Протокол асимметричного чередования битов

апрель 2004 г.

1 Введение

Протокол чередующихся битов (енг. *Alternating Bit Protocol*) хорошо известен. Но это симметричный протокол для двух участников одного ранга/уровня.

Описанный здесь протокол представляет собой асимметричный вариант протокола чередующихся битов. Его следует использовать, когда один из участников опрашивает другого (енг. *polling*). Этот участник обычно называется **мастером** (енг. *master*). Таким образом, нам необходимо проверить как ведущую, так и ведомую сторону на коммуникационной шине.

Один мастер может общаться с несколькими ведомыми через одну и ту же шину. Для каждого ведомого у ведущего есть отдельный экземпляр протокола.

В последующем описании предполагается, что проверка правильности передачи сообщений выполняется с помощью CRC, но это всего лишь распространенный вариант использования. Протокол не определяет, какой метод будет использоваться для этой проверки — например, «интернет-контрольная сумма» является опцией.

2 С точки зрения мастера

При первом опросе ведомого ведущее устройство устанавливает бит FIB в 1.

Всякий раз, когда он опрашивает ведомое устройство, мастер проверяет, есть ли какие-либо сообщения для отправки указанному ведомому устройству, и отправляет их, если они есть. Если их нет, мастер отправит заполняющее (пустое) сообщение. В любом случае он отправляет бит FIB, как и предполагалось.

После передачи ведущий ожидает ответа от ведомого.

Если ответ истекает или приходит с ошибкой (в первую очередь с ошибкой CRC), бит FIB останется «как есть», и отправленное сообщение не будет выпущено. Таким образом, он будет снова отправлен на следующий опрос для этого подчиненного устройства. Разумеется, сообщение с ошибкой игнорируется.

Если приходит ответ, мастер сверяет полученный бит FIB с отправленным. Если они совпадают, то для целей этого протокола время ожидания ответа истекло (для диагностики ведомое устройство «живое»).

Если полученный бит FIB отличается (инвертирован/альтернирован) от отправленного, то мастер предполагает, что сообщение было получено правильно на ведомой стороне, и

отправленное сообщение будет освобождено. Затем мастер инвертирует бит FIB для этого ведомого, чтобы подготовиться к следующему опросу.

Таким образом заканчивается опрос одного ведомого и ведущий переходит к следующему. Как только будут опрошены все остальные ведомые устройства, ведущий вернется к этому.

В целях диагностики, если ведомое устройство не отвечает на опрос несколько раз подряд, оно считается «неработающим».

3 С точки зрения раба

Во время инициализации ведомое устройство устанавливает бит FIB в 1.

Если он принял сообщение правильно (проверка CRC прошла успешно), ведомое устройство проверяет, полученный бит FIB такой же, как и его собственный бит FIB, или отличается от него.

Если FIB соответствует ожиданиям, это означает, что полученное сообщение является новым и что наше сообщение (из последнего опроса) было принято. Ведомое устройство выпускает это сообщение из последнего опроса и инвертирует/меняет бит FIB. Затем он отправляет в ответ на опрос свое собственное сообщение или заполняющее (пустое) сообщение с таким битом FIB.

Если FIB отличается от ожидаемого, это означает, что мастер неправильно получил ответ при последнем опросе. Только что полученное сообщение игнорируется, и ведомое устройство отправляет то же сообщение, что и в последнем опросе, сохраняя тот же бит FIB, что и в этом последнем опросе.

Если сообщение получено с ошибкой CRC, обработка аналогична — ведомое устройство отправляет то же сообщение, что и в последнем опросе, с тем же битом FIB (и, конечно, игнорирует сообщение с ошибкой CRC).

4 Обсуждение

Если ведомое устройство установит FIB в 0 во время инициализации, протокол потеряет самое первое сообщение. Но на практике это не обязательно, потому что инициализация ведомого устройства асинхронна с инициализацией ведущего устройства. Таким образом, фактическая установка FIB в 1 во время инициализации ведомого устройства не является 100% гарантией того, что первое сообщение не будет потеряно.

Поэтому, когда мастер делает первый опрос, он должен отправить пустое (заполняющее) сообщение, чтобы можно было не сомневаться в его потере.

К сожалению, даже это не решит *все возможные случаи*. Так как инициализация может занять много времени и во время нее делается много всего, все еще возможно, что время таково, что сообщения теряются. Решить это было бы возможно, но предполагается, что ведомое устройство не часто инициализируется, поэтому оно не стоит усилий.

5 SPIN модель протокола

Мы представляем исходный код Promela для модели SPIN этого протокола. Его следует сохранить в отдельный файл, скажем, `FIB_BIT_KOM` (файлы Promela обычно не имеют расширения). Если такой файл ”пропустить” через SPIN model checker, то он покажет, что протокол исправен.

Этот исходный код написан на «жаргоне» системы SRCE, поэтому KOP является ведущим, а RP — подчиненным в контексте протокола, который мы описываем в этом документе.

