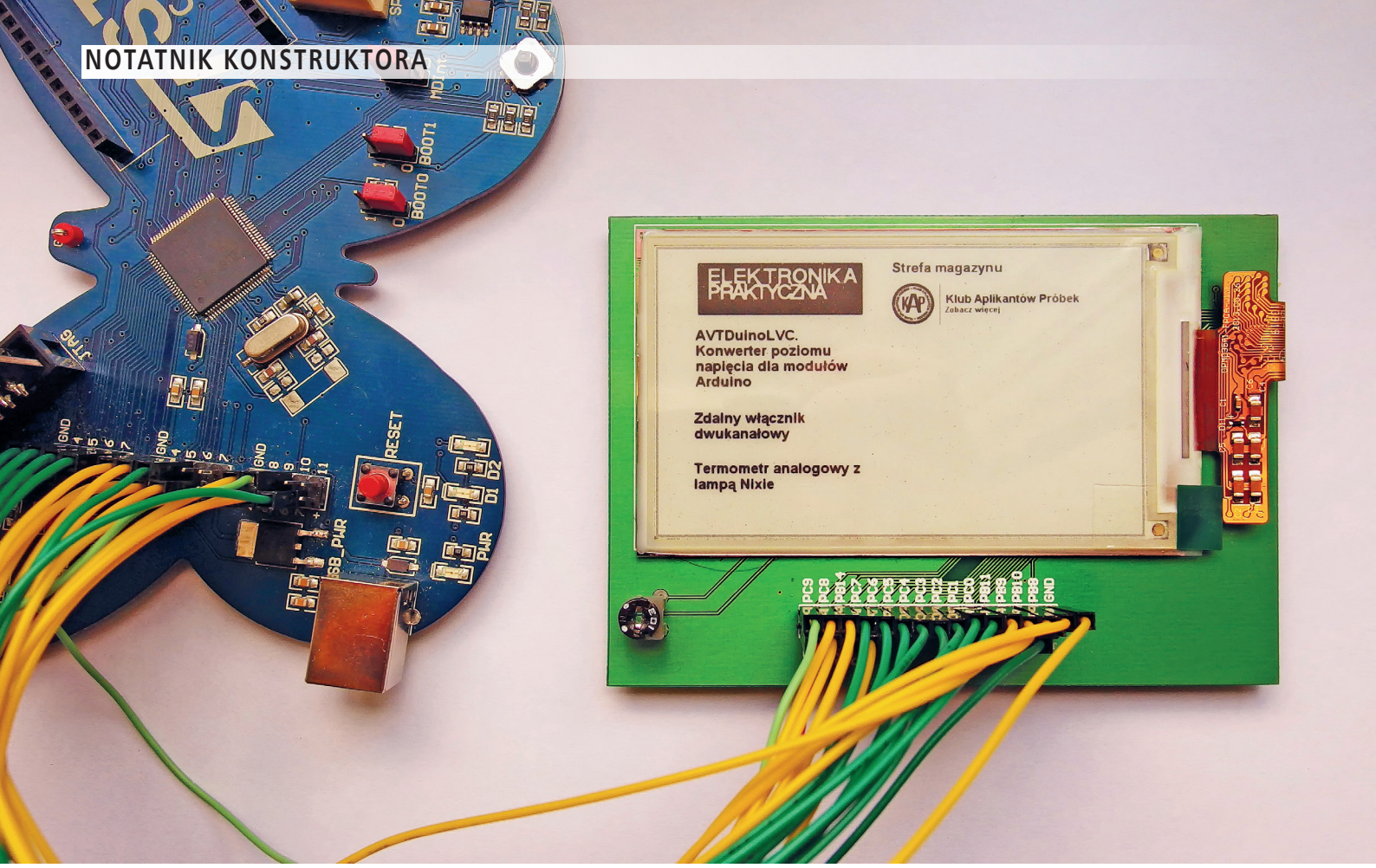


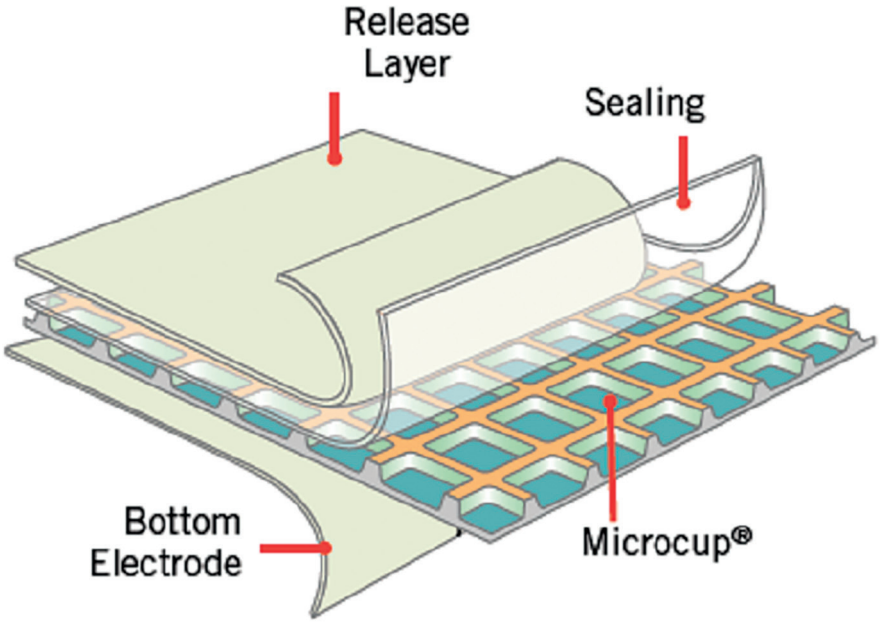
Wyświetlacze e-paper – teoria i praktyka

Na naszych oczach odbywa się teraz rewolucja związana z jednymi z ważniejszych elementów współczesnej elektroniki – wyświetlaczami graficznymi. Mają one coraz lepsze parametry: relatywnie duże przekątne matrycy, duże głębie kolorów i wierność odtwarzania barw oraz co ważne dla konstruktorów i programistów – stosunkowo łatwo nimi sterować. Jednocześnie spada cena, a konstruktorzy mogą liczyć na wsparcie (często darmowe) w postaci bibliotek graficznych. Do znanych i popularnych wyświetlaczy LCD , TFT i OLED w ostatnim czasie dołączyły bardzo ciekawe elementy nazywane popularnie e-papier lub EPD.

