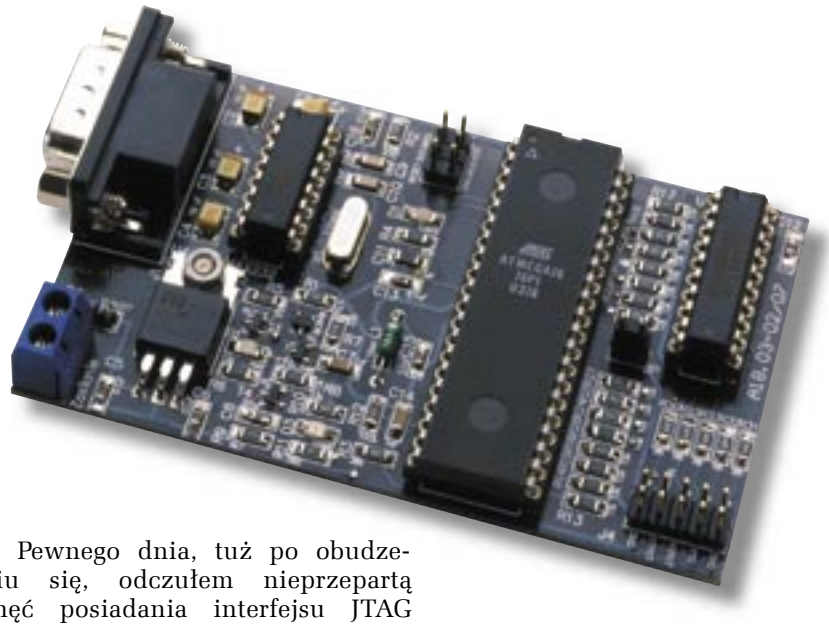
Interfejs JTAG do procesorów AVR

AVT-581



Od chwili pojawienia się pierwszego mikrokontrolera na rynku, zauważalna jest tendencja do implementowania w jego strukturze możliwie wszystkich wykorzystywanych w praktyce komponentów. Spowodowało to – szczególnie po powszechnym wprowadzeniu pamięci programu typu Flash – że zewnętrzna magistrala danych i adresowa stały się zbędne. Niestety dla konstruktora oznaczało to kłopoty z dostępem do zasobów mikrokontrolera. Panaceum na te dolegliwości mogą być emulatory z interfejsem JTAG.

***Rekomendacje**: Niezwykle użyteczny element wyposażenia warsztatu, w którym powstają konstrukcje oparte na mikrokontrolerach AVR, znacznie podnoszące komfort prac uruchomieniowych.*

Pewnego dnia, tuż po obudzeniu się, odczułem nieprzepartą chęć posiadania interfejsu JTAG do procesorów AVR. Po porannych ablucjach i małym co nieco zasiadłem do komputera, aby dowiedzieć się, w którym sklepie internetowym i za ile mogę nabyć takie urządzenie. Szybko skonstatowałem, że zakup oryginalnego JTAG-a [1] wygenerowałby ogromną „dziurę" w moim domowym budżecie. Produkty mniej renomowanych firm [2] nie wzbudziły mojego zaufania, „pachnąc" z daleka nieoczekiwanymi kłopotami związanymi z ich eksploatacją w miarę pojawiania się na rynku kolejnych wersji AVR Studio. W miarę upływu czasu przekonałem się o słuszności moich podejrzeń [3].

Z literatury dokumentacji procesorów AVR mogących współpracować z interfejsem JTAG jasno wynika, że zrobienie takiego urządzenia od podstaw jest praktycznie niemożliwe ze względu na brak opisu czterech instrukcji debugera ($8?B). Odpowiednia dokumentacja jest dostarczana przez Atmela jedynie wybranym producentom. Zatem jak mawiał kapral Kuraś „jakbyś się nie kręcił, d... zawsze z tyłu". Niewątpliwie uczy to, jak monopol na informacje może utrzymać wysoką cenę produktu oraz „trzymać w szachu" ubiegających się o licencję.

W Internecie funkcjonuje bądź funkcjonowało kilka stron, na których znajdowały się opisy [4, 5] jak w domowych warunkach, za pomocą AVR Studio, zbudować interfejs JTAG. Oba urządzenia zbudowałem i przetestowałem z negatywnym wynikiem. Nawet usunięcie ewidentnych błędów software'owych [5] występujących w przypadku programu Boot Strap Loadera dla procesora Atmega16 nie przyniosło spodziewanych rezultatów. Rozwiązania pozwalające korzystać ze starych wersji AVR Studio [6] zignorowałem z oczywistych względów.

Jak widać, stanąłem przed dylematem, czy kontynuować zabawę, licząc się z fiaskiem całego przedsięwzięcia i stratą czasu, czy też wyładować swoje niezadowolenie na żonie i dzieciakach. W obawie o wszelkie możliwe represje ze strony rodziny wybrałem pierwszą opcję, a wynikami mojej pracy mogę podzielić się z Państwem.

