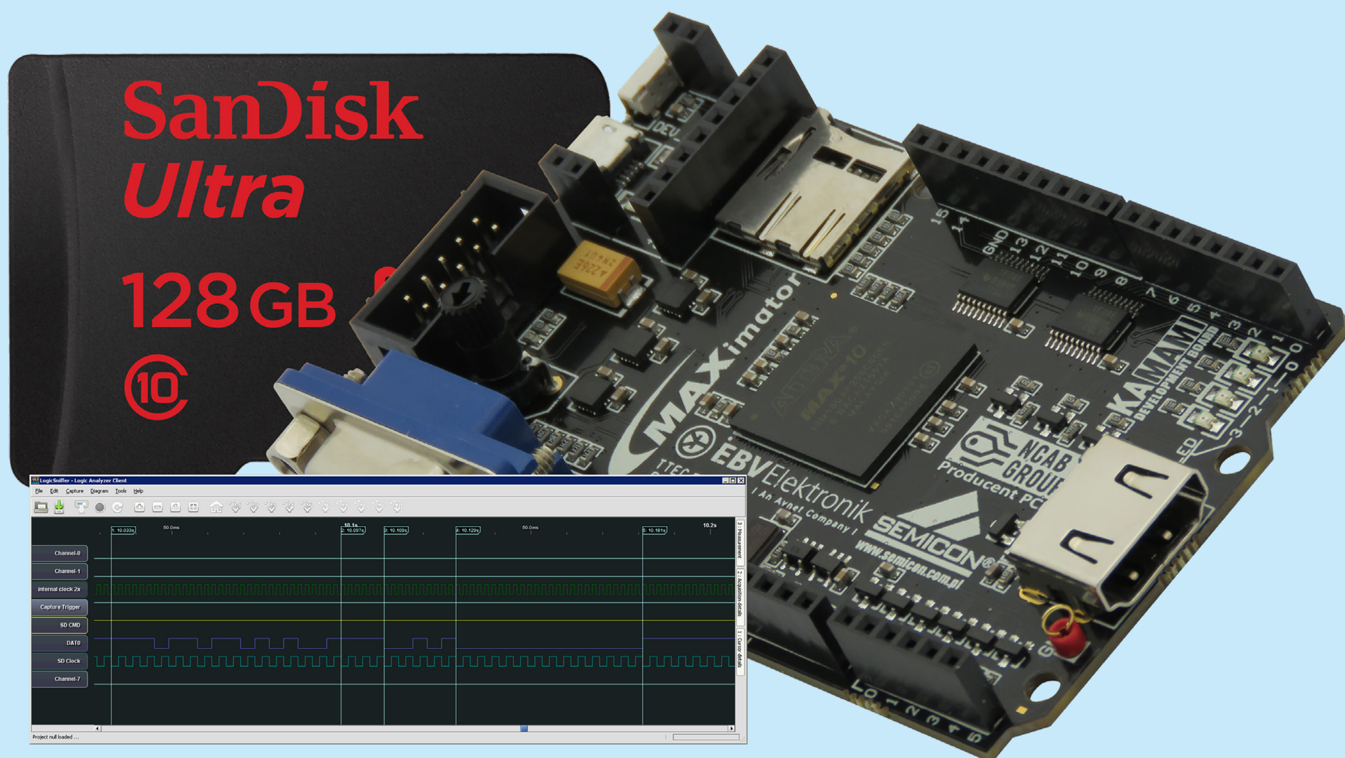


NIOS II na maXimatorze, czyli mikroprocesor w układzie FPGA (12)

Więcej o pamięci, czyli o obsłudze kart SD

Niejednokrotnie w naszym projekcie potrzebujemy przechować, a następnie udostępnić dla komputera większą ilość danych. Oczywiście, dane można przechowywać np. w pamięci Flash lub EEPROM, a następnie przesłać do komputera, jednak dobrym rozwiązaniem jest zastosowanie kart pamięci SD. Zatem budowa rejestratorów danych nie będzie już dla nas stanowiła wyzwania! Dodatkowo czas, aby nauczyć się także, jak przenosić przygotowane moduły Platform Designer'a pomiędzy projektami.