Wyświetlacze e-papier są dobrze znane z rynku konsumenckiego. Wykorzystuje się je w popularnych czytnikach książek dystrybuowanych w formie elektronicznej, czyli tzw. e-booków. Równolegle z nazwą e-papier jest stosowany termin wyświetlacz bistabilny. Co to oznacza?

Obraz na ekranie zostaje zachowany bez zmiany jakości (kontrastu, jasności) po wyłączeniu napięcia zasilającego. Ta właściwość pozwala na tworzenie interfejsów użytkownika o ekstremalnie małym poborze mocy. Wyświetlacz potrzebuje energii tylko w momencie zmiany wysiedlanej treści, a potem można ją wyłączyć aż do następnej zmiany. W aplikacjach, w których zmiany na wyświetlaczu nie zachodzą zbyt często to olbrzymia zaleta. W typowym wyświetlaczu LCD treść – niezależnie czy się zmienia, czy nie – musi być odświeżana ok. 30 razy na sekundę. Kolejnym czynnikiem znacznie ograniczającym pobór mocy jest brak wymogu podświetlania. Odczytywanie informacji z ekranu e-papier odbywa się dokładnie tak samo, jak w przypadku czytania zadrukowanej kartki. Jeżeli w pomieszczeniu jest wystarczająco jasno, to podświetlenie nie jest potrzebne. Matryce LCD do działania potrzebują sporych ilości energii przeznaczonej dla podświetlenia.

„Atrament" elektroniczny składa się z milionów maleńkich mikrokapsułek o średnicy ludzkiego włosa. Każda mikrokapsułka zawiera dodatnio naładowane cząstki białe i naładowane ujemnie cząstki czarne. Cząstki są zawieszone w przeźroczystej cieczy. Jeżeli poddamy mikrokapsułki działaniu pola elektrycznego, to białe i czarne cząstki będą się przemieszczać. Wyświetlacz ma wbudowane 2 elektrody. Jedna przeźroczysta jest umieszczona na powierzchni ekranu wyświetlacza, a druga pod mikrokapsułkami (**rysunek 1**). Kiedy górna elektroda będzie miała potencjał dodatni a dolna ujemny, to ujemnie naładowane cząstki czarne zostaną przez pole elektryczne przesunięte do góry i wyświetlacz w tym miejscu będzie zaczerniony. Odwrotnie, kiedy górna elektroda będzie



Rysunek 1. Budowa matrycy EPD

ujemna a dolna dodatnia, to cząstki białe, naładowane dodatnio będą się gromadziły przy powierzchni ekranu wyświetlacza, a naładowane ujemnie w głębi i wyświetlacz będzie „świecił" na biało. Po zaniku napięcia na elektrodach cząstki pozostają w swoim ostatnim położeniu i informacja na ekranie pozostaje bez zmian (**rysunek 2**).

Na wyjściu standardowego drivera sterującego elektrodą przy powierzchni wyświetlacza mogą występować trzy napięcia względem elektrody umieszczonej pod mikrokapsułkami:

- Napięcie dodatnie (ok. +15 V) – piksel jest czarny.
- Napięcie ujemne (ok. -15 V) – piksel jest biały.
- Napięcie 0 V – piksel się nie zmienia.

Sterowanie kapsułek jest zorganizowane matrycowo. Przykładowy wyświetlacz GDE035A4 ma 600 linii po 800 kapsułek, czyli inaczej mówiąc ma rozdzielczość 800×600 pikseli. Trudno sobie wyobrazić, by elektrody sterujące dla tak wielkiej liczby elementów, nawet w połączeniu matrycowym, były wyprowadzone na jakieś złącze. Dlatego sterowaniem driverów kapsułek zajmują się specjalizowane układy scalone mające bardzo dużą liczbę wyprowadzeń sterujących. W GDE035A4 jest wbudowany układ HX8705, który zgodnie z dokumentacją ma 1683 wyprowadzenia! Schemat blokowy sterownika pokazano na **rysunku 3**.

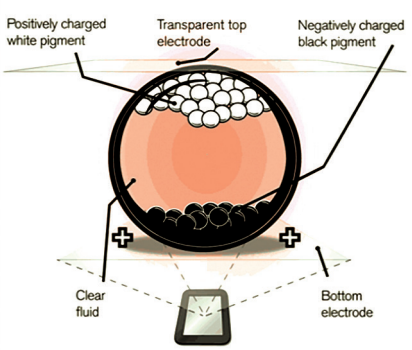
Elektrody sterujące również są matrycowo: do każdego z pikseli jest doprowadzona jedna z 800 elektrod drivera SOURCE i jedna z 600 elektrod drivera GATE. Driver SOURCE jest przeznaczony do sterowania elektrodami kolumn matrycy i ma 800 wyprowadzeń pracujących z trzema napięciami wyjściowymi: +15 V, -15 V i 0 V. Z driverem jest połączony dwukierunkowy rejestr o długości 800 słów 2-bitowych. Rejestr jest podzielony na 4 grupy. Każde słowo 2-bitowe odpowiada za napięcie na wyjściu drivera według zależności pokazanej w **tabeli 1**.

Każdy bajt zapisany do rejestru odpowiada za poziomy wyjściowe 4 driverów. Żeby zapisać cała linijkę o długości 800 pikseli, trzeba wpisać do rejestru 200 bajtów (200×8/2=800). Na wyjściu rejestru przesuwnego jest umieszczony rejestr LATCH, tak aby poziomy sterujące buforem wyjściowym mogły się pojawić dopiero po zapisaniu całego bufora i uaktywnieniu sygnałów: LE (*Latch Enable*) przepisującego zawartość rejestru przesuwnego do rejestru LATCH i aktywującego wyjścia rejestru LATCH OE (*Latch Oputput Enable*). Ponieważ rejestr jest dwukierunkowy, to jest potrzebny dodatkowy sygnał sterujący kierunkiem przepływu danych. Tę rolę spełnia sygnał R/L.