Opis układu

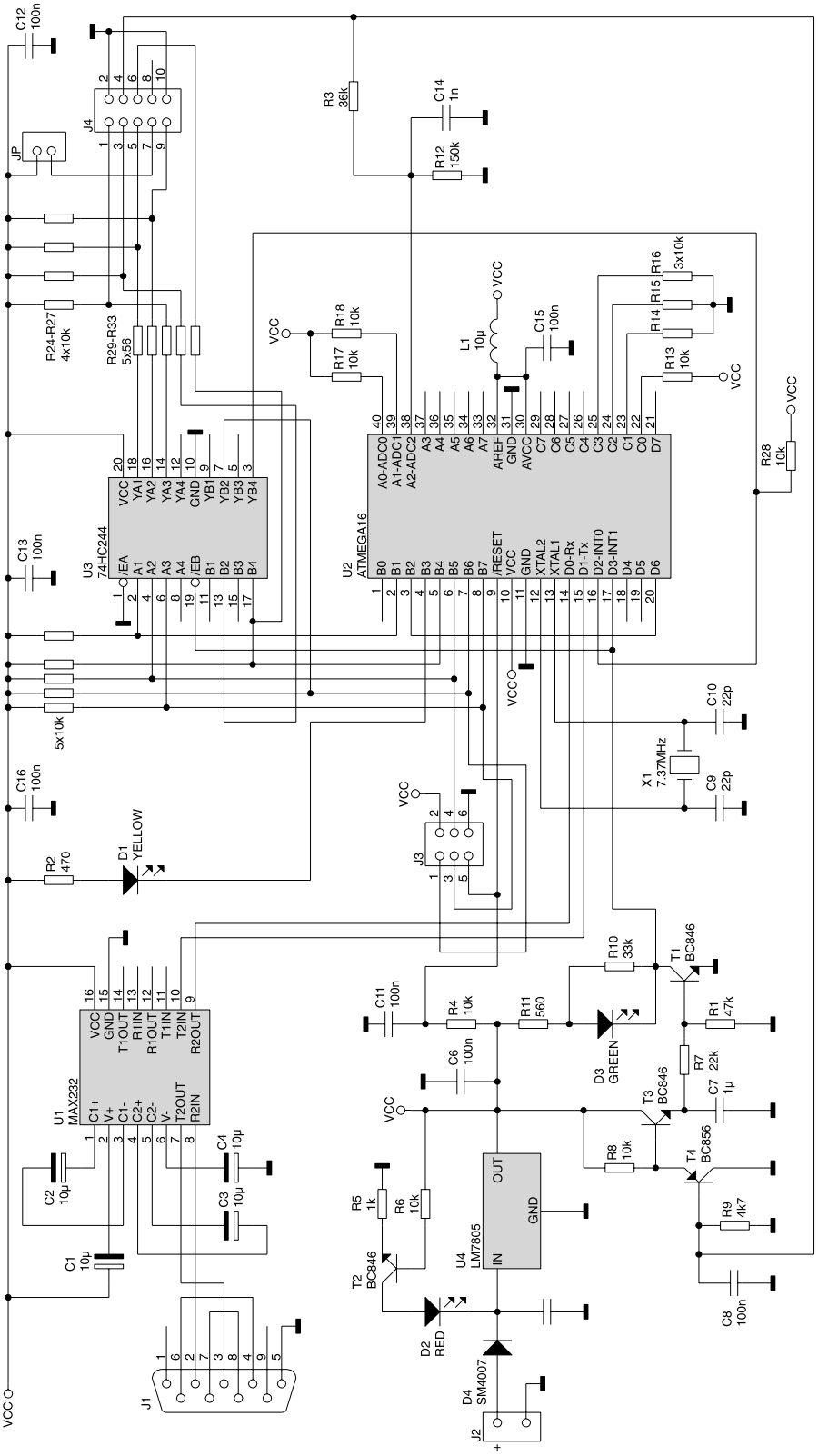
Schemat elektroniczny układu jest w dużej mierze odwzorowaniem układu prezentowanego w Internecie [4] opartego na schematach starszych wersji interfejsu JTAGICE (**rys. 1**). Wprowadzone zmiany dotyczą usunięcia drugie-

go procesora sprawującego funkcję loadera programu do głównego procesora JTAG-a oraz niewielkiej przeróbki połączeń pozwalającej na bezbłędne działanie układu ISP (prawdopodobna, hardware'owa przyczyna wadliwej weryfikacji programu wprowadzonego do procesora [5]). Warto zaznaczyć, że jednoprocesorowy układ jest stosowany obecnie w interfejsie JTAGICE produkowanym przez Atmela [7].

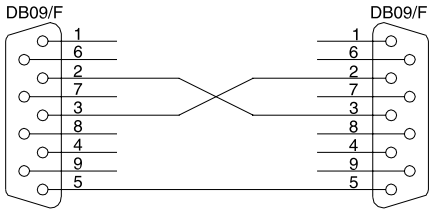
Sercem układu jest procesor Atmega16 komunikujący się z AVR Studio w komputerze za pomocą kabla *null modem* (**rys**. **2**, patrz Pomoc Windows) przez konwerter poziomów napięć MAX232. Procesor jest połączony z emulowanym układem, przez bufor SN74HC244 za pomocą dziesięciożyłowego przewodu. Stabilizator napięcia LM7805 wytwarza napięcie 5 V do zasilania interfejsu i emulowanego układu, gdy założona jest zwora na wyprowadzenia JP. W ogólnym przypadku emulowany układ może być zasilany z innego źródła (JP rozwarty). Dopuszczalne jest również zasilanie interfejsu z emulowanego układu (JP zwarty), jednak ze względu na dane katalogowe użytych elementów, napięcie zasilania powinno pozostawać w zakresie 4,5...5,5 V. W rozważanym przypadku nie wolno podłączać gniazda J2 do innego źródła zasilania (możliwe uszkodzenie stabilizatora).

Dioda D1 (żółta) zapala się w momencie wymiany informacji pomiędzy interfejsem i AVR Studio. Sygnalizatorem podłączenia napięcia zasilającego do gniazda J2 jest dioda D2 (czerwona). Natomiast dioda D3 (zielona) sygnalizuje, że podłączone jest napięcie zasilania do emulowanego układu. Kolory diod są zgodne z zastosowanymi w interfejsie JTAGICE Atmela.