Nasza przygodę z gromadzeniem danych zacznijmy od spojrzenia na to, czym tak na prawdę jest karta pamięci. Z punktu widzenia elektroniki to po prostu kolejny rodzaj pamięci nieulotnej, w której możemy przechowywać jakieś bajty – tak, mówimy tylko o bajtach, a nie plikach, gdyż warstwa sprzętowa tak naprawdę nic nie wie o tym w jaki sposób „poukładamy" zapisane dane – tym zajmie się warstwa systemu plików, która z kolei nie musi nic wiedzieć o tym jak dane są zapisane.

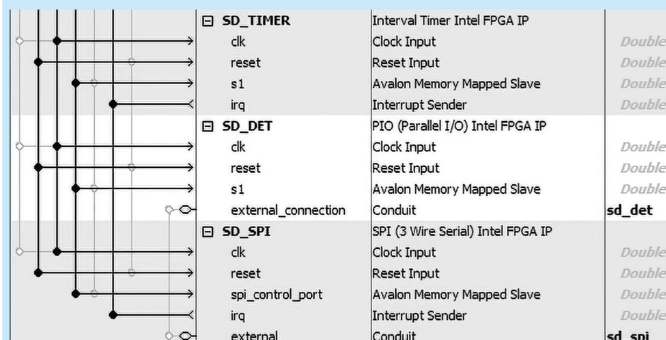
Karty pamięci, systemy plików – jak się nie pogubić...

Karty pamięci SD, którymi się zajmiemy wyposażone są zwykle w dwa rodzaje interfejsów, które współdzielą ze sobą wyprowadzenia – są to SDIO oraz SPI. Pierwszy z nich jest zaawansowanym interfejsem, umożliwiającym szybki transfer danych, jednak w zamian za to jego implementacja jest dosyć skomplikowana. Drugi z interfejsów to SPI – tak, dokładnie ten sam interfejs, który już poznaliśmy.

Niestety, mimo iż interfejs jest nieskomplikowany, to sam sposób zapisywania i odczytywania danych jest już mocno skomplikowany. Na szczęście nie będziemy musieli rozgryzać tej łamigłówki od podstaw – nie ma sensu wyważania otwartych drzwi! Co więcej interfejs SPI kart SD jest kompatybilny z interfejsem SPI kart MMC, więc nie zdziwcie się, że już kilkanaście linijek dalej będziemy używać plików z *MMC* w nazwie.

Zakładając, że możemy zapisywać i odczytywać bajty z karty pamięci, czas zastanowić się nad tym, jak to robić. Rzecz jasna moglibyśmy w dowolny sposób skorzystać z dostępnych gigabajtów

Poprzednie części kursu i dodatkowe materiały dostępne są na na stronie www.media.avt.pl



Rysunek 1. Moduły dodane do obsługi karty SD

sd_det_export[1]	Input	PIN_A14
sd_det_export[0]	Input	PIN_A15
sd_spi_MISO	Input	PIN_A2
sd_spi_MOSI	Output	PIN_A4
sd_spi_SCLK	Output	PIN_A5
sd_spi_SS_n	Output	PIN_A3

Rysunek 2. Prawidłowo przypisane wyprowadzenia do komunikacji z kartą SD

pamięci, ale chyba chcielibyśmy w wygodny sposób dostać się do naszych danych na komputerze oraz móc odczytać dane w ten sposób zapisane na karcie. W tym celu nasze bajty musimy układać w taki sposób, aby komputer mógł je zinterpretować jako dysk sformatowany w konkretny sposób zawierający jakieś pliki. Do tego celu posłużmy się systemem plików FAT32. Znów nie musimy zagłębiać się w to, jak działa ten system plików – na szczęście dostępna jest biblioteka FatFs, która rozwiąże większość problemów. Co ciekawe, system plików FAT32 możemy wykorzystać na dowolnej pamięci, więc jeśli z jakichś powodów przypadnie nam do gustu możemy go wygodnie używać.

