Dorota Rabczuk
Akademia Morska w Gdyni

PRAKTYCZNE NAUCZANIE SYSTEMÓW WBUDOWANYCH Z WYKORZYSTANIEM PLATFORMY ARDUINO

Artykuł stanowi podsumowanie doświadczeń dydaktycznych w dziedzinie praktycznego nauczania systemów wbudowanych z wykorzystaniem platformy Arduino. Nauczanie zostało podzielone na dwa etapy poświęcone: 1) poznaniu rejestrów mikrokontrolera i konfigurowaniu portów i interfejsów na podstawie bezpośrednich wpisów do rejestrów; 2) wykorzystaniu magistral komunikacyjnych UART, SPI, I2c, 1-wire, Ethernet oraz samodzielnemu pisaniu bibliotek do transmisji danych po wybranych magistralach. W fazie uruchamiania projektu zastąpiono proste środowisko Arduino środowiskiem AtmelStudio, wyposażonym w możliwość debuggowania programu, integrując z nim przydatne biblioteki Arduino.

***Słowa kluczowe**: systemy wbudowane, mikrokontrolery, programowanie, język C/C++.*

WSTĘP

Szybki rozwój w dziedzinie systemów wbudowanych implikuje pojawianie się na rynku platform z dostosowanym sprzętem i kompletem bibliotek pozwalających w łatwy sposób złożyć działający układ, nawet jeśli konstruktor nie ma pełnej wiedzy na temat wykorzystywanych podzespołów. Jednak nieznajomość programowania w języku C i zasad pracy mikrokontrolera uniemożliwi konstruktorowi wprowadzenie jakichkolwiek modyfikacji w gotowe rozwiązania celem ich przystosowania do potrzeb użytkownika.

Platforma sprzętowa i programowa przydatna w praktycznym nauczaniu mikrokontrolerów i systemów wbudowanych powinna zapewnić możliwość nauczania na wielu poziomach [1]: od prostej obsługi wejść i wyjść przez wpisy do rejestrów, pisanie własnych bibliotek dla magistral komunikacyjnych do wykorzystania gotowych bibliotek w zaawansowanych aplikacjach z rozbudowanymi urządzeniami zewnętrznymi. Wybrano dwie uzupełniające się platformy: Arduino i Raspberry Pi, które są zorientowane na inne obszary wykorzystania, dzięki czemu mogą zapewnić hierarchiczność w nauczaniu. Platforma Arduino bazuje na systemie wejść/wyjść (GPIOs) i nadaje się przede wszystkim do zadań związanych z bezpośrednim sterowaniem i szybkim odczytem stanów środowiska zewnętrznego, podczas gdy platforma Raspberry Pi bazująca na Linuxie znajduje zastosowanie przede wszystkim w interfejsach użytkownika (monitor, głośnik) oraz cyfrowym przetwarzaniu sygnałów.

1. CHARAKTERYSTYKA PLATFORMY ARDUINO W ASPEKCIE WYKORZYSTANIA W LABORATORIUM SYSTEMÓW WBUDOWANYCH

Platforma Arduino dostarcza zarówno sprzęt z szeroką gamą sensorów, jak i narzędzia programistyczne do realizacji projektów na zasadzie dostępności kodu (*open source*) [1]. Rodzina układów ewaluacyjnych Arduino jest wyposażona w mikrokontrolery 8-bitowe Atmel z zasilaniem 5 V: Atmega32U4, ATMega 168, ATMega328 lub ATMega2560. Wyjątek w rodzinie Arduino stanowi płytka Arduino Due, na której zamontowano 32-bitowy 3V3-woltowy procesor Atmel SAM3X8E ARM Cortex-M3. Płytki bazowe udostępniają wejścia i wyjścia cyfrowe i analogowe (do przetwarzania ADC oraz DAC) oraz interfejsy komunikacyjne, do których można podłączyć płytki rozszerzeń (*extension boards*). Złącza płyty głównej są tak dopasowane, że umożliwiają podpięcie płytek rozszerzeń bez konieczności lutowania. Dla płyt bazowych przygotowano kilka nakładek (*shields*): moduł Ethernet, moduł łączności radiowej, moduł bezprzewodowej sieci radiowej Xbee, *driver* obciążeń indukcyjnych, np. silników krokowych. Płytki rozszerzeń z rozmaitymi czujnikami są dostarczane przez różnych producentów, zachowując kompatybilność wyprowadzeń z macierzystą płytą Arduino.

