**Listing 1. Przykładowa procedura obsługi przerwania od WDG**

/\* "czas" - define czau zadziałania WDG, dostępne wartości:

WDTO\_30MS, WDTO\_60MS, WDTO\_120MS, WDTO\_250MS, WDTO\_500MS, WDTO\_1S, WDTO\_1S

\*funkcja - adres funkcji wywołanej przed wyjśiem z przerwania od WDG

W zmiennej globalnej "adrCallWdg" znajduje sie adres z którego nastąpiło przerwanie WDG. Jeśli "funkcja" = 0 to skok nie będzie wykonany

void InitWdgI**(** byte czas**,** long **\***funkcja **)**

**{**

byte cSREG**;**

cSREG **=** SREG**;**

cli**();**

InitWdg**(** czas **);**

#if defined(\_\_AVR\_ATmega640\_\_) || defined(\_\_AVR\_ATmega1280\_\_) || defined(\_\_AVR\_ATmega1281\_\_) || defined(\_\_AVR\_ATmega2560\_\_) || defined(\_\_AVR\_ATmega2561\_\_)

WDTCSR **|=** **(**1**<<**WDIE**);** // Aby ustawić WDIE nie może być ustawiony bit WDTON w fuses

adrUserCallWdg **=** funkcja**;**

adrUserCallWdg **<<=** 1**;**

#else

#error "Tan procesior nie generuje IRQ od WDG. Uzyj funkcji 'InitWdg()'."

#endif

SREG **=** cSREG**;**

**}**

// "czas" - define czau zadziałania WDG, dostępne wartości:

void InitWdg**(** byte czas **)**

**{**

byte cSREG**;**

cSREG **=** SREG**;**

cli**();**

wdt\_enable**(** czas **);** // Ustawienie WDG (IRQ muszą byc wyłączone)

SREG **=** cSREG**;**

**}**

// WDIE jest automatycznie kasowany, dlatego następne zadziałanie WDG wywoła reset

#if defined(\_\_AVR\_ATmega164\_\_) || defined(\_\_AVR\_ATmega640\_\_) || defined(\_\_AVR\_ATmega1280\_\_) || defined(\_\_AVR\_ATmega1281\_\_) || defined(\_\_AVR\_ATmega2560\_\_) || defined(\_\_AVR\_ATmega2561\_\_)

#ifdef WDGI\_NAKED

ISR**(**WDT\_vect**,** ISR\_NAKED **)**

#else

SIGNAL**(** WDT\_vect **)**

#endif

**{**

byte **\***ptr**,** ofs**=**0**,** adr**[**3**];**

word stack**,** cmd**;**

#if defined(\_\_AVR\_ATmega2560\_\_) || defined(\_\_AVR\_ATmega2561\_\_)

long vect**;** // Ponad 128kB adres jest 3-bajtowy

#else

word vect**;**

#endif

#ifndef WDGI\_NAKED

nop**();** // Pierwszy rozkaz procedury (kod maszynowy rozkazu = $0000)

#endif

vect **=** WDT\_vect**;** // Adres wektora przerwania od WGD

vect **<<=** 1**;** // Adres w słowach więc mnożymy przez 2

#ifndef WDGI\_NAKED

//×××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

**for(** byte x**=**0**;** x**<**32**;** x**++** **)** // Liczymy liczbe rozkazów push

**{**

#if defined(\_\_AVR\_ATmega1280\_\_) || defined(\_\_AVR\_ATmega1281\_\_) || defined(\_\_AVR\_ATmega2560\_\_) || defined(\_\_AVR\_ATmega2561\_\_)

cmd **=** pgm\_read\_word\_far**(**vect**);**// Ponad 64KB "far"

#else

cmd **=** pgm\_read\_word**(** vect **);**

#endif

vect **+=**2**;**

**if** **(** **!**cmd **)** **break;** // Jeśli kod rozkazu "nop"

**if** **(** **((**cmd**&**0xFF00**)==**0x9200**)** **||** // Jeśli rozkaz push R0..15

**((**cmd**&**0xFF00**)==**0x9300**)** **)** // Jeśli rozkaz push R16..31

**{**

ofs**++;**

**}**

**}**

#endif

stack **=** SP **+** ofs**;**

stack**++;** // Stos wskazuje pierwszy wolny bajt więc zwiększamy o 1

ptr **=** stack**;**

adr**[**0**]** **=** **\***ptr**++;**

adr**[**1**]** **=** **\***ptr**++;**

adr**[**2**]** **=** **\***ptr**++;**

#if defined(\_\_AVR\_ATmega2560\_\_) || defined(\_\_AVR\_ATmega2561\_\_)

adrCallWdg **=** **((**adr**[**0**]** **<<** 8**)** **<<**8**);** // Adres powrotu dla cpu ponad 128kB

adrCallWdg **|=** adr**[**1**]** **<<** 8**;**

adrCallWdg **|=** adr**[**2**];**

#else

adrCallWdg **=** adr**[**0**]** **<<** 8**;** // Adres powrotu dla cpu do 128kB

adrCallWdg **|=** adr**[**1**];**

#endif

adrCallWdg **\*=** 2**;** //PC wskazuje nr słowa więc adres jest 2 razy większy

//×××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

**if** **(** adrUserCallWdg **)**

**{**

long adrCall **=** adrUserCallWdg **>>** 1**;**

**(\*((**void**(\*)(**void**))**adrCall**))();** // CALL do funkcji o adresie "adrUserCallWdg"

**}**

#ifdef WDGI\_NAKED

**while(** true **);**

#endif

**}**

#endif

//Deklaracje:

#define DEF\_RST\_SOFT 0x534f4654

#define DEF\_RST\_UNKOWN (DEF\_RST\_SOFT ^ 0xaa55a55a)

#define WDGI\_NAKED //Po obsłudze IRQ wykonanie resetu (krótszy kod).

long IdSoftReset NOINIT**;**

long adrUserCallWdg**=**0**,** adrCallWdg**=**0**;**

extern void InitWdgI**(** byte czas**,** long **\***funkcja **);** // Init IRQ od WDG

extern void InitWdg**(** byte czas **);** // Init WDG

#define SoftReset() IdSoftReset=DEF\_RST\_SOFT; IdSoftReset=DEF\_RST\_UNKOWN; while(true);

**Listing 2. Procedura konwersji zmiennej long na łańcuch znaków**

void sPrintfLongDec**(** char **\***str**,** long dec **)**

**{**

void sPrintfLongDec**(** char **\***str**,** long dec **)**

**{**

ulong w**;**

byte z **=** TRUE**;**

**if** **(** dec **&** 0x80000000 **)** // Jeśli liczba ujemna

**{**

dec **=** **~**dec **+** 1**;**

**\***str**++** **=** '-'**;**

**}**

// 2147483647 (0x7FFFFFFF)

w **=** dec **/** 1000000000**;**

dec **%=** 1000000000**;**

**if** **(** w **||** **!**z **)**

**{**

**\***str**++** **=** **(** w **+** '0' **);**

z **=** FALSE**;**

**}**

w **=** dec **/** 100000000**;**

dec **%=** 100000000**;**

**if** **(** w **||** **!**z **)** **{**

**\***str**++** **=** **(** w **+** '0' **);**

z **=** FALSE**;**

**}**

w **=** dec **/** 10000000**;**

dec **%=** 10000000**;**

**if** **(** w **||** **!**z **)** **{**

**\***str**++** **=** **(** w **+** '0' **);**

z **=** FALSE**;**

**}**

w **=** dec **/** 1000000**;**

dec **%=** 1000000**;**

**if** **(** w **||** **!**z **)** **{**

**\***str**++** **=** **(** w **+** '0' **);**

z **=** FALSE**;**

**}**

w **=** dec **/** 100000**;**

dec **%=** 100000**;**

**if** **(** w **||** **!**z **)** **{**

**\***str**++** **=** **(** w **+** '0' **);**

z **=** FALSE**;**

**}**

w **=** dec **/** 10000**;**

dec **%=** 10000**;**

**if** **(** w **||** **!**z **)** **{**

**\***str**++** **=** **(** w **+** '0' **);**

z **=** FALSE**;**

**}**

w **=** dec **/** 1000**;**

dec **%=** 1000**;** f **(** w **||** **!**z **)** **{**

**\***str**++** **=** **(** w **+** '0' **);**

z **=** FALSE**;**

**}**

w **=** dec **/** 100**;**

dec **%=** 100**;**

**if** **(** w **||** **!**z **)** **{**

**\***str**++** **=** **(** w **+** '0' **);**

z **=** FALSE**;**

**}**

w **=** dec **/** 10**;**

dec **%=** 10**;**

**if** **(** w **||** **!**z **)** **{**

**\***str**++** **=** **(** w **+** '0' **);**

z **=** FALSE**;**

**}**

**\***str**++** **=** **(** dec **+** '0' **);** **\***str **=** 0**;**

**}**

void sPrintfWordDec**(** char **\***str**,** word dec **)**

**{**

word w**;**

byte z **=** TRUE**;**

w **=** dec **/** 10000**;** dec **%=** 10000**;** **if** **(** w **)** **{** **\***str**++** **=** **(** w **+** '0' **);** z **=** FALSE**;** **}**

w **=** dec **/** 1000**;** dec **%=** 1000**;** **if** **(** w **||** **!**z **)** **{** **\***str**++** **=** **(** w **+** '0' **);** z **=** FALSE**;** **}**

w **=** dec **/** 100**;** dec **%=** 100**;** **if** **(** w **||** **!**z **)** **{** **\***str**++** **=** **(** w **+** '0' **);** z **=** FALSE**;** **}**

w **=** dec **/** 10**;** dec **%=** 10**;** **if** **(** w **||** **!**z **)** **{** **\***str**++** **=** **(** w **+** '0' **);** z **=** FALSE**;** **}**