Przygotowania

Zanim zaczniemy pracę z kartami pamięci warto wybrać kartę, która nie zawiera cennych danych i wcześniej ją sformatować (za pomocą narzędzia SDFormatter, formatowanie usuwa nieodwracalnie wszelkie dane z karty). Oczywiście możemy wykorzystać kartę z jakimiś danymi, jednak w wypadku jakiejś nieoczekiwanej sytuacji dane te możemy utracić.

Ponadto, pobierzmy ze strony Elm-Chan (***http://elm-chan.org/fsw/ff/00index_e.html***) bibliotekę FatFs oraz przykłady dla różnych platform (na chwilę pisania tego artykułu najnowszą wersją biblioteki była R0.13c).

Przygotowanie systemu

Na początku musimy dodać do naszego systemu moduły, które umożliwią komunikację z kartą SD i jej obsługę. W tym celu dodajemy doskonale znane już nam moduły (niewymienione opcje pozostawiamy bez zmian):

1. Interval Timer:
   a. Nazywamy go *SD_TIMER.*
   b. Ustawiamy okres na 10 ms.
   c. Wybieramy opcje *No Start/Stop control bits* oraz *Fixed period.*
   d. Odznaczamy opcję *Readable snapshot.*
2. PIO (Parallel I/O):
   a. Nazywamy *SD_DET.*
   b. Szerokość ustawiamy na 2 bity.
   c. Kierunek: wejście.
   d. Eksportujemy wejścia jako: *sd_det.*
3. SPI (3 Wire Serial):
   a. Nazywamy *SD_SPI.*
   b. *SCLK* 500 000 Hz.
   c. *Clock polarity*: 0.
   d. *Clock phase*: 0.
   e. Eksportujemy wejścia jako: *sd_spi.*

Po tych czynnościach nowo dodane moduły powinny wyglądać w sposób pokazany na **rysunku 1**. Moduły te dołączamy w standardowy sposób do systemu, wykonujemy przypisanie adresów bazowych, numerów przerwań, generujemy *HDL*. Wykonujemy *Analysis & Synthesis*, a następnie dokonujemy przypisania wyprowadzeń układu, nie zapominając o zmianie standardu na *3.3-V LVTTL*.

Funkcje wyprowadzeń karty SD w trybie SPI oraz połączenia na płytce maXimator przedstawiono w **tabeli 1**. Wszystkie używane piny karty muszą być podciągane za pomocą rezystorów do napięcia zasilania karty, czyli 3,3 V. Piny DET1 i DET2 to mechaniczne przełączniki służące do wykrywania obecności karty w slocie. Prawidłowo przypisane piny pokazano na **rysunku 2**. Po tym kompilujemy plik i generujemy projekt i BSP w *Eclipse*.

Oprogramowanie

Po utworzeniu standardowego projektu przyszła kolej na dodanie wsparcia dla obsługi kart SD oraz systemu plików FAT32. Jak już wspomniałem w tym celu posłużymy się biblioteką FatFs. Na początku w naszym projekcie utwórzmy nowy folder i nazwijmy go fatFs, a następnie skopiujmy do niego zawartość katalogu source z pobranej biblioteki FatFs. Dodawanie folderu i wklejanie plików wykonajmy poprzez menu w Eclipse, dzięki czemu odpowiednie pliki zostaną dodane automatycznie do projektu. Dodatkowo musimy dodać folder fatFs do ścieżki *Nios II Application Paths* → *Application include directories*.

