Stacjonarny odtwarzacz MP3 z budzikiem (3)

Informacje dla dociekliwych

Trzecia część artykułu o odtwarzaczu MP3 przeznaczona jest dla Czytelników, którym nie wystarcza złożenie urządzenia i zaprogramowanie mikrokontrolera gotowym programem wynikowym. Do lektury tej części zachęcam wszystkich Czytelników, którzy chcą lepiej poznać działanie odtwarzacza lub zmodyfikować jego kod źródłowy. Omawiana jest tutaj zasada działania najważniejszych bloków odtwarzacza oraz opisane są sposoby modyfikacji fragmentów kodu, które warto dopasować do własnych wymagań.

AVT–5195

W ofercie AVT:
AVT–5195A – płytka drukowana

PODSTAWOWE PARAMETRY
• Układ zbudowany na 3 płytkach: klawiatura, płytka główna, przetwornik C/A
• Zdalnie sterowany pilotem podczerwieni
• Dopasowany do możliwości typowego, domowego zestawu audio
• Mikrokontroler AT91SAM9260
• Odtwarzanie utworów nagranych na karcie SD
• Zegar czasu rzeczywistego z funkcją budzika

PROJEKTY POKREWNE wymienione artykuły są w całości dostępne na CD

Tytuł artykułu	Nr EP/EdW	Kit
Kieszonkowy odtwarzacz MP3	EP 3-4/2003	---
Magnetofon cyfrowy DAR-001	EP 6-9/2007	---
Yampp-7 Kieszonkowy odtwarzacz MP3	EP 3-4/2003	---
Yampp-3	EP 12/2002-2/2003	---
Odtwarzacz MP3 z układem STA013	EP 8-9/2003	---
Yampp 3	EP 9-10/2002	---

Narzędzia i organizacja projektu

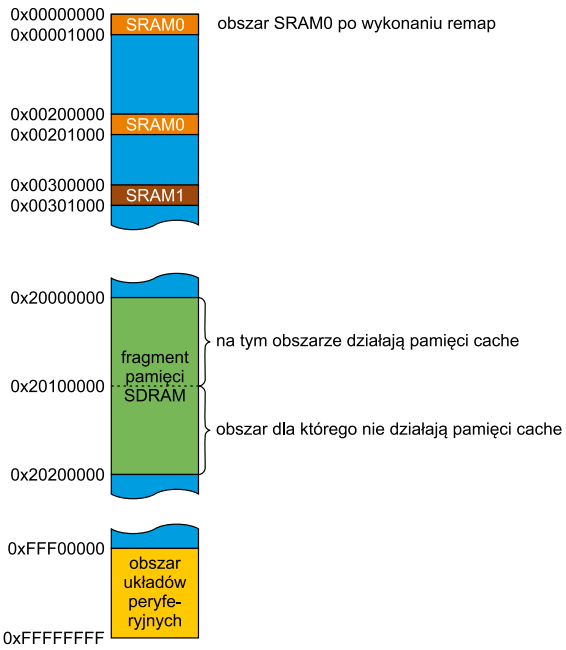
Wykaz wszystkich katalogów projektu i jego najważniejszych plików wraz z krótkimi ich opisami znajduje się w **tab. 1**. Może on być pomocy w lokalizacji interesujących fragmentów kodu. Cały program odtwarzacza MP3 kompilowany był za pomocą darmowego kompilatora GCC z pakietu WinARM. Jeśli używamy już WinARM-a do pracy z popularnymi mikrokontrolerami ARM7TDMI (np. AT91SAM7, LPC2k itp), to zasiadając do pracy z AT91SAM9260, nie musimy wykonywać żadnej dodatkowej konfiguracji tego pakietu. Wszystkie potrzebne ustawienia zawarte zostały w pliku Makefile projektu odtwarzacza.

Odtwarzacz MP3 dysponuje bardzo dużą ilością pamięci RAM, lecz wykorzystywana jest jedynie jej niewielka część. Omawiany w pierwszej części artykułu program bootloadera MP3_Player_Bootloader inicjalizuje jednostkę MMU mikrokontrolera w taki sposób, aby dla kodu odtwarzacza dostępna była pamięć SDRAM (32 MB od adresu 0x20000000), pamięć SRAM0 (od adresu 0x00000000 oraz od 0x00200000), SRAM1 (od 0x00300000) i układy peryferyjne (najwyższy megabajt przestrzeni adresowej). Natomiast skrypt linkera odtwarzacza (plik sdram.lds) pozwala na zagospodarowanie jedynie pierwszych dwóch megabajtów pamięci SDRAM. Przy dostępach do zakresu adresów 0x20000000...0x200FFFFF (pierwszy megabajt SDRAM) rdzeń będzie używał pamięci ICache oraz DCache (są to odpowiednio pamięci cache dla instrukcji i dla danych). Natomiast w przypadku wyższych adresów 0x20100000...0x201FFFFF (drugi megabajt SDRAM) pamięci cache nie będą działały.