Driver GATE jest przeznaczony do sterowania elektrodami wierszy matrycy i ma 600 wyprowadzeń. W odróżnieniu od drivera SOURCE, na jego wyjściach mogą wystę-

Tabela 1. Programowanie napięcia wyjściowego drivera SOURCE

Bit D[2n+1] (starszy)	Bit D[2n] (młodszy)	Napięcie na wyjściu bufora[n+1=4k]
0	0	VSSA (AGND)
0	1	VPOS (+15V)
1	0	VNEG(-15V)
1	1	VSSA(AGND)
n=0…3, k=0…199		



Rysunek 2. Zasada działania wyświetlacza e-paper

Tabela 2. Wyprowadzenia wyświetlacza GDE035A4

1	VNEG	Ujemne napięcie zasilające driver SOURCE
2	VPOS	Dodatnie napięcie zasilające driver SOURCE
3	VSS	Masa
4	VDD	Napięcie zasilające układy cyfrowe
5	CLK	Zegar rejestru drivera SOURCE
6	LE	Latch Enable drivera SOURCE
7	OE	Output Enable drivera SOURCE
8	SHR (R/L)	Kierunek przesuwania danych rejestru drivera SOURCE
9	STH	Start Pulse rejestru drivera SOURCE
10	D0	Dane wejściowe rejestru SOURCE
11	D1	Dane wejściowe rejestru SOURCE
12	D2	Dane wejściowe rejestru SOURCE
13	D3	Dane wejściowe rejestru SOURCE
14	D4	Dane wejściowe rejestru SOURCE
15	D5	Dane wejściowe rejestru SOURCE
16	D6	Dane wejściowe rejestru SOURCE
17	D7	Dane wejściowe rejestru SOURCE
18	COMMON	Punkt wspólny
19	VGH	Dodatnie napięcie zasilające driver GATE
20	VGL	Ujemne napięcie zasilające drivera GATE
21	MODE1	Tryb pracy drivera GATE
22	STV	Start Pulse drivera GATE
23	CPV	Zegar rejestru drivera GATE
24	VBORDER	Wyprowadzenie BORDER

pować tylko 2 poziomy: aktywny lub nieaktywny. Podobnie jak w przypadku SOURCE, driver jest połączony z 2-kierunkowym rejestrem przesuwnym.

Interfejs komunikacyjny HX8705 jest zbudowany z 8-bitowej magistrali danych oraz sygnałów sterujących: LE, OE, SHR i STH drivera SOURCE i STV, CPV drivera GATE. Oprócz linii sterujących wyświetlaczem na 24-pinowym wyprowadzeniu umieszczono też linie zasilające (**tabela 2**).

Zapis rejestru SOURCE polega na zapamiętaniu 200 bajtów przez równoległy, 8-bitowy interfejs komunikacyjny. Dane z linii D0…D7 są zapisywane do rejestru przy narastającym zboczu na linii CLK. Zostało to pokazane na **rysunku 4**.

Nasz wyświetlacz jest sterowany przez moduł ewaluacyjny *STM32 Conectivity* linie z mikrokontrolerem z rodziny STM32F107VB. Na **listingu 1** pokazano definicje linii sterujących i magistrali danych. Dane do bufora SOURCE są zapisywane przez zamieszczoną na **listingu 2** funkcję *SendSOURCEData*.

Sekwencja zapisu rozpoczyna się od ustawienia (LE=1) i wyzerowania (LE=0) oraz ustawienia OE (OE=1). Każda zmiana stanu tych linii sterujących wymaga jednego cyklu zegara. Potem jest zapisywanych 200 bajtów z wcześniej przygotowanego bufora *pArray*. Każdy bajt jest wpisywany narastającym zboczem sygnału zegara CL (**rysunek 6**). Wprowadzenie zmiennej *old_data* zapobiega niszczeniu stanów nieużywanych linii portu przy zapisywaniu linii magistrali danych. Po zapisaniu 200 bajtów jest aktywowany sygnał OE i zawartość rejestru LATCH pojawia się na wyjściu bufora SOURCE. Jednocześnie na linii CPV jest wymuszane zbocze opadające a potem narastające.

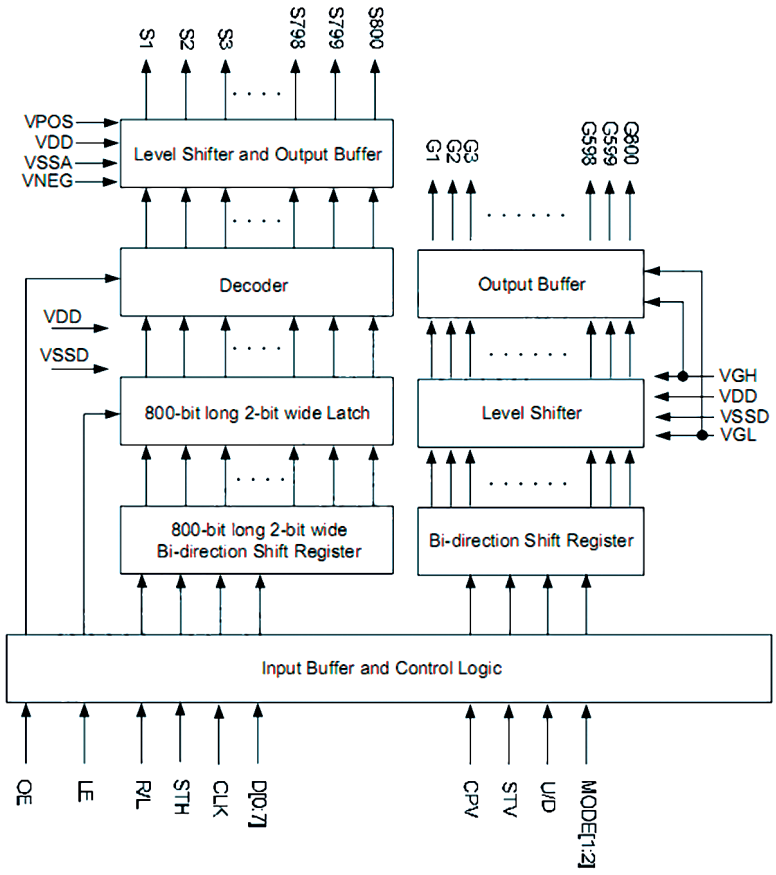
Po każdym zapisaniu wiersza trzeba zmienić zawartość licznika wierszy, czyli rejestru GATE, aby następny wiersz był zapisywany w kolejnej pozycji na wyświetlaczu. Cykl uaktywnienia zapisywania rejestru, aby sterował pierwszym wierszem (G1), rozpoczyna się od wyzerowania wyjścia STV i wymuszenia zbocza narastającego na linii zegarowej CPV (**rysunek 6**). Każda sekwencja zapisu rejestrów wyświetlacza powinna się zaczynać od wykonania procedury *EPD_StartScan()* wymuszającej taką sekwencję (**listing 3**).

