|  |  |
| --- | --- |
|  |  |
|  | Politechnika Wrocławska  Artur Zochniak  nr albumu 184725  Projekt do laboratorium kursu  Podstawy Technik Mikroprocesorowych  prowadzący: dr inż. Jacek Mazurkiewicz |

|  |
| --- |
| **[Dekoder kodu morse’a]** |
| Projekt zrealizowany na mikrokontrolerze ARM STM32F103VC (z architekturą Cortex-m3) |



# Wstęp

Celem projektu było stworzenie oprogramowania na 32-bitowy mikrokontroler ARM umożliwiającego dekodowanie kodu morsa.

Odkryty przez Samuela Morse'a i Alfreda Vaila sposób reprezentacji alfabetu, cyfr i znaków specjalnych za pomocą dźwięków, błysków światła, impulsów elektrycznych lub znaków popularnie zwanych kreską i kropką. Wszystkie znaki reprezentowane są przez kilkuelementowe serie sygnałów – krótkich (kropek) i długich (kresek). Kreska powinna trwać co najmniej tyle czasu, co trzy kropki. Odstęp pomiędzy elementami znaku powinien trwać jedną kropkę. Odstęp pomiędzy poszczególnymi znakami – jedną kreskę. Odstęp pomiędzy grupami znaków – trzy kreski.[[1]](#footnote-1)

Tabela 1. reprezentacja alfabetu łacińskiego w alfabecie Morse'a

|  |  |  |  |
| --- | --- | --- | --- |
| **Litery alfabetu łacińskiego** | | | |
| Litera | Kod | **Litera** | **Kod** |
| A | • — | N | — • |
| B | — • • • | O | — — — |
| C | — • — • | P | • — — • |
| D | — • • | Q | — — • — |
| E | • | R | • — • |
| F | • • — • | S | • • • |
| G | — — • | T | — |
| H | • • • • | U | • • — |
| I | • • | V | • • • — |
| J | • — — — | W | • — — |
| K | — • — | X | — • • — |
| L | • — • • | Y | — • — — |
| M | — — | Z | — — • • |

# Użyte oprogramowanie

Do stworzenia programu użyto:

* CooCox CoIDE,
* biblioteki do obsługi wyświetlacza LCD TFT HY32D zgodnego ze sterownikiem SSD1289,
* biblioteki GLCD uławiającej obsługę wyświetlacza,
* programów przykładowych dostarczonych z środowiskiem CooCox IDE,
* schematu układu (zaznajomienie się z podłączeniem portów do wyjść procesora),
* datasheet firmy STMicroelectronics dla układów STM32F103xC STM32F103xD STM32F103xE.

# Reprezentacja alfabetu Morse’a w pamięci

Alfabetu Morse’a w pamięci przechowywany jest w postaci dwóch tablic. W jeden widnieje reprezentacja sygnałowa, w drugim odpowiedniki z alfabetu łacińskiego.

Pojedyncza jedynka ‘1’ oznacza nadawanie sygnału przez jeden okres. Podkreślenie ‘\_’ oznacza nie nadawanie sygnału przez jeden okres. Zgodnie z konwencją po nadaniu każdej litery należy odczekać 3 okresy (co reprezentują trzy znaki

podkreślenia na końcu definicji każdego zapisu litery w alfabecie sygnałowym).