Dla ścisłości należy dodać, że w projekcie odtwarzacza MP3, mimo zastosowania mikrokontrolera z jednostką MMU, nie jest wykorzystywana żadna wyszukana translacja adresów. Podane wcześniej zakresy adresów są zarówno adresami wirtualnymi, jak i fizycznymi. Oznacza to, że edytując kody źródłowe

Tab. 1. Wykaz najważniejszych katalogów i plików wchodzących w skład projektu odtwarzacza

Katalog	Pliki	Krótki opis zawartości
Główny	main.c	funkcja main, inicjalizacja podstawowych sterowników, utworzenie zdania głównego
	board.h, common.h	często wykorzystywane i globalne dla całego programu makrodefinicje oraz dyrektywy #include
	syscalls.c	funkcje umożliwiające działanie niektórych bibliotek języka C
	FreeRTOSConfig.h	makrodefinicje konfiguracyjne dla systemu FreeRTOS
	user_if.c, user_if.h	realizacja interfejsu użytkownika odtwarzacza, realizacja innych ważnych zadań takich jak obsługa modułów odtwarzacza
	player_task.c, player_task.h	zadanie systemu operacyjnego realizujące odtwarzanie MP3 oraz funkcje wysokiego poziomu sprawujące kontrolę nad odtwarzaniem
	player_core.c, player_core.h	„rdzeń" odtwarzacza MP3: funkcje realizujące obsługę biblioteki Helix, nadzorujące pracę przetwornika DAC i pobieranie danych z plików MP3
	id3.c, id3.h	odczyt tagów ID3v1 z plików MP3
include	AT91SAM9260.h, lib_AT91SAM9260.h	pliki nagłówkowe z makrodefinicjami specyficznymi dla układów peryferyjnych mikrokontrolera, strukturami danych reprezentującymi te układy i funkcjami inline do niskopoziomowej ich obsługi
	sdram.lds	skrypt linkera GCC (LD) umożliwiający wykonywanie programu i przechowywanie danych w zewnętrznej pamięci SDRAM
mp3dec	biblioteka dekodowania plików MP3 (Helix MP3 decoder)	
	mp3dec.c	wysokopoziomowe funkcje do dekodowania i analizy strumienia danych w formacie MP3 („interfejs użytkownika" biblioteki)
freertos	pliki wchodzące w skład systemu operacyjnego FreeRTOS oraz jego portu dla mikrokontrolerów AT91SAM	
fat	elementy biblioteki FatFS do obsługi systemu plików FAT	
drivers	moduły sterowników układów peryferyjnych	
	backlite.c, backlite.h	obsługa podświetlenia wyświetlacza LCD za pomocą modułu Timer Counter
	cstartup.S, Cstartup_SAM9.c	niskopoziomowa inicjalizacja mikrokontrolera oraz obsługa niektórych wyjątków
	dbgu.c, dbgu.h	funkcje odpowiedzialne za realizację komunikacji „serwisowej" przez port szeregowy z modułu DBGU
	delay.c, delay.h	opóźnienia w trybie polling uzyskane za pomocą modułu Timer Counter
	i2c.c, i2c.h	programowa implementacja komunikacji przez I²C przy użyciu linii PIO
	keyboard.c, keyboard.h	obsługa klawiatury
	rc5_ir.c, rc5_ir.h	odbiór kodów RC5 z pilota zdalnego sterowania
	remote_control.c, remote_control.h	„most" pomiędzy funkcjami obsługi pilota a pozostałymi elementami systemu; w tych plikach można podmienić obsługę pilota
	rtc.c, rtc.h	obsługa układu odmierzania czasu rzeczywistego (tutaj DS1307)
	stereo_dac.c/h, stereo_dac_isr.c/h	obsługa przetwornika cyfrowo-analogowego
	time.c, time.h	funkcje pomocnicze do odmierzania czasu rzeczywistego



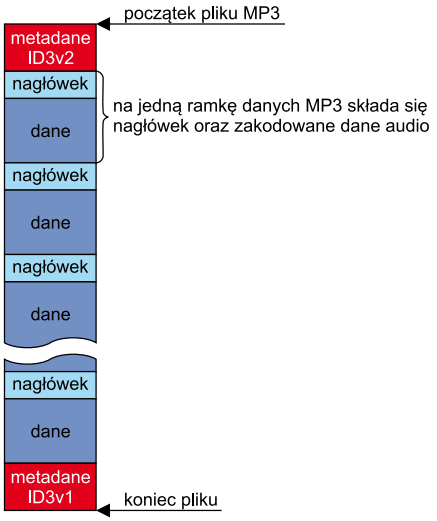
Rys. 12. Organizacja przestrzeni adresowej w projekcie odtwarzacza

odtwarzacza, możemy postrzegać przestrzeń adresową tak jak w małych, prostych i dobrze już znanych mikrokontrolerach. Na **rys. 12** przedstawiona została graficzna reprezentacja przestrzeni adresowej widzianej w głównym programie odtwarzacza.

Dostarczony Czytelnikom program bootloadera MP3_Player_Bootloader.bin omawiany w pierwszej części artykułu jest wyposażony w opcję uruchamiania programów testowych z pamięci SDRAM bez konieczności każdorazowego programowania pamięci DataFlash. Wejście do trybu uruchamiania programu testowego odbywa się przez przytrzymanie wciśniętego przycisku SW1 (STOP/BACKLITE) w momencie zerowania lub włączania zasilania odtwarzacza. Uruchamianie programu testowego polega na umożliwieniu użytkownikowi wysłania dowolnego wykonywalnego programu do pamięci SDRAM (pod adres 0x20000000) za pomocą programu terminalowego (protokół XModem), a następnie uruchomieniu skopiowanego programu. Ładowane w ten sposób programy nie trafiają do pamięci DataFlash. Dlatego ponowne uruchomienie odtwarzacza bez wciśniętego przycisku SW1 spowoduje załadowanie domyślnego programu znajdującego się w pamięci DataFlash – w ten sposób po uruchamianiu programu testowego nie będzie śladu.