W teorii mamy wszystko co potrzeba, aby wyświetlić dowolną bitmapę. Zapisujemy 600 razy rejestr SOURCE o pojemności 800×2 bity jednocześnie zmieniając wpisy do rejestru GATE. Zależnie od zwartości 2 bitów programujących napięcie wyjściowe dla określonego piksela (kapsułki) punkt na ekranie będzie czarny lub biały, jednak w praktyce nie jest to takie łatwe. Okazuje się, że do zmiany

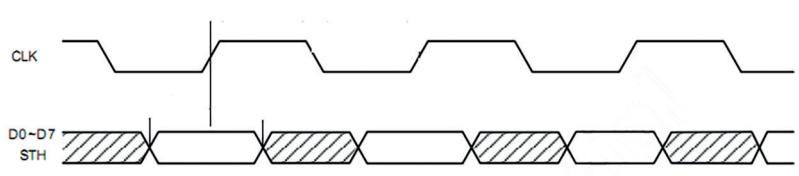
Listing 1. Definicje linii portów magistrali danych i sygnałów sterujących

```
//linie sterujące
#define EPD_CL_PORT    GPIOE
#define EPD_CL         GPIO_Pin_0  //CL
#define EPD_OE_PORT    GPIOE
#define EPD_OE         GPIO_Pin_1  // OE
#define EPD_LE_PORT    GPIOE
#define EPD_LE         GPIO_Pin_2  //LE
#define EPD_SHR_PORT   GPIOE
#define EPD_SHR        GPIO_Pin_3  //SHR
//lnie magistrali danych
#define EPD_DB_PORT    GPIOD
#define EPD_DB_0       GPIO_Pin_0  //PD0
#define EPD_DB_1       GPIO_Pin_1  //PD1
#define EPD_DB_2       GPIO_Pin_2  //PD2
#define EPD_DB_3       GPIO_Pin_3  //PD3
#define EPD_DB_4       GPIO_Pin_4  //PD4
#define EPD_DB_5       GPIO_Pin_5  //PD5
#define EPD_DB_6       GPIO_Pin_6  //PD6
#define EPD_DB_7       GPIO_Pin_7  //PD7
#define EPD_CL_H       EPD_CL_PORT->BSRR =EPD_CL
#define EPD_CL_L       EPD_CL_PORT->BRR =EPD_CL
#define EPD_LE_H       EPD_LE_PORT->BSRR =EPD_LE
#define EPD_LE_L       EPD_LE_PORT->BRR =EPD_LE
#define EPD_OE_H       EPD_OE_PORT->BSRR =EPD_OE
#define EPD_OE_L       EPD_OE_PORT->BRR =EPD_OE
```

„świecenia" punktu na ekranie potrzeba czegoś więcej, niż tylko zmiany polaryzacji pomiędzy elektrodami sterującymi. Po to, aby obraz wyświetlił się i w dodatku mógł pozostawać w tym stanie przez długi czas, potrzeba kilku zdefiniowanych sekwencji zapisywania całego wyświetlacza. Na elektrodach sterujących kapsułkami trzeba wymusić coś w rodzaju przebiegu (*waveform*). Długość przebiegu i jego kształt zależy od producenta matrycy, a nawet od konkretnego modelu wyświetlacza. Przebiegi sterujące będą wyglądały inaczej dla zmiany „świecenia" kapsułki, a inaczej dla sekwencji odświeżenia już wyświetlanego obrazu.



Rysunek 3. Schemat blokowy układu HX8705



Rysunek 4. Przesyłanie danych do rejestru SOURCE

Podanie przebiegu na każdą z elektrod sterujących sprowadza się do wykonania kilku zapisów (sekcji) całego wyświetlacza, jak to zostało pokazane na **rysunku 7**. Okres skanowania wyświetlacza jest określony przez czas od wysterowania G0 do wysterowania G599, czyli czas potrzebny do zapisania całego wyświetlacza. Przykład zapisu piksela pokazano na **rysunku 8**. Każdy z pikseli jest zapisywany 12 razy (12 sekcji). Każda sekcja to jeden zapis całego wyświetlacza.

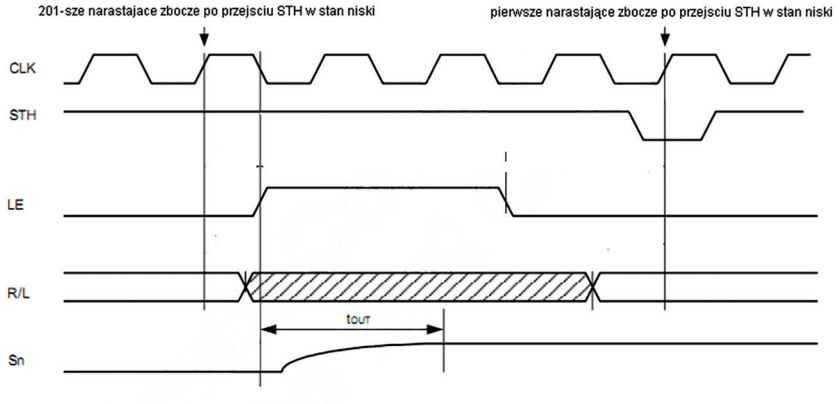
W czasie sterowania wyświetlaczem Każdy piksel może zmieniać swój stan:

- Z białego na biały.
- Z białego na czarny.
- Z czarnego na biały.
- Z czarnego na czarny.