**\***str**++** **=** **(** dec **+** '0' **);** **\***str **=** 0**;**

**}**

void PrintLongHex**(** long hex **)**

**{**

PrintWordHex**(** **((**hex**>>**8**)>>**8**)** **);**

PrintWordHex**(** hex **);**

**}**

void PrintWordHex**(** word hex **)**

**{**

PrintByteHex**(** hex **>>** 8 **);**

PrintByteHex**(** hex **);**

**}**

void PrintByteHex**(** byte hex **)**

**{**

static byte hexmem**;**

hexmem**=**hex**;**

PrintNibleHex**(** hex **>>** 4 **);**

PrintNibleHex**(** hex **);**

**}**

void PrintNibleHex**(** byte hex **)**

**{**

hex **&=** 0x0F**;**

static byte nibmem**;**

nibmem**=**hex**;**

**if** **(** hex **<=** 9 **)** Usart1\_Transmit**(** hex **+** '0' **);**

**else** Usart1\_Transmit**(** hex**-**10 **+** 'A' **);**

**}**

**Listing 3. Przykładowy sposób określenia zużycia pamięci**

// Sekcja ".initX" (ZERO zainicjalizowane)

unsigned char DnoStosu NOINIT**;** // Kontrola stosu

void ClrIntRam**(**void**)** \_\_attribute\_\_ **((**naked**))** \_\_attribute\_\_ **((**section **(**".init3"**)));**

void ClrIntRam**(**void**)**

**{**

unsigned char **\***AdrRam**;**

// Zapisujemy od ostatniej zajetej komórki ram do wierzchołka stosu -32 bajty rezerwy

**for** **(**AdrRam**=&**DnoStosu**;** AdrRam **<** **(**unsigned char**\*)**RAMEND**-**32**;** AdrRam**++)** // Wpisanie do IntRam $FF

**{**

**\***AdrRam **=** 0xFF**;**

**}**

DnoStosu **=** '@'**;** // Kontrola stosu

**}**

//Przykład sprawdzania zajętości pamięci. Najlepiej wywoływać cyklicznie, na przykład co 100ms w pętli głównej programu:

void TestStosu**()**

**{**

unsigned char **\***AdrRam**;**

unsigned int cnt**=**0**;**

// Sprawdzamy od ostatniej zajetej komórki ram do wierzchołka stosu -32 bajty rezerwy

**for** **(**AdrRam**=(&**DnoStosu**)+**1**;** AdrRam **<** **(**unsigned char**\*)**RAMEND**-**32**;** AdrRam**++)** // Wpisanie do IntRam $FF

**{**

cnt**++;**

**if** **(\***AdrRam **!=** 0xFF**)**

**{**

FreeRam **=** cnt**;** // Wolny obszar RAM'u

**if** **(**FreeRam **<** 32 **)**

**{** // Jeśli za mały obszar to generuj błąd

PrintError**(** ERR\_STOS **);**

LedErrorOn**();**

sprintf\_P**(**str**,** PSTR**(**" Free $%04x RAM"CRLF**),** FreeRam**);** PrintString**(**str**);**

**}**

**return;**

**}**

**}**

**}**

**Listing 4. Procedura do szacowania częstotliwości taktowania**

/\* Oszacowanie częstotliwości taktowania mikrokontrolera. Funkcja musi być

umieszczona jako pierwsza w kodzie (za main). W innych miejscach może działać

nieprawidłowo lub niepotrzebnie wydłużyc start programu (pomiar musi

spowodować zadziałanie WDG. \*/

word fcpuCalculate**()**

**{**

word fcpu**;**

**if** **(** PomiarFCLK **!=** DEFPWRRST **)** // Jeśli pomiar nie był wykonany

**{**

PomiarFCLK **=** DEFPWRRST**;**

f\_us **=** 0**;**

cli**();** // Koniecze jeśli funkcja wywołania po włączeniu IRQ

wdt\_enable**(** WDTO\_30MS **);** // Ustawienie WDG (IRQ muszą byc wyłączone)

wdt\_reset**();** // Koniecze jeśli funkcja wywołania gdy WDG był już w użyciu lub włączony bit WDTON w fuses

**while(** true **)**

**{**

f\_us**++;** // Zmienną zwiększamy co 1ms do czasu aż WDG zrestartuje mikrokontroler

\_delay\_ms**(**1**);**

**}**

**}**

**else** // Pomiar zakończony, interpretuj wynik

**{**

// FCPU\_TPOM - czas pomiaru WDTO\_30MS (uwagi w pliku ".h"

fcpu **=** **(**long**)**F\_CPU**/**1000 **\*** f\_us **/** FCPU\_TPOM**;**

// Procentowa odchyłka częstotliwości

fcpuDeviation **=** **((**long**)**fcpu **\*** 100 **/** **(**F\_CPU**/**1000**))** **-** 100 **;**

**}**

**return(** fcpu **);**

**}**

//Definicje:

byte volatile f\_us NOINIT**;**

int fcpuDeviation**;**

long PomiarFCLK NOINIT**;**