Częścią softwarową platformy Arduino jest środowisko IDE (*Integrated Development Environment*) do tworzenia i ładowania programów. Środowisko wspiera języki programistyczne C/C++, ładowanie skompilowanego programu odbywa się z udziałem programu rezydentnego *bootloader* po magistrali szeregowej UART lub USB. Ze względu na rozmaitość gotowych do użycia płytek rozszerzeń dostarczanych łącznie z bibliotekami platforma Arduino stanowi narzędzie do łatwego tworzenia projektów z dziedziny mikroprocesorowego nadzoru czujników i sterowania urządzeniami po różnych magistralach, w tym Ethernet oraz drogą radiową.

Kod programu napisany w języku C lub C++ jest w środowisku Arduino kompilowany za pomocą avr-gcc do pliku obiektowego, który następnie jest linkowany z bibliotekami Arduino. W rezultacie powstaje plik Intel hex, który zostaje wpisany do pamięci programu mikrokontrolera za pośrednictwem *bootloadera*, który już wcześniej się tam znajdował. Środowisko Arduino automatycznie generuje prototypy wszystkich użytych funkcji, w tym funkcji setup() i loop(), które w trakcie kompilacji są łączone w funkcję main() wg zasady: main() { setup(); while(1) loop();}. Rozbudowane biblioteki środowiska Arduino są wygodne w użyciu i zapewniają łatwe oprogramowanie różnych urządzeń zewnętrznych na wielu magistralach: I2C, SPI, UART, 1-wire, Ethernet, CAN, jednak brak możliwości debuggowania programu w trybie symulacji i w czasie rzeczywistym utrudnia uruchamianie programów. Brak narzędzi w środowisku Arduino można uzupełnić, przenosząc projekt utworzony w Arduino IDE do środowiska Atmel Studio 6. W tym celu należy wyszukać wśród wielu plików tworzonych podczas kompilacji w środowisku Arduino IDE plik zbioru prekompilowanych bibliotek core.a i przenieść go do katalogu nowo tworzonego projektu w Atmel Studio. Ponadto trzeba skopiować kod projektu z Arduino IDE

do Atmel Studio – nie należy korzystać z kodu źródłowego Arduino *.pde, lecz pobrać plik *.cpp z katalogu wynikowego kompilacji, ponieważ zawiera on prototypy używanych funkcji. W efekcie można skorzystać z profesjonalnych narzędzi do debuggowania programu w czasie rzeczywistym z opcjami pracy krokowej i punktów docelowych (*breakpoint*).

Platforma Raspberry Pi wykorzystuje procesory ARM7, 9 lub 11 lub w najnowszej wersji – procesor ARM Cortex-A7 900 MHz, z 1 GB pamięci RAM. Procesory te posiadają moce obliczeniowe predysponujące je do obliczeń DSP. Płyty bazowe Raspberry Pi są wyposażone w linie GPIOs, porty USB, HDMI, Ethernet, slot kart micro SD, interfejsy ekranu DSI oraz kamery CSI.

W systemach wbudowanych najlepiej sprawdza się połączenie możliwości platform Arduino i Raspberry Pi. Na rynku dostępna jest płyta Arduino w postaci rozszerzenia do płyty Raspberry Pi. W projekcie wykorzystującym obie platformy Rasberry Pi powinno zajmować się wizualizacją, przetwarzaniem obrazu i innych złożonych algorytmów, podczas gdy Arduino powinno być odpowiedzialne za sterowanie urządzeniami na różnych magistralach i szybką reakcję na zmiany, np. unikanie zagrożeń w środowisku zewnętrznym.

W początkowym etapie nauczania dobrze jest skorzystać z płytki ewaluacyjnej Arduino Esplora, wyposażonej w 8-bitowy 5 V mikrokontroler Atmel ATMega32u4 o 32 kB pamięci programu *flash* (w tym 4 kB zajmuje *bootloader*), 2,5 kB pamięci ulotnej SRAM, 1 kB nieulotnej pamięci EEPROM, z taktowaniem kwarcowym 16 MHz. Płytka jest wygodna do użycia w laboratorium, ponieważ posiada szereg urządzeń zewnętrznych, od których tradycyjnie rozpoczyna się nauczanie systemów wbudowanych: diodę LED i diodę RGB, cztery przyciski samopowrotne, joystick analogowy z centralnym przyciskiem, przetwornik ADC, na którego wejście można podać sygnał z jednego z zamontowanych na płycie czujników: potencjometru, mikrofonu, czujnika temperatury lub czujnika światła, akcelerometr 3-osiowy (w warunkach laboratoryjnych można wykorzystać do pomiaru kąta odchylenia od poziomu), *buzzer*, interfejs myszy i klawiatury po USB oraz gniazdo do podłączenia wyświetlacza z funkcją dotykową. Bogate wyposażenie płytki Arduino Esplora bardzo upraszcza organizację nauczania praktycznego w laboratorium w porównaniu z płytkami ewaluacyjnymi, na których znajduje się tylko procesor ze źródłem zasilania i taktowania, a urządzenia zewnętrzne trzeba podłączać na płytkach uniwersalnych lub lutować.

