2.1. КОНТРОЛЕН ПРИМЕР ЗА ИЗПОЛЗВАНЕ НА ОПЕРАТОРА *PLACED PAR*

Поставената в работата основна цел предполага явно използване на глобалния структурен паралелизъм на изпълнителната среда *XCORE/XC*. Разглежданият контролен пример съдържа няколко важни от тази гледна точка момента: връзката с апаратните ресурси, обявяването на апаратна нишка, използването на копие на апаратната нишка, паралелната композиция на процесите[1].

Формалната дефиниция на паралелната система *S* от контролния пример се дава чрез следната *CSP* спецификация:

$$S = \{P_1 \,||P_2||P_3\,||P_4\}$$

$$P_1 = P_2 = P_3 = P_4 = P \qquad (2.1.1)$$

$$P = \{x = 1;* \{leds = x; delay; x = \bar{x}\}\}$$

Съгласно спецификацията (2.1.1), паралелната система *S* е композиция от четири процеса P_1, P_2, P_3 и P_4. Те са тъждествени на процеса *P*, което означава идентичност на кода им. Процесът *P* е цикличен и се заключва в периодичното извеждане на променливата *x* на светодиодния индикатор *leds*. След всяко извеждане се отработва времевата задръжка *delay*, след което променливата *x* се инвертира.

Решението се основава на тясната връзка между *CSP*, като език за формална спецификация, и езика на реализацията *XC*. За разлика от *CSP* обаче, при реализацията се налага предварителното деклариране на системните константи, типове, функции, осигуряване на връзката с хардуера:

1 Понятията *процес*, *нишка* и *задача* не се разграничават в нашия случай и се използват като синоними. В друг контекст обаче, те могат да обозначават различен тип активни обекти. Например, в областта на операционните системи процесите са тежки единици работа със собствено адресно пространство; докато нишките са "олекотени" процеси и поделят общото адресно пространство на родителския процес. Това разделяне при операционните системи произтича от липсата на пълна апаратна поддръжка и необходимостта от минимизиране на незибежните системни издръжки.

```
#include <platform.h>

// 200 ms = 0.2 sec = 20E6 x 10E-9 = 200E-3
#define FLASH_PERIOD 20000000

on stdcore[0]: out port x0ledA = PORT_LED_0_0;
on stdcore[0]: out port x0ledB = PORT_LED_0_1;
on stdcore[1]: out port x1ledA = PORT_LED_1_0;
on stdcore[1]: out port x1ledB = PORT_LED_1_1;
```

Включването на заглавния файл *platform.h* е продиктувано от необходимостта за връзка със системния хардуер, базиран на фамилията *XS1*. Чрез този файл се достъпват логическите адреси на апаратните ресурси. Генерира се автоматично от развойната среда *xTIMEComposer Studio* чрез описанието на апаратните ресурси в конфигурационния *XN* файл към проекта. От своя страна, *XN* файлът се генерира автоматично при създаването на проект, базиран на конкретен развоен кит. Ако се работи със специализирана целева система, *XN* файла се подготвя от конструктора [Б.11, Б.17, Б.18].

За отработване на задръжката от *200 ms*, посочена в уравнението за *P* като *delay*, се налага да се използва системен таймер. Интервалът `FLASH_PERIOD` между текущата стойност на таймера и стойността, при която таймерът отново трябва да сработи, се изчислява чрез израза

$$T = \frac{200 \times 10^{-3}}{10 \times 10^{-9}} = 200 \times 10^{5} \qquad (2.1.2)$$

Знаменателят в израза (2.1.2) се обяснява с периода от *10 ns* на тактуване на системните таймери, включени в ядрото *XCORE*.

Достъпът до еднобитовите светодиодни индикатори на *Възел*$_0$ и *Възел*$_1$ се извършва чрез обръщение към логическите адреси на портовете им. Тези адреси се присвояват на съответните променливи, имащи вградения в *XC* тип **`out port`**. Модификаторът **`on`** `stdcore[0]` в началото на декларацията показва възела (физическото ядро), с което е свързан дадения апаратен ресурс, в случая – изходния порт.

Обявяването на апаратна нишка[2] предполага деклариране на прототипа на главната ѝ функция, последвано от определяне тялото на тази функция. Интересна особеност е възможността за

2 Когато се говори за *апаратна нишка* просто се подчертава вградената на апаратно ниво поддръжка на паралелизма. При това, броят на нишките (разбирай също *процесите*, *задачите*) не може да надвишава стойността, непосредствено поддържана от архитектурата. За един възел от типа *XCORE*, тази стойност е *8*.

използване на няколко копия на една и съща главна функция. Тук по спецификация имаме един процес *P* и не е нужно да се обявяват и дефинират четири процеса с един и същ код[3]. Декларира се прототипа на главната функция на процеса *P*

```
void flashLED (out port led, int delay);
```

Тялото на тази функция, определено по-долу, отговаря на уравнението за *P* от формалната *CSP* спецификация

```
void flashLED(out port led, int delay)
{
  timer tmr;
  unsigned t;
  unsigned ledOn = 1;

  tmr :> t;

  while (1)
  {
    led <: ledOn;

    t += delay;
    tmr when timerafter(t) :> void;

    ledOn = !ledOn;
  }
}
```

Разбира се, реализацията съдържа повече детайли от спецификацията, поради по-ниското йерархично ниво, на което се намира. Променливата `ledOn` отговаря на *x* от спецификацията.

