李沛冀，1212300209，Unix系统分析作业4

1. HSI - High-speed Synchronous Serial Interface

2

3 1. Introduction

4 ~~~~~~~~~~~~~~~

5

6 High Speed Syncronous Interface (HSI) is a fullduplex, low latency protocol,

7 that is optimized for die-level interconnect between an Application Processor

8 and a Baseband chipset. It has been specified by the MIPI alliance in 2003 and

9 implemented by multiple vendors since then.

10

11 The HSI interface supports full duplex communication over multiple channels

12 (typically 8) and is capable of reaching speeds up to 200 Mbit/s.

13

14 The serial protocol uses two signals, DATA and FLAG as combined data and clock

15 signals and an additional READY signal for flow control. An additional WAKE

16 signal can be used to wakeup the chips from standby modes. The signals are

17 commonly prefixed by AC for signals going from the application die to the

18 cellular die and CA for signals going the other way around.

19

20 +------------+ +---------------+

21 | Cellular | | Application |

22 | Die | | Die |

23 | | - - - - - - CAWAKE - - - - - - >| |

24 | T|------------ CADATA ------------>|R |

25 | X|------------ CAFLAG ------------>|X |

26 | |<----------- ACREADY ------------| |

27 | | | |

28 | | | |

29 | |< - - - - - ACWAKE - - - - - - -| |

30 | R|<----------- ACDATA -------------|T |

31 | X|<----------- ACFLAG -------------|X |

32 | |------------ CAREADY ----------->| |

33 | | | |

34 | | | |

35 +------------+ +---------------+

36

37 2. HSI Subsystem in Linux

38 ~~~~~~~~~~~~~~~~~~~~~~~~~

39

40 In the Linux kernel the hsi subsystem is supposed to be used for HSI devices.

41 The hsi subsystem contains drivers for hsi controllers including support for

42 multi-port controllers and provides a generic API for using the HSI ports.

43

44 It also contains HSI client drivers, which make use of the generic API to

45 implement a protocol used on the HSI interface. These client drivers can

46 use an arbitrary number of channels.

47

48 3. hsi-char Device

49 ~~~~~~~~~~~~~~~~~~

50

51 Each port automatically registers a generic client driver called hsi\_char,

52 which provides a charecter device for userspace representing the HSI port.

53 It can be used to communicate via HSI from userspace. Userspace may

54 configure the hsi\_char device using the following ioctl commands:

55

56 \* HSC\_RESET:

57 - flush the HSI port

58

59 \* HSC\_SET\_PM

60 - enable or disable the client.

61

62 \* HSC\_SEND\_BREAK

63 - send break

64

65 \* HSC\_SET\_RX

66 - set RX configuration

67

68 \* HSC\_GET\_RX

69 - get RX configuration

70

71 \* HSC\_SET\_TX

72 - set TX configuration

73

74 \* HSC\_GET\_TX

75 - get TX configuration

linux-4.2.2/Documentation/hsi.txt

翻译：

HSI高速同步串行接口

简介

高速同步接口（HSI）是一个全双工，低延迟的协议，应用处理器之间的芯片级互连优化和基带芯片组。这是2003由MIPI联盟指定多个供应商实施以来。HSI接口支持全双工通信的多渠道（通常为8），时速可以达到200 Mbit/s。串行协议采用两路信号，数据和标志作为数据和时钟相结合信号和一个附加的准备信号的流量控制。附加的唤醒信号可以用来唤醒芯片从待机模式。信号是通常以交流信号从模具的应用

20 +------------+ +---------------+

21 | Cellular | | Application |

22 | Die | | Die |

23 | | - - - - - - CAWAKE - - - - - - >| |

24 | T|------------ CADATA ------------>|R |

25 | X|------------ CAFLAG ------------>|X |

26 | |<----------- ACREADY ------------| |

27 | | | |

28 | | | |

29 | |< - - - - - ACWAKE - - - - - - -| |

30 | R|<----------- ACDATA -------------|T |

31 | X|<----------- ACFLAG -------------|X |

32 | |------------ CAREADY ----------->| |

33 | | | |

34 | | | |

35 +------------+ +---------------+

36。在Linux HSI子系统，在Linux内核的HSI系统应该使用HSI设备。恒生指数HSI控制器子系统包含包括驱动程序的支持多端口控制器提供了一个通用的API使用HSI口。它还含有HSI客户端驱动，利用通用的API来实现一个使用HSI接口协议。这些客户端驱动程序可以

