МФТИ ИСП РАН

Метод обеспечения устойчивости функционирования бортовой операционной системы реального времени

Иван Дорошенко, Виталий Чепцов. 23 мая 2023 г.

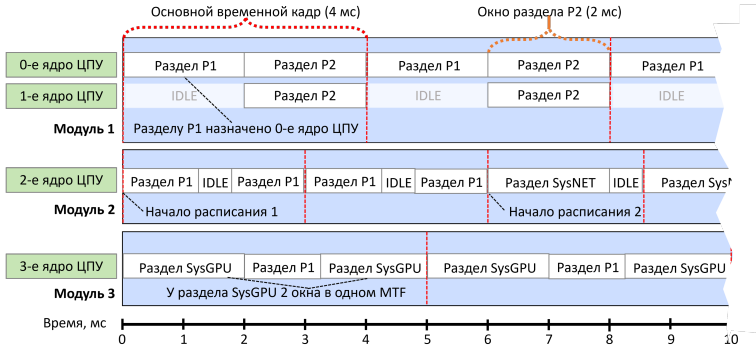
Архитектура ИМА

Операционная система реального времени (ОСРВ) — операционная система, способная обеспечить требуемый уровень сервиса в определённый промежуток времени. **Бортовые ОСРВ** используются для управления бортовым оборудованием.

Одним из стандартов на программное обеспечение, используемое в бортовом оборудовании, является **ARINC 653**, регламентирующий разделение ресурсов между **разделами**, которые сопоставляются функциональным узлам.

Институт Системного Программирования РАН разрабатывает ряд ОСРВ для ответственного применения, ориентированных на соответствие ARINC 653, в том числе для аэрокосмической отрасли.

Планирование в ARINC 653



- Расписания распределяют время по окнам между разделами в рамках модуля.
- Окна расписания повторяются каждый основной временной кадр.
- Ядра ЦПУ назначаются разделу в рамках расписания.

WDT как часть BITE

BITE (Built-In Test Equipment, встроенная система самопроверок) — подсистема, ответственная за запуск тестов, позволяющих своевременно обнаружить аномальное поведение ОСРВ, связанное с ошибками или аппаратными сбоями.

WatchDog Timers (WDT, сторожевые таймеры) — один из обособленных аппаратных компонентов BITE. Система, однажды запустив такой таймер, периодически взводит (**взводит**) его заново. Если взвод не был произведен вовремя, WDT генерирует прерывание или перезагружает систему.

Цель

Цель: разработка метода обеспечения устойчивости функционирования бортовой ОСРВ, реализующей стандарт ARINC 653, основывающийся на применении WDT.

WatchDog должен защищать функции операционной системы, такие к системный таймер и системные разделы, так как для обнаружения и исправления ошибок прикладного ПО используются механизмы ARINC 653, такие как подсистема Health Monitor и Time Management Services.

Нельзя «переусложнять» этот метод, так как это может понизить надёжность подсистемы, представляющей собой «последний рубеж».

Возможности WDT на разных платформах

Arch	Board	Кол-во	Управление	Pre-init	Реарм	Разрешение	Таймаут	Значения таймаута	Прерывания	Вкл / выкл	Debug mode
arm	imx6	2	mmap regs	Сбросить один бит	Когда угодно	0.5 с	Int и reset отдельно	0.5 с – 128 с	Есть	Таймер on/off, прерывания только при первой записи в регистр	Есть
aarch64	Платформа 1	1+1	mmap regs	Нет	Когда угодно	Тики pclk без делителя	Через T int, ещё через T reset	16 вариантов от 0x0000ffff до 0x7fffffff тиков	Есть	Все можно включать и выключать	Нет
armv7m	Платформа 2	1 (IWDT)	mmap regs	Нет	Когда угодно	С 0.125 до 8 мс по степеням двойки	Только для ресета	В преддепенных тиках от 0 до 2^12 - 1	Нет	Все можно включить и нельзя выключить	Есть
		1 (WWDT)			Только в окно (настраивается на <= 65 тиков до сброса)	t_PCLK1 * 4096 * 2^(WDGTB)	Прерывание только за тик, ресет настраивается	Ресет таймаут не более 65 тиков	Есть		
mips	Платформа 3	1	mmap regs	Включить режим WDT	Когда угодно по запутанной схеме	Тики тактового генератора, 8-бит прескейп	Только для прерывания	От 1 до 2^32 прескейленных тиков	Только прерывания	Все можно включать и выключать	Нет
	Платформа 4	3 унив. Тайм.	mmap regs	Сделать из универсального таймера сторожевой	Когда угодно	50 нс (1 / 24MHz)	Int и reset отдельно	От 0 до 2^32-1 тиков	Есть	Все можно включать и выключать	Нет
PPC	Платформа 5	1	mmap regs	Нет	Когда угодно	Тики одного из выбранных осциллпторов без скейпа	Через T int, через еще T reset	На выбор из 4 вариантов	Есть	Все можно включать и выключать, хардресет нельзя выключить	Есть
	Платформа 6	1	Специальные регистры + mtspr	Нет	**Не предусмотрен** Очистка int бита после прерывания	Тики основного таймера	Через T int, через еще T reset	Степень двойки на выбор (2^0 - 2^63)	Есть	Все можно включать и выключать, хардресет нельзя выключить	Есть

Возможности WDT на разных платформах

Некоторые общие возможности, имеющие отношение к проектированию метода:

- Возможности таймеров в настройке таймаута отличаются очень сильно, но почти все допускают таймаут хотя бы в 20 секунд.
- Некоторые таймеры генерируют прерывания спустя половину времени, отведенного под взвод, и не допускают отдельной настройки таймаута для прерываний.
- Таймер можно включать и выключать на всех платформах в произвольное время.
- Взвод можно производить в любое время.

