Wersja z dnia:
10 czerwca 2005

Andrzej Kuroś
Marcin Krzesiński

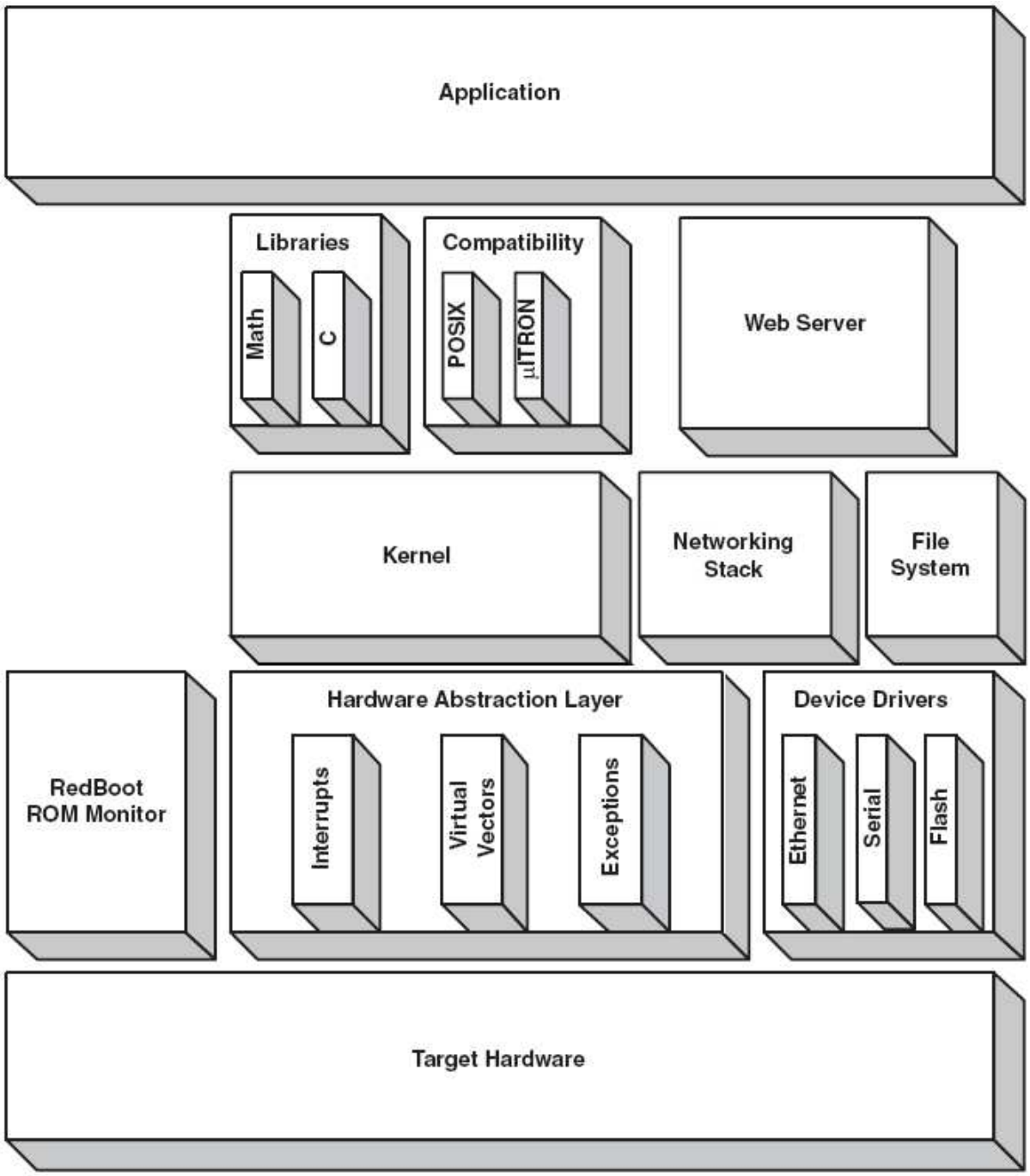
System operacyjny ECOS na platformie ZL1ARM z mikrokontrolerem LPC2124

1. O systemie ECOS słów kilka[1]

System operacyjny **eCOS** (Embedded Configurable Operating System) jest systemem czasu rzeczywistego (RTOS – Real-Time Operating System). Napisano go w językach C/C++, a najniższą warstwę bezpośrednio w asemblerach dla odpowiednich mikroprocesorów. Ta wielość języków nie powinna przerażać, gdyż system dostarcza biblioteki API (Application Programming Interface) w języku C. System składa się z szeregu różnych modułów, z których wyróżnić można kilka podstawowych grup:

- **HAL** (Hardware Abstraction Layer) – warstwa odpowiedzialna za dostarczenie w miarę prostego interfejsu do sprzętu na możliwie najniższym poziomie. Jest ona zrealizowany jako **makra języka C** i jako taka właściwie nie musi być używana przez programistę piszącego aplikację w systemie eCOS.
- **Jądro** (Kernel) – system eCOS jest prawdziwym wielozadaniowym systemem z wywłaszczeniem. Jądro zajmuje się obsługą wątków, szeregowaniem i synchronizacją zadań, itp...
- **Sterowniki** (Drivers) – są to fragmenty kodu pozwalające na dostęp do peryferyjnych urządzeń wejścia/wyjścia w zunifikowany sposób. Każde takie urządzenie powinno mieć swój osobny sterownik, który wnosi zarówno obsługę specyficznych przerwań, rezerwuje pamięć na bufory i dostarcza funkcjonalności.
- **Biblioteki standardowe** (Standard Libraries) – są to fragmenty kodu, dostarczające funkcjonalności dla standardu języka C (np. funkcje printf, rozpoczynanie aplikacji użytkownika od funkcji main, itp...)
- **Biblioteki dodatkowe** (Auxiliary Libraries) – są to fragmenty kodu, dostarczające dodatkowej funkcjonalności, np. programowy stos TCP/IP, kompresja/dekompresja ZIP itp...

[1] Dostępny jest w sieci dobry podręcznik: Anthony J. Massa, "Embedded Software Development with ECOS™", Prentice Hall ProfessionalTechnical Reference, New Jersey 2003. Polecamy dokładnie przeczytać, szczególnie pierwsze rozdziały.



Rys. 1. Diagram modułów funkcjonalnych systemu eCOS.

Istnieje wiele platform, na których system eCOS został uruchomiony. Każda z nich ma swoją specyfikę, dlatego kod źródłowy podzielony jest tak, aby poszczególne fragmenty kodu dla różnych platform znajdowały się w różnych katalogach[2].

[2] Struktura katalogów objaśniona jest w [1]

System ECOS jest w bardzo wysokim stopniu konfigurowalny. Jest tak dzięki specjalnemu stylowi pisania, używającemu (czasem nadużywającemu) dyrektyw typu

#define

#ifdef

które pozwalają na warunkowe kompilowanie kodu (dyrektywy preprocesora). Dzięki temu możliwe jest bardzo staranne wybranie tego kodu, który jest potrzebny, a usunięcie nieużywanego i optymalizacja wynikowego kodu, minimalizująca ilość zużytej pamięci danych i programu. Jest to szczególnie istotne w systemach wbudowanych, w których występują znaczne ograniczenia sprzętowe. Dostarczone graficzne narzędzie do konfiguracji zwalnia programistę od konieczności ręcznego definiowania setek parametrów.

Uruchamiano wersję 2.0 systemu eCOS dostępną jako kompletną dystrybucję.

2. Mikrokontroler LPC2124

Mikrokontroler LPC2124 produkcji firmy Phillips jest oparty na rdzeniu ARM7TDMI[3]. Posiada 256KB pamięci FLASH i 16KB SRAM oraz kilka rodzajów standardowych portów komunikacyjnych i GPIO (General Purpose Input Output). Rdzeń ARM7TDMI jest 32-bitowym mikroprocesorem typu RISC o architekturze **von Neumanna**. Pogram i dane są dostępne w tej samej przestrzeni adresowej, a dostęp do nich możliwy jest tylko poprzez operacje odczytu, zapisu i wymiany. Mikroprocesor ARM jest dość wydajny, w przypadku operacji arytmetyczno-logicznych, należy mieć jednak świadomość **ograniczeń** co do szybkości dostępu do pamięci:

- Nie ma instrukcji arytmetyczno-logicznych na argumentach w pamięci (zwyczajna cecha RISC'ów).
- Nie ma instrukcji zapisu/odczytu z użyciem natychmiastowego adresu, w zamian występuje adresowanie względem wskazanego rejestru, do którego dodawane jest przesunięcie.
- Używając jakiegokolwiek dostępnego trybu adresacji, przesłania z użyciem pamięci zajmują minimum 2 takty zegara (obliczenie adresu i wykonanie), nie licząc narzutu spowodowanego na konieczność załadowania określonych rejestrów.

Ponadto mikrokontrolery z rodziny LPC2xxx mają wbudowany mostek z podzielnikiem zegara, przy dostępie do układów peryferyjnych, co dodatkowo zmniejsza wydajność (szczególnie przy dostępie do GPIO). Dokumentacja (w tym [4]) do LPC2xxx jest dość uboga i przemilcza niektóre fakty, między innymi to, że zapis do rejestru GPIO trwa 7 taktów[4]. Problem ten przeanalizowano w [5].