Program „ładowany" do procesora Atmega16 jest zwykłym Boot Strap Loaderem (BSL) mającym za zadanie odpowiednie zainicjowanie programu AVRProg z AVR Studio tak, aby plik upgrade.ebn mógł być następnie załadowany do Flasha procesora Atmega16 za pomocą tegoż programu BSL. Protokół transmisji pomiędzy AVR-Prog i programatorem (BSL) jest powszechnie znany, a opis działania wszystkich używanych komend jest dostępny w literaturze [8...10]. Program BSL (**list**. **1**) nie wykorzystuje wszystkich komend, tym niemniej prawie wszystkie z nich zostały w nim umieszczone, aby ułatwić czytelnikowi ewentualne modyfikacje software'u w przyszłości. Zamieszczony program może być skompilowany za pomocą dowolnej wersji AVR Studio. Należy jednak pamiętać, że jego symulacja nie jest możliwa ze względu na brak w AVR Studio 3.5x lub jedynie fragmentaryczną obsługę w AVR Studio 4.0x wszystkich opcji rozkazu SPM.



Rys. 1. Schemat interfejsu JTAG do procesorów AVR



Rys. 2. Schemat kabla *null-modem* (oba wtyki żeńskie)

Po włączeniu napięcia zasilania interfejsu, przełączanie oprogramowania z programu BSL do programu JTAGICE zachodzi jedynie wówczas, gdy podłączone jest napięcie zasilania do emulowanego układu (pin 4 – VTG, gniazdo J4). Jeśli nie występuje tam napięcie wyższe od około 2,7 V (zależnie od współczynnika β tranzystorów T4, T3 i T1), „panem sytuacji" pozostaje program BSL i możliwe jest zaprogramowanie lub przeprogramowanie interfejsu.

Montaż i uruchomienie układu JTAG-a

Ze względu na częściowe zastosowanie elementów SMD, ich montaż wykonujemy począwszy od najmniejszych, aż do największych (widok płytki drukowanej wraz z rozmieszczeniem elementów opublikujemy w EP7/2004). Pozwala to uniknąć nadtopienia dużych podzespołów (np. gniazda) w czasie lutowania pomiędzy nimi małych rezystorów lub kondensatorów. Układy scalone warto wyposażyć w podstawki (precyzyjne), mając na uwadze przyszłe modyfikacje układu lub ewentualne „tragiczne" błędy w podłączaniu interfejsu do komputera lub emulowanych układów. Koszt niewielki, a komfort napraw jest nie do pogardzenia.

Po zmontowaniu i doprowadzeniu zasilania (+8...12 V względem masy) warto sprawdzić woltomierzem poziomy napięć na końcówkach kondensatorów C1 i C4 dołączonych do układu scalonego MAX232 (odpowiednio ok. –6...10 i +6...10 V względem masy). Należy też sprawdzić, czy podłączenie pinu 4 w gnieździe J4 do VCC (+5 V) skutkuje zapaleniem się diody D3 (zielonej). Na tym praktycznie kończy się uruchomienie hardware'u.

Za pomocą AVR Studio należy skompilować program BSL

List. 1

```
; Program Boot Strap Loader'a do skonstruowanego interfejsu JTAG dla procesorów AVR
; współpracuje poprawnie z AVR STUDIO v.4.06 build 223
;                                                                              4.07 build 240
;                                                                              4.08 build 310
;                                                                              4.09 build 338
;---------------------------------------
.INCLUDE „m16def.inc"
;
   .cseg
   .org 0x1C00
;
.def D0  = r0
.def D1  = r1
.def AD0 = r15
.def AD1 = r16
.def TMP2 = r17
.def COMM = r18
.def TMP1 = r19
;
   sbis PIND,PIND3
   jmp 0x0000
   rjmp INIT
;
   .cseg
   .org 0x1E00
;
INIT:
   cli
   ldi TMP1,0xFF              ;
   out SPL,TMP1
   ldi TMP1,0x03              ;
   out SPH,TMP1
   ldi TMP1,0x00              ;
   out UBRRH,TMP1             ; Transfer rate of UART 19200 BAUD (7.3728 MHz)
   ldi TMP1,0x17              ;
   out UBRRL,TMP1             ; Transfer rate of UART 19200 BAUD (7.3728 MHz)
   ldi TMP1,0x40              ;
   out UCSRA,TMP1             ; TXC = 1, USART Transmit Complete
   ldi TMP1,0x18              ;
   out UCSRB,TMP1             ; RXEN = 1, Receiver Enable, TXEN = 1, Transmitter Enable, 8 bit
   ldi TMP1,0x86              ;
   out UCSRC,TMP1             ; URSEL = 1, Register Select, UCSZ1:0 = 1, 8 bit for TRx use
   ldi TMP1,0x08
   out DDRB,TMP1              ; Activate PB3 for output
   out PORTB,TMP1             ; Send „one" to PB3
MAIN:                         ; Read command from RS232
   rcall GET_UART
   mov COMM,TMP1
   cpi COMM,0x70              ; 'p' Programmer type ?
   brne O_com01               ; Go to other commands analysis (O_com01)
   ldi TMP1,0x53              ; 'S'
TERM_P:
   rcall PUT_UART             ; Send 'S'erial programmer info
   rjmp MAIN                  ; Read next command from RS232
O_com01:
   cpi COMM,0x61              ; 'a' Report autoincrement address
   brne O_com02               ; Go to other commands analysis (O_com02)
   ldi TMP1,0x59              ; Set 'Y'
   rjmp TERM_P                ; Send 'Y'es for autoincrement address
O_com02:
   cpi COMM,0x53              ; 'S' Return software identifier (7 characters)
   brne O_com03               ; Go to other commands analysis (O_com03)
   ldi TMP1,0x41              ; 'A'
   rcall PUT_UART
   ldi TMP1,0x56              ; 'V'
   rcall PUT_UART
   ldi TMP1,0x52              ; 'R'
   rcall PUT_UART
   ldi TMP1,0x4E              ; 'N'
   rcall PUT_UART
   ldi TMP1,0x4F              ; 'O'
   rcall PUT_UART
   ldi TMP1,0x43              ; 'C'
   rcall PUT_UART
   ldi TMP1,0x44              ; 'D'
   rjmp TERM_P
O_com03:
   cpi COMM,0x74              ; 't' Return supported device codes
   brne O_com04               ; Go to other commands analysis (O_com04)
   ldi TMP1,0x74              ; Set 't' Device code for ATmega16
   rcall  PUT_UART            ; Send device code
   ldi TMP1,0x00              ; Device code terminated by 0x00
   rjmp TERM_P                ; Send termination
O_com04:
   cpi COMM,0x56              ; 'V' Return software version
   brne O_com05               ; Go to other commands analysis (O_com05)
   ldi TMP1,0x33              ; '3'
   rcall PUT_UART
   ldi TMP1,0x31              ; '1'
   rjmp TERM_P
O_com05:
   cpi COMM,0x54              ; 'T' Select device type
   brne O_com06               ; Go to other commands analysis (O_com06)
   rjmp TERM_G
O_com06:
   cpi COMM,0x41              ; 'A' Set address
   brne O_com07               ; Go to other commands analysis (O_com07)
   rcall GET_UART             ; Get high byte of address
   mov ZH,TMP1                ; Set high byte of address
   rcall GET_UART             ; Get low byte of address
   mov ZL,TMP1                ; Set low byte of address
   lsl ZL ; Convert from word
   rol ZH ; to byte address
TERMINATE_BY_0D:              ; Commands termination by 0x0D
   ldi TMP1,0x0D
   rjmp TERM_P
O_com07:
   cpi COMM,0x50              ; 'P' Enter programming mode
   brne O_com08               ; Go to other commands analysis (O_com08)
   rjmp TERMINATE_BY_0D       ; Send termination code 13d (0D)
O_com08:
   cpi COMM,0x4C              ; 'L' Leave programming mode
```