Początkowy etap nauczania systemów wbudowanych wymaga odwoływania się do rejestrów mikrokontrolera w celu podania ich przeznaczenia i wykonywanych na nich konfiguracji pracy układów peryferyjnych. Do najczęściej używanych należą rejestry obsługujące linie wejścia i wyjścia (GPIOs), które są zgrupowane w porty A, B, itd. każdy po 8 linii. Każdy bit rejestru jest odpowiedzialny za zarządzanie jedną linią wejściowo/wyjściową. Ustawienia wstępne dla linii GPIO wymagają wyboru kierunku dla linii (wejście lub wyjście) i ewentualnie podwieszenia linii wejściowych. W sumie każda linia *in/out* jest konfigurowana przez trzy bity leżące w trzech rejestrach: kierunkowym DDR (*Direction Register*), sterującym PORT oraz wejściowym PIN. Rejestr PIN odzwierciedla stany fizyczne

na liniach wejściowych, dlatego jest stosowany wyłącznie w trybie odczytu. Zarządzanie urządzeniami peryferyjnymi (timery, interfejsy komunikacyjne) oraz funkcjami (wyjątki, przerwania) jest również możliwe przez bezpośrednie wpisy do rejestrów mikrokontrolera.

Nauczanie programowania przez wpisy do rejestrów (tab. 1) wymaga wprawy w stosowaniu operacji arytmetyki binarnej tj.: koniunkcji &, alternatywy |, negacji ~ , sumy modulo 2 ^ , obrotu w prawo >> i lewo << . Każdy bit liczby daje się łatwo skojarzyć z jedną linią portu, jeśli liczby są zapisywane w systemie hex (szestnastkowym) lub binarnym. Konfigurowanie przez wpisy do rejestrów daje możliwość bezpośredniego wpływu na pracę mikrokontrolera na niskim poziomie.

Tabela 1. Porównanie konfiguracji linii portów przez bezpośredni wpis do rejestrów oraz przy wykorzystaniu funkcji bibliotecznych

Table 1. *Comparison of port configuration by direct writing to configuration registers to the usage of port library functions*

Wpis do rejestru	**Wykorzystanie funkcji bibliotecznych**
Pin 2 na porcie D konfigurowany jako wejście podwieszone wewnętrznie (*Pin 2 internally pulled up*)	
pin 2 = PORTD, 2 DDRD = DDRD & 0xFB; PORTD = PORTD | 0x04;	pinMode(2, INPUT_PULLUP); pinMode(2, INPUT_PULLUP);
Pin 2 na porcie D konfigurowany jako wejście w stanie wielkiej impedancji (*Pin 2 as high-Z input*)	
pin 2 = PORTD, 2 DDRD = DDRD & 0xFB; PORTD = PORTD & 0xFB;	pinMode(2, INPUT);
Pin 5 na porcie B konfigurowany jako wyjście (*Pin 13 as output*)	
pin 13 = PORTB, 5 DDRB = DDRB | 0x20;	pinMode(13, OUTPUT);
PIN 5 na porcie B sterowany naprzemiennie zerem i jedynką (*Pin 13 driven alternately 0 and 1*)	

pin 13 = PORTB, 5
void loop {
PORTB = PORTB | 0x20;
 	
PORTB = PORTB & 0xDF;}
	
	void loop {
	digitalWrite(13, HIGH);
	
	digitalWrite(13, LOW);}

Środowisko Atmel Studio ułatwia zrozumienie procesów zachodzących podczas wykonywania programu poprzez wizualizację stanu rejestrów mikrokontrolera. Debuggowanie można prowadzić w trybie symulacji (bez podłączania mikrokontrolera) lub w czasie rzeczywistym (program wgrany w mikrokontroler jest wykonywany krokowo pod nadzorem środowiska).