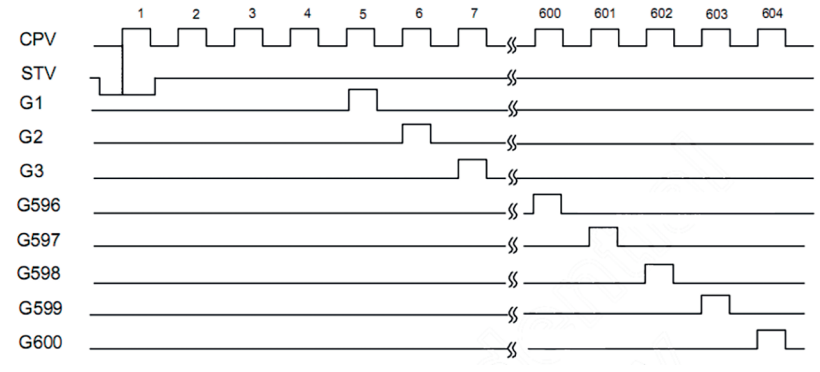
Dla każdej z takich sekwencji trzeba zdefiniować przebieg sterujący (**rysunek 9**). W naszym wypadku definicja sekwencji sterujących jest wykonywana przez funkcję *make_vave_table* (**listing 4**). W wyniku działania tej funkcji jest zapisywana dwuwymiarowa tablica *vave_end_table* o wymiarach 256×12. Procedurę wyświetlania całego obrazu pokazano na **listingu 5**. Rozpoczyna się od włączenia zasilania matrycy (procedura *EPD_PowerOn()*). Liczba sekcji zapisu jest określona przez wartość FRAME_END_LEN. W każdej sekcji zapisu najpierw jest wywoływana funkcja *EPD_Start_Scan()* ustawiająca skanowanie linii na początek, czyli od aktywnego sygnału na linii G0. Potem program wywołuje 600 razy w pętli funkcję *linie_end_pic* (**listing 6**). W wyniku działania tej funkcji jest tworzona tablica *g_dest_data[]* o rozmiarze 200 bajtów. Zawartość tej tablicy jest zapisywana do rejestru SOURCE przez funkcję *SendSOURCEData()*. Sekwencja zapisu jest kończona wyłączeniem zasilania matrycy – *procedura EPD_PowerOff()*. Na **fotografii 10** pokazano przykładową bitmapę na wyświetlaczu.

Listing 2. Procedura zapisywania jednego wiersza wyświetlacza

```
void SendSOURCEData(u8 *pArray )
{
    EPD_LE_H;  //LE =1
    EPD_CL_L;  //CL=0
    EPD_CL_H;  //CL=1
    EPD_CL_L;  //CL=0
    EPD_CL_H;  //CL=1
    EPD_LE_L;  //LE=0
    EPD_CL_L;  //CL=0
    EPD_CL_H;  //CL=1
    EPD_CL_L;  //CL=0
    EPD_CL_H;  //CL=1
    EPD_OE_H;  //OE =1
    EPD_CL_L;  //CL=0
    EPD_CL_H;  //CL=1
    EPD_CL_L;  //CL=0
    EPD_CL_H;  //CL=1
    EPD_SPH_L;  //SPH=0
    old_data = (uint16_t)(EPD_DB_PORT->ODR);
    old_data &= 0xFF00;
    for(column= 0; column < 200; column++)  // zapisanie 200bajtów z tablicy
wiersza
    {
        new_data = old_data | (u16)pArray[column];
        (EPD_DB_PORT->ODR) = new_data;
        EPD_CL_L;  //CL=0   takt zegara rejestru SOURCE
        EPD_CL_H;  //CL=1
    }
    EPD_SPH_H;  //STH=1
    EPD_CL_L;  //CL=0
    EPD_CL_H;  //CL=1
    EPD_CL_L;  //CL=0
    EPD_CL_H;  //CL=1
    EPD_CPV_L;  //CPV=0 - zegar taktujacy bufor GATE
    EPD_OE_L;  //OE =0
    EPD_CL_L;  //CL=0
    EPD_CL_H;  //CL=1
    EPD_CL_L;  //CL=0
    EPD_CL_H;  //CL=1
    EPD_CPV_H;  //CPV=1
}
```



Rysunek 5. Sekwencja sygnałów sterujących po zapisaniu całego rejestru SOURCE



Rysunek 6. Zapisywanie rejestru GATE

Układ zasilania

Układ zasilania matrycy jest rozbudowany, ponieważ musi dostarczać napięć +15V(VPOS) i -15V(VNEG) do zasilania bufora SOURCE oraz +20V(VGH) i -20V (VGL) do zasilania bufora GATE. Zależności pomiędzy tymi napięciami pokazano na **rysunku 11**. Sterowanie układem zasilania musi umożliwiać wykonanie sekwencji włączania i wyłączania napięć zasilających, tak jak to zastało pokazane na **rysunku 12**.

Listing 3. Wymuszenie początku skanowania linii wyświetlacza

```
void EPD_Start_Scan(void)
{
    EPD_STV_H;  //STV=1
    loop = 2;
    while(loop--)
    {
        EPD_CPV_L;  //CPV=0
        Delay(0xf);
        EPD_CPV_H;  //CPV=1
        Delay(0xf);
    }
    EPD_STV_L;  //STV=0
    loop = 2;
    while(loop--)
    {
        EPD_CPV_L;  //CPV=0
        Delay(0xf);
        EPD_CPV_H;  //CPV=1
        Delay(0xf);
    }
    EPD_STV_H;  //STV=1
    loop = 2;
    while(loop--)
    {
        EPD_CPV_L;
        Delay(0xf);
        EPD_CPV_H;
        Delay(0xf);
    }
}
```

Żeby było to możliwe, zasilacz został wyposażony w tranzystorowe klucze włączające i wyłączające niezależnie każde z napięć wyjściowych.

Z oczywistych względów najpierw ma się pojawić napięcie zasilające układy cyfrowe: VDD, potem ujemne napięcia zasilające bufor GATE: VGL i w końcu pozostałe napięcia VNEG, VPOS i VGH. Przy wyłączaniu wyświetlacza jako ostatnie jest wyłączane napięcie VGL. Taka sekwencja ma zapobiegać zjawisku zatrzaskiwania (*latch-up*) i w konsekwencji możliwości uszkodzenia układu zasilania lub sterownika wyświetlacza.

