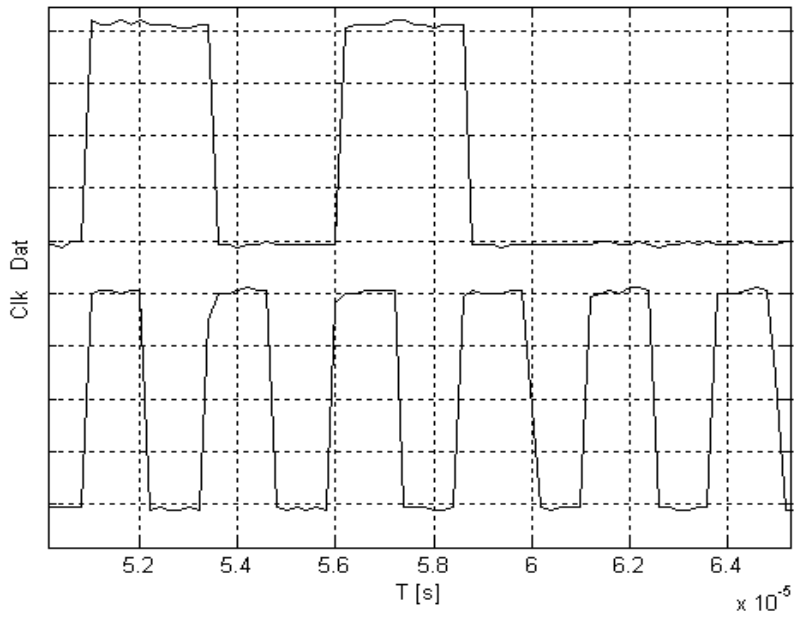

Fig. 3. Maximum frequency synchronous transmission

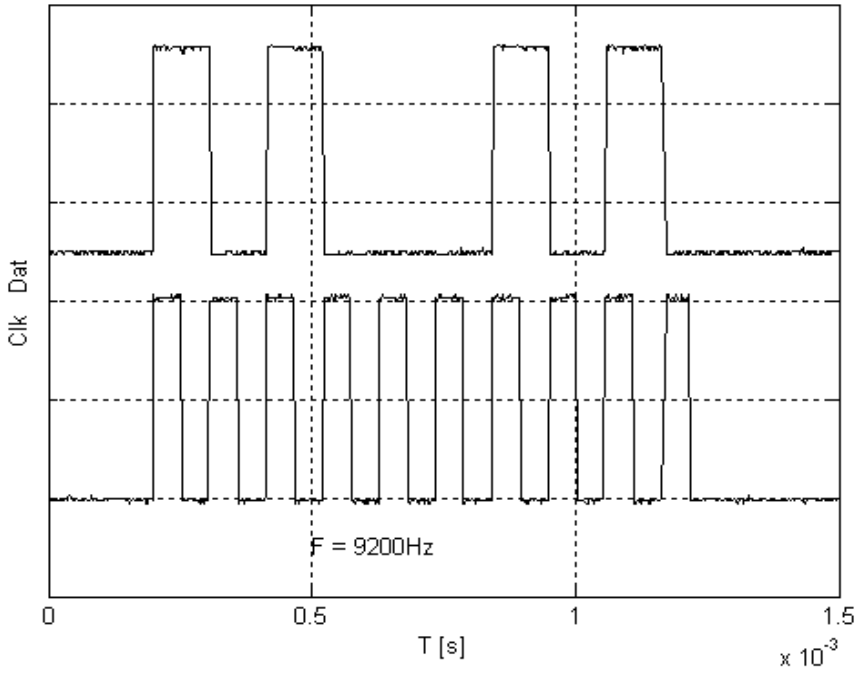

Fig. 4. Asynchronous transmission

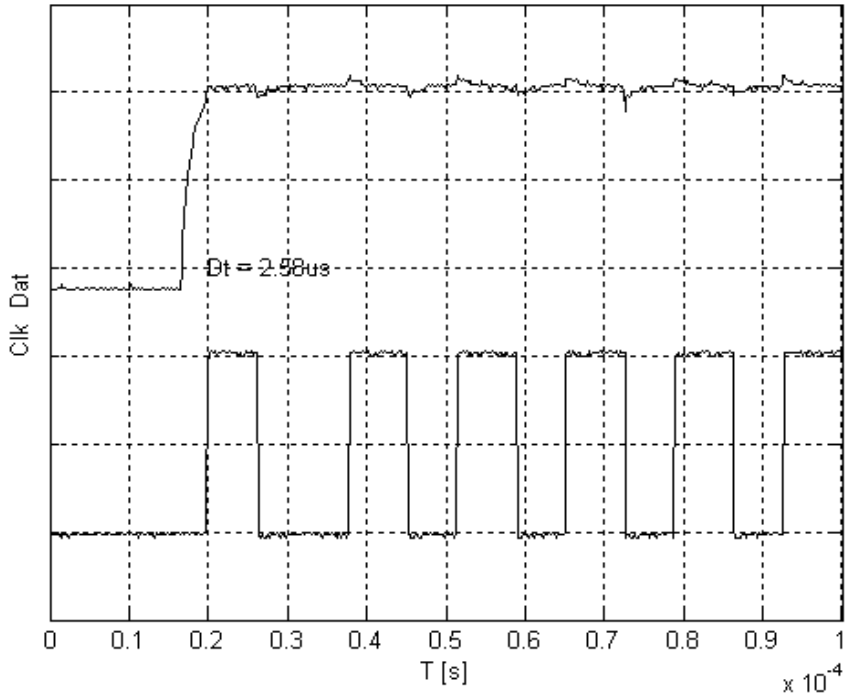

Fig. 5. Reaction time for external event

It can be seen that reaction time is very short. Internal programmer trigger occurs only 2.58 μs after the rising edge. This time is much shorter then needed for most cases (acceptable value is about 200 μs), and it was never a problem during normal use. If someone examine clock signal closely he/she will find that the first clock period is longer then others. This is not the mistake but the required synchronization for asynchronous data format. The first sent bit is always a "start bit" then data bits follow it. Then programmer waits during start bit time plus ½ of the next bit time – this gives the best sampling time occurring in the middle of the transmitted bit.