2. SAMODZIELNE TWORZENIE BIBLIOTEK KOMUNIKACYJNYCH

Po nabyciu podstawowej praktycznej wiedzy o posługiwaniu się rejestrami mikrokontrolera należy przejść do drugiego poziomu nauczania – praktycznego wykorzystania magistral komunikacyjnych, ze szczególnym zwróceniem uwagi na magistrale typowe dla mikrokontrolera, obsługiwane w trybie softwarowym (SPI, I2C, UART, 1-wire) oraz hardwarowym (Ethernet, USB). W środowisku Arduino dostępne są biblioteki do obsługi magistral komunikacyjnych, można też przystosować biblioteki pobrane z innych źródeł. Z dydaktycznego punktu widzenia studenci powinni napisać samodzielnie przynajmniej jedną bibliotekę dla wybranej magistrali, co wymaga wnikliwego przyjrzenia się sekwencjom komunikacyjnym i równocześnie stanowi ćwiczenie z języka C/C++ dla mikrokontrolerów.

Biblioteka napisana samodzielnie często jest prostsza i szybciej wykonuje polecenia komunikacyjne, ponieważ ma mniej zabezpieczeń. Przykładem mogą być autorskie biblioteki dla magistral 1-wire, SPI oraz I2C w tabelach 2, 3 oraz 4, będące przedmiotem ćwiczeń laboratoryjnych (w Arduino prototypy generowane automatycznie).

Biblioteka dla magistrali SPI jest prosta ze względu na charakter transmisji po tej magistrali, po której ruch bitów jest dookólny i odbywa się za pośrednictwem rejestrów przesuwnych przez oba urządzenia: *master* i *slave* uczestniczące w transmisji.

Biblioteka magistrali SPI składa się z jednej procedury wspólnej dla nadawania odbioru – każdemu nadaniu bitu po linii MOSI towarzyszy odbiór bitu po linii MISO tej magistrali w zamkniętym kółku. Wykorzystanie procedury komunikacyjnej dla SPI (tab. 3) wymaga wcześniejszego skonfigurowania interfejsu SPI w mikrokontrolerze.

Tabela 2. Biblioteka komunikacyjna dla magistrali 1-wire

Table 2. *Communication library for 1-wire interface*

```
// pin 2 = PORTD, 2   linia magistrali 1-wire
#include<util/delay.h>

void reset (void)
{
  pinMode(2, OUTPUT);
  digitalWrite(2, HIGH);
  delayMicroseconds(5);
  digitalWrite(2, LOW);
  delayMicroseconds(500);
  digitalWrite(2, HIGH);
  delayMicroseconds(60);
  pinMode(2, INPUT_PULLUP);
  if(digitalRead(2)) ; //brak urzadzenia
  else ; //jest urzadzenie
  delayMicroseconds(500);
  pinMode(2, OUTPUT);
  digitalWrite(2, HIGH);
}

void impuls0 (void)
{
   pinMode(2, OUTPUT);
  digitalWrite(2, HIGH);
  delayMicroseconds(5);
  digitalWrite(2, LOW);
  delayMicroseconds(80);
  digitalWrite(2, HIGH);
  delayMicroseconds(10);
}

void impuls1(void)
{
  digitalWrite(2, LOW);
  delayMicroseconds(10);
  digitalWrite(2, HIGH);
  delayMicroseconds(80);
}

void nadaj (char A)
{
   pinMode(2, OUTPUT);
  digitalWrite(2, HIGH);

  for(int i=0; i<8; i++)
  {
    if(A & 0x01){
    impuls1(); }
    else{
    impuls0(); }
    A=A>>1;
  }
}
char odbierz (void)
{
 char B=0;
 for(int i=0; i<8; i++)
 {
   B=B>>1;
   pinMode(2, OUTPUT);
   digitalWrite(2, HIGH);
   delayMicroseconds(5);
   digitalWrite(2, LOW);
   delayMicroseconds(2);
   pinMode(2, INPUT_PULLUP);
   delayMicroseconds(9);

   if(digitalRead(2))
   B |= 0b10000000;
   else
   B &= 0b01111111;
   delayMicroseconds(80);
 }
 pinMode(2, OUTPUT);
 digitalWrite(2, HIGH);
 return B;
}
```

Tabela 3. Biblioteka komunikacyjna dla magistrali SPI

Table 3. *Communication library for SPI interface*

```
int spi_transfer (int address, int transmitB)
 {
   int receiveB;
   digitalWrite(SS, HIGH); / / lub  digitalWrite(SS, LOW); w zależności od polaryzacji
   SPDR = address;
   while (!(SPSR && (1<<SPIF)));
   SPDR = transmitB;
   while (!(SPSR && (1<<SPIF)));
   receiveB = SPDR;
   digitalWrite(SS, LOW);
   return receiveB; // lub digitalWrite(SS, HIGH); w zależności od polaryzacji
 }
```

Biblioteka dla magistrali I2C (tab. 4) została napisana dla komunikacji realizowanej w 100% programowo, bez odwoływania się do interfejsu I2C, może być zatem wykorzystana również na mikrokontrolerach nieposiadających takiego interfejsu.