Programowanie mikrokontrolera może odbywać się za pomocą sondy JTAG (obecnie trudno dostępne ze względu na cenę) lub za pomocą wbudowanego **boot-loadera** (program zapisujący pamięć FLASH programu) przez 0 port szeregowy. Rozwiązanie układowe zastosowane w platformie ZL1ARM wydaje się być bardzo dobre i ekonomiczne, a dodatkowo może być stosowane w innych aplikacjach układów LPC2xxx.

[3] Dokumentacja firmy ARM [2]

[4] Nie sprawdziliśmy tego doświadczalnie.

2.1. Tryb ładowania programu

Mikrokontrolery z rodziny LPC2xxx po włączeniu zasilania rozpoczynają od wykonania testu, polegającego na sprawdzeniu stanu pinu P0.14 (wariant LPC2124). Należy zatem uznać, że wyprowadzenie to jest specjalnego przeznaczenia. Jeżeli podczas startu pin ten jest w stanie wysokim, to wbudowane oprogramowanie zmieni tryb działania mikrokontrolera na *User Flash Mode* i uruchomi aplikację zapisaną w pamięci FLASH (wykona załadowany program użytkowy). Jeżeli stan będzie niski, to mikrokontroler będzie wykonywał dalszą część boot-loadera, umożliwiającą we współpracy z dostarczonym przez producenta oprogramowaniem, zapisanie pamięci FLASH nowym programem. Jest to tzw. funkcjonalność ISP (In-System Programming).

Aby używać ISP i/lub pętli fazowej zegar zewnętrzny musi być z zakresu 10 ÷ 25 MHz. Platforma ZL1Arm używa zegara **12 MHz**, co wydaje się być dobrym rozwiązaniem, ze względu na łatwość osiągnięcia maksymalnej częstotliwości 60MHz przez pętlę fazową (12 * 5 = 60).

3. Środowisko do kompilacji/uruchamiania – przewodnik instalacji

Aby uruchomić system operacyjny ECOS konieczne jest posiadanie szeregu programów narzędziowych zainstalowanych na komputerze PC[5]. Wszystkie używane narzędzia są dostępne w Internecie jako oprogramowanie darmowe. Sam system eCOS dostępny jest na licencji GNU-GPL.

Uwagi ogólne:

- instalacji jakiegokolwiek programu należy dokonać do katalogu o nazwie nie zawierającej spacji
- należy mieć na dysku łącznie ok. **1.2GB** wolnego miejsca

3.1. Środowisko Cygwin

Jest to zestaw programów emulujących system unix na platformie Windows. Jest niezbędne do prawidłowego funkcjonowania niektórych narzędzi, w szczególności narzędzia do konfiguracji eCOS'a.

1) rozpakować plik CygwinSrc.zip do katalogu tymczasowego, powstanie tam podkatalog CygWinSrc
2) uruchomić cygwinsetup.exe
3) wybrać opcję instalacji z lokalnego katalogu
4) podać katalog instalacji całego środowiska (REKOMENDOWANE!!! **'C:\Cygwin'** Spotkano się z narzędziami, które zakładają istnienie Cygwin'a wyłącznie w takim katalogu)
5) teraz można usunąć tymczasowe pliki powstałe w kroku 1)
6) uruchomić powłokę (powinna znajdować się ikona na pulpicie, skrót do 'C:\Cygwin\cygwin.bat'), zostanie dokończone generowanie ustawień

[5] Używano AMD Duron 700MHz, 160MB RAM z systemem WindowsXP

Zainstalowane środowisko zajmuje do 130MB. Istnieje oczywiście możliwość dodania innych pakietów i programów standardowych unixa, wtedy należy ściągnąć pakiety z sieci, stosując instalację z Internetu.

Narzędzia GNU będą uruchamiane wsadowo z linii poleceń powłoki Cygwina, co nie jest wygodną, ale jedyną drogą do używania tych narzędzi (przynajmniej w załączonej wersji).

3.2. Źródła systemu ECOS

Dystrybucja systemu eCOS znajduje się w pliku eCos-050320.tar.gz (w Internecie mogą być nowsze). Należy rozpakować całość do katalogu (przykładowo D:\Arm\eCOS'). Rozpakowana dystrybucja zajmuje 180MB, choć ze względu na rozmiar klastra rozmiar może sięgnąć nawet 400MB. Katalog ten należy traktować jako repozytorium z plikami nie przeznaczonymi do modyfikacji. Narzędzie do konfiguracji wyekstrahuje z niego wszystko, co potrzebne podczas kompilacji bibliotek systemu eCOS.