Na tym etapie dodaliśmy do projektu bibliotekę obsługującą system plików FAT32, teraz przyszła kolej na nieco bardziej pracochłonną część, czyli dodanie obsługi kart SD. Na początek z pośród przykładów odnajdźmy folder avr i z niego skopiujmy pliki *diskio.c* oraz *.h*, zamieniając pliki wcześniej skopiowane wraz z biblioteką. Następnie skopiujmy pliki *mmc_avr.h* oraz *mmc_avr_spi.c* i zamieńmy ich nazwy na *mmc.h* oraz *mmc.c*.

Teraz pora na ostateczne porządki w nowo dodanych plikach. Zacznijmy od pliku *diskio.c*. W nim usuwamy wszelkie fragmenty objęte dyrektywą preprocesora dotycząca definicji *DRV_CFC*, gdyż nie będziemy zajmować się obsługą kart CF, kasujemy także instrukcje warunkowe preprocesora dotyczące definicji *DRV_MMC*, pozostawiając w programie objęte nimi instrukcje. Kasujemy także niepotrzebne, zdublowane po takich operacjach funkcje *return* oraz funkcje *switch* które staną się niepotrzebne. Przykładowo poniższa funkcja oryginalnie wyglądała jak na **listingu 1**, a po naszych modyfikacjach będzie wyglądała w sposób pokazany na **listingu 2**.

Podobnie postępujemy z innymi instrukcjami. Jak widzimy biblioteka FatFs pozwala na wsparcie dla wielu nośników danych – wystarczy wtedy zrobić tylko „użytek" z parametru *pdrv*. Dodatkowo zmieniamy nazwę dołączanego pliku nagłówkowego z *mmc_avr.h* na *mmc.h*.

Tabela 1. Funkcje wyprowadzeń karty SD w trybie SPI oraz połączenia na płytce maXimator

wyprowadzenie karty SD	funkcja w trybie SPI	połączenie maXimator
DAT0	MISO	A2
DAT1	-	-
DAT2	-	-
DAT3	CS	A3
CMD	MOSI	A4
CLK	SCK	A5
DET1	wykrywanie obecności karty	A15
DET2	wykrywanie obecności karty	A14

Teraz czas na finalne dostosowanie programu do naszych potrzeb i modyfikację pliku *mmc.c*. Zaczynamy od zmodyfikowania dołączanych w tym pliku plików nagłówkowych (**listing 3**). Następnie uzupełniamy wymagane makrodefinicje (oznaczone zachęcającymi *To be filled*) – **listing 4**. Jak widzicie, definicję związaną z wykrywanie ochrony zapisu ustawiłem stale na 0, gdyż karty microSD nie posiadają mechanicznej blokady zapisu. Także definicje funkcji zmieniających częstotliwość magistrali SPI pozostawiłem puste, gdyż w dostarczonym przez producenta module SPI nie mamy możliwości zmiany częstotliwości pracy po syntezie. Następnie uzupełniamy w następujący sposób funkcje odpowiedzialne za zasilanie karty (**listing 5**). Na płytce maXimator nie mamy możliwości sterowania zasilaniem karty, więc jedyne co robimy w jednej z funkcji (która wywoła się na pewno przed jakąkolwiek komunikacją z kartą) to ustawiamy prawidłowy układ do aktywacji linii CS/SS. Dalej musimy zmienić funkcję realizującą wymianę jednego bajta danych z karta pamięci za pomocą protokołu SPI – na podstawie bibliotecznej funkcji producenta możemy przygotować taki fragment kodu (**listing 6**).

Nie stosujemy tu używanej wcześniej funkcji, gdyż nie pozwala ona (jak wspominałem jakiś czas temu) na jednoczesne nadawanie i odbiór innych danych w tym samym czasie. Dalej, już bez zbędnych komplikacji modyfikujemy kolejne dwie funkcje, dla uproszczenia korzystając z przygotowanej wcześniej funkcji (nie jest to rozwiązanie optymalne, ale za to eleganckie) (**listing 7**).