使用任意数量的渠道。

HSI字符设备，每个端口自动注册一个通用的客户端驱动程序称为hsi\_char，提供了一个字符的设备用户代表输入端口。可通过HSI从用户空间通信。用户可以配置hsi\_char装置使用下列ioctl命令：

\* HSC\_RESET:

57 - flush the HSI port

58

59 \* HSC\_SET\_PM

60 - enable or disable the client.

61

62 \* HSC\_SEND\_BREAK

63 - send break

64

65 \* HSC\_SET\_RX

66 - set RX configuration

67

68 \* HSC\_GET\_RX

69 - get RX configuration

70

71 \* HSC\_SET\_TX

72 - set TX configuration

73

74 \* HSC\_GET\_TX

75 - get TX configuration

# 2. [linux-4.2.2](http://lxr.oss.org.cn/source/)/[Documentation](http://lxr.oss.org.cn/source/Documentation/)/[bus-devices](http://lxr.oss.org.cn/source/Documentation/bus-devices/)/[ti-gpmc.txt](http://lxr.oss.org.cn/source/Documentation/bus-devices/ti-gpmc.txt)

1 GPMC (General Purpose Memory Controller):

2 =========================================

3

4 GPMC is an unified memory controller dedicated to interfacing external

5 memory devices like

6 \* Asynchronous SRAM like memories and application specific integrated

7 circuit devices.

8 \* Asynchronous, synchronous, and page mode burst NOR flash devices

9 NAND flash

10 \* Pseudo-SRAM devices

11

12 GPMC is found on Texas Instruments SoC's (OMAP based)

13 IP details: http://www.ti.com/lit/pdf/spruh73 section 7.1

14

15

16 GPMC generic timing calculation:

17 ================================

18

19 GPMC has certain timings that has to be programmed for proper

20 functioning of the peripheral, while peripheral has another set of

21 timings. To have peripheral work with gpmc, peripheral timings has to

22 be translated to the form gpmc can understand. The way it has to be

23 translated depends on the connected peripheral. Also there is a

24 dependency for certain gpmc timings on gpmc clock frequency. Hence a

25 generic timing routine was developed to achieve above requirements.

26

27 Generic routine provides a generic method to calculate gpmc timings

28 from gpmc peripheral timings. struct gpmc\_device\_timings fields has to

29 be updated with timings from the datasheet of the peripheral that is

30 connected to gpmc. A few of the peripheral timings can be fed either

31 in time or in cycles, provision to handle this scenario has been

32 provided (refer struct gpmc\_device\_timings definition). It may so

33 happen that timing as specified by peripheral datasheet is not present

34 in timing structure, in this scenario, try to correlate peripheral

35 timing to the one available. If that doesn't work, try to add a new

36 field as required by peripheral, educate generic timing routine to

37 handle it, make sure that it does not break any of the existing.

38 Then there may be cases where peripheral datasheet doesn't mention

39 certain fields of struct gpmc\_device\_timings, zero those entries.

40

41 Generic timing routine has been verified to work properly on

42 multiple onenand's and tusb6010 peripherals.

43

44 A word of caution: generic timing routine has been developed based

45 on understanding of gpmc timings, peripheral timings, available

46 custom timing routines, a kind of reverse engineering without

47 most of the datasheets & hardware (to be exact none of those supported

48 in mainline having custom timing routine) and by simulation.

49

50 gpmc timing dependency on peripheral timings:

51 [<gpmc\_timing>: <peripheral timing1>, <peripheral timing2> ...]