W rozwiązaniu modelowym wykorzystano zastosowano przetwornicę AME5142 oraz sterowne cyfrowo układy włączania/wyłączania napięć wyjściowych (**rysunek 13**). Procedurę sekwencji włączenia zasilania pokazano na **listingu 7**, a wyłączania na **listingu 8**.

Bitmapa – przygotowanie i konwersja

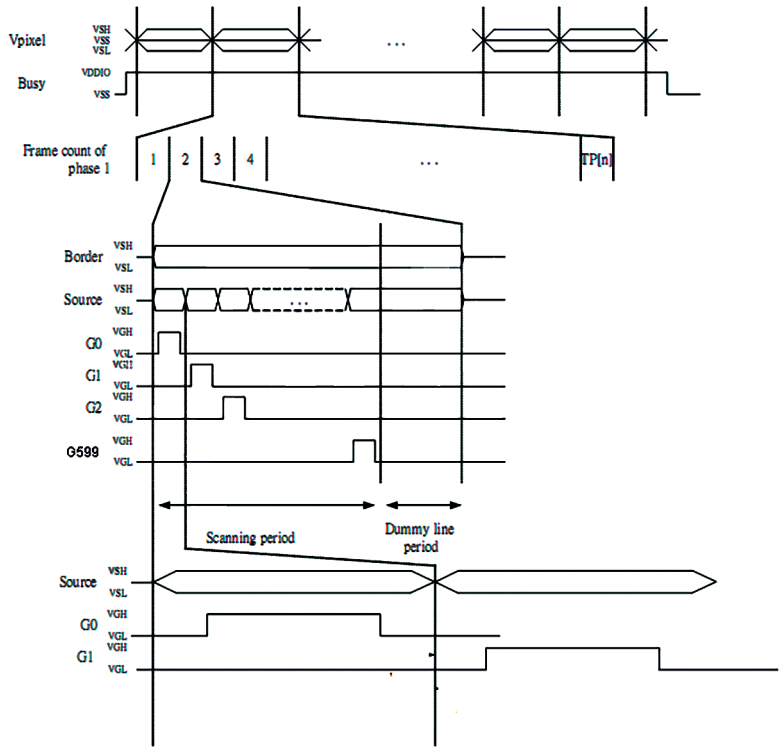
Przygotowanie bitmapy wyświetlanej przez nasz wyświetlacz e-paper odbiega od przygotowania bitmapy dla wyświetlaczy LCD. Standardowa bitmapa charakteryzuje się rozmiarem proporcjonalnym do rozmiaru wyświetlacza w pikselach i do głębi koloru. Dla wyświetlaczy monochromatycznych bez odcieni szarości jeden bit bitmapy odpowiada jednemu pikselowi, ale dla kolorowych wyświetlaczy z 16-bitową głębią kolorów potrzeba aż 2 bajty bitmapy, żeby zapisać informację o jednym pikselu.

Wiemy, że nasz e-paper jest wyświetlaczem monochromatycznym i że napięcie na każdym z pikseli jest zakodowane na 2 bitach. Można próbować tworzyć do wyświetlania bitmapy o 1-bitowej głębi koloru i w trakcie przesyłania do rejestrów SOURCE każdy bit bitmapy konwertować na 2 bity zgodnie z zasadą wyjaśnioną na rys. 4. Przy pierwszych próbach z wyświetlaczem tak właśnie zrobiłem. Niestety okazało się, że ten sposób nie jest dobry. Bitmapa co prawda wyświetlała się, ale z przekłamaniami i z bardzo małym kontrastem. Bitmapę trzeba było przekonwertować do postaci monochromatycznej z 2-bitową głębią kolorów i wtedy wyświetla się prawidłowo.

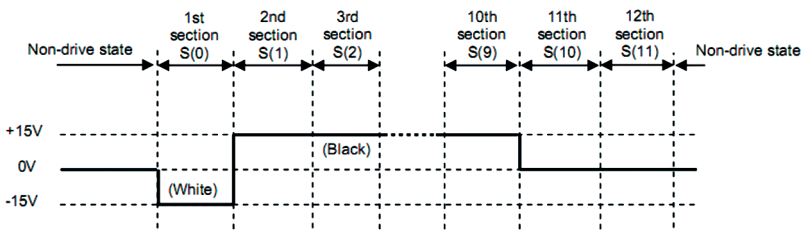
Przeznaczona do wyświetlania bitmapa o odpowiedniej wielkości (w naszym wypadku 800×600 pikseli) musi być w pierwszym kroku przekonwertowana z kolorowej na monochromatyczną z zachowaniem odcieni szarości. Do tego celu bardzo dobrze nadaje się dowolny program graficzny potrafiący wykonać taka konwersję. Może to być na przykład GIMP, Photoshop lub inny. W Photoshopie, po otwarciu pliku wykonujemy konwersję na obraz monochromatycz-

Listing 4. Tworzenie tablicy definicji sekwencji sterującej

```
void make_wave_table(void)
{
    int frame, num;
    unsigned char tmp,value;
    for(frame=0; frame<FRAME_END_LEN; frame++)
    {
        for(num=0; num<256; num++)
        {
            tmp = 0;
            tmp = wave_end[(num>>6)&0x3][frame];
            tmp = tmp<< 2;
            tmp &= 0xfffc;
            tmp |= wave_end[(num>>4)&0x3][frame];
            tmp = tmp<< 2;
            tmp &= 0xfffc;
            tmp |= wave_end[(num>>2)&0x3][frame];
            tmp = tmp<< 2;
            tmp &= 0xfffc;
            tmp |= wave_end[(num)&0x3][frame];
            value = 0;
            value = (tmp <<6) & 0xc0;
            value += (tmp<<2) & 0x30;
            value += (tmp>>2) & 0x0c;
            value += (tmp>>6) & 0x03;
            wave_end_table[num][frame] = value;
        }
    }
}
```