Dalej musimy dokonać jednej obowiązkowej modyfikacji w pliku *ffconf.h*, który zawiera definicje pozwalające na włączenie lub wyłączenie niektórych funkcji biblioteki FatFs. Odnajdujemy tam zapis *FF_FS_NORTC 0* i zmienaimy wartość na *1*. Informujemy tym samym bibliotekę, że nie będziemy dostarczać funkcji podającej aktualny czas i datę. W widocznych pod tą definicją kolejnych definicjach możemy zdefiniować według własnego uznania datę, która ma zostać domyślnie użyta dla datowania plików na karcie. Możemy już wrócić do pliku *main.c*. W nim dołączamy odpowiednie pliki nagłówkowe oraz definiujemy funkcję obsługi przerwania od timera (**listing 8**). W funkcji *main* musimy włączyć jedynie przypisać tę funkcję do obsługi timera oraz włączyć jego przerwanie:

```
alt_ic_isr_register(SD_TIMER_IRQ_INTERRUPT_
CONTROLLER_ID, SD_TIMER_IRQ, sdTimerInterrupt,
NULL, NULL);
IOWR_ALTERA_AVALON_TIMER_
CONTROL(SD_TIMER_BASE, ALTERA_
AVALON_TIMER_CONTROL_ITO_MSK);
```

W tej chwili możemy już w pełni korzystać z biblioteki FatFs. Aby zamontować kartę i odczytać zawartość zapisanego na niej pliku o nazwie SD.txt (nazwa może być dowolna) możemy się posłużyć następującym programem (dane wymieniać będziemy za pomocą *JTAG UART* i *Nios II Console* – **listing 9**.

W przykładowym programie zawarłem też instrukcje pozwalające zapisać dane do pliku, albo poprzez dopisanie ich na końcu istniejącego pliku, albo poprzez całkowite zastąpienie jego zawartości. Myślę, że w tym miejscu mogę każdego z czytelników pozostawić, zachęcając do samodzielnego zapoznania się z bardziej zaawansowanymi funkcjami biblioteki FatFs, na przykład pozwalającymi na pracę z katalogami, ich przeszukiwanie itp. Na koniec musze jeszcze wspomnieć o kwestii optymalizacji – w pliku *ffconf.h* możemy włączyć lub wyłączyć niektóre funkcje biblioteki, a przez to zoptymalizować zasoby przez nią zużywane. W wypadku

Listing 1. Oryginalna funkcja *disk_status*

```
DSTATUS disk_status (
    BYTE pdrv   /* Physical drive nmuber to identify the drive */
)
{
    switch (pdrv)
    {
#ifdef DRV_CFC
    case DRV_CFC :
        return cf_disk_status();
#endif
#ifdef DRV_MMC
    case DRV_MMC :
        return mmc_disk_status();
#endif
    }
    return STA_NOINIT;
}
```

Listing 2. Funkcja *disk_status* po modyfikacji

```
DSTATUS disk_status (
    BYTE pdrv   /* Physical drive nmuber to identify the drive */
)
{
    return mmc_disk_status();
}
```

Listing 3. Wykaz dołączonych plików nagłówkowych

```
#include „alt_types.h"
#include „system.h"
#include „sys/alt_sys_wrappers.h"
#include „altera_avalon_spi_regs.h"
#include „altera_avalon_spi.h"
#include „altera_avalon_pio_regs.h"
#include „ff.h"
#include „diskio.h"
#include „mmc.h"
```

Listing 4. Uzupełnione makrodefinicje

```
/* Peripheral controls (Platform dependent) */
#define CS_LOW()   IOWR_ALTERA_AVALON_SPI_CONTROL(SD_SPI_BASE, ALTERA_AVALON_SPI_CONTROL_SSO_MSK)  /* Set MMC_CS = low */
#define  CS_HIGH()  IOWR_ALTERA_AVALON_SPI_CONTROL(SD_SPI_BASE, 0)  /* Set MMC_CS = high */
#define MMC_CD     (IORD_ALTERA_AVALON_PIO_DATA(SD_DET_BASE) == 3)  /* Test if card detected.   yes:true, no:false, default:true */
protected. yes:true, no:false, default:false
#define  FCLK_SLOW()  /* Set SPI clock for initialization (100-400kHz) */
#define  FCLK_FAST()  /* Set SPI clock for read/write (20MHz max) */
```