|  |  |
| --- | --- |
| const char \*codes[29]=  {"1\_111\_\_\_\0",  "111\_1\_1\_1\_\_\_\0",  "111\_1\_111\_1\_\_\_\0",  "111\_1\_1\_\_\_\0",  "1\_\_\_\0",  "1\_1\_111\_1\_\_\_\0",  "111\_111\_1\_\_\_\0",  "1\_1\_1\_1\_\_\_\0",  "1\_1\_\_\_\0",  "1\_111\_111\_111\_\_\_\0",  "111\_1\_111\_\_\_\0",  "1\_111\_1\_1\_\_\_\0",  "111\_111\_\_\_\0",  "111\_1\_\_\_\0",  "111\_111\_111\_\_\_\0",  "1\_111\_111\_1\_\_\_\0",  "111\_111\_1\_111\_\_\_\0",  "1\_111\_1\_\_\_\0",  "1\_1\_1\_\_\_\0",  "111\_\_\_\0",  "1\_1\_111\_\_\_\0",  "1\_1\_1\_111\_\_\_\0",  "1\_111\_111\_\_\_\0",  "111\_1\_1\_111\_\_\_\0",  "111\_1\_111\_111\_\_\_\0",  "111\_111\_1\_1\_\_\_\0",  "111\_\_\_\0",  "\0"}; | char letters[27]={  'A',  'B',  'C',  'D',  'E',  'F',  'G',  'H',  'I',  'J',  'K',  'L',  'M',  'N',  'O',  'P',  'Q',  'R',  'S',  'T',  'U',  'V',  'W',  'X',  'Y',  'Z',  ' ',  0}; |

# Implementacja programowa

## Odmierzanie okresów

odmierzanie czasu (tożsame z odmierzającym czas każdego okresu paskiem). Po odmierzeniu każdego odcinka czasu następuje sprawdzenie czy wciśnięto przycisk oznaczający nadawanie czy nie.

//returns if there was new turn

int proceedBar(int isPressed) {

static int barx = 144;

static int bary = 4;

static const int barWidth = 90;

static int bar=0;

static int which=0;

static int first=1;

int h=5;

int i;

if (bar>barWidth/2 || first) {

bar = 0;

for(i=h;i>=0; --i)

LCD\_DrawLine(barx, bary+i, barx + barWidth, bary+i, Green);

if(first) {

first=0;

return 0;

} else {

return 1;

}

} else {

for(i=h;i>=0; --i) LCD\_SetPoint(barx + bar, bary+i, Black);

for(i=h;i>=0; --i) LCD\_SetPoint(barx + barWidth-bar, bary+i, Black);

++bar;

return 0;

}

}

Funkcja, której każde wywołanie odpowiada osobnej iteracji pętli, z tym, że praca pętli została rozbita na osobne iteracje. Podejście takie ułatwia wychwycenie momentu kiedy skończył się czas aktualnego okresu i należy podjąć decyzję czy było to nadawanie sygnału.

# Wprowadzanie danych

Sygnały nadawane są przez przycisk USER\_KEYB na obrzeżu obudowy.

Jego inicjalizacja wygląda następująco:

void GPIO\_init() {

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB, ENABLE);

GPIO\_InitStructure.GPIO\_Pin = USER\_KEYB;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IN\_FLOATING;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_Init(USER\_KEYB\_PORT, &GPIO\_InitStructure);

## Dopasowywanie ciągów sygnałów do ich odpowiedników

Mając ciąg wejściowy sygnałów należy znaleźć odpowiedniki ich w alfabeie łacińskim. Realizuje to funkcja działająca w czasie rzeczywistym i wyświetlająca efekty przypasowania na bieżąco. Za każdym wprowadzonym sygnałem zawęża ona zbiór wyświetlanych symboli oraz pokazuje na bieżąco użytkownikowi czy ma nadawać czy milczeć, aby odbiorca zrozumiał to jako dany symbol.

int printTable(char \*morseBuf) {

for(i=0; i<26; ++i)

{

int x=(i/14)\*120;

int y=(i%14)\*15+20;

buf[0]=letters[i];

buf[1]=0;

int res=strlen(morseBuf)==0 || strncmp(morseBuf, codes[i], strlen(morseBuf))==0;

int ncmp=(strcmp(morseBuf, codes[i])==0);

if(res && ncmp) ok=i; //i-ty element jest rozwiazaniem

if(res && ok<0) ok=-1; //jest nadal szansa

int werdykt=(res!=0) || (ncmp!=0);

…

## Sygnalizacja pracy

Użytkownik jest cały czas informowany o tym, czy mikrokontroler działa dzięki pracy dwóch diod sterowanych przerwaniami z licznika czasowego.