3.3. Narzędzie konfiguracji systemu ECOS

Uruchomić instalator configtool-2.11-setup.exe. Dalej postępować wg instrukcji programu. Przykładowy katalog: 'D:\ARM\Programy\eCos Configuration Tool' (tu spacje są dopuszczalne). Narzędzie to jest najważniejsze na etapie budowania określonej konfiguracji systemu eCOS. Do działania wymaga środowiska Cygwin.

UWAGA: W narzędziu istnieje mały błąd polegający na tym, że aby zmiana parametru została „zauważona” trzeba kliknąć po zmianie gdzieś poza aktualnie zmienianą opcję.

3.4. Narzędzia GNU dla mikroprocesora ARM

Narzędzia GNU są jedynymi (aktualnie) umożliwiającymi prawidłową kompilację systemu eCOS. Wymaga ich narzędzie do konfiguracji podczas kompilacji. Zbiór programów - kompilator gcc, linker ld itp... - należy zainstalować przy pomocy programu GnuARM3xx.exe[6]. Tutaj katalog docelowy NIE MOŻE zawierać spacji (narzędzia źle działają)!

Używana dystrybucja GnuARM jest skompilowana dla środowiska Cygwin, jednakże może działać również bez niego dzięki temu, że zawiera już niezbędne biblioteki *.dll. Jednakże, w przypadku używania razem ze środowiskiem Cygwin występuje konflikt pomiędzy biblioteką zawartą w GnuARM a tą z Cygwina. Sygnalizowany jest błąd, którego rozwiązanie polega na usunięciu biblioteki `cygwin1.dll` z katalogu, w którym zainstalowany jest kompilator.

Jeżeli instalator nie zmodyfikuje zmiennej systemowej PATH, to dobrze dodać do niej katalog Cygwina (przykładowo ‘C:\Cygwin\bin’).

[6] Jest to instalator skompilowanej dla środowiska Cygwin wersji narzędzi GNU. Istnieją oczywiście inne kompilacje, ale spośród testowanych tylko ta zadziałała razem z narzędziem do konfiguracji.

Narzędzia GnuARM będziemy wykorzystywać wsadowo z linii komend powłoki Cygwina, dlatego konieczne jest ustalenie ścieżki dostępu. Realizuje to poniższa komenda bash'a (przykładowo dla katalogu '`D:\ARM\Pragramy\GnuARM332`'):

```
export PATH=/cygdrive/d/ARM/Programy/GnuARM332/bin
```

Można również skopiować plik '.bashrc' do katalogu:

```
C:\Cygwin\home\NAZWA_ZALOGOWANEGO_UŻYTKOWNIKA\
```

Plik ten zawiera już ustawione odpowiednie ścieżki dostępu do programów (dla przykładowej konfiguracji katalogów). Narzędzia GnuARM zajmą na dysku ok. 200MB.

3.4.1 WinARM 4.0.0

Jest to skompilowana wersja narzędzi GnuARM w wersji 4.0.0. Przetestowano dystrybucję '`WinARM-20050427-2`'. Nie działa ona z graficznym narzędziem konfiguracji (prawdopodobnie nie zadziała również z 'make' z linii komend), z powodu nierozumienia unixowych ścieżek dostępu do plików w systemie Windows (oddzielanie katalogów znakiem '/' zamiast '\'). Ta kompilacja narzędzi działa jednak bez żadnej zależności od środowiska Cygwin i może być używana do budowania aplikacji w oparciu o już wygenerowane biblioteki eCOS'a (np. przy pomocy starszej wersji). Dodatkowo dużą zaletą tej niezależnej dystrybucji jest możliwość integracji z popularnymi uniwersalnymi edytorami, dedykowanymi do pisania kodów w językach C/C++ (np. darmowe *Syn*, czy *Dev-C++*). Wydaje się, że ta dystrybucja w połączeniu z darmowym uniwersalnym środowiskiem zintegrowanym *Eclipse* ma ogromne możliwości, aczkolwiek nie zostało to sprawdzone.

3.5. Debugger/symulator

System można uruchamiać przy pomocy debuggera GNU – programu `gdb`, jednakże nie testowaliśmy jego działania. W Internecie znaleźć można liczne informacje na temat wadliwego działania zarówno `gdb`, jak i jego graficznej nakładki – `insight`.

3.6. LPC2000 Flash Utility

Narzędzie to dostarcza firma Phillips i służy ono do zapisu i odczytu pamięci w mikrokontrolerze (zarówno SRAM jak i FLASH). Działa tylko w systemie Windows i współpracuje z boot-loaderem w mikrokontrolerze. Na platformie ZL1ARM można użyć maksymalnej szybkości transmisji 38400 bit/s.