List. 1. cd.

```
   brne O_com09                ; Go to other commands analysis (O_com09)
   rjmp TERMINATE_BY_0D        ; Send termination code 13d (0D)
O_com09:
   cpi COMM,0x73               ; 's' Read signature bytes (3 bytes)
   brne O_com0A                ; Go to other commands analysis (O_com0A)
   ldi TMP1,0x1E               ;
   rcall PUT_UART
   ldi TMP1,0x94               ;
   rcall PUT_UART
   ldi TMP1,0x03               ; 0x03 for ATmega16
   rjmp TERM_P
O_com0A:
   cpi COMM,0x64               ; 'd' Read data memory (EEPROM)
   brne O_com0B                ; Go to other commands analysis (O_com0B)
   out EEARL,ZL                ; Set address ZH:ZL
   out EEARH,ZH                ;
   sbi EECR,EERE               ; Set bit EERE - EEPROM Read Enable (in EECR register)
   in TMP1,EEDR                ; Read data
   adiw ZH:ZL,0x01             ; Increment address by 0x01
   rjmp TERM_P                 ; Send read EEPROM data to RS232
   rjmp MAIN                   ; Read next command from RS232
O_com0B:
   cpi COMM,0x52               ; 'R' Read program memory
   brne O_com0C                ; Go to other commands analysis (O_com0C)
   ldi TMP1,0x11               ;Read-while-write Section Read Enable activation
   out SPMCR,TMP1              ; within 4 cycles after SPM instruction
   spm
   lpm; Read byte (Flash) from address ZH:ZL
   mov D1,D0
   adiw ZH:ZL,0x01
   lpm    ; Read byte (Flash) from address ZH:ZL
   adiw ZH:ZL,0x01
   mov TMP1,D0
   rcall PUT_UART              ; Send low byte of program to RS232
   mov TMP1,D1
   rjmp TERM_P                 ; Send high byte of program to RS232
O_com0C:
   cpi COMM,0x63               ; 'c' Write program memory, low byte
   brne O_com0D                ; Go to other commands analysis (O_com0D)
   rcall GET_UART              ; Read byte of data
   mov AD0,TMP1                ; Store data in AD0
   mov AD1,ZL                  ; Store ZL
   andi AD1,0x7F               ; Hide most significant bit of AD1
   tst AD1                     ; Test if 7 less significant bites of address ZL are zero
   brne SETD0                  ; If they are not zero set D0(r0) by data for next SPM instruction
   ldi TMP1,0x03               ;
   out SPMCR,TMP1              ; Page erase (TMP1=0x03) within 4 cycles
   spm
PER:
   in TMP1,SPMCR               ; Set SPMCR for erasing
   sbrc TMP1,SPMEN             ; SPMEN - Store Program Memory Enable
   rjmp PER                    ; Erasing complete ?
SETD0:
   mov D0,AD0                  ; Set D0(r0) by data
   rjmp TERMINATE_BY_0D        ; Send termination code 13d (0D)
O_com0D:
   cpi COMM,0x43               ; 'C' Write program memory, high byte
   brne O_com0E                ; Go to other commands analysis (O_com0E)
   rcall GET_UART              ; Read byte of data
   mov D1,TMP1                 ; Store data in D1
   ldi TMP1,0x01               ;
   out SPMCR,TMP1              ; Write D0, D1 content to Flash address ZH:ZL
   spm
   adiw ZL,0x02                ; Increment ZH:ZL by 2
   rjmp TERMINATE_BY_0D        ; Send termination code 13d (0D)
O_com0E:
   cpi COMM,0x6D               ; 'm' Issue Page Write
   brne O_com0F                ; Go to other commands analysis (O_com0F)
   ldi TMP1,0x05               ;
   out SPMCR,TMP1              ; Page write from buffer (beginning address by ZH:ZL)
   spm
   rjmp TERMINATE_BY_0D        ; Send termination code 13d (0D)
O_com0F:
   cpi COMM,0x65               ; 'e' Chip erase
   brne O_com10                ; Go to other commands analysis (O_com10)
   rjmp EREASE
O_com10:
   cpi COMM,0x55               ; 'U' Chip erase II
   brne O_com11                ; Go to other commands analysis (O_com11)
EREASE:
   ldi ZL,0x00                 ; Set low byte of address
   ldi ZH,0x00                 ; Set high byte of address
   ldi AD1,0x00                ; Set first page number
PAGE_ER:
   ldi TMP1,0x03               ;
   out SPMCR,TMP1              ; Page erase (TMP1=0x03) within 4 cycles
   spm
   subi ZL,0x80                ; Set address of next page low byte
   sbci ZH,0xFF                ; Set address of next page high byte
   cpi AD1,0x70                ; Compare with last page number
   breq ER_COM
   inc AD1                     ; Set next page number
   rjmp PAGE_ER
ER_COM:   ; Erasing completed
   in TMP1,SPMCR
   sbrc TMP1,SPMEN             ; Erasing completed ?
   rjmp ER_COM
   rjmp TERMINATE_BY_0D        ; Send termination code 13d (0D)
O_com11:
   cpi COMM,0x78               ; 'x' Set LED
   brne O_com12                ; Go to other commands analysis (O_com12)
   rjmp TERM_G
O_com12:
   cpi COMM,0x79               ; 'y' Clear LED
   brne O_com13                ; Go to other commands analysis (O_com13)
   sbis PINB,PINB3
   rjmp LED1
   cbi PORTB,PB3               ; When PB3=1 then clear it
   rjmp TERM_G
LED1:
   sbi PORTB,PB3               ; When PB3=0 then set it
TERM_G:   ; Procedure of termination code 13d (0D) prior data receive
   rcall GET_UART              ; Receive data
```