Przed wejściem w tryb uruchamiania programu testowego należy podłączyć port DBGU mikrokontrolera portu szeregowego komputera PC w taki sposób, jak do programowania pamięci odtwarzacza przy jego uruchamianiu (procedura opisana w pierwszym artykule). Wejście do trybu testowego zasygnalizowane będzie na terminalu (115200 baud, 8n1) następującymi komunikatami:

MP3 Player by RABW, 2009



Rys. 13. Schemat przykładowej zawartości pliku MP3

SDRAM Bootloader. Send file to SDRAM

Następnie na terminalu zaczną pojawiać się znaki „C" oznaczające gotowość urządzenia do odbioru danych za pomocą protokołu XModem z 16-bitowymi sumami kontrolnymi. Wysłany do odtwarzacza program (plik wynikowy typowo z rozszerzeniem .bin) zostanie umieszczony w pamięci SDRAM, a następnie uruchomiony przez skok do adresu 0x20000000. Gdy opracujemy już zadowalającą nas nową wersję oprogramowania dla odtwarzacza, nowy program można będzie umieścić w pamięci DataFlash wg algorytmu opisanego w pierwszej części artykułu. Z pamięci DataFlash będzie on już automatycznie uruchamiany z każdym włączeniem zasilania bądź resetowaniem odtwarzacza.

Zawartość plików MP3

Przystępując do analizy działania odtwarzacza, warto wiedzieć, co znajduje się w najczęściej spotykanych plikach MP3. Typowy plik MP3 składa się z danych audio oraz dodatkowych informacji zawartych np. w tagach ID3 (**rys. 13**). Tagi, które można znaleźć najczęściej na początku (np. ID3v2) i na końcu (ID3v1) pliku, pozwalają przechowywać dane m.in. o albumie, wykonawcy i tytule utworu.

Właściwe dane audio przechowywane są „strumieniowo" w tzw. ramkach (frames). Każda ramka składa się z nagłówka (header) i bloku danych (data block). Ramki z danymi nie muszą mieć takich samych rozmiarów, dlatego trudno jest jednoznacznie określić, gdzie w danym pliku znajdzie się początek kolejnej ramki. Do rozpoznawania początku ramki oraz do wstępnej analizy jej zawartości służy nagłówek ramki. Nagłówek taki ma rozmiar 32 bitów i rozpoczyna się od pola bitowego obowiązkowo zawierającego sekwencję 12 bitów „1". W omawianym odtwarzaczu MP3, początki ramek są wstępnie rozpoznawane po wystąpieniu tej sekwencji. Mimo że liczba zakodowanych danych w jednej ramce nie musi być jednakowa nawet w obrębie jednego pliku, to jednak liczba danych po zdekodowaniu jest zdefiniowana i może wynosić: 384, 576 lub 1152 próbek z jednej ramki.

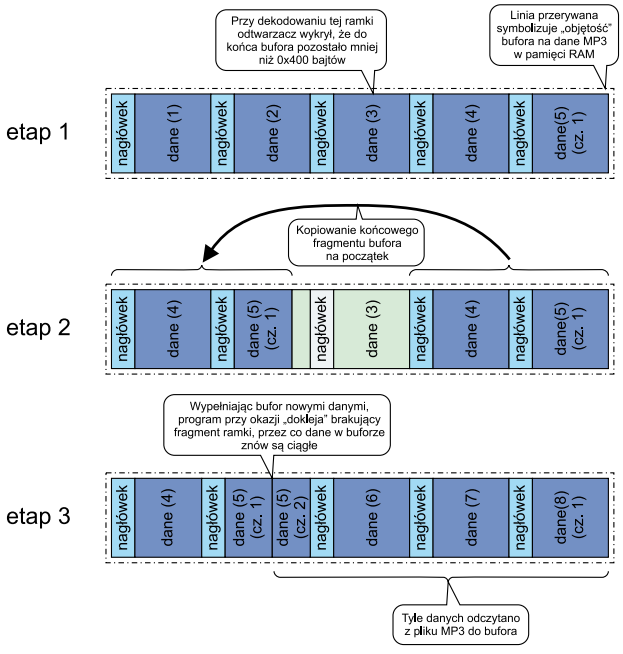
Realizacja dekodowania MP3 za pomocą dekodera Helix

Algorytm dekodowania danych MP3 daleko wykracza poza ramy tego artykułu, więc w dalszej części skoncentruję się na opisie realizacji odtwarzania MP3 za pomocą dekodera Helix zastosowanego w odtwarzaczu. W tym momencie także znajomość samego algorytmu dekodowania MP3 jest mało istotna dla Czytelników chcących zbudować własny odtwarzacz bądź zmodyfikować projekt odtwarzacza zaproponowany w części praktycznej artykułu. Mimo że dekodera Helix można używać jako czarnej skrzynki, to i tak jego zastosowanie we własnym odtwarzaczu jest nieco trudniejsze od wykorzystania sprzętowych dekoderów MP3, takich jak np. układy VS1001.

Dekodowanie danych MP3 w omawianym odtwarzaczu realizowane jest w funkcji playerCore w module palyer_core.c za pomocą wywołania funkcji o nazwie MP3Decode. Jednym z argumentów pobieranych przez MP3Decode jest „podwójny wskaźnik" (wskaźnik do wskaźnika) do bufora z danymi MP3. Wskaźnik ma być tak ustawiony, aby pokazywał na początek ramki MP3. W typowych warunkach wskaźnik ten przesuwany jest automatycznie przez dekoder (po to stosowany jest wskaźnik podwójny). Wspomniany bufor z danymi MP3 (tablica mp3Buffer typu uint8_t) w omawianej implementacji ma stały rozmiar 5 KB (makrodefinicja MP3_BUFFER_SIZE).