6. EXAMPLES OF SUCCESSFUL APPLICATION OF THE PROGRAMMER

The programmer was generally intended to be a very universal replacement for commercial programming devices. However it was often clear that much other tasks can be

done if we treat it as a programmable digital I/O for the PC with a good and fast driver. Some examples of successful applications are presented below.

6.1. Debugger for micro controller and other applications

Many applications designed today (and all designed by the author) have some form of debugging or "service" connector. This is usually a small plug that can be used for application testing, reprogramming of the processor or other tasks. This greatly simplifies debugging process. Up to now about 15 different projects were debugged this way. Often it was even possible to find hardware faults (bad connection, damaged output MOSFET) without using the voltmeter or oscilloscope. It is clear that additional work that have to be done to provide "service" connector and driver software will pay off during debugging phase.

6.2. RF receiver/transmitter

This was done by connecting Links RF modules to the programmer. General idea was to test the range and transmission error rate of the modules in building environment. PC (or driver) was set up to constantly send and receive a "packed" of known data. The other end was equipped with the same Links modules and PIC micro-controller (final application) that was moved around the building. At the end log files on the PC could be examined and it was possible to find out where the error rate is to high or the average range.

6.3. Peltier module controller and temperature regulator

During this project a laser diode temperature controller was build. The programmer was connected to DS1820 temperature sensor (One Wire interface) and to power amplifier that provided 12 V, 2 A bipolar supply for Peltier module. Cooling or heating power was regulated by PWM drive. In PC a PID controller was realized with the possibility of changing P, I or D factors on the fly. Reference temperature was set up form keyboard, and the regulation algorithm could be started. PWM frequency was about 10 Hz – more then needed, and the temperature could be stabilized to about 10 percent. This is not very accurate but mostly the limiting factor was a temperature meter. It was not possible to make a good "thermal connection" between the meter and the diode, and the accuracy of the meter (not intended for such use) was low.

The above are only a few examples of possible programmer uses. Generally it can be seen as quite fast reliable digital input/output module with nice features like integrated power supply or high power output lines. It is interesting that so small and simple device can serve so many different functions – and only a good OS and software driver is needed. This is very good example of how fast new PC computer is. Unfortunately most of the popular software that is used today does not have ability to use real time capabilities of the hardware. Many commercial manufacturers (even those well know) does not even try to provide optimized drivers for their equipment, simply calling for a faster PCs, that often can not overcome some basic faults and simplifications in the drivers.

7. SUMMARY

Carried tests and application examples shows that presented programmer and it's controlling software is very handy and useful debugging tool. Almost every project that is equipped with TTL service port can be debugged or tested by using a PC – all results are directly presented on the screen and all parameters can be altered from keyboard. In addition the hardware is cheap and easy to fix when some damage occurs (up to now only 2 faults were noticed).

Generally the programmer can by characterized by:
- very simple hardware design,
- parallel port (Centronics) connection to the PC,
- special software driver and library functions that uses real time operating system running on PC,
- very robust hardware and software design to minimize the risk of damaging programmed ICs,
- provided library simplifies new protocols design and new standards addition – often it is enough to set pulses lengths or polarity,
- uUse of QNX or RTAI Linux minimize risk of software crash – and unknown consequences for programmed ICs.

Original driver was written for QNX, but implementation for RTAI Linux is concurrently in progress. Linux programs for almost all tasks were written and they are typically well maintained by the authors.

References

[1] Mielczarek W.: *Szeregowe interfejsy cyfrowe.* Gliwice, Helion 1993
[2] Intel: *Designing for On-Board Programming Using the IEEE 1149.1 (JTAG) Access Port.* Application Note AP-630. Intel, 1996
[3] Altera: *IEEE1149.1Boundary Scan Testing in ALTERA Devices.* ALTERA, 2000, www.altera.com
[4] Microchip: *In Circuit Serial Programming (ICSP) Guide.* Microchip Technology Inc., 2000, www.microchip.com
[5] Kernighan B.W., Ritchie D.M.: *Język ANSI C.* Warszawa, WNT 1998
[6] Metzger P.: *Anatomia PC.* Gliwice, Helion 1993
[7] QNX Software System: *QNX Operating System, System Architecture.* QNX Software System 1997, www.qnx.com

Wpłynęło: 6.03.2005

Michał Adam WIDLOK

Was born in 1979 in Kraków, Poland. He received M.Sc. degree in electrical engineering from the University of Science and Technology in Kraków on the subject "Testing and analysis of high speed interfaces with specially designed ICSP programming device" in 2003.Actually he is on the second year of Ph.D. Studies on AGH University of Science and Technology in Krakov. His research directions are power electronics and micro-controller control systems.

e-mail: widlok@uci.agh.edu.pl