52

53 1. common

54 cs\_on: t\_ceasu

55 adv\_on: t\_avdasu, t\_ceavd

56

57 2. sync common

58 sync\_clk: clk

59 page\_burst\_access: t\_bacc

60 clk\_activation: t\_ces, t\_avds

61

62 3. read async muxed

63 adv\_rd\_off: t\_avdp\_r

64 oe\_on: t\_oeasu, t\_aavdh

65 access: t\_iaa, t\_oe, t\_ce, t\_aa

66 rd\_cycle: t\_rd\_cycle, t\_cez\_r, t\_oez

67

68 4. read async non-muxed

69 adv\_rd\_off: t\_avdp\_r

70 oe\_on: t\_oeasu

71 access: t\_iaa, t\_oe, t\_ce, t\_aa

72 rd\_cycle: t\_rd\_cycle, t\_cez\_r, t\_oez

73

74 5. read sync muxed

75 adv\_rd\_off: t\_avdp\_r, t\_avdh

76 oe\_on: t\_oeasu, t\_ach, cyc\_aavdh\_oe

77 access: t\_iaa, cyc\_iaa, cyc\_oe

78 rd\_cycle: t\_cez\_r, t\_oez, t\_ce\_rdyz

79

80 6. read sync non-muxed

81 adv\_rd\_off: t\_avdp\_r

82 oe\_on: t\_oeasu

83 access: t\_iaa, cyc\_iaa, cyc\_oe

84 rd\_cycle: t\_cez\_r, t\_oez, t\_ce\_rdyz

85

86 7. write async muxed

87 adv\_wr\_off: t\_avdp\_w

88 we\_on, wr\_data\_mux\_bus: t\_weasu, t\_aavdh, cyc\_aavhd\_we

89 we\_off: t\_wpl

90 cs\_wr\_off: t\_wph

91 wr\_cycle: t\_cez\_w, t\_wr\_cycle

92

93 8. write async non-muxed

94 adv\_wr\_off: t\_avdp\_w

95 we\_on, wr\_data\_mux\_bus: t\_weasu

96 we\_off: t\_wpl

97 cs\_wr\_off: t\_wph

98 wr\_cycle: t\_cez\_w, t\_wr\_cycle

99

100 9. write sync muxed

101 adv\_wr\_off: t\_avdp\_w, t\_avdh

102 we\_on, wr\_data\_mux\_bus: t\_weasu, t\_rdyo, t\_aavdh, cyc\_aavhd\_we

103 we\_off: t\_wpl, cyc\_wpl

104 cs\_wr\_off: t\_wph

105 wr\_cycle: t\_cez\_w, t\_ce\_rdyz

106

107 10. write sync non-muxed

108 adv\_wr\_off: t\_avdp\_w

109 we\_on, wr\_data\_mux\_bus: t\_weasu, t\_rdyo

110 we\_off: t\_wpl, cyc\_wpl

111 cs\_wr\_off: t\_wph

112 wr\_cycle: t\_cez\_w, t\_ce\_rdyz

113

114

115 Note: Many of gpmc timings are dependent on other gpmc timings (a few

116 gpmc timings purely dependent on other gpmc timings, a reason that

117 some of the gpmc timings are missing above), and it will result in

118 indirect dependency of peripheral timings to gpmc timings other than

119 mentioned above, refer timing routine for more details. To know what

120 these peripheral timings correspond to, please see explanations in

121 struct gpmc\_device\_timings definition. And for gpmc timings refer