W projekcie odtwarzacza MP3 dane do bufora pobierane są z plików znajdujących się na karcie SD/SDHC, a obsługiwanym systemem plików jest FAT (biblioteka FatFS). Ze specyfiki systemu plików FAT wynika, że najszybszy jest odczyt pliku od jego początku w kierunku końca, natomiast próba „cofnięcia się" z odczytem danych z pliku może trwać dłużej. Teraz pojawia się pytanie, co zrobić, gdy wypełniając bufor danymi z karty, nie zmieścimy w nim całkowitej liczby ramek? Sytuacja jest jak najbardziej prawdopodobna, ponieważ ramki mogą mieć różne rozmiary, a odczytując „hurtowo" dane z karty SD do bufora mp3Buffer, nie wiemy, gdzie zaczyna się, a gdzie kończy ramka MP3. Teoretycznie można by odczytywać z pliku dane „bajt po bajcie" w poszukiwaniu nagłówka ramki, lecz jest to raczej karkołomne zadanie – tak dla programisty, jak dla mikroprocesora. Jedno z rozwiązań tego problemu polega na sprawdzaniu, ile bajtów z bufora danych MP3 pozostało jeszcze do odtworzenia. Jeśli bufor będzie się „wyczerpywał", odtwarzacz przejdzie do procedury ponownego wypełniania bufora. Jako graniczną liczbę danych, poniżej której trzeba ponownie napełnić bufor, przyjęto wartość 1024 (0x400) bajty, czyli 1/5 całej objętości bufora. Procedura ponownego napełniania bufora realizowana jest w funkcji getMoreMp3Data znajdującej się w pliku player_core.c. Ponowne wypełnianie bufora polega na skopiowaniu danych, które w nim pozostały do odtworzenia na jego początek. Następnie do skopiowanej zawartości „doklejane" są kolejne dane odczytane z pliku MP3. Odtwarzanie kolejnych ramek rozpocznie się od dokładnie tych samych danych, lecz w innym miejscu bufora. Graficzna reprezentacja tego algorytmu znajduje się na **rys. 14**.

Kolejnym zadaniem w oprogramowywaniu odtwarzacza MP3 jest odpowiednie znajdowanie początków ramek i radzenie sobie z ewentualnymi sytuacjami, gdy nagłówek ramki nie zostanie znaleziony. Nagłówki wstępnie znajdowane są za pomocą funkcji MP3FindSyncWord wchodzącej w skład biblioteki Helix. Funkcja ta jest dość prosta, bo poszukuje jedynie synchronizacyjnego ciągu 12 bitów o wartości „1" w buforze z danymi MP3. Ramki ustawione są „jedna po drugiej", więc typowo ciąg synchronizacji powinien być bez problemu odnajdywany najczęściej w aktualnej pozycji wskaźnika bufora (dekoder sam przesuwa wskaźnik). Czasem jednak poprawne odnalezienie ciągu 12 bitów ustawionych na „1" nie musi oznaczać znalezienia poprawnego nagłówka ramki. Dlatego po odnalezieniu początku nagłówka poprawność danych jest jeszcze przed zdekodowaniem dodatkowo sprawdzana funkcją MP3GetNextFrameInfo. Funkcja MP3GetNextFrameInfo przy okazji uaktualnia dane w strukturze mp3Frame-



Rys. 14. Sposób pobierania danych z pliku MP3 do bufora w pamięci RAM

Info typu MP3FrameInfo przekazanej w jednym z argumentów, dzięki czemu można sprawdzić parametry ramki (np. liczbę kanałów, próbek wyjściowego sygnału – definicja typu MP3FrameInfo znajduje się w pliku mp3dec.h). Jeśli na tym etapie pojawi się błąd, to, w zależności od liczby danych pozostałych w buforze, podejmowane są próby uzupełnienia bufora albo poszukiwania kolejnych nagłówków.

Po znalezieniu i sprawdzeniu nagłówka, przeprowadzone zostaje dekodowanie ramki za pomocą funkcji MP3Decode. Najistotniejsze argumenty przekazywane do tej funkcji to: wspomniany wcześniej podwójny wskaźnik do bufora z danymi MP3 oraz zwykły wskaźnik do bufora na wyjściowy sygnał audio (waveBuffer). Nie jest przypadkiem, że format danych wyjściowych z dekodera dokładnie odpowiada formatowi danych, które przyjmuje przetwornik DAC. Jest to bardzo typowe ułożenie danych spotykane np. w plikach popularnego formatu WAVE o jakości nazywanej „jakością CD". Próbka jednego kanału kodowana jest na 16 bitach ze znakiem. Próbki kanału lewego i prawego umieszczone są naprzemiennie. Do funkcji MP3Decode oprócz wskaźników do buforów przez referencję przekazywana jest także zmienna bytesLeft zawierająca liczbę jeszcze nieodczytanych bajtów z bufora oraz struktura danych mp3DecoderData typu HMP3Decoder zawierająca aktualne dane dekodera.

Funkcja MP3Decode zwraca wartość informującą program o przebiegu dekodowania. Definicje wszystkich wartości zwracanych przez MP3Decode znajdziemy w pliku mp3dec.h. Wartość 0 (ERR_MP3_NONE) oznacza poprawnie przeprowadzone dekodowanie i możliwość wysłania uzyskanych danych audio do przetwornika cyfrowo-analogowego. Wartości ujemne stanowią komunikaty o statusie dekodowania wymagającym uwagi czy bardziej lub mniej dotkliwych w skutkach błędach.