Модель имеет три процесса:

1. КОП - мастер
2. RP - ведомый (только один, т.к. он одинаков для всех и любого RP)
3. коммуникационная шина, для имитации ошибок связи

Средство проверки модели SPIN может установить, что протокол передает сообщения правильно (по порядку) и не блокируется. Чтобы SPIN не сообщал о бесконечно плохой коммуникационной шине (которая теряет *все* сообщения) как о тупике протокола, мы пометили соответствующие строки кода метками `progress...`.

```
/** Maksimal'noye kolichestvo soobshcheniy, dolzhno byt'
    bol'she dvukh. */
#define MAX    10

/** Vspomogatel'noye simvolicheskoye opredeleniye
    «plokhogo CRC» */
#define BAD_CRC  0

/** Vspomogatel'noye simvolicheskoye opredeleniye
    «pravil'noy/khoroshey CRC» */
#define GOOD_CRC 1

/** KOP, vedushchiy, oprashivayet RP, vedomyye ustroystva
    na shine. Dlya etoy modeli nam nuzhen tol'ko odin RP,
    tak kak on odinakov dlya vsekh.
    @param in Kanal, po kotoromu KOP poluchayet soobshcheniya
    @param out Kanal, po kotoromu KOP otpravlyayet soobshcheniya
    */
proctype kop(chan in, out)
{
    byte by; /* Soobshcheniye dlya otpravki; v testa chislo */
    bit fib;
    bit fib_prim; /* FIB v poluchennom otvetnom soobshchenii */
    byte by_prim; /* Poluchennoye otvetnoye soobshcheniye */

    by = MAX - 1; /* V pervoy iteratsii stanovitsya 0 */
    fib = 1;

    do
    :: by = (by + 1) % MAX;
prozivka:
        out!fib, by, GOOD_CRC;

        if
        :: timeout -> goto prozivka
        :: in?fib_prim, by_prim, BAD_CRC -> goto prozivka
        :: in?fib_prim, by_prim, GOOD_CRC ->
            if
            :: (fib == fib_prim) -> goto prozivka
            :: (fib != fib_prim) ->
```

```
                assert(by_prim == (by + 1) % MAX);
                goto progress
            fi
        fi;
progress:
        fib = !fib
    od
}

/** Protsess RP na shine. RP otvechayet na opros. Eta model'
    ne sovsem tochna, RP imeyet ochered' soobshcheniy,
    poetomu otvet ne mozhet byt' na tol'ko chto poluchennoye
    soobshcheniye. No zdes' eto ne printsipial'no,
    soobshcheniya zdes' prosto vspomogatel'nyye schetchiki
    dlya proverki validnosti protokola.
    @param in Kanal, po kotoromu RP poluchayet soobshcheniya
    @param out Kanal, po kotoromu RP otpravlyayet soobshcheniya
    */
proctype rp(chan in, out)
{
    bit fib;
    byte by; /* Soobshcheniye dlya polucheniya; v testa chislo
        */
    bit fib_prim; /* FIB poluchennogo soobshcheniya */
    byte by_prim; /* Poluchennoye soobshcheniye */

    fib = 1;
    by = 0;

    do
    :: in?fib_prim,by_prim,BAD_CRC -> out!fib,by,GOOD_CRC
    :: in?fib_prim,by_prim,GOOD_CRC ->
        if
        :: (fib_prim != fib) -> out!fib,by,GOOD_CRC
        :: (fib_prim == fib) ->
            assert(by_prim == by);
            by = (by + 1) % MAX;
progress:   fib = !fib;
            out!fib,by,GOOD_CRC;
        fi
    od
}


/** Kommunikatsionnaya shina. Eto vnosit sluchaynyye
    oshibki, libo poteryu soobshcheniya, libo povrezhdeniye
    soobshcheniya, chto privodit k nepravil'noy proverke CRC.
    Konechno, eto ne imitiruyet oshibki, kotoryye CRC ne
    poymayet, no eta model' ne opredelyayet faktycheskuyu
```

```
    (CRC) proverku, poetomu razrabotchik dolzhen ispol'zovat'
    nailuchshuyu vozmozhnuyu proverku, chtoby poymat'
    naibol'sheye kolichestvo oshibok.

    @param kop_in Kanal, po kotoromu KOP poluchayet
        soobshcheniya
    @param kop_out Kanal, po kotoromu KOP otpravlyayet
        soobshcheniya
    @param rp_in Kanal, po kotoromu RP poluchayet soobshcheniya
    @param rp_out Kanal, po kotoromu RP otpravlyayet
        soobshcheniya
    */
proctype magistrala(chan kop_in, kop_out, rp_in, rp_out)
{
    bit fib;
    byte by;
    bit crc;

    do
    ::kop_out?fib,by,crc ->
        if
        ::rp_in!fib,by,GOOD_CRC
        ::rp_in!fib,by,BAD_CRC; progress_sgreskom1:skip;
        ::skip; progress_gubitak1:skip;
        fi
    ::rp_out?fib,by,crc ->
        if
        ::kop_in!fib,by,GOOD_CRC
        ::kop_in!fib,by,BAD_CRC; progress_sgreskom2:skip;
        ::skip; progress_gubitak2:skip;
        fi
    od
}


/** Inicijalizacija modela */
init {
    chan kop_predaja = [1] of {bit, byte, bit};
    chan kop_prijem = [1] of {bit, byte, bit};
    chan rp_predaja = [1] of {bit, byte, bit};
    chan rp_prijem = [1] of {bit, byte, bit};

    atomic {
        run kop(kop_prijem, kop_predaja);
        run magistrala(kop_prijem, kop_predaja, rp_prijem,
            rp_predaja);
        run rp(rp_prijem, rp_predaja);
    }
}
```