Listing 5. Funkcje włączające/wyłaczajace zasilanie karty SD

```
static
void power_on (void)
{
    IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(SD_SPI_BASE, 0b1);
}

static
void power_off (void)
{
}
```

Listing 6. Wymiana bajta pamięci za pomocą SPI

```
/* Exchange a byte */
static
BYTE xchg_spi (  /* Returns received data */
    BYTE dat     /* Data to be sent */
)
{
    BYTE rdDat;
    // odczytaj dane na wszelki wypadek, aby opróżnic bufor
    IORD_ALTERA_AVALON_SPI_RXDATA(SD_SPI_BASE);
    // czekaj aż układ będzie gotowy do nadawania
    while ((IORD_ALTERA_AVALON_SPI_STATUS(SD_SPI_BASE) &
        ALTERA_AVALON_SPI_STATUS_TRDY_MSK) == 0);
    // wyślij bajt
    IOWR_ALTERA_AVALON_SPI_TXDATA(SD_SPI_BASE, dat);
    // czekaj aż zostanie odebrany jeden bajt danych (podczas wysyłania)
    while ((IORD_ALTERA_AVALON_SPI_STATUS(SD_SPI_BASE) &
        ALTERA_AVALON_SPI_STATUS_RRDY_MSK) == 0);
    // odczytaj odebrane dane
    rdDat = IORD_ALTERA_AVALON_SPI_RXDATA(SD_SPI_BASE);
    // czekaj, dopóki nie zostanie zakończony cały cykl transmisji
    while ((IORD_ALTERA_AVALON_SPI_STATUS(SD_SPI_BASE) &
        ALTERA_AVALON_SPI_STATUS_TMT_MSK) == 0);
    return rdDat;
}
```

Poprzednie części kursu i dodatkowe materiały dostępne są na na stronie www.media.avt.pl

```
Listing 7. Wymiana bloku danych za pomocą SPI
/* Receive a data block fast */
static
void rcvr_spi_multi (
    BYTE *p,   /* Data read buffer */
    UINT cnt   /* Size of data block */
)
{
    do
    {
        *p++ = xchg_spi(0xFF);
        *p++ = xchg_spi(0xFF);
    } while (cnt -= 2);
}

/* Send a data block fast */
static
void xmit_spi_multi
(
    const BYTE *p,   /* Data block to be sent */
    UINT cnt         /* Size of data block */
)
{
    do
    {
        xchg_spi(*p++);
        xchg_spi(*p++);
    } while (cnt -= 2);
}
```

konieczności dalszych oszczędności możemy posłużyć się biblioteką Petit FatFs. Jest ona bliźniaczo podobna do omawianej przez nas pełnej wersji, jednak zawiera pewne ograniczenia jak np. brak możliwości tworzenia plików czy zmiany ich rozmiaru. W zamian za to zajmuje ona znacznie mniej zasobów, a jej funkcjonalność w zupełności wystarczy nam np. do odczytu danych z karty w jakimś odtwarzaczu.

Współdzielenie modułów

Do tej pory stworzyliśmy już pewną bazę modułów, jednak jak zauważyliście, są one przypisane do projektu, przy którym je tworzyliśmy – niezbyt wygodne…

Pierwsza możliwością współdzielenia modułów jest ich kopiowanie pomiędzy projektami w takim stanie, w jakim są, czyli kopiujemy folder z plikami źródłowymi (np. dla sterownika wyświetlacza 7-segmentowego był to folder *SEG7* umieszczony w katalogu naszego projektu) oraz plik *.tcl* zawierający konfigurację modułu (we wspomnianym przypadku będzie to plik *SEG7_hw.tcl*). Rozwiązanie takie jednak nie jest wygodne, gdyż kopiować musimy 2 elementy i co ważniejsze – utrzymywać między nimi odpowiednie względne położenie.