Библиография

- Mallachiev K.M., Pakulin N.V., Khoroshilov A.V. Design and architecture of real-time operating system. Trudy ISP RAN/Proc. ISP RAS, vol.28, issue 2, 2016
- Goldberg, A., & Horvath, G. Software Fault Protection with ARINC 653. 2007 IEEE Aerospace Conference.
- Tomayko, James E. Computers in spaceflight: The NASA experience. 1988
- MicroSin.net: Руководство по сторожевым таймерам для встраиваемых систем

Анализ существующих подходов

Существующие подходы можно классифицировать двумя способами:

1. место расположения менеджера WDT, ответственного за взвод: ядро или выделенный процесс (раздел в ARINC 653)
2. условия взвода: безусловный взвод или взвод только при выполнении некоторых условий (при нарушении которых система считается «нездоровой»)

Во втором подходе под условиями, необходимыми для сброса, подразумевается то, что периодически выполняющиеся разделы должны вовремя отчитываться.

Анализ существующих подходов

Такие подходы были описаны в первую очередь для защиты от зависания прикладного ПО в системах, предоставляющих WDT как отдельное устройство:

- Отдельный процесс менеджера позволяет не переписывать существующий код ОС.
- Условный взвод предназначен для обнаружения зависаний как отдельных процессов, так и совместных (например, дедлоки).

Добавление условного взвода с участием системных разделов может быть целесообразным, потому что если системный раздел даже не получает управление, то система функционирует некорректно.

При этом выделение менеджера в отдельный раздел потребует обеспечения коммуникации этого раздела с другими, что значительно понизит надежность метода.

Предлагаемый метод

Кратко опишем подход, предложенный после проведенного анализа:

1. Менеджер WDT, ответственный за его взвод, находится в ядре. Таким образом, для него нет проблемы собирать «отчёты» от раздела в конце его frame.
2. Менеджер хранит битовую маску, в которую сохраняются «отчеты». В конце каждого major time frame менеджер проверяет, что все биты в этой маске установлены, и в таком случае производит сброс и обнуляет маску.
3. Системный интегратор определяет, какие биты может устанавливать тот или иной системный раздел. Раздел совершает специальный системный вызов, чтобы «отчитаться», т.е. установить некоторый бит.
4. Специально отмеченные разделы, которые выполняют очень критичную работу, могут отключать WDT (например, при перепрошивке).
5. Конфигурация доступных битов, времени таймаута WDT и других параметров определяется в XML-файлах конфигурации и не изменяется в течение работы модуля.

Пример конфигурирования

```
<!-- Module config -->
<Watchdog Period="300ms"/>

<WDog_Bit_List>
    <WDog_Bit Index="0">
        <WDog_Partition_Bit_Ref PartitionNameRef="P1" Id="p1_bit"/>
    </WDog_Bit>

    <WDog_Bit Index="1" Shareable="true">
        <WDog_Partition_Bit_Ref PartitionNameRef="P1" Id="common_bit"/>
        <WDog_Partition_Bit_Ref PartitionNameRef="P2" Id="wdt_bit"/>
    </WDog_Bit>
</WDog_Bit_List>

<!-- Partition 1 config -->
<WDog_Partition_Bit_List>
        <WDog_Partition_Bit Id="p1_bit"/>
        <WDog_Partition_Bit Id="common_bit"/>
</WDog_Partition_Bit_List>
```

Пример использования в userspace

```
uint64_t mask = 0;
GET_WATCHDOG_PARTITION_MASK(WATCHDOG_PARTITION_MASK: &mask, RETURN_CODE: &ret);
printf(format: "1:mask: %llx\n", mask);

WATCHDOG_BIT_INDEX_TYPE p1_bit, common_bit;
CREATE_WATCHDOG_BIT(WATCHDOG_BIT_ID: "p1_bit", WATCHDOG_BIT_INDEX: &p1_bit, RETURN_CODE: &ret);
CREATE_WATCHDOG_BIT(WATCHDOG_BIT_ID: "common_bit", WATCHDOG_BIT_INDEX: &common_bit, RETURN_CODE: &ret);

// ...

while(1)
{
  // Полезная работа

  SET_WATCHDOG_BIT(WATCHDOG_BIT_INDEX: p1_bit, RETURN_CODE: &ret);
  SET_WATCHDOG_BIT(WATCHDOG_BIT_INDEX: common_bit, RETURN_CODE: &ret);
}
```

Результаты и дальнейшие планы

Результаты:

1. Разработана спецификация метода.
2. Спроектирован API для реализации метода:
   - на уровне конфигурации ARINC 653 (XML)
   - на уровне интерфейса функционального ПО ARINC 653 (C)
   - на уровне интерфейса драйвера ядра для WDT (C)
3. Написан черновой прототип для i.MX6.

Дальнейшие планы:

1. Обобщить прототип и подготовить к code-review (до 31.05).
2. Выполнить интеграцию с другими драйверами WatchDog, разрабатываемыми в ИСП РАН.
3. Внедрить в ОСРВ ИСП РАН (до конца лета).

МФТИ ИСП РАН

Вопросы?

Иван Дорошенко, Виталий Чепцов