Systemy wbudowane

Wykład 3: Implementacja programów wbudowanych

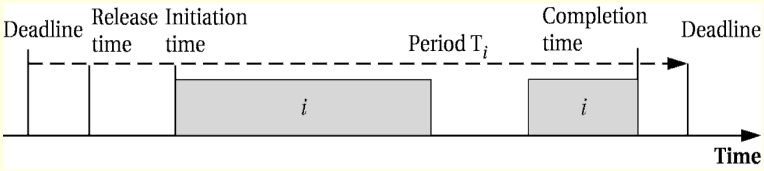
Problemy implementacji oprogramowania wbudowanego

- Szeregowanie zadań
- System operacyjny
- Obsługa przerwań

Szeregowanie w systemach czasu rzeczywistego

- Statyczne
  - Metody konstrukcyjne
  - Metody rafinacyjne
- Dynamiczne
  - Zadania mają priorytety
  - Priorytety statyczne lub dynamiczne
- Quasi-statyczne

Ograniczenia czasowe

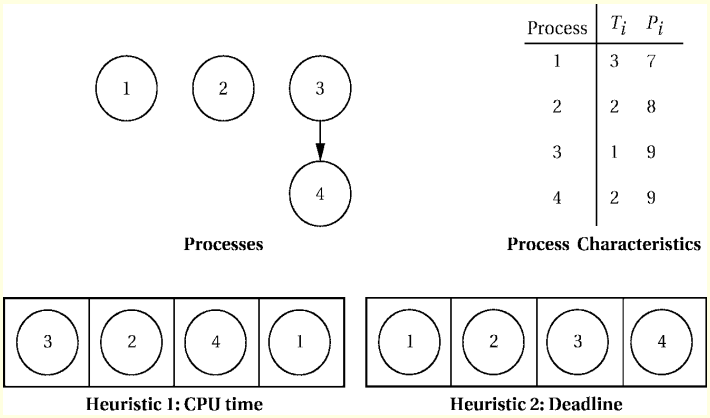


Szeregowanie statyczne

- ASAP
  - As Soon As Possible
- ALAP
  - As Late As Possible

Szeregowanie list

- Najczęściej stosowane



Szeregowanie dynamiczne

- Każdy proces ma priorytet
- Proces może być w 2 stanach: aktywny i oczekujący
- Jako następny wybierany jest oczekujący proces o najwyższym priorytecie
- Priorytety mogą być stałe lub zmienne

RMS(RMA) – Rate Monotonic Scheduling(Analysis)

- Założenia:
  - Nie ma zależności pomiędzy zadaniami
  - Czas przełączania zadań jest nieistotny
  - Czas wykonania zadania jest stały
  - Proces rozpoczyna się na początku swojego okresu
- Stałe priorytety
  - Im wyższa częstotliwość wykonywania zadania tym wyższy priorytet

Earliest-deadline-first (EDF) scheduling

- Priorytet dynamiczny
  - Proces mający mniej czasu do ograniczenia czasowego ma wyższy priorytet
- Aktualizacja priorytetów po każdym przełączeniu zadań
- Nie zapewnia poprawności uszeregowania!

Szeregowanie LLF

- Least Laxity First
  - Procesy mające najmniej swobody czasowej mają wyższy priorytet
- Uwzględnia czas wykonania (w przeciwieństwie do EDF)

System operacyjny czasu rzeczywistego

- Szeregowanie procesów:
  - Zapewnienie spełnienia ograniczeń czasowych
  - W uniwersalnych SO celem jest równy podział czasu procesora pomiędzy zadania
- SO dla systemów wbudowanych:
  - Windows CE, µCLinux, MicroC/OSII, eCos itp.