Aby temu zaradzić skopiujmy folder *SEG7* z jednego z naszych starszych projektów i wklejmy go do folderu w aktualnym projekcie, w którym znajduje się nasz główny plik projektu **.qsys*. Po tym ze starego projektu skopiujmy plik *SEG7_hw.tcl* i wklejmy go do poprzednio wklejonego folderu *SEG7*. Teraz otwórzmy

```
Listing 8. Obsługa przerwania timera
#include „system.h”
#include „sys/alt_stdio.h”
#include „altera_avalon_timer_regs.h”
#include „sys/alt_irq.h”
#include „ff.h”
#include „diskio.h”

void sdTimerInterrupt(void* context)
{
    IOWR_ALTERA_AVALON_TIMER_STATUS(SD_TIMER_BASE, 0);
    disk_timerproc();
}
```

plik *SEG7_hw.tcl* w ulubionym edytorze tekstowym (polecam Notepad++) i odnajdźmy w nim linijki, zawierające ścieżki dostępu do plików źródłowych:

```
add_fileset_file SEG7.vhd VHDL PATH ../SEG7/SEG7.vhd
TOP_LEVEL_FILE
…
add_fileset_file SEG7.vhd VHDL PATH ../SEG7/SEG7.vhd
…
add_fileset_file SEG7.vhd VHDL PATH ../SEG7/SEG7.vhd
```

Zmieniamy ścieżki dostępu do plików tak, aby wskazywały na pliki znajdujące w tym samym folderze co nasz plik **.tcl*:

```
add_fileset_file SEG7.vhd VHDL PATH SEG7.vhd
TOP_LEVEL_FILE
…
add_fileset_file SEG7.vhd VHDL PATH SEG7.vhd
…
add_fileset_file SEG7.vhd VHDL PATH SEG7.vhd
```

Po zapisaniu pliku i otwarciu naszego projektu w *Platofrm Designer* powinniśmy widzieć na szczycie listy nasz moduł możliwy do dodania w projekcie. teraz aby przenieść taki moduł/bibliotekę gdzie indziej wystarczy tylko skopiować cały folder do folderu z nowym projektem i gotowe!

```
Listing 9. Obsługa pliku o nazwie SD.TXT
if(disk_initialize(0) != 0)  //inicjalizujemy warstwę sprzętową
{
    alt_putstr(„SD card ERROR\r\n”);
    waitUser();
    continue;
}
alt_putstr(„SD card initialized\r\n”);

if(f_mount(&Fatfs, „”, 0) != FR_OK)  // inicjalizujemy system plików
{
    alt_putstr(„File System ERROR\r\n”);
    waitUser();
    continue;
}

alt_putstr(„File system FATfs initialized\r\n”);
if(f_open(&File, „SD.txt”, FA_READ|FA_OPEN_ALWAYS) != FR_OK)  // otwieramy plik do odczytu,
// jeśli plik nie istnieje to tworzymy nowy plik
{
    alt_putstr(„File Open for reading ERROR\r\n”);
    waitUser();
    continue;
}

alt_putstr(„File SD.txt opened for reading\r\n”);
UINT i=0;
while(1)
{
    if(f_read(&File, &Buff, BUFF_SIZE, &i) == FR_OK)  // odczytujemy maksymalnie 255 znaków z pliku
    {
        alt_printf(„0x%x bytes read:\r\n”, i);  // i je wyświetlamy
        if(i == 0) break;
        UINT j = 0;
        while(j<i && Buff[j])
        {
            alt_putchar(Buff[j++]);
        }
        alt_putstr(„\r\n”);
    } else {
        alt_putstr(„File Read ERROR\r\n”);
        break;
    }
}

if(f_close(&File) != FR_OK)  // zamykamy plik
{
    alt_putstr(„File Close ERROR\r\n”);
    waitUser();
    continue;
}
```