4. Jak skompilować i uruchomić aplikację?

Zakładamy, że wszystkie narzędzia, omówione poprzednio zostały zainstalowane. Napisane kody programów stanowią załącznik niniejszego dokumentu. Źródła aplikacji znajdują się w katalogu `work` załącznika.
Jako pierwszą wprawkę polecam skompilować test1 (katalog '`work\1`'). Aby to zrobić należy uruchomić powłokę Cygwina. Następnie należy przejść do katalogu z projektem wpisując komendę podobną do poniższej:

```
cd /cygdrive/d/arm/work/1
```

Ścieżka dostępu wprowadzana jest zgodnie z regułami systemu Unix z tą różnicą, że litery duże i małe nie są rozróżnialne. Aby przejść do katalogu na innym dysku stosujemy przedrostek `/cygdrive/litera_dysku`. Następnie należy skompilować projekt. Stosujemy kompilator arm-elf-gcc wywołując komendę:

```
arm-elf-gcc -g -Os -IEcosConfig/KernelMinimum_install/include
test1.c -LEcosConfig/KernelMinimum_install/lib -Ttarget.ld -
nostdlib
```

Kompiluje ona program zawarty w pliku test1.c. Flagi oznaczają:

`-g`	Generacja symboli do debugowania
`-Os`	Optymalizacja pod względem wielkości
`-Ikatalog`	Powoduje, że pliki standardowe nagłówkowe, dołączane za pomocą dyrektywy #include <plik.h> (nawiasy trójkątne) są wyszukiwane najpierw w podanym katalogu
`-Lkatalog`	Powoduje, że wszelkie biblioteki i skrypty linkera są poszukiwane najpierw we wskazanym katalogu
`-Tskrypt.ld`	Używa podanego skryptu podczas linkowania

Pełny opis opcji kompilatora gcc znajduje się w dokumentacji do niego. Jeżeli kompilacja i linkowanie poszło bezbłędnie, wówczas powstał plik a.out (nazwa standardowa, można ją zmienić używając odpowiedniej flagi). Ten plik należy przekonwertować do postaci akceptowalnej przez *LPC2000 Flash Utility*, czyli do formatu intel-hex (ihex):

```
arm-elf-objcopy -O ihex a.out a.hex
```

Powstały plik a.hex można już podać do *LPC2000 Flash Utility* i zaprogramować pamięć flash mikrokontrolera. Możemy jeszcze dowiedzieć się czegoś o tym, co zostało skompilowane. Listing asemblerowy generujemy używając komendy:

```
arm-elf-objdump -D a.out > a.lst
```

Powstanie plik a.lst. Możemy jeszcze dowiedzieć się, ile pamięci zajmują poszczególne sekcje skompilowanego programu:

```
arm-elf-size
```

Wszystkie wymienione kroki można zautomatyzować zapisując je w skrypcie powłoki. We wszystkich eksperymentach posługiwano się skryptem „m", który kompilował, konwertował i wyświetlał informacje. Z niewielkimi modyfikacjami można go używać do każdego programu. Kompilację poprzez skrypt „m" wykonujemy szybko wpisując komendę:

```
./m
```

5. Aplikacje w systemie ECOS

Aby uruchomić jakąkolwiek aplikację w systemie ECOS konieczne jest utworzenie konfiguracji. Ta ostatnia jest niczym innym jak pewną konkretną wersją systemu, zawierającą zarówno kod startowy dla konkretnego mikroprocesora, kod inicjujący układy peryferyjne, układy akceleracji pamięci itp... (czyli to, czym zajmuje się warstwa HAL), jądro, sterowniki jak i biblioteki standardowe oraz dodatkowe. Możliwe jest zbudowanie konfiguracji minimalnej, nie zawierającej niczego oprócz HAL'a i wywołania punktu startowego aplikacji użytkownika oraz znacznie bardziej rozbudowanej, zawierającej wszystkie sterowniki, biblioteki, jądro i to jeszcze w wersji „debug", z dodatkową funkcjonalnością ułatwiającą uruchamianie. Można dodać również tzw. *GDB stub*, czyli komponent odpowiedzialny za komunikację z oprogramowaniem debuggera na komputerze PC.

5.1. Zbudowanie żądanej konfiguracji systemu ECOS

Budowanie systemu ECOS sprowadza się do wygenerowania żądanej konfiguracji przy pomocy narzędzia Configtool, a następnie kompilacji do postaci biblioteki. Ta ostatnia jest żądaną wersją systemu operacyjnego, z którą aplikacja użytkowa jest łączona (ang. link) w sposób statyczny. W ten sposób powstaje plik który po odpowiedniej konwersji do formatu `ihex` może być załadowany do pamięci Flash mikrokontrolera (oczywiście nie tylko LPC2xxx).