Czasem zdarza się, że pliki MP3 zawierają błędne dane, mają nadzwyczaj rozbudowane obszary metadanych, takie jak tagi ID3v2 (szczególnie uciążliwe mogą być metadane na początku pliku), czy wreszcie mają „urwane" zakończenie. Od dobrego odtwarzacza wymaga się, aby potrafił zignorować długie obszary metadanych, nie zawiesił się w przypadku błędnych ramek oraz potrafił znaleźć kolejne poprawne ramki w razie błędu. Obsługa błędów w omawianym projekcie odtwarzacza dostrojona została w dużej mierze w oparciu o testy z różnymi plikami MP3, lecz mimo wszystko nie daje to gwarancji, że odtwarzacz będzie zawsze niezawodnie działał w każdej sytuacji. W kodzie obsługi błędów pozostało również trochę niepotrzebnego kodu z czasów, kiedy odtwarzacz był projektem bardziej eksperymentalnym niż użytkowym (np. sprawdzającego pewnego rodzaju błędy tam, gdzie raczej one nie wystąpią). Jeśli stwierdzimy, że urządzenie na pewno powinno poprawnie odtwarzać dany plik MP3, a z jakiegoś powodu np. pomija ten plik, warto wyświetlić za pomocą programu terminalowego i przeanalizować komunikaty diagnostyczne wysyłane przez port DBGU mikrokontrolera oraz zajrzeć do zawartości m.in. funkcji playerCore w celu sprawdzenia przebiegu programu.

Jeśli dekodowanie danej ramki MP3 zakończy się powodzeniem, to funkcja playerCore oblicza liczbę zdekodowanych próbek, które należy odtworzyć oraz dostarcza zdekodowane dane audio do przetwornika DAC (wywołanie funkcji dacFeed).

Tempo odtwarzania narzucane jest przez moduł SSC (Synchronous Serial Controller) skonfigurowany tak, by uzyskać popularny interfejs I²S dla przetwornika DAC. Oczywiście przy wysyłaniu danych do przetwornika wykorzystany jest kanał DMA stowarzyszony z SSC. Skorzystano tutaj z możliwości podwójnego buforowania oferowanej przez kontroler PDC (Peripheral DMA Controller) nadzorujący transfery DMA. W czasie, gdy jeden z buforów jest wysyłany, trwa również dekodowanie strumienia MP3 – dzięki temu bez przerwy w czasie odtwarzania jeden z buforów jest wysyłany do przetwornika, a drugi służy dekoderowi MP3 do wpisywania zdekodowanych danych audio.

Gotowość modułu SSC do przyjęcia nowych danych powoduje zgłoszenie odpowiedniego przerwania oraz (w funkcji jego obsługi) ustawienie semafora systemu operacyjnego FreeRTOS. Ustawiony tam semafor (obiekt dacSemaphore w pliku stereo_dac_isr.c) jest znakiem dla funkcji playerCore wywoływanej w pętli zadania playerTask (moduł player_task.c), że należy przystąpić do dekodowania kolejnej ramki oraz uzyskane dane audio „zarejestrować" za pomocą funkcji dacFeed jako oczekujące na transfer DMA.

Jak widać, mikrokontroler w czasie odtwarzania MP3 musi wykonywać wiele czynności na raz. Dane z pamięci SDRAM do przetwornika DAC muszą być stale wysyłane, aby uzyskać ciągły sygnał audio – bez trzasków czy przerw. Jednocześnie mikrokontroler musi wykonywać dekodowanie kolejnych ramek MP3. Zarówno bufor z danymi MP3 odczytywanymi, z karty jak i wykonywany program mikrokontrolera znajdują się w pamięci SDRAM. W takim projekcie jak omawiany odtwarzacz pamięci cache mikroprocesora mają naprawdę duże znaczenie. Dzięki nim korzystamy z zalet harwardzkiej architektury rdzenia ARM926EJ-S oraz odciążamy kontroler pamięci SDRAM od ciągłego pobierania danych (szczególnie programu). Analizując kod odtwarzacza, warto zwrócić uwagę na fakt, że bufor na zdekodowane dane audio (waveBuffer) zdefiniowany jest tak, aby linker umieścił go w specjalnie utworzonej sekcji o nazwie .buffers. Sekcja ta umieszczona jest w obszarze SDRAM, dla którego nie są aktywne pamięci cache, ponieważ zawartość waveBuffer ma być transmitowana przez kanał DMA. W niektórych konfiguracjach pamięci cache odczyt przez kanał DMA danych, których kopia znajduje się w pamięci cache, może prowadzić do niezgodności danych odczytanych z RAM i zawartych w cache.

Komunikacja pomiędzy zadaniami odtwarzacza

Jak zostało wspomniane, funkcja playerCore stanowiąca „rdzeń" odtwarzacza wy-

Tab. 2. Wykaz kodów pilota podczerwieni

przycisk	kod RC5	mnemonik	stan „odtwarzanie"	stan „zatrzymany"	stan „ustawianie zegara"
SW1	0x0C	STOP_ENTER	przejście do stanu „zatrzymany"	wyłączenie/włączenie podświetlenia LCD	zapisanie ustawień i przejście do stanu „zatrzymany"
SW2	0x37	PLAY_PAUSE_ESC	restart odtwarzanego utworu	przejście do stanu „odtwarzanie"	przejście do stanu „zatrzymany" bez zapisywania ustawień
SW3	0x3C	FAST_FORWARD	przewijanie do przodu	godzina alarmu ++	godzina zegara ++
SW4	0x3F	REWIND	przewijanie do tyłu	godzina alarmu —	godzina zegara —
SW5	0x10	NEXT_TRACK	następny utwór	minuta alarmu ++	minuta zegara ++
SW6	0x11	PREVIOUS_TRACK	poprzedni utwór	minuta alarmu —	minuta zegara —
SW7	0x20	NEXT_DIR	następny katalog	przejście do stanu „ustawianie zegara"	–
SW8	0x21	PREVIOUS_DIR	poprzedni katalog	włączenie/wyłączenie alarmu	–
–	0x2E	NEXT_10_DIRS	skok 10 katalogów do przodu	–	–
–	0x2B	PREVIOUS_10_DIRS	skok 10 katalogów do tyłu	–	–
–	0x1E	PLAY_ALL_DIRS	odtwarzanie utworów z jednego katalogu/odtwarzanie utworów ze wszystkich katalogów	–	–