Poprzednie części kursu i dodatkowe materiały dostępne są na stronie www.media.avt.pl

Jeszcze więcej porządku

Na razie moduł mamy jeden. Dodając te, które stworzyliśmy wspólnie do tego momentu uzbierałoby się ich kilka. Ale po jakimś czasie pracy… mielibyśmy ich prawdziwe zatrzęsienie, w którym trudno byłoby się odnaleźć. Dlatego warto moduły pogrupować! Aby to zrobić, w *Platform Designer* otwieramy okienko edycji modułu i w zakładce *Component Type* w polu *Group* możemy wpisać nazwę grupy, lub nawet całą ścieżkę, ja np. wpisałem *Custom/Display*. Po kliknięciu *Finish…* i odpowiedzeniu twierdząco na pytanie o zapis moduł pojawi się na liście w odpowiednim folderze.

Jednak nadal nie pozbyliśmy się jednego problemu – musimy tę „bibliotekę" kopiować do każdego projektu… Aby poradzić sobie z tym problemem wystarczy, że przeniesiemy cały ten folder do odpowiedniego katalogu, w którym przechowywane są biblioteki. Jest to ścieżka: *folder_instalacji_quartus\wersja_oprogramowania\ip* (u mnie: *C:\intelFPGA_lite\18.0\ip*). Tam możemy tworzyć dowolną strukturę katalogów – oprogramowanie przeszuka ten folder rekursywnie. Dla porządku możemy tam zastosować taką strukturę folderów, jak grupa, w której wyświetlany jest komponent. Teraz, po restarcie *Platform Designer*, nasza grupa *Custom/Display* wyświetli się już pod grupą nadrzędną *Library* zamiast *Project*.

Dodatkowo, warto aby zablokować możliwość „grzebania" w module, który chcemy współdzielić między projektami (aby potem nie było zaskoczenia, że coś zmieniliśmy i w jednym projekcie działa, a w N starszych nie). Możemy to zrobić znowuż edytując plik **.tcl* i zmieniając wartość parametru

`set_module_property EDITABLE`

na *`false`*.

Podsumowanie

W czasie naszego dzisiejszego spotkania przede wszystkim nauczyliśmy się jak poradzić sobie z przystosowaniem kolejnej, dosyć złożonej, biblioteki do pracy z naszym systemem i powierzchownie poznaliśmy niektóre możliwości tej biblioteki – mowa tu oczywiście o FatFs. W drugiej, krótkiej części, zajęliśmy się uporządkowaniem powstającego powoli bałaganu związanego z tworzeniem przez nas kolejnych bibliotek.

W ramach zadania domowego tym razem cała paleta możliwości:

1. Zebrać finalne wersje wszystkich stworzonych przez nas komponentów:
   a. bufor od I^2C,
   b. moduł PWM z definiowaną liczbą kanałów,
   c. sterownik wyświetlacza 7-segmentowego,
   d. dekoder impulsów z enkoderów,
   e. zoptymalizowany sterownik do diod WS2812, i dodać je w ten sposób, aby były widoczne jako elementy biblioteczne we wszystkich projektach.
2. Do naszego projektu dodać wybrany przez nas moduł (np. sterownik WS2812) i odczytywać z karty SD sposób sterowania takim modułem (np. stworzyć animację na matrycy diod WS2812 lub choćby wyświetlać kolejne liczby odczytane z karty na wyświetlaczu 7-segmentowym.

Trzymam kciuki za kreatywność! A w kolejnej części zajmiemy się… obsługą wyjścia VGA – szykujcie monitory!

Piotr Rzeszut, AGH

REKLAMA

100% elektroniki na avt.pl/prenumerata

Prenumerujesz Elektronikę Praktyczną
+ Elektronikę dla Wszystkich?
Skorzystaj z promocji 1+1=3
i zamów bezpłatną prenumeratę Elektronika

Poprzednie części kursu i dodatkowe materiały dostępne są na na stronie www.media.avt.pl