Na platformę ZL1ARM należy użyć szablonu Keil Evaluation Board MCB2100, z którą jest ona kompatybilna. Można eksperymentować z różnymi wersjami szablonu (wybór dostępny w graficznym narzędziu do konfiguracji systemu). Najlepszym punktem startu wydaje się być wersja „Kernel", gdyż nie dołącza standardowego punktu startu, czyli wywołania funkcji main(), i biblioteki ISO C, dla której alokowany jest aż 8KB stos. Nie stosując elementów biblioteki ISO C można z powodzeniem używać mniejszego stosu. W celu zainstalowania systemu eCOS na innej platformie z mikrokontrolerem LPC2xxx należałoby utworzyć nowy szablon (rozdział dot. przenoszenia), lub usunąć ręcznie niektóre linie kodu źródłowego eCOS'a (m. in. te w HAL'u, które odpowiadają za używanie sygnalizacji LED, specyficznej dla platformy).

5.2. Krytyczne punkty konfiguracji

Najważniejszymi, z punktu widzenia stabilności systemu, parametrami są ustawienia stosów dla poszczególnych procesów. Sprawdzono, że rozmiar stosu mniejszy niż 1024 bajty w pewnych sytuacjach był zbyt mały. Przyczyn nie ustalono. Testowano również przy ustawieniach 1024 oraz 2048, z pozytywnym wynikiem.

Nie ustalono, co ma największy wpływ na ilość potrzebnej pamięci potrzebną na stos. Na pewno używanie funkcji ze standardowej biblioteki C (np. funkcja printf) wymaga większego stosu. Również stosowanie głębokich zagnieżdżeń funkcji jest niewskazane, a rekurencja poza mocno uzasadnionymi przypadkami – wręcz zakazana.

Kolejnymi parametrami są ustawienia czasowe. W systemie eCOS oprócz zegara taktującego pobieranie instrukcji (poprzez ustawienia pętli fazowej) możemy skonfigurować okres tzw. timeslice'a, czyli najmniejszego kwantu czasu odmierzanego przez systemowy zegar czasu rzeczywistego. Odbywa się to w definicji stałych:
CYGNUM_HAL_RTC_NUMERATOR,
CYGNUM_HAL_RTC_DENOMINATOR,
CYGNUM_HAL_RTC_PERIOD.
Ich znaczenie opisane jest w [1]. Domyślne ustawienie to kwant czasu 10ms. Wartość ta wynika jeszcze z ustawień pętli fazowej:
mnożnik CYGNUM_HAL_ARM_LPC2XXX_PLL_MUL,
zegar zewnętrzny CYGNUM_HAL_ARM_LPC2XXX_XTAL_FREQ.
Stąd system eCOS wie z jaką szybkością działa i jak ustawić zegar czasu rzeczywistego, by odmierzał właściwie. W przypadku jądra planującego zadania z wielopoziomową kolejką i podziałem czasowym możemy jeszcze ustawić po ilu kwantach nastąpi wywłaszczenie wątku (domyślnie po 5 kwantach). Połączenie kwantu czasu z liczbą do wywłaszczenia nie może dać zbyt krótkiego okresu czasu, ponieważ spowoduje to niedopuszczalny wzrost mocy przetwarzania zmarnowanej na przełączanie kontekstów. Testy były prowadzone przy pozostałych ustawieniach domyślnych.

Biblioteka ISO C dostarcza funkcjonalności w zakresie środowiska zodnego ze standardem, tzn. funkcje standardowe (np. printf) i standardowy punkt wejścia – funkcja main. Ta ostatnia jest wywoływana jako oddzielny wątek z oddzielnym stosem. W przypadku nie używania tej funkcji należy koniecznie usunąć cały moduł, gdyż domyślnie rezerwowanych jest dla niej aż 8KB pamięci RAM na stos! Pisanie aplikacji rozpoczynamy od funkcji `cyg_user_start`, która jest standardowym dostępnym punktem wejścia. Należy pamiętać, że planista (*task scheduler*) uruchamiany jest dopiero po zakończeniu tej funkcji, więc wszystkie uruchamiane wewnątrz wątki rozpoczną działanie dopiero po jej zakończeniu.

5.3. Pierwsza aplikacja – migające diody LED

Kod źródłowy, wraz z konfiguracją systemu eCOS, zawiera katalog `'work\1'`. Jest to prosta dwuwątkowa aplikacja, która rozpoczyna się od startu systemu eCOS. Warstwa HAL konfiguruje wszystkie urządzenia, między innymi również wyjścia portów LED. Po wystartowaniu system przechodzi do funkcji

cyg_user_start, w której mamy sekwencyjne „przemiatanie” wszystkich diod LED, a następnie uruchamiane są dwa wątki. Jeden miga diodą nr 7 a drugi diodą nr 6.

Na tym programie przeprowadzano eksperymenty o charakterze jakościowym z ustawieniami systemu ECOS.