За отработването на задръжката *delay* се налага да се използва системен таймер, като част от посочената в т.1.1 базова концепция *софтуерно-дефиниран силиций* (фиг. 1.8).

Самото отработване на задръжката е пример за вградената на апаратно ниво поддръжка на *механизма на събитията*, като средство за синхронизация с асинхронни външни събития[4]. В конкретния случай, операторът **when** и функцията `timerafter()` са просто езиков интерфейс към посочения апаратен механизъм. Процесът

3 Такова множествено определение е допустимо, но при него се губи част от красотата на паралелното решение.

4 Наличието на апаратен механизъм за обработка на събитията в ядрото *XCORE* го определя като *събитийно* (event-driven) и съответно *DLP* ядро.

P се блокира на оператора **when**, докато не настъпи събитието, т.е. докато текущата стойност на таймера не стане равна на стойността на параметъра `t`. След настъпването на това събитие, процесът се разблокира[5], а в `t` се записва следващия момент на сработване на таймера.

Командата за паралелна композиция се реализира чрез паралелния оператор **par** на *XC* и се разполага в тялото на главната функция **main**(), т.е. на най-високото ниво на декомпозиция на системата, отговарящо на *CSP* дефиницията на *процеса S*

```
int main(void)
{
  par
  {
    on stdcore[0]: flashLED(x0ledA, FLASH_PERIOD);
    on stdcore[0]: flashLED(x0ledB, FLASH_PERIOD);
    on stdcore[1]: flashLED(x1ledA, FLASH_PERIOD);
    on stdcore[1]: flashLED(x1ledB, FLASH_PERIOD);
  }

  return 0;
}
```

Използваната в случая разновидност на паралелния оператор **par** се нарича *placed par* и може да се използва само във функцията **main**(). Служи за задаване разположението на процесите във възлите на системата, задача, известна при ОС като *tasks allocation*. Детерминираният характер на изпълнението на процесите от ядрото *XCORE*, както и апаратната поддръжка на паралелизма, опростяват тази задача. За да се определи правилното привързване на процесите с апаратните възли е достатъчно да се отчита следното:

1. Да не се надхвърлят преодставяните от ядрото апаратни ресурси[6].

2. Ако в рамките на едно ядро се изпълняват най-много $n = 4$ нишки, всяка от тях гарантирано се изпълнява с максималната производителност[7] на ядрото. Ако броят на нишките *n* надхвърли *4*, на всяка от тях се отделят *1/n* машинни цикли и гарантираната производителност на нишка намалява [Б.6, Б.17]. Тази особеност се

5 *Блокирането* (деактивирането) и *разблокирането* (активирането) на процесите също е част от вградената на апаратно ниво поддръжка на паралелизма и се извършва не от операционна система, а апаратно.

6 Изпълнението на това условие се контролира от развойната среда *xTIMEcomposer Studio*. При нарушаването му, компилацията на проекта завършва с грешка.

7 Производителността се измерва в *MIPS*.

обяснява с дължината на *изпълнителния конвейер* на ядрото, която при *XS1* е *4*. Освен това, диспечеризацията на нишките е с *фина гранулация* (*fine-grained MTP*) - активните нишки се превключват на всеки машинен цикъл. За разлика от многоядрените процесори с общо предназначение, от рода на *Intel x86*, които са с *едра гранулация* (*coarse-grained MTP*) - активните нишки се превключват на всеки системен квант, зададен от ОС.

В основата на конструкцията *placed par* са операторът **`on`** и глобалната векторна променлива `stdcore[]`. При четиривъзлова паралелна система, като разглежданата, променливата `stdcore[]` има размерност *4* и `stdcore[i]` отговаря на *Възел*$_i$.

Разполагането на обръщение към главната функция `flashLED()` на процеса *P* в оператора **`par`** води до автоматичното създаване на апаратни нишки. Първите две от тях се изпълняват във *Възел*$_0$, а другите две - във *Възел*$_1$. Фактическите аргументи, предавани на всяко едно копие на главната функция на апаратната нишка, служат за неговата параметризация – алгоритъмът на копията е един и същ, но чрез различните параметри се отчита спецификата на всяко отделно копие.

Може да се направи извод за тясната връзка между конструкцията **`par`** на *XC* и паралелната команда на *CSP*. Модификацията *placed par* на паралелния оператор служи за явно изобразяване на процесите върху апаратните възли.

Операторите за въвеждане `:>` и извеждане `<:` на *XC* изцяло отговарят на *CSP* командите за изпращане *!* и получаване *?* на съобщения. При това те са напълно независими от физическото разположение на процесите по възли.

Наличието на вградените в *XC* типове **`timer`**, **`chan`** и **`port`** е пряко следствие на апаратната поддръжка в *XCORE* на ресурсите, необходими за работа в реално време, на средствата за взаимодействие между процесите и на интегрирания в ядрото вход/изход, т.е. съществена част от концепцията *софтуерно-дефиниран силиций*.