Rysunek 7. Skanowanie wyświetlacza

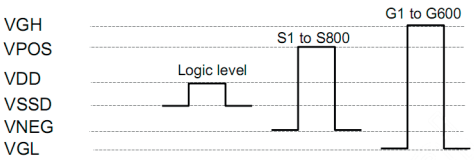


Rysunek 8. Przykładowe ustawienie piksela w czasie 12 zapisów

Waveform Name		Waveform Parameter	S(0)	S(1)	S(2)	S(3)	S(4)	S(5)	S(6)	S(7)	S(8)	S(9)	S(10)	S(11)
W0	(White→ White)	Output Level	+15V	-15V	-15V	0V	0V	0V	0V	0V	0V	0V	0V	0V
W1	(White→ Black)	Output Level	-15V	+15V	+15V	+15V	+15V	+15V	+15V	+15V	0V	0V	0V	0V
W2	(Black→ White)	Output Level	+15V	-15V	-15V	-15V	-15V	-15V	-15V	-15V	0V	0V	0V	0V
W3	(Black→ Black)	Output Level	-15V	+15V	+15V	0V	0V	0V	0V	0V	0V	0V	0V	0V

Rysunek 9. Przykładowe sekwencje zmiany stanu pikseli

Fotografia 10. Przykładowa bitmapa



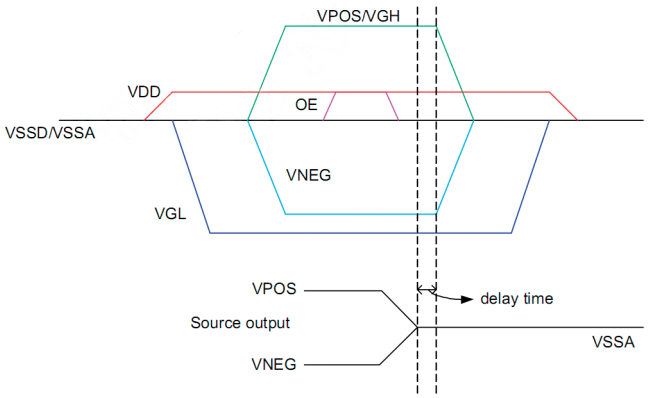
Rysunek 11. Zależności pomiędzy poziomami napięć zasilających wyświetlacz

ny poleceniem *Obraz → Dopasowania → Czarno-biały*. Potem za pomocą krzywych opcjonalnie można przekonwertowany obraz przyciemnić lub zwiększyć kontrast. Tak przygotowaną bitmapę trzeba przekonwertować na obraz o 2-bitowej głębi kolorów i zapisać w postaci tablicy, którą można będzie skompilować za pomocą kompilatora języka C. Do tego celu wykorzystamy darmowy program *uC/GUI – Bitmap Convert* (**rysunek 14**). Plik otwieramy za pomocą menu *File → Open*. Potem konwertujemy ją na wersję o 2-bitowejk głębi kolorów za pomocą polecenia *Image → Convert Into → Gray4*.

Bitmapę po konwersji trzeba zapisać poleceniem *File → SaveAs*. Do wyboru są formaty: *C bitmap File*, *Windows Bitmap File*, *C stream*. Dla naszych celów wybieramy format *C bitmap file*. Program generuje plik tekstowy z rozszerzeniem „.c" zawierający tablicę *static GUI_CONST_STORAGE unsigned char* z zapisanymi bajtami przekonwertowanej bitmapy. Trzeba ten plik otworzyć w edytorze tekstowym, skopiować zawartość tablicy i wkleić ją do tablicy programu. Po skompilowaniu i przesłaniu kodu wynikowego do pamięci Flash mikrokontrolera otrzymamy na ekranie obraz pokazany na **fotografii 15**.

Listing 5. Procedura wyświetlania całego obrazu
```
void EPD_Display_PIC(void)
{
    int line, frame;
    bool led_flag = TRUE;
    unsigned char *ptr;

    EPD_PowerOn();  //włączenie zasilania
    ptr = (unsigned char *)(ac);
    for(frame=0; frame<FRAME_END_LEN; frame++)
    {
        EPD_Start_Scan();
        for(line=0; line<600; line++)
        {
            line_end_pic(ptr + line*200, frame);  //42ms
            SendSOURCEData ( g_dest_data );  //40ms
        }
        SendSOURCEData ( g_dest_data );
    }
    EPD_PowerOff();//wyłączenie zasilania
}
```