Zarządzanie pamięcią

- Rzadziej stosowana w RTOS
- Zastosowania:
  - Organizacja pamięci wirtualnej (w pamięci FLASH)
  - Rezerwacja pamięci dla zadań (np. W przypadku uruchamiania zadań zewnętrznych)
- Windows CE:
  - Adresy 32-bitowe
  - Dolne 2GB dla programów użytkownika
  - Górne 2GB dla SO

Mechanizmy zapewniające RT

- Przerwania
  - Implementują priorytety
- Szeregowanie
  - Zapewnia spełnienie wymagań czasowych

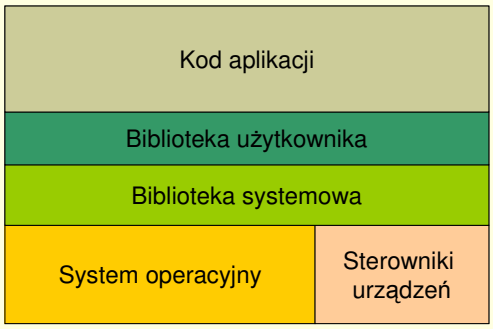
Obsługa przerwań

- Przerwania sprzętowe:
  - Wyższy priorytet niż przerwania programowe
  - Obsługa (ISR) powinna być jak najszybsza
- Przerwania programowe:
  - Obsługa (IST) – mechanizm szeregowania

Komunikacja i synchronizacja pomiędzy procesami

- Mailbox – przesyłanie komunikatów
- Kolejki komunikatów
- Semafory
- Zdarzenia/sygnały

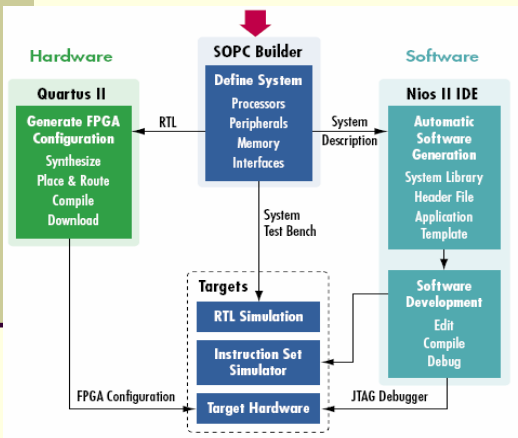
Program wbudowany



Obsługa urządzeń zewnętrznych

- Poprzez system operacyjny
  - Sterowniki zgodne z wymaganiami RTOS
  - Standardowe funkcje C (np. fread(), fwrite())
- Bezpośrednio
  - Własne sterowniki
  - Programowanie niskopoziomowe

NIOS II EDS



Hardware:
- procesor Nios II (32b, RISC),
- szyna Avalon,
- moduły IP: kontrolery RAM, DMA, moduły DSP, itp.
- automatyczna konfiguracja systemu.

Software:
- system operacyjny MicroC/OSII,
- stos TCP/IP,
- automatyczna generacja szablonu projektu,
- debuger rzeczywistego systemu.

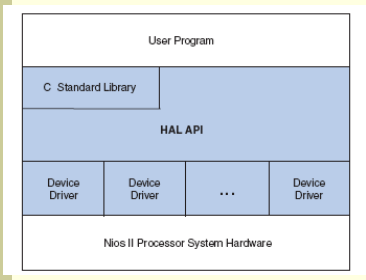
Nios II EDS

- Język C/C++
- System operacyjny MicroC/OS II
- Stos NicheStack TCP/IP
- System plików RO ZIP
- Debugger

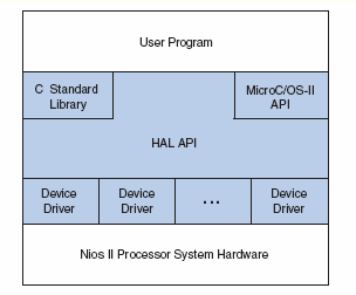
Struktura systemu (1)

- Kod aplikacji
- Biblioteka aplikacji
- Biblioteka systemowa BSP:
  - HAL (Hardware Abstraction Layer)
  - NewLib C Standard Library
  - Opcjonalne pakiety (np. TCP/IP)
  - RTOS

Struktura systemu (2)