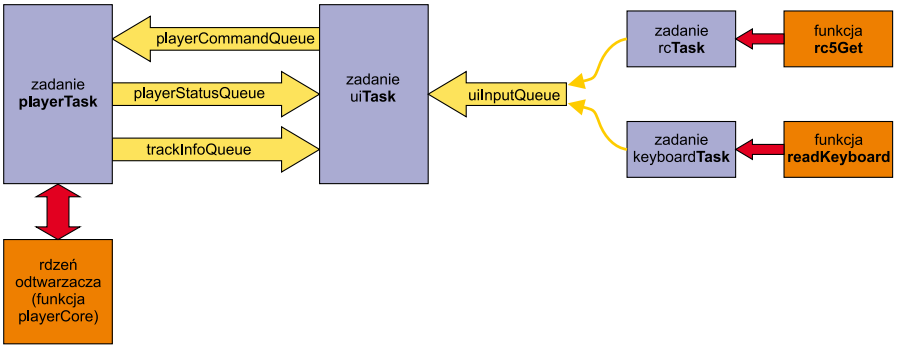
wołowana jest w pętli zadania playerTask. Natomiast zadaniem odpowiedzialnym za zapewnienie interfejsu użytkownika jest uiTask zdefiniowany w pliku user_if.c. Dzięki systemowi operacyjnemu FreeRTOS oba te zadania wykonywane są pozornie „jednocześnie". Samo wywoływanie funkcji playerCore w zadaniu playerTask nie wystarcza do poprawnego działania odtwarzacza. Trzeba także jakoś kontrolować odtwarzanie: użytkownik musi mieć możliwość uruchamiania, zatrzymywania odtwarzania lub przełączania aktualnego utworu.

Do realizacji komunikacji pomiędzy zadaniem playerTask a uiTask służą trzy kolejki (**rys. 15**). Pierwsza z nich reprezentowana jest przez obiekt playerCommandQueue i jest przeznaczona do wysyłania poleceń z interfejsu użytkownika do modułu odtwarzacza. Moduł odtwarzacza może wysyłać informacje zwrotne lub niezależne komunikaty za pomocą kolejki reprezentowanej obiektem playerStatusQueue. Trzecia kolejka (trackInfoQueue) służy do przesyłania informacji na temat odtwarzanego utworu – w zadaniu uiTask dane przez nią przesłane służą do wyświetlenia nazwy wykonawcy oraz tytułu albumu i utworu.

Moduł interfejsu użytkownika pobiera dane z dwóch urządzeń wejściowych: z klawiatury oraz odbiornika sygnału pilota zdalnego sterowania. Oba te urządzenia przekazują swoje dane do zadania uiTask za pomocą jednej kolejki reprezentowanej obiektem uiInputQueue. Takie rozwiązanie pozwala w prosty sposób zmodyfikować dotychczasowe sterowniki urządzeń wejściowych lub nawet dopisać niezależny sterownik innego urządzenia wejściowego (można np. zaimplementować sterowanie odtwarzacza za pomocą portu szeregowego).

Odbiorem kodu RC5 z pilota zdalnego sterowania zajmują się funkcje z pliku rc5_ir.c. Do interpretacji sygnału RC5 używane jest zewnętrzne przerwanie IRQ1 oraz moduł układu Timer Counter 0 (TC0). Funkcją „interfejsową" pliku rc5_ir.c jest rc5Get – tej funkcji używa moduł remote_control.c pośredniczący w wymianie danych pomiędzy pilotem a pozostałymi modułami odtwarzacza. Moduł remote_control.c odbiera dane z pilota RC5, tłumaczy odebrane kody na polecenia, które będą zrozumiałe dla interfejsu użytkownika, a tak przygotowane polecenia wysyła do kolejki uiInputQueue. Za ciągły odbiór danych z pilota odpowiada w tym module zadanie o nazwie rcTask.

Zmiana przypisania kodów RC5 z pilota do konkretnych poleceń w najprostszy sposób może być przeprowadzona przez edycję funkcji convertRcToUiData z pliku remote_control.c. W funkcji tej znajduje się jedynie nieco rozbudowana instrukcja switch realizująca przypisanie kodów pilota (wartości wybierane) do kodów rozumianych przez moduł



Rys. 15. Schemat przepływu danych pomiędzy głównymi zadaniami odtwarzacza

interfejsu użytkownika (wartości zwracane). Kody pilota wymieniono w **tab. 2**.

Funkcje interfejsu użytkownika

Najważniejsze funkcje odpowiadające za zapewnienie interfejsu użytkownika znajdują się w pliku user_if.c.

Podstawowe ustawienia odtwarzacza przechowywane są w strukturze danych o nazwie playerSettings. Zawartość całej tej struktury może być przechowywana w pamięci RAM układu zegara czasu rzeczywistego DS1307 ze względu na podtrzymanie bateryjne jego pamięci. Dzięki temu, nawet po całkowitym odłączeniu zasilania sieciowego odtwarzacz może automatycznie przywrócić swoje ustawienia. Struktura playerSettings jest kopiowana do pamięci RAM układu RTC za pomocą funkcji saveSettings, a przywracanie jej z powrotem do pamięci operacyjnej mikrokontrolera odbywa się przez wywołanie funkcji restoreSettings. Elementy struktury playerSettings przechowują następujące informacje:

- godzina i minuta działania budzika (elementy alarmHour i alarmMin),
- aktywność budzika (alarmOn),
- odtwarzany utwór (element track) oraz przypisany mu katalog (dir),
- całkowita liczba katalogów na karcie (numberOfDirs) i liczba plików MP3 w aktualnym katalogu (numberOfTracks),
- aktywność opcji odtwarzania wszystkich utworów z katalogu (allDirs),
- wartość „magiczna" (element magic o domyślnej wartości 0xABCDEF58) używana do prostego sprawdzania, czy odczytana zawartość struktury playerSettings jest poprawna.