Wpisanie informacji o generowanych przerwaniach licznika do tablicy przerwań (Nested Vectored Interrupt Controller (NVIC)).

void NVIC\_conf() {

NVIC\_InitStructure.NVIC\_IRQChannel = TIM1\_CC\_IRQn;

NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority = 0;

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 1;

NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE;

NVIC\_Init(&NVIC\_InitStructure);

}

Konfiguracja licznika czasowego wygląda następująco:

void TIM\_Conf(void)

{

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseStructure;

TIM\_OCInitTypeDef TIM\_OCInitStructure;

TIM\_TimeBaseStructure.TIM\_Period = 65535;

TIM\_TimeBaseStructure.TIM\_Prescaler = 1440/100;//72Mhz/1440\*100=5000kHz

TIM\_TimeBaseStructure.TIM\_ClockDivision = 0;

TIM\_TimeBaseStructure.TIM\_CounterMode = TIM\_CounterMode\_Up;

TIM\_TimeBaseInit(TIM1, &TIM\_TimeBaseStructure);

TIM\_OCInitStructure.TIM\_OCMode = TIM\_OCMode\_Timing;

TIM\_OCInitStructure.TIM\_OutputState = TIM\_OutputState\_Enable;

TIM\_OCInitStructure.TIM\_Pulse = CC1;

TIM\_OCInitStructure.TIM\_OCPolarity = TIM\_OCPolarity\_High;

TIM\_OC1Init(TIM1, &TIM\_OCInitStructure);

TIM\_OC1PreloadConfig(TIM1, TIM\_OCPreload\_Disable);

TIM\_ITConfig(TIM1, TIM\_IT\_CC1, ENABLE);

TIM\_Cmd(TIM1, ENABLE);

}

Zdefiniowanie obsługi przerwania:

void TIM1\_CC\_IRQHandler() {

static int counter=1;

if(TIM\_GetITStatus(TIM1, TIM\_IT\_CC1) != RESET) {

TIM\_ClearITPendingBit(TIM1, TIM\_IT\_CC1);

if(counter%70==0)

GPIO\_Blink();

else if(counter%72==0) {

GPIO\_Blink();

}

if (counter%80==0 || counter%88==0) {

LED1\_PORT->ODR^=LED1\_PIN;

}

if(counter==88) counter=1;

counter++;

}

}

# Korzystanie z programu

W celu nadania danego symbolu należy obserwować znak znajdujący się po prawej od litery w alfabecie łacińskim. Możliwe są dwie reprezentacje:

* ‘\_’ – podkreślenie - oznacza milczenie przez najbliższy okres,
* ‘1’ – cyfra jeden – oznacza nadawanie przez najbliższy okres.

Po poprawnym nadaniu wszystkich symboli na dole ekranu pojawi się litera w notacji alfabetu łacińskiego. Obok niej podczas dalszej będą pojawiać się kolejne litery, tworząc w ten sposób przekaz.

Kiedy użytkownik popełni błąd – dotychczasowa praca zostaje usunięta i użytkownik otrzymuje ponowną szansę nadania symbolu.

# Bibliografia

1. Krzysztof Paprocki „Mikrokontrolery STM32 w praktyce”, Wydawnictwo BTC, Legionowo 2009, ISBN 978-83-60233-52-8
2. http://en.radzio.dxp.pl/stm32vldiscovery/lesson1,first,program,blinking,leds.html (dostępne 5. maja 2012)
3. http://www.elektroda.pl/rtvforum/viewtopic.php?t=2065241
4. http://en.sourceforge.jp/projects/sfnet\_tpflowmeter/downloads/Archivos/TouchScreen/HY32D.rar/
5. http://www.datasheetdir.com/SSD1289+LCD-Drivers-Controllers

1. źródło: http://pl.wikipedia.org/wiki/Kod\_Morse'a [↑](#footnote-ref-1)