122 IP details (link above).

翻译：

GPMC（通用存储器控制器）： GPMC是一个统一的内存控制器专用接口外

内存设备，由

\*异步SRAM一样的记忆和特定应用集成电路装置。

\*异步，同步，和页面模式突发或闪存设备NAND闪存

\*伪SRAM器件

是德克萨斯仪器GPMC SOC的发现（的）

GPMC通用时间计算：有一定的时序，GPMC可以编程为合适功能的外周，而周边有另一套的时间。在GPMC有外周工作，周时有被转化为形式的GPMC可以理解。它必须是

翻译依赖于连接的外设。还有一个依赖在GPMC时钟频率一定的GPMC计时。因此个通用时序例程开发，以达到上述要求。通用例程提供计时计算GPMC泛型方法周的时间更快。结构域有gpmc\_device\_timings是从外围数据表，定时更新连接到GPMC。几周的时间可以喂食在时间或周期中，提供处理这种情况的规定提供的（指结构gpmc\_device\_timings定义）。它可以这样发生时间为周指定的数据表不存在在时序结构中，在这种情况下，尝试关联外围可供使用的时间。如果不工作，尝试添加一个新的领域所需的外围，教育通用时序例程处理它，确保它不会打破任何现有的。有可能在数据表中没有提到的情况外结构gpmc\_device\_timings 39某些领域，零这些条目。个通用时序程序已被验证工作正常多个OneNAND和tusb6010外设。一个字的谨慎：通用时序例程已开发为基础在GPMC计时，了解周围时，可用自定义定时例程，一种逆向工程无需大部分的数据和硬件（是那些支持不精确在主线具有自定义定时程序）和模拟。周的时间在GPMC时间依赖：

[<gpmc\_timing>: <peripheral timing1>, <peripheral timing2> ...]52

1. 常见的

cs\_on：t\_ceasu adv\_on：t\_avdasu，t\_ceavd

1. 同步共同

sync\_clk：时钟

page\_burst\_access：t\_bacc

clk\_activation：t\_ces，t\_avds

1. 读异步混合

adv\_rd\_off：t\_avdp\_r

oe\_on：t\_oeasu，t\_aavdh

访问：t\_iaa，t\_oe，t\_ce，t\_aa

rd\_cycle：t\_rd\_cycle，t\_cez\_r，t\_oez

1. 读异步非混合

adv\_rd\_off：t\_avdp\_r

oe\_on：t\_oeasu

访问：t\_iaa，t\_oe，t\_ce，t\_aa

rd\_cycle：t\_rd\_cycle，t\_cez\_r，t\_oez

1. 读同步混合

adv\_rd\_off：t\_avdp\_r，t\_avdh

oe\_on：t\_oeasu，t\_ach，cyc\_aavdh\_oe

访问：t\_iaa，cyc\_iaa，cyc\_oe

rd\_cycle：t\_cez\_r，t\_oez，t\_ce\_rdyz

1. 读同步非混合

adv\_rd\_off：t\_avdp\_r

oe\_on：t\_oeasu

访问：t\_iaa，cyc\_iaa，cyc\_oe

rd\_cycle：t\_cez\_r，t\_oez，t\_ce\_rdyz

1. 写异步混合

adv\_wr\_off：t\_avdp\_w

we\_on，wr\_data\_mux\_bus：t\_weasu，t\_aavdh，cyc\_aavhd\_we

we\_off：t\_wpl

cs\_wr\_off：t\_wph

wr\_cycle：t\_cez\_w，t\_wr\_cycle

1. 写异步非混合

adv\_wr\_off：t\_avdp\_w

we\_on，wr\_data\_mux\_bus：t\_weasu

we\_off：t\_wpl

cs\_wr\_off：t\_wph

wr\_cycle：t\_cez\_w，t\_wr\_cycle

1. 写同步混合

adv\_wr\_off：t\_avdp\_w，t\_avdh

we\_on，wr\_data\_mux\_bus：t\_weasu，t\_rdyo，t\_aavdh，cyc\_aavhd\_we

we\_off：t\_wpl，cyc\_wpl

cs\_wr\_off：t\_wph

wr\_cycle：t\_cez\_w，t\_ce\_rdyz

1. 写同步非混合

adv\_wr\_off：t\_avdp\_w

we\_on，wr\_data\_mux\_bus：t\_weasu，t\_rdyo

we\_off：t\_wpl，cyc\_wpl

cs\_wr\_off：t\_wph

wr\_cycle：t\_cez\_w，t\_ce\_rdyz

注意：许多GPMC计时取决于其他GPMC定时（几GPMC定时纯粹依赖其他GPMC定时，一个原因一些GPMC定时失踪以上），这将导致间接依赖周围计时GPMC时间以外上面提到的，指的是更多细节的时间程序。知道什么这周时间对应，请解释结构gpmc\_device\_timings定义。和GPMC定时参考知识产权详情（以上）。