„Lista odtwarzania" utworzona jest za pomocą dwóch dwuwymiarowych tablic. Jedna z tych tablic nosi nazwę dirList i służy do przechowywania listy wszystkich nazw katalogów na karcie SD/SDHC. Jej zawartość można określić jako „statyczną", ponieważ tworzona jest podczas inicjalizacji odtwarzacza i nie zmienia się w trakcie jego działania (nie została zaimplementowana możliwość zmiany karty pamięci w czasie działania urządzenia). Drugą tablicą „listy odtwarzania" jest tablica o nazwie trackList. Znajdują się w niej nazwy plików z katalogu, z którego w danym momencie odtwarzane są utwory. Zawartość tablicy trackList jest wpisywana na nowo wraz z każdą zmianą aktualnego katalogu. W obecnej wersji oprogramowania odtwarzacza nie zaimplementowano żadnego algorytmu sortowania katalogów ani plików w „listach odtwarzania". Katalogi i pliki w tablicach ułożone są w taki sposób, w jaki są one odczytywane z karty SD/SDHC, więc także w kolejności ich nagrywania na kartę. Wypełnianie tablic danymi odbywa się przez wywołanie funkcji fsListAllFiles zdefiniowanej w pliku fs_tools.c z podkatalogu fat projektu.

Elementy track i dir struktury danych playerSettings zawierają indeksy odpowiednio w tablicy trackList i dirList odpowiadające aktualnie odtwarzanemu utworowi. Z kolei elementy numberOfDirs i numberOfTracks określają liczbę wszystkich wpisów w tablicach dirList i trackList. W celu „namierzenia" danego utworu na karcie pamięci program musi znać jego indeksy w obu tablicach. Znając indeksy, można odczytać nazwę katalogu i nazwę pliku utworu, który ma być odtwarzany. W ten sposób program odtwarzacza „składa" ścieżkę otwieranego pliku MP3 z nazwy katalogu, znaku „/" i nazwy pliku.

Do wyświetlania informacji o aktualnie odtwarzanym utworze używane są: albo nazwa pliku i katalogu zawarte w omawianych tablicach, albo dane odczytane z taga ID3, jeśli tag ID3 został poprawnie odczytany. O ile nazwy odczytane z metadanych ID3v1 pomimo swoich ograniczeń (np. długość nazw do 30 znaków) są czytelne, o tyle nazwy plików i katalogów zawartych tablicach trackList i dirList czytelnymi nazwać nie można. Jest tak z powodu stosowania nazw plików i katalogów określanych jako „nazwy MS-DOS" – tylko taki format obsługiwany jest bowiem przez zastosowaną w odtwarzaczu wersję biblioteki obsługi systemu plików FAT.

Jeśli zechcemy zmienić wygląd ekranu odtwarzacza, możemy to uczynić, edytując następujące funkcje z pliku user_if.c:

- displayCurrentTrackInfo (ekran w trybie odtwarzania),
- displayIdleScreen (ekran w trybie „zatrzymany"),
- displaySplashScreen (ekran „powitalny"),

– displayTimeSettingScreen (ekran przy ustawianiu zegara),
– displayTimeMessage (ekran komunikatu po zakończeniu ustawiania zegara).

Posługiwanie się funkcjami obsługi wyświetlacza LCD nie powinno nastręczać większych trudności. Do obsługi wyświetlacza zastosowałem prościutką, lecz dość funkcjonalną „bibliotekę graficzną" własnego autorstwa. Wszystkie pliki wchodzące w jej skład znajdziemy w podkatalogu drivers/lcd. Minibibliotekę LCD można z powodzeniem „rozgryźć", choćby pobieżnie analizując treść wyżej wymienionych funkcji wyświetlających komunikaty w odtwarzaczu lub przeglądając kody źródłowe głównie plików lcd_basic.c i lcd_print.c z katalogu drivers/lcd.

Tagi ID3

Pozostało do omówienia jeszcze jedno, dość ciekawe zagadnienie – tagi ID3. Są to metadane dopisywane do plików MP3 w celu ułatwienia przechowywania tekstowych informacji na temat utworu. Metadane można tutaj rozumieć jako „opis zawartości". Dzięki nim w pliku MP3 można zapisać w czytelnej dla człowieka postaci m.in.: pełny tytuł utworu, nazwę wykonawcy, tytuł albumu, z którego pochodzi utwór, rok wydania lub przypisany utworowi numer ścieżki na płycie CD. Metadanymi dopisywanymi do utworu MP3 nie muszą być wyłącznie tagi ID3, lecz są one jednym z najpopularniejszych nośników informacji o utworach.

Pierwotnie używano tagów ID3 w wersji 1, nazywanych także ID3v1. Cechowały się one przede wszystkim prostotą implementacji: zarówno jeśli chodzi o umieszczanie ich w pliku, jak i procedurę odczytu. Ten rodzaj taga to nic innego, jak 128 bajtów danych (głównie tekstowych) „dopisanych" na koniec pliku MP3. Kodowanie tekstu odbywa się w najprostszy sposób, czyli za pomocą znaków ASCII. Prostota działania ID3 polega na tym, że każde pole taga ma na stałe przypisany dla siebie offset (położenie względem początku taga). W **tab. 3** znajduje się wykaz wszystkich pól taga ID3. Nazwy pól odpowiadają nazwom elementów struktury danych typu T_ID3_Tag_v1 reprezentującej tag w projekcie odtwarzacza. Struktura ta zdefiniowana jest w pliku id3.c. W tym module także odbywa się analiza zawartości tagów ID3v1 za pomocą funkcji readId3v1.