Tabela 4. Biblioteka komunikacyjna dla magistrali I2C

Table 4. *Communication library for I2C interface*

```
//pin 18 = PORTC, 4  SDA
//pin 19 = PORTC, 5  SCL
#define mySCL 19
#define mySDA 18


void impuls_SCL ()
{
_delay_us(5);
digitalWrite(mySCL, LOW);
_delay_us(5);
digitalWrite(mySCL, HIGH);
_delay_us(5);
digitalWrite(mySCL, LOW);
_delay_us(5);
}



void I2C_start (void)
{
_delay_us(5);
digitalWrite(mySDA, HIGH);
digitalWrite(mySCL, HIGH);
_delay_us(5);
digitalWrite(mySDA, LOW);
_delay_us(5);
digitalWrite(mySCL, LOW);
_delay_us(5);
}


void I2C_stop (void)
{
_delay_us(5);
digitalWrite(mySDA, LOW);
digitalWrite(mySCL, LOW);
_delay_us(5);
digitalWrite(mySCL, HIGH);
_delay_us(5);
digitalWrite(mySDA, HIGH);
_delay_us(5);
}
```

```
void I2C_transmitB (char data)
{
digitalWrite(mySCL, LOW);
for (int i=0; i<8; i++)
{
if (data&0b10000000)
digitalWrite(mySDA, HIGH);
else
digitalWrite(mySDA, LOW);
impuls_SCL();
data=data<<1;
}
pinMode(mySDA, INPUT_PULLUP);
digitalWrite(mySCL, HIGH);
_delay_us(5);
if(digitalRead(mySDA)) ;
else;
digitalWrite(mySCL, LOW);
_delay_us(5);
pinMode(mySDA, OUTPUT);
digitalWrite(mySDA, HIGH);
}

char I2C_receiveB (bool ack)
{
char received=0;
pinMode(mySDA, INPUT_PULLUP);

for (int i=0; i<8; i++)
{
received = received <<1;
digitalWrite(mySCL, HIGH);
_delay_us(5);

if (digitalRead(mySDA))
received |= 0b00000001;

else
received &= 0b11111110;

digitalWrite(mySCL, LOW);
_delay_us(5);

}
pinMode(mySDA, OUTPUT);
if (ack)
digitalWrite(mySDA, LOW);
else
digitalWrite(mySDA, HIGH);
_delay_us(5);
impuls_SCL();

return received;
}
```

PODSUMOWANIE

Rozwiązania proponowane w artykule zapewniające efektywność nauczania praktycznego systemów wbudowanych zostały z dobrym skutkiem sprawdzone w trakcie zajęć ze studentami. Rozmaitość urządzeń zewnętrznych, które można podłączyć na magistralach komunikacyjnych, daje możliwość zindywidualizowania projektów studenckich i gwarancję, że każdy student będzie zaangażowany w swój projekt, zyskując pożyteczną praktykę inżynierską.

LITERATURA

1. Jamieson P., *Arduino for Teaching Embedded Systems. Are Computer Scientists and Engineering Educators Missing the Boat?* Proceedings FECS'10 Conference on Frontiers in Education, Las Vegas 2010.
2. Rabczuk D., *Podsumowanie doświadczeń nauczania programowania mikrokontrolerów 8-bitowych AVR w środowisku AVRStudio*, Zeszyty Naukowe Akademii Morskiej w Gdyni, 2011, nr 70.

PRACTICAL TEACHING OF EMBEDDED SYSTEMS ON THE BASIS OF ARDUINO PLATFORM

Summary

The article is aimed at summarizing author's experience in practical teaching of embedded systems using Arduino platform. The teaching process is divided into two levels: first level is devoted to getting experience in programming by direct entry to configuration and port registers of the microcontroller, the 2-nd level is devoted to using communication busses UART, SPI, I2C, 1-wire, Ethernet and writing own libraries to hold communication on these busses. While putting things in operation the simple Arduino platform is changed to more sophisticated AtmelStudio platform with capability of debugging the project (Arduino libraries are integrated in it).

Keywords*: embedded systems, microcontrollers, programming, C/C++ languages.*