(zawarty w pliku bsl.asm) do postaci IntelHex (bsl.hex). Podłączyć programator STK200/300 [11] do układu JTAG-a i włączyć jego zasilanie. Po skonfigurowaniu oprogramowania PonyProg [12] (kalibracja i ustawienie na „AVR micro" oraz Atmega16) „wgrać wsad" do procesora JTAG-a. Za pomocą opcji *Configuration and Security Bits* zaprogramować bity: JTAGEN, BOOTSZ1, BOOTSZ0, BOOTRST. Należy pamiętać o „odprogramowaniu" bitu CKSEL0. Jest to istotne, bowiem procesory Atmega16 mają ten bit zaprogramowany w chwili zakupu. Po tych operacjach programator nie będzie już potrzebny.

Po podłączeniu interfejsu JTAG do komputera za pomocą kabla z rys. 2 i włączeniu napięcia zasilania układu, należy uruchomić AVR Studio (wersje 4.06...09). Nie jest wymagane otwieranie żadnego projektu. W opcji Tools trzeba wybrać AVR Prog..., a po wyświetleniu się menu programatora klawiszem *Browse* należy wskazać plik upgrade.ebn znajdujący się w katalogu: ...\Program Files\Atmel\AVR Tools\JTAGICE. Uwidocznienie pliku upgrade.ebn następuje po włączeniu opcji *Pliki typu All files* (*.*). Następnie po naciśnięciu klawisza *Program* w opcji *Flash* rozpocznie się wprowadzanie „wsadu" do procesora interfejsu, sygnalizowane paleniem się diody D1 (żółtej). Wraz ze zgaśnięciem diody (koniec programowania) interfejs JTAG jest gotów do współpracy z daną wersją AVR Studio. Można to sprawdzić, bez podłączania emulowanego układu, łącząc pin 4 gniazda J4 z VCC i powtórnie włączając napięcie zasilania. Diody D1 (żółta) i D3 (zielona) zapalą się, a po wywołaniu w AVR Studio opcji *Tools* i *STK500/AVRISP/JTAGICE* zgaśnie dioda D1 i pojawi się na ekranie komunikat o wadliwym działaniu interfejsu polegającym między innymi na braku zasilania w emulowanym układzie.

Rozkład wyprowadzeń w gnieździe J4 jest zgodny ze standardem stosowanym przez Atmela w układzie uruchomieniowym STK500 i opisanym w instrukcji obsługi JTAGICE [13]. Po podłączeniu opisywanego interfejsu do emulowanego układu (zastosowałem tutaj procesor Atmega16)

i wywołaniu w AVR Studio opcji *Tools* i *STK500/AVRISP/JTAGICE* pojawi się okno umożliwiające sprawdzenie poprawności działania (programowanie, kasowanie Flasha, ustawianie *fuse* i *lock* bitów, czytanie sygnatury procesora itp.) oraz skonfigurowanie JTAGICE. Cóż, nie jest to jeszcze pełny sukces całego przedsięwzięcia, ale daje już obraz ogromnych możliwości interfejsu.

W następnym kroku, otwarcie za pomocą AVR Studio nowego projektu z pliku testowego test.asm (**list. 2**) pozwala na przetestowanie działania interfejsu JTAG. W tym celu po wejściu do opcji *Debug* należy wybrać rodzaj emulatora i procesor w *Select Platform and Device*. W naszym przypadku jest to JTAGICE i Atmega16. W opcji tej (wersje 4.09 i 4.08) zaznaczając okienko *Open Platform Options* możemy skonfigurować, między innymi, szybkość z jaką interfejs będzie komunikował się z emulowanym procesorem – częstotliwość zegara JTAG-a nie może przekraczać 1/4 częstotliwości zegara emulowanego procesora. Po dokonaniu wyboru i zamknięciu okienka naciśnięcie klawisza *Start Debugging* powoduje uruchomienie emulatora. W poprzednich wersjach AVR Studio (4.06 i 4.07) konfiguracji emulatora można dokonać dopiero po opcji *Start Debugging,* wywołując opcję *JTAG ICE Options*. Warto zaznaczyć, że wersje 4.09 i 4.08 również w tym samym miejscu posiadają powyższą opcję pozwalającą na ewentualną rekonfigurację emulatora. Jednak opcja ta zostaje „dodana" do menu debugera dopiero po wykonaniu opcji *Select Platform and Device*.