Konfiguracja zgodnie z szablonem „Keil...MCB2100” wersja „Kernel”. Stos kontekstu przerwań[7]: 1024 bajty
Konfigurację nazwano „Kernel.ecc”. Uzyskano zajętość pamięci nic nie robiącego programu (pusta funkcja cyg_user_start, bez dodatkowych definicji zmiennych):

Text	Data	Bss	Flash	RAM
41464	932	4492	42396	5424

Zajętość pamięci Flash wynosi zatem 42KB. Zużywamy ponad 5KB RAM’u na nic nie robiącą aplikację! Kluczowe jest zatem ustawienie wielkości stosu dla wątku tła (*idle thread*). Domyślne ustawienie stosu dla wątku tła to 2048 bajtów. Zmniejszamy to ustawienie do 512 (minimalna dozwolona wartość) bajtów i uzyskujemy:

Text	Data	Bss	Flash	RAM
41328	932	3564	42260	4496

Zajętość pamięci RAM wyznaczono jako sumę sekcji Data i Bss

Uruchomienie nowego wątku wymaga zadeklarowania dla niego:

- stosu – można to zrobić statycznie poprzez deklarację tablicy lub poprzez wskaźnik do dynamicznie alokowanego obszaru
- uchwytu do wątku (zmienna typu `cyg_handle_t`)
- deskryptora wątku (zmienna typu `cyg_thread`)

Pełna aplikacja *test1* (w katalogu ‘`work\1`’) jest skompilowana z wersją KernelMinimal systemu ECOS (stos dla przerwań 512B). Obserwujemy tutaj działanie dwóch wątków, możliwość wywłaszczenia (Planista skonfigurowany jako wielopoziomowa kolejka z podziałem czasowym) itp. Na uwagę zasługuje porównanie działania pętli oczekującej z funkcją systemową `cyg_thread_delay`. Ta pierwsza obciąża procesor i nie należy jej stosować. W przypadku różnych priorytetów zadanie na wyższym priorytecie, które działa w sposób ciągły i jawnie nie odda procesora (funkcja `cyg_thread_delay` oddaje procesor na pewien czas), nie zostanie wywłaszczone przez wątek o niższym priorytecie. W przypadku równych priorytetów zachodzi wywłaszczenie po okresie określonym ustawieniami systemu ECOS (liczba timeslice’ów do wywłaszczenia – domyślnie 5). Wyższy priorytet ma zadanie o niższej wartości. Działanie planisty szczegółowo omawia [1].

5.4. Interfejs diagnostyczny

Interfejs diagnostyczny badano, posługując się zmodyfikowaną aplikacją migającą diodami. System eCOS dostarcza prosty[8] interfejs diagnostyczny umożliwiający wysłanie ciągu znaków poprzez port diagnostyczny za pomocą funkcji

[7] Stos kontekstu przerwań jest używany nie tylko podczas obsługi w przypadku wspólnego stosu dla różnych przerwań, ale również dla procedur startowych, gdy planista nie jest jeszcze uruchomiony i nie działają żadne wątki (te mają swoje oddzielne stosy), czyli m. in. podczas funkcji cyg_user_start.
[8] Jak wspomniano w rozdziale 3.5 nie używamy GDB

`diag_printf`. Port szeregowy jest konfigurowany podczas startu w sposób niewidoczny dla programisty. Parametry połączenia są ustawiane w konfiguracji systemu eCOS (domyślnie transmisja **38400** kbit/s, format 1+8+1). Dostępne jest również makro `hal_diag_led` wyświetlające podaną kombinację na diodach LED, jednak jego użycie ma sens jedynie na platformie posiadającej taką sygnalizację.

Przykład użycia interfejsu diagnostycznego pokazano w aplikacji *test2* (w katalogu `'work\2'`). Jest to zmodyfikowana wersja migająca diodami, wysyłająca dodatkowo poprzez interfejs szeregowy informacje diagnostyczne o starcie poszczególnych funkcji.

5.5. Interfejs szeregowy

System ECOS udostępnia sterowniki do wszelkiego typu układów peryferyjnych wejścia-wyjścia znajdujących się na platformie. Mogą to być wszelkiego rodzaju porty szeregowe, kontrolery Ethernetu, USB, portów równoległych, IRDA, itp. Dostęp do funkcji transmisyjnych jest zunifikowany z punktu widzenia biblioteki API dla każdego rodzaju urządzenia. Platforma ZL1ARM wyposażona jest w dwa porty szeregowe, do których dostęp możliwy jest poprzez wywołanie funkcji sterownika bądź bezpośrednio. Dostęp bezpośredni polega na możliwości modyfikacji rejestrów kontrolno sterujących, ale jest niepotrzebny ze względu na istniejące już gotowe funkcje.