Bez RTOS



Z systemem MicroC/OS-II

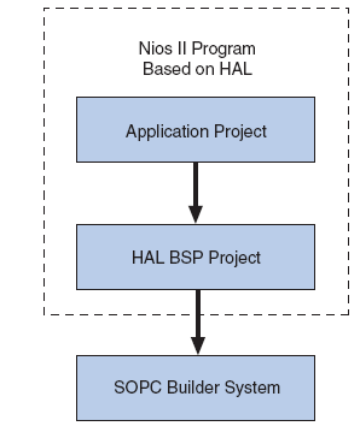
HAL - zadania

- Implementacja standardowych funkcji I/O języka C
- Dostęp do urządzeń I/O
- API dla języka C
- Inicjalizacja systemu
- Inicjalizacja urządzeń I/O

HAL – klasy urządzeń

- Znakowe (np. UART)
- Timer
- System plików
- Ethernet
- DMA
- Pamięć FLASH

Tworzenie projektu



Pliki tworzone przez projektanta

Pliki tworzone przez NiosII IDE (BSP Settings)

Pliki tworzone przez SOPC Builder (.ptf .sopc)

Plik system.h

- Konfiguracje urządzeń
- Adresy bazowe
- Przerwania
- Nazwy symboliczne

```
/*
* sys_clk_timer configuration
*
*/
#define SYS_CLK_TIMER_NAME "/dev/sys_clk_timer"
#define SYS_CLK_TIMER_TYPE "altera_avalon_timer"
#define SYS_CLK_TIMER_BASE 0x00920800
#define SYS_CLK_TIMER_IRQ 0
#define SYS_CLK_TIMER_ALWAYS_RUN 0
#define SYS_CLK_TIMER_FIXED_PERIOD 0
```


Typy danych i funkcje I/O

- alt_types.h:
  - alt_8, alt_u8, alt_16, alt_u16, …
- Standardowe funkcje:
  - Standardowe funkcje C (np. fopen())
  - Funkcje unixowe (np. open())

Przykład: pomiar czasu wykonania

```
#include <stdio.h>
#include "sys/alt_timestamp.h"
#include "alt_types.h"
int main (void)
{
      alt_u32 time1;
      alt_u32 time2;
      alt_u32 time3;
      if (alt_timestamp_start() < 0) {
            printf ("No timestamp device available\n");
      } else {
            time1 = alt_timestamp();
            func1(); /* first function to monitor */
            time2 = alt_timestamp();
            func2(); /* second function to monitor */
            time3 = alt_timestamp();
            printf ("time in func1 = %u ticks\n",(unsigned int) (time2 – time1));
            printf ("time in func2 = %u ticks\n",(unsigned int) (time3 – time2));
            printf ("Number of ticks per second = %u\n",(unsigned int)alt_timestamp_freq());
      }
      return 0;
}
```


Tworzenie własnych sterowników

- Utworzenie pliku nagłówkowego
- Implementacja sterownika
- Testowanie kodu
- Integracja z HAL

Plik nagłówkowy

- <altera_inst>\ip\sopc_builder_ip\<component>
- .\inc\<component>_regs.h
  - IORD_<component>_<register>(adres_bazowy)
  - IOWR_<component>_<register>(adres_bazowy, dane)
  - IOADDR_<component>_<register>(adres_bazowy)
  - <component>_<register>_<field>_MSK
  - <component>_<register>_<field>_OFST

Sterowniki dla urządzeń znakowych

- Utworzenie instancji urządzenia:

```
typedef struct {
alt_llist llist; /* for internal use */
const char* name;
int (*open) (alt_fd* fd, const char* name, int flags, int mode);
int (*close) (alt_fd* fd);
int (*read) (alt_fd* fd, char* ptr, int len);
int (*write) (alt_fd* fd, const char* ptr, int len);
int (*lseek) (alt_fd* fd, int ptr, int dir);
int (*fstat) (alt_fd* fd, struct stat* buf);
int (*ioctl) (alt_fd* fd, int req, void* arg);
} alt_dev;
```

- Rejestracja urządzenia:

  int alt_dev_reg (alt_dev* dev)

Integracja sterowników z HAL

- Struktura kartoteki:
  - <component>
    - HAL
      - inc - <component>.h
      - src - component.mk
    - inc

Inicjalizacja

- alt_sys_init.c
  - alt_sys_init():
    - #include ”component.h”
    - <component>_INSTANCE (NAME, dev)
    - <component>_INIT(NAME, dev)

Koniec