Wykonując testowy program (test.asm lub odpowiednio test.hex) rozkaz po rozkazie (kolejne kliknięcia klawisza F11), możemy obserwować na ekranie zmiany w rejestrach procesora. Praktycznie rzecz biorąc praca z JTAG-iem wygląda dokładnie tak samo, jak z Symulatorem AVR Studio. Jedyną różnicę stanowi fakt zapalania się diody D1 (żółtej) w momencie przeładowywania zawartości rejestrów z emulowanego procesora do komputera. No, może jeszcze jedno rzuca się w oczy, a mianowicie w przypadku symulatora mamy na starcie „pięknie" wyzerowane wszystkie rejestry (r0...r31), natomiast JTAG pokazuje, jak wiele jest w nich „brudu" w chwili rozpoczęcia wykonywania programu. Warto dodać, że interfejs JTAG przeładowuje zawartość wszystkich rejestrów z emulowanego układu do AVR Studio. Jednakże, podobnie jak w czasie pracy Symulatora, czerwonym kolorem podświetlone są jedynie te, których zawartość została zmodyfikowana w wyniku działania ostatniej instrukcji.

W przypadku, gdy zmieniamy wersję AVR Studio z np. 4.06 na 4.07, powinien również ulec

List. 1. cd.

```
   rjmp TERMINATE_BY_0D         ; Send termination code 13d (0D)
O_com13:
   cpi COMM,0x1B                ; „ESCAPE ?" Clear UART buffers
   brne O_com14                 ; Go to other commands analysis (O_com14)
   rjmp MAIN                    ; Read next command from RS232
O_com14:
   cpi COMM,0x6C                ; 'l' Set BLB and LB bits
   brne O_com15
   rcall GET_UART               ; Receive data
   mov D0,TMP1
; rcall BLB_SET                 ; Always deactivate Boot and Lock Bits setting
   rjmp TERMINATE_BY_0D         ; Send termination code 13d (0D)
O_com15:
   cpi COMM,0x76                ; 'v' Hardware version (1.2)
   brne O_com16
   ldi TMP1,0x61                ; '1' + 0x30
   rcall PUT_UART
   ldi TMP1,0x62                ; '2' + 0x30
   rjmp TERM_P
O_com16:
   cpi COMM,0x44                ; 'D' Write data memory (EEPROM)
   brne O_com17
   rcall GET_UART               ; Receive data
   out EEDR,TMP1                ; Set data for EEPROM writing
   out EEARL,ZL                 ; Set low byte of address
   out EEARH,ZH                 ; Set high byte of address
PEE0:
   in TMP1,SPMCR
   sbrc TMP1,SPMEN              ; SPMEN - Store Program Memory Enable
   rjmp PEE0                    ; SPM complete ?
PEE1:
   sbic EECR,EEWE
   rjmp PEE1                    ; EEPROM writing complete ?
   sbi EECR,EEMWE               ; EEPROM writing activate
   sbi EECR,EEWE                ; EEPROM write
   adiw ZH:ZL,0x01              ; Increment address by 0x01
   rjmp TERMINATE_BY_0D         ; Send termination code 13d (0D)
O_com17:
   cpi COMM,0x46                ; 'F' Read fuse and lock bits
   brne O_com18
   ldi TMP1,0xFF
   rjmp TERM_P
O_com18:
   cpi COMM,0x66                ; 'f' Write fuse bits
   brne O_com19
   rjmp TERM_G
O_com19:
   cpi COMM,0x66                ; ':' Universal command
   brne O_com1A
   rcall GET_UART               ; Get 1st byte of data
   rcall GET_UART               ; Get 2nd byte of data
   rcall GET_UART               ; Get 3rd byte of data
   ldi TMP1,0xFF
   rcall PUT_UART               ; Send reply
   rjmp TERMINATE_BY_0D         ; Send termination code 13d (0D)
O_com1A:
   cpi COMM,0x2E                ; '.' New universal command
   brne O_com1B
   rcall GET_UART               ; Get 1st byte of data
   rcall GET_UART               ; Get 2nd byte of data
   rcall GET_UART               ; Get 3rd byte of data
   rcall GET_UART               ; Get 4th byte of data
   ldi TMP1,0xFF
   rcall PUT_UART               ; Send reply
   rjmp TERMINATE_BY_0D         ; Send termination code 13d (0D)
O_com1B:
   cpi COMM,0x5A                ; 'Z' Special test command
   brne O_com1C
   rcall GET_UART               ; Get 1st byte of data
   rcall GET_UART               ; Get 2nd byte of data
   ldi TMP1,0xFF
   rjmp TERM_P                  ; Send reply
O_com1C:
   ldi TMP1,0x3F                ; '?'
   rjmp TERM_P                  ; Send „I beg your pardon ?"
      ;
GET_UART:                       ; UART receiving
   sbis UCSRA,RXC               ; RXC - USART Receive Complete
   rjmp GET_UART                ; UART ready for receiving ?
   in TMP1,UDR                  ; Receive data from RS232
   ret
      ;
PUT_UART:                       ; UART sending
   sbis UCSRA,UDRE              ; UDRE - USART Data Register Empty
   rjmp PUT_UART                ; UART ready for sending ?
   out UDR,TMP1                 ; Send data to RS232
   ret
      ;
BLB_SET:  ; Lock bits setting
   ldi TMP1,0x09                ; Set for BLB activation
   out SPMCR,TMP1               ; BLB activate
   spm    ; BLB activate
BLB_SET1:                       ; Monitoring completion of Writing
   in TMP1,SPMCR
   sbrc TMP1,SPMEN              ; SPMEN - Store Program Memory Enable
   rjmp BLB_SET1                ; SPM complete ?
   ret
;
```