Sterowniki portów szeregowych w systemie ECOS umożliwiają buforowanie, a rozmiar buforów jest konfigurowalny. Porty szeregowe są wstępnie skonfigurowane i gotowe do działania. Szybkość transmisji wybieramy podczas konfiguracji systemu eCOS, Możemy ją również łatwo zmienić poprzez wywołanie funkcji konfigurującej port już podczas działania aplikacji.

Interfejs szeregowy testowano posługując się aplikacją test3 (katalog ‘`work\3`’). Konfiguracja systemu ECOS zmieniła się od poprzedniej dodaniem sterowników portów szeregowych dla UART0 i UART1. Rozmiary buforów wynoszą po 128 bajtów. Konieczne było zwiększenie rozmiaru stosu o 192 bajty dla każdego z wątków, uzyskując 1216 bajtów. Przy zbyt małym rozmiarze aplikacja zawieszała się, co jest zrozumiałe, gdyż zamazaniu ulec mogły zmienne systemowa.

Z niewyjaśnionych przyczyn port szeregowy, który nie jest portem diagnostycznym nie chce wysyłać żadnych danych, jeżeli wpierw wprost nie wpiszemy jakiejkolwiek danej do rejestru nadawczego (w sposób bezpośredni). Spowodowane jest to najprawdopodobniej błędem w inicjalizacji sterownika (port diagnostyczny inicjuje zupełnie inny kod). Problem ten poruszany jest w Internecie na listach dyskusyjnych i w chwili pisania niniejszego dokumentu nie było znane rozwiązanie.

Możliwe jest dodanie sterowników portu szeregowego wyższej warstwy – terminala. Korzystają one z funkcjonalności sterowników sprzętowych. Dodanie polega na zaznaczeniu opcji podczas konfiguracji systemu ECOS. Sterownik terminala różni się od podstawowego portu szeregowego głównie tym, że wysyła echo dla każdego odebranego bajtu.

5.5.1. Dlaczego sterownik interfejsu szeregowego działa wadliwie? Próba rozwiązania problemu.

Wada sterownika szeregowego polega na tym, że instrukcja `cyg_io_write` nie działa na porcie, do którego wcześniej wprost nie zapisano (bezpośrednio do rejestru nadawczego UART'u). Objawia się to tym, że np. instrukcje `diag_printf` działają dobrze, instrukcje `cyg_io_write` na tym samym porcie, na którym jest diagnostyka, również, ale nie działają na drugim (np. UART1). Okazało się, że przy ustawieniu rozmiaru bufora na 0 (czyli bez buforowania – pooling) obie funkcje działają jak trzeba (sterownik jest ok.). **Problem dotyczy jedynie buforowanej transmisji** (tej bardziej atrakcyjnej), sterowanej przerwaniami.

Problem jest o tyle trudny do rozwiązania, że każdy sterownik ma swoje funkcje, których adresy są zapisane w strukturze, która jest elementem tablicy __DEVTABLE__. Dalej nie udało się ustalić, co się dzieje z tymi adresami. Ustalono, że do wysłania znaku używana jest funkcja `serial_write` zaimplementowana w pliku `'packages\io\serial\current\src\common\serial.c'` (linia 309). W zależności, czy jest buforowanie, czy nie ma korzysta albo wprost z funkcji putc (tak nazywa się jedynie wskaźnik w tablicy funkcji, nie ustalono jednak jak nazywa się przypisana do niego funkcja), lub korzysta z bardziej rozbudowanego mechanizmu

Podsumowanie

Ze wstępnej analizy wynika, że system eCOS charakteryzuje się bardzo dużymi możliwościami. Posiada wszystkie podstawowe elementy wielozadaniowego systemu operacyjnego czasu rzeczywistego. Możliwość przenoszenia systemu na różne architektury (procesory) oraz platformy jest bardzo atrakcyjna. eCOS jest darmowy, na licencji GNU-GPL ale istnieje dystrybucja komercyjna z pełnym wsparciem technicznym. Ilość potrzebnej pamięci danych jest dość duża, co utrudnia wykorzystanie systemu na platformach z 16KB RAM. Jako minimum należałoby uznać raczej 32KB. Wnioski te mają charakter ogólny, gdyż zajętość pamięci bardzo zależy od aplikacji.

Literatura

[1] Anthony J. Massa, "Embedded Software Development with ECOS™", Prentice Hall Professional Technical Reference, New Jersey 2003

[2] ARM, "ARM7TDMI Technical Reference Manual", Revision 4p1, 2004

[3] ARM, "ARM Architecture Reference Manual", 2000

[4] Phillips, "LPC2114/2124/2212/2214 User Manual", 2003 May 03

[5] J. Szczesiul, "Czy ARM zawsze znaczy szybki?", EP 5/2005