Dodatkowego komentarza wymagać może pole taga o nazwie zeroByte. Bajt ten może należeć do pola comment razem z elementem trackNumber – w takim przypadku zeroByte może zawierać wartość niezerową. Jeśli jednak do zeroByte wpiszemy wartość 0, to kolejne pole (trackNumber) będzie interpretowane jako przechowujące numer utworu w albumie. Warto także mieć na uwadze, że niewykorzystane bajty w polach title, artist, album i comment mogą zostać wyzerowane lub wypełnione spacjami.

Wystarczy spojrzeć na sposób kodowania zawartości tagów ID3v1, by dostrzec jego mankamenty. Przede wszystkim kodowanie tytułów i nazwy wykonawcy na 30 znakach może nie być wystarczające – nieraz zdarza się, że tytuł utworu lub albumu przekracza tę liczbę znaków. Wtedy nie pozostaje nic innego jak „obciąć" tekst tytułu, co może być nieco denerwujące przy jego wyświetlaniu. Drugim ważnym mankamentem tagów ID3 w wersji 1 było kodowanie gatunku muzycznego w jednym bajcie – z racji ogromnych ilości gatunków dość oczywiste jest, że nie da się ich wszystkich zaszufladkować, tym bardziej używając do tego 256 wartości. Kolejnym problemem jest umieszczenie taga typowo na końcu utworu, co jest jednak bardziej dokuczliwe przy przesyłaniu utworu „strumieniowo" (np. przez sieć Internet) niż w odtwarzaczach dysponujących od razu całymi plikami MP3 (np. na kacie pamięci lub dysku).

Tab. 3. Pola taga ID3 w wersji 1

Nazwa pola	Rozmiar pola w bajtach	Opis
id	3	nagłówek identyfikujący taga ID3v1 – trzyznakowy tekst „TAG"
title	30	tytuł utworu
artist	30	nazwa wykonawcy
album	30	tytuł albumu
year	4	rok wydania
comment	28	komentarz tekstowy
zeroByte	1	jeśli ten bajt zawiera wartość 0, to kolejny bajt (trackNumber) odczytany zostanie jako numer ścieżki
trackNumber	1	numer ścieżki w albumie
gerneCode	1	kod gatunku muzyki

Niektóre z wyżej wymienionych problemów są rozwiązane w tagach ID3 w wersji 2, lecz jest to okupione pewną komplikacją algorytmu kodowania i odczytu ich danych. Tagi ID3v2 umieszczane są typowo na początku utworu MP3. Mogą one przechowywać bardzo dużo informacji, ponieważ zezwalają na zmienne długości pól, a same pola danych mogą w nich być bardzo różnorodne. Niestety pociąga to za sobą konieczność rozpoznawania początków pól poprzez analizę ich zawartości – nie można rozpoznać pola na podstawie jego położenia, jak miało to miejsce w przypadku ID3v1.

Co dalej?

Prezentowany odtwarzacz MP3 jeszcze rok temu był jedynie projektem eksperymentalnym typu „proof of concept", a właściwie efektem poszukiwań ciekawego zastosowania dla mikrokontrolera z 32-bitowym rdzeniem taktowanym z częstotliwością prawie 200 MHz. Od tego czasu znacząco wyewoluował: np. w pierwszych wersjach zastosowany był archaiczny wręcz wyświetlacz alfanumeryczny 2×16 znaków, nie było funkcji budzika, obsługiwane były wyłącznie „stare" karty SD... Obecnie projekt staje się coraz bardziej rozbudowany, choć do „idealnego odtwarzacza domowego" jeszcze trochę mu brakuje.

Ciekawym pomysłem na rozbudowę odtwarzacza może być np. implementacja usuwania wybranego pliku MP3. Nawet naszym ulubionym artystom zdarza się bowiem umieścić na płycie utwór, którego mimo najlepszych chęci i sympatii do wykonawcy nie jesteśmy w stanie słuchać. Przy obecnej wersji oprogramowania w takiej sytuacji należy wyjąć kartę z odtwarzacza, umieścić ją w czytniku i usunąć nielubiany utwór. Innym udogodnieniem dla użytkownika mogłaby być implementacja parsowania tagów ID3 w wersji 2. Jest to zadanie nieco trudniejsze niż odczyt tagów ID3v1, lecz możemy dzięki temu uzyskać dalszy wzrost funkcjonalności odtwarzacza. Jeszcze bardziej ambitnym zadaniem byłaby implementacja wyświetlania aktualnie odtwarzanej minuty i sekundy utworu.

Odtwarzanie MP3 dla takiego mikrokontrolera jak AT91SAM9260 to bułka z masłem, więc pozostaje do wykorzystania duży zapas mocy obliczeniowej. Pozwala to na dołączenie do odtwarzacza innych dekoderów dla bardziej nowoczesnych i zaawansowanych formatów plików muzycznych niż „klasyczne" MP3. Przykładem tutaj może być format AAC. Niewykorzystaną moc obliczeniową można także spożytkować do zrealizowania dodatkowych efektów dźwiękowych lub do implementacji algorytmów wizualizacji. W końcu nic nie stoi na przeszkodzie, aby mikrokontroler odtwarzający MP3 sterował także jasnością np. diod LED RGB dużej mocy.

Robert Brzoza-Woch
rabw@poczta.fm