List. 2
```
; Program Test.asm do testowania działania
;interfejsu JTAG
.include „m16def.inc”
   .cseg
   .org 0x0000
.def   TMP1 = r16
   cli
   ldi TMP1,0xFF
   out DDRB,TMP1
   ldi TMP1,0x01
PETLA:
   out PORTB,TMP1
   rol TMP1
   rjmp   PETLA
```

aktualizacji program w interfejsie JTAG. Przy wywołaniu w nowym AVR Studio opcji *Tools* i *STK500/AVRISP/JTAGICE* pojawi się okienko informujące o konieczności takiej zmiany (musi być podłączony emulowany układ!). Program BSL został przygotowany w taki sposób, że postępowanie zgodne z pojawiającym się na ekranie opisem przyniesie pożądany efekt. Oczywiście, uaktualnienie zachodzić będzie poprawnie pomiędzy kolejnymi wersjami 4.06...4.09. Warto dodać, że w przypadku uruchomienia AVR Studio w wersji np. 4.06 z interfejsem JTAG „załadowanym” zawartością pliku upgrade.ebn z wersji 4.09, program ten nie domaga się „obniżenia” wersji programu JTAG-a, ale poprawnie współpracuje.

Czas pokaże, czy w przypadku następnych wersji AVR Studio aktualizacja przebiegnie również poprawnie. Jeśli pojawią się błędy, czy to we wprowadzaniu „wsadu” do procesora JTAG-a, czy też w działaniu interfejsu, trzeba będzie albo zakasać rękawy i udoskonalić program BSL, albo korzystać z dotychczasowej wersji AVR Studio. Wybór pozostawiam Czytelnikowi.

Uwagi końcowe

Ze względu na szczupłość miejsca uważam, że nie warto tutaj prezentować gotowych przepisów dotyczących posługiwania się interfejsem JTAG. Wszelkie informacje na ten temat zamieścił Atmel na stronach „AVR JTAG ICE User Guide” [13]. Chociaż na jedną rzecz chciałbym zwrócić uwagę. Mianowicie, jeśli w tekście znajduje się zapis, że coś może być (*something may be*), to z bardzo wysokim prawdopodobieństwem należy przyjąć, iż na pewno nie jest. Na przykład opcjonalność linii sterującej nSRST świadczy o tym, że ona nie działa. Łatwo można sprawdzić, że emulator po wykonaniu hardware'owego zerowania emulowanego układu idzie w przysłowiowe maliny. Nieco dalej autor „AVR JTAG ICE User Guide” wyjaśnia, że działanie tej linii nie jest jednak konieczne do poprawnej emulacji. Biorąc pod uwagę to, że zarówno AVR Studio, jak też i współpracujące z nim urządzenia są stale udoskonalane, powinniśmy wybaczyć komercyjne wstawki w instrukcjach użytkownika, zachwalające potencjalną użyteczność oferowanych produktów.

Jak informuje Atmel, w czasie pracy z interfejsem JTAG możliwe jest uszkodzenie programu znajdującego się w procesorze. Nie oznacza to, że urządzenie uległo całkowitemu zniszczeniu, ale że istnieje możliwość „reanimowania” interfejsu poprzez powtórne „wgranie wsadu” z pliku upgrade.ebn. W tym celu należy powtórzyć wykonanie operacji „wgrywania” opisanej wyżej. Należy zaznaczyć, że powtórne „wgrywanie” programu BSL nie jest niezbędne.

Oprócz automatycznej aktualizacji „wsadu” procesora w interfejsie JTAG możliwe jest również jego „ręczne” uaktualnienie. W tym celu należy postępować zgodnie z opisaną już procedurą albo skorzystać z opisu zamieszczonego w literaturze [14]. Warto zaznaczyć, że funkcjonalna zgodność programu BSL z firmowym Boot Strap Loaderem z interfejsu JTAGICE pozwala na bezproblemowe stosowanie wielu (a może wszystkich?) literaturowych „receptur” dotyczących tegoż urządzenia.

Testując skonstruowany prototyp interfejsu stwierdziłem, że bywają sytuacje (rzadko, ale jednak), gdy po wywołaniu opcji *Start Debugging* w oknie *Messages* pojawiają się Uwagi (oznaczone żółtą kropką) lub informacje o błędach (oznaczone czerwoną kropką) dotyczące niemożności podjęcia współpracy emulatora z AVR Studio. Najczęstszą przyczyną tego zjawiska są „słabe” połączenia przewodów z gniazdami i niska jakość użytych kabli (zarówno tego od RS232 jak i „tasiemki”). Po sprawdzeniu połączeń, do podjęcia poprawnej współpracy konieczne jest powtórne uruchomienie (włączenie zasilania) interfejsu, a także powtórne wywołanie AVR Studio. Należy zwrócić uwagę, że fabryczny JTAGICE nie jest wolny od tej wady. Wystarczy przeczytać *Troubleshooting Guide* w Helpie do AVR Studio.