Rysunek 12. Sekwencja włączania i wyłączania napięć zasilających wyświetlacz

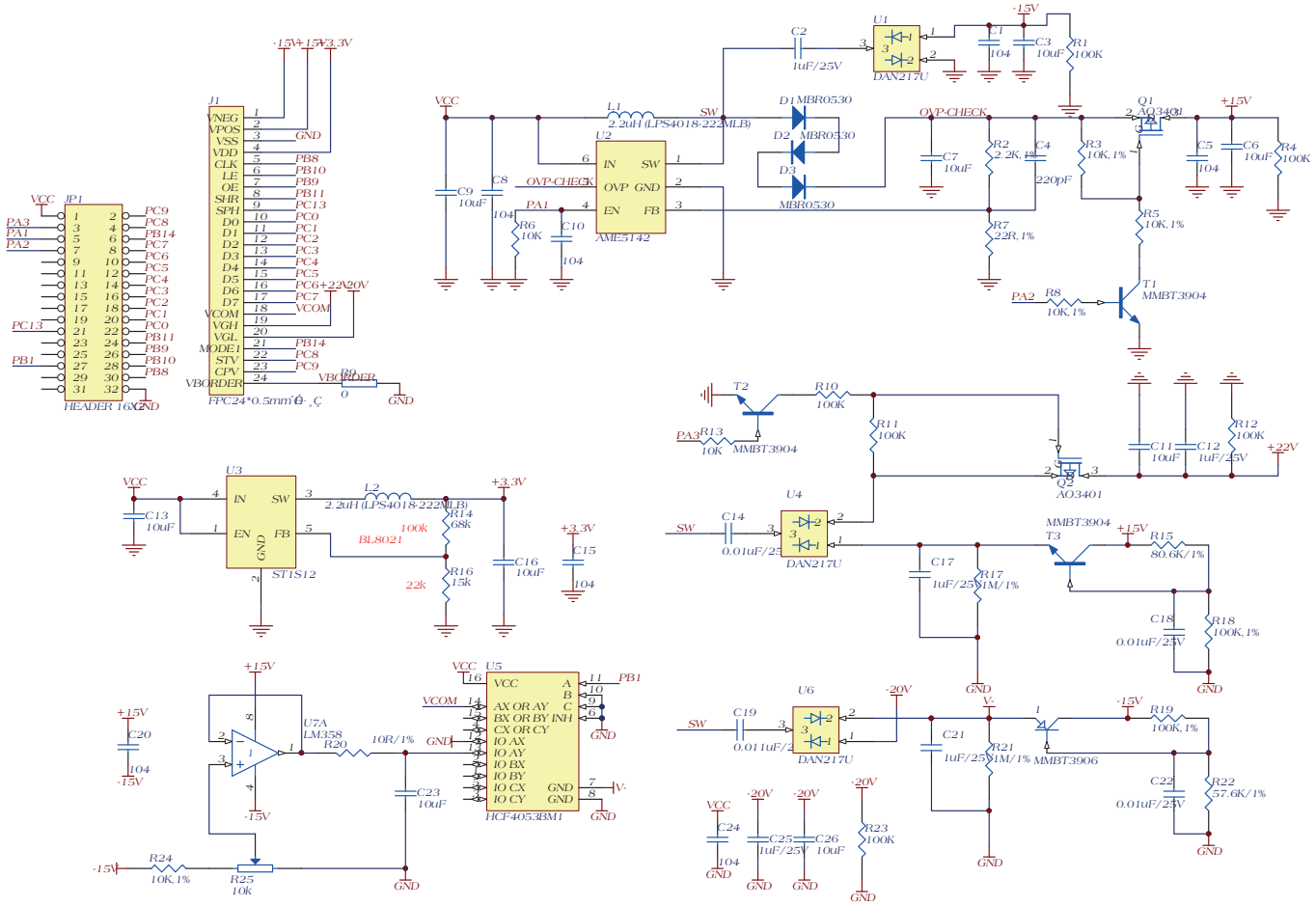
Listing 6. Procedura linie_end_pic
```
void line_end_pic(u8 *new_pic, u8 frame)
{
    int i;
    for(i=0; i<200; i++)
    {
        g_dest_data[i] = wave_end_table[new_pic[i]][frame];
    }
}
```

Listing 7. Włączenie napięć zasilających
```
void EPD_PowerOn(void)
{
    GPIO_SetBits(VNEGGVEE_CTR_PORT, VNEGGVEE_CTR);  //pa1
    Delay(0xf);
    GPIO_SetBits(VPOS15_CTR_PORT, VPOS15_CTR);  //pa2
    Delay(0xf);
    GPIO_SetBits(GVDD22_CTR_PORT, GVDD22_CTR);  //pa3
    Delay(0xf);
    GPIO_SetBits(VCOM_CTR_PORT, VCOM_CTR);  //VCOM = -V
    Delay(0xff);
}
```

Listing 8. Wyłączenie napięć zasilających
```
void EPD_PowerOff(void)
{
    GPIO_ResetBits(VCOM_CTR_PORT, VCOM_CTR);  //VCOM = 0
    Delay(0xff);
    GPIO_ResetBits(GVDD22_CTR_PORT, GVDD22_CTR);
    Delay(0xf);
    GPIO_ResetBits(VPOS15_CTR_PORT, VPOS15_CTR);
    Delay(0xf);
    GPIO_ResetBits(VNEGGVEE_CTR_PORT, VNEGGVEE_CTR);
    Delay(0xff);
}
```

Na koniec

Opisałem sposób, którym można w praktyce posłużyć się do sterowania wyświetlaczem e-paper. W testowanym wyświetlaczu producent zastosował driver HX8705 pozwalającym tylko na zapisywanie danych do re-



Rysunek 13. Schemat układu zasilania

Rysunek 14. Bitmapa otwarta w programie uC/GUI – Bitmap Convert

jestru przesuwnego odpowiadającego za wyświetlanie jednej linii i na automatyczne wybieranie kolejnych linii. Wykonywanie wszystkich sekwencji wyświetlania oraz organizacja pamięci obrazu spoczywa na programie sterującym. Nie jest to zadanie trudne do wykonania dla szybkiego i dobrze wyposażonego mikrokontrolera, ale zajmuje czas procesora. Jednak w układach z uruchomionym systemem RTOS lub w aplikacjach opartych na przerwaniach może to nie mieć większego znaczenia. Jak zwykle w przypadku wyświetlaczy o dużych rozdzielczościach, problemem jest duża wielkość plików z bitmapami. W praktyce zapewne trzeba będzie wyposażyć układ sterowania w pamięć masową, na przykład kartę SD zapisanymi bitmapami lub złącze USB, po którym będą przesyłane dane bitmap.

Sam wyświetlacz charakteryzuje się znakomitym kontrastem oraz dość długim czasem „wyświetlania” informacji bez odświeżania. Jednocześnie trzeba sobie zdawać sprawę z tego, że zmiana wyświetlanej informacji wymaga (w porównaniu z typowym LCD) relatywnie długiego czasu, a w trakcie zmiany wyświetlacz najpierw robi się czarny, a potem biały i w końcu pojawia się wyświetlana informacja.

Przedstawiony tutaj sposób wyświetlania uwzględnia tylko zapisanie całej bitmapy. Nie ma możliwości zapisania tylko pewnego obszaru po to, aby na przykład wyświetlić fragment tekstu. By realizować takie funkcje trzeba by było utworzyć bufor w pamięci RAM mieszczący całą bitmapę, modyfikować ten bufor i w całości zapisywać do wyświetlacza. Bitmapa wielkości 800×600 pikseli zajmuje 120 tys. bajtów. Realizacja tak dużego bufora w pamięci RAM mikrokontrolera może być trudna. Dlatego taki wyświetlacz najlepiej sprawdzi się w aplikacjach, które nie wymagają częstych zmian wyświetlanej informacji, ale ważny jest za to bardzo mały pobór prądu oraz duży kontrast przy braku podświetlania. Mamy tu też do czynienia z właściwością do tej pory niespotykaną w elementach tego typu: wyświetlana informacja pozostaje czytelna przez długi czas (przez dni a nawet tygodnie) po całkowitym zaniku zasilania i to bez utraty jakości! Daje to możliwość wielu nowych zastosowań.

Tomasz Jabłoński, EP