WYKAZ ELEMENTÓW

Rezystory
R1: 47kΩ
R2: 470Ω
R3: 36kΩ
R4, R6, R8, R13...R28: 10kΩ
R5: 1kΩ
R7: 22kΩ
R9: 4,7kΩ
R10: 33kΩ
R11: 560Ω
R12: 150kΩ
R29...R33: 56Ω

Kondensatory
C1...C4: 10 μF/16V tantalowy
C5, C6, C8, C11...C13, C15, C16: 0,1μF/50V
C7: 1μF/25V
C9, C10: 22pF/50V
C14: 1nF/50V

Półprzewodniki
D1: LED żółta LG-150Y
D2: LED czerwona LG-150UR
D3: LED zielona LG-150UG
D4: SM4007
T1...T3: BC846
T4: BC856
U1: MAX232
U2: Atmega16
U3: SN74HC244
U4: LM7805

Różne:
X1: rezonator kwarcowy 7,3728 MHz
J1: gniazdo DB09RA/M
J2: gniazdo zasilające Terminal Block 1x2
J3: gniazdo IDC6 (męskie) 2x3
J4: gniazdo IDC10 (męskie) 2x5
JP: gniazdo SIP2 lub IDC2 (męskie) 1x2
L1: 10 μH
Podstawka DIP16 (opcjonalnie)
Podstawka DIP20 (opcjonalnie)
Podstawka DIP40 (opcjonalnie)
Śruba M3x6 z nakrętką sześciokątną
Zwora SIP2

Ograniczenia

Przejdźmy teraz do opisu hardware'owych ograniczeń opisanego urządzenia. Prąd pobierany z interfejsu do emulowanego układu nie powinien przekraczać około 200 mA. W innym przypadku stabilizator LM7805 będzie się dość mocno grzał i może (przy długo-

trwałej pracy) ulec uszkodzeniu. Pewną poprawę może dać zastosowanie radiatora, ale ze względu na szczupłość miejsca nie może on być zbyt duży.

W przypadku zasilania emulowanego układu z oddzielnego źródła, jego napięcie nie może być zbyt niskie tak, aby bufor U3 był w stanie prawidłowo odróżniać stany „0" i „1". Zależy to w dużej mierze od egzemplarza układu scalonego SN74HC244. Skonstruowany prototyp pracował poprawnie w zakresie napięcia zasilania emulowanego układu 3,3÷6 V. Warto podać, że JTAGICE Atmela pracuje w zakresie napięć 3,3÷5,5 V.

Gdy interfejs JTAG zasilany jest z emulowanego układu, jego napięcie zasilania nie powinno być niższe od około 4 V. Poziom ten jest uwarunkowany poprawną pracą układów U1...U3. Zamieniając układy scalone z MAX232 na MAX3232 (konieczna zmiana kondensatorów C1...C4), Atmega16 na Atmega16L i SN74HC244 na SN74AHC244, można przesunąć dolny zakres napięcia do około 3,3 V, ale zdobycie w polskich warunkach ostatniego z układów w obudowie DIP jest trudne. Warto zaznaczyć, że w przypadku zasilania interfejsu z emulowanego układu, napięcie zasilania pokazywane przez AVR Studio może być zafałszowane.

Przewód łączący gniazdo J4 (gniazdo interfejsu JTAG zgodne ze standardem Atmela) to zwykła dziesięciożyłowa „tasiemka". W czasie testowania prototypu stwierdziłem, że jej długość powinna zawierać się w granicach 10÷30 cm. Krótsze połączenia są niewygodne, a dłuższe np. 50 cm prowadzą do problemów w komunikacji pomiędzy interfejsem i emulowanym układem. Kłopoty te są szczególnie widoczne przy wysokich częstotliwościach zegara JTAG-a (np. 2 MHz).

W przypadku AVR Studio wersje 3.55 i 3.56 i opisanego interfejsu możliwe jest jedynie korzystanie z „opcji" programowania procesorów wyposażonych w JTAG-a. Emulacja w układzie procesorów nie jest możliwa! Prawdopodobną przyczyną jest to, że wyżej wymienione wersje były przygotowane w czasach, gdy interfejsy JTAG produkowano w oparciu o procesory Atmega163 (obecnie wycofane z produkcji). Przypuszczam, że wersja hardware'u identyfikowana jest w oparciu o typ zastosowanego procesora.

Jak widać moje pragnienie sprzed kilku tygodni zostało zaspokojone. Czy warto było wkładać tyle pracy w jego urzeczywistnienie? Osądźcie to sami.

Jacek A. Michalski
SP5IMO, WX3V

Wzory płytek drukowanych w formacie PDF są dostępne w Internecie pod adresem: ***pcb.ep.com.pl*** *oraz na płycie CD-EP6/2004B w katalogu* ***PCB****.*

Literatura

1. http://seguro.pl/cgibin/shop?show=P2201&sort=id&sid=08e3a8ca
2. http://www.olimex.com/dev/index.html
3. http://www.olimex.com/dev/avr-jtag-upgrade.html
4. http://jtag-avr.port5.com/ (strona w obecnej chwili już niedostępna)
5. http://avr.openchip.org/bootice/old_index.html
6. http://www.hw.cz/out.php3?www.mcu.cz/modules/news/article.php?storyid=353
7. Fred Eady, Still swimming with the STK500 – Onto the JTAGICE, Circuit Cellar – The Magazine for Computer Applications, 143 (czerwiec 2002), 1-6, (http://www.circuitcellar.com),
8. http://www.atmel.com/dyn/resources/prod_documents/DOC0943.PDF, Nota aplikacyjna AVR910 z programem AVR910.asm, In System Programming, Atmel,
9. http://www.atmel.com/dyn/resources/prod_documents/doc1644.pdf, Nota aplikacyjna AVR109 ze zbiorem AVR109.zip, Self-Programming, Atmel,
10. http://www.avrfreaks.net/Tools/showtools.php?ToolID=328, Nota projektowa, AVR Boot Loader, AVR Freaks Net,
11. AVT-871, Zbigniew Raabe, Programator procesorów AVR do kompilatora BASCOM AVR, Elektronika Praktyczna, 06/2000, str. 55...58,
12. http://www.lancos.com/prog.html, Program PonyProg i schemat programatora STK200/300,
13. http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2737, AVR JTAG ICE User Guide, Atmel,
14. http://www.avrfreaks.net/Tools/showtools.php?ToolID=248, Nota projektowa, Manual JTAG ICE Firmware Upgrade, AVR Freaks Net