Readme

Click the link below, **choose the readme** in the root of the project, and **click Recompile**:

- `00-readme-for-module-pm.tex`
- `01-readme-for-writer.tex`
- `02-readme-for-reviewer.tex`

`https://www.overleaf.com/read/bwdjvfjksvjy/`

Todo list

Todo resolved

目录

1 比特传输控制器

ESP32-P4 中有大量支持 DMA（直接存储器访问）的外设，它们可以在 CPU 不参与的情况下将数据从存储器传输到外设或从外设传输到存储器，但这需要外设传输的数据格式与软件支持的数据格式相同，如果格式不同，则需要 CPU 重写数据格式，如交换字节、反转字节和左右移位数据。

由于位操作通常相当耗费 CPU 资源，而设计 DMA 的初衷是在传输过程中避免使用 CPU，因此 ESP32-P4 集成了两个比特传输控制器 (BitScrambler)，专门用于修改存储器和外设之间传输数据的格式，一个传输控制器用于存储器到外设（或存储器到存储器）方向的传输，另一个传输控制器用于外设到存储器方向的传输。除此之外，比特传输控制器还是一个灵活的可编程状态机，能够执行更高级的操作。

1.1 概述

在 ESP32-P4 中，支持 DMA 的外设是指能够通过 DMA 通道直接从存储器读取数据和/或直接向存储器写入数据的外设。这些数据的读取/写入格式取决于外设，灵活性较低。对于 ESP32-P4 中大多数支持 DMA 的外设来说，可以在这些路径中挂载比特传输控制器来提高灵活性。

当某个外设挂载比特传输控制器后，比特传输器就可以修改与该外设连接的 DMA 通道上的任何数据。写入 DMA 通道的数据会出现在比特传输控制器的输入端，而比特传输控制器输出端的数据将沿着 DMA 通道继续传输。比特传输控制器内置指令存储器，存储的指令用于控制输入和输出数据格式的映射关系。因此，通过编写合适的比特传输控制器程序，可以让比特传输控制器改写源数据。

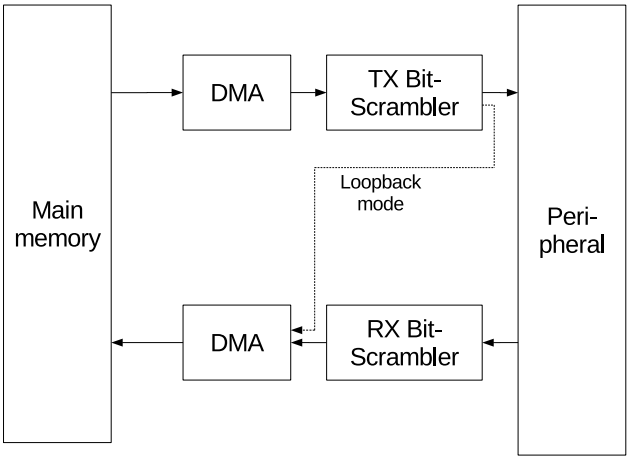


图 1-1. 比特传输控制器系统架构图

1.2 特性

比特传输控制器有以下特性：

- 两个比特传输控制器，一个用于 RX（外设到存储器），一个用于 TX（存储器到外设）
- 支持存储器到存储器的传输
- 每个 DMA 时钟周期最多可处理 32 位数据
- 数据路径由存储在指令存储器中的比特传输控制器程序控制
- 输入寄存器每个时钟周期可读取 0、8、16 或 32 位
- 输出寄存器：

  - 每个时钟周期可写入 0、8、16 或 32 位
  - 输出寄存器位的数据源：64 位输入数据、两个计数器、LUT RAM 数据、上个周期的数据输出、比较器
  - 32 位输出寄存器位中的每一位可以来自数据源的任意位
- 8 x 257 位指令存储器，用于存储八条指令，配置控制流和数据路径
- 2048 字节查找表（LUT）存储器，可配置为不同的字宽

1.3 架构

比特传输控制器可以分为数据路径、控制路径以及寄存器，CPU 可通过这些寄存器配置数据路径和控制路径。数据路径在 DMA 输入流、内部寄存器和 DMA 输出流之间传输比特。控制路径解析指令存储器中的指令，以控制数据路径，并处理程序流程。

1.3.1 数据路径

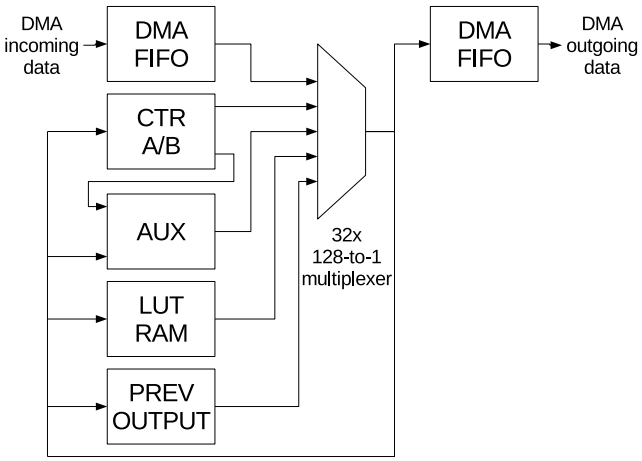


图 1-2. 比特传输控制器数据路径图

比特传输控制器的数据路径从 DMA 输入流中获取数据，根据存储在指令存储器中的程序代码处理数据，然后将其写入 DMA 输出流中。数据从 DMA 输入流保存到一个 64 位寄存器中。在每个指令周期结束时，根据指令的配置从 DMA 输入流中读取 N 位（N 可以是 0、8、16 或 32）。然后，寄存器中的数据向寄存器的最低有效位（LSB）方向移动，最低有效位的 N 位消失。最后，将最新读取的数据保存在最高的 N 位。在比特传输控制器启动时，可以选择预先自动读取前 64 位到寄存器中。

在比特传输控制器的另一端，数据存入一个 32 位输出寄存器。在每个指令周期结束时，根据指令的配置将最低的 0、8、16 或 32 位写入 DMA 输出流。

比特传输控制器的名称来源于其可以将输入比特放置在输出中的任意位置，这是通过 32 个 128 选 1 多路复用器 (128-to-1 multiplexer) 来实现的。对于输出寄存器中的 32 位的每一位，都有一个多路复用器根据比特传输控制器控制路径的输出，从相应的数据源中选择输入信号。这些数据源包括：

- 从输入 FIFO 移入的 64 位数据（即图 1-2 中左侧的 DMA FIFO 模块）。如前所述，来自 DMA 输入流的数据被存储在此处。为了便于循环输入比特，指令可以启用相对寻址。在这种寻址模式下，任何从输入 FIFO 寄存器获取比特的多路复用器实际上会获取偏移计数器寄存器 A 位的比特（即图 1-2 中的 CTR A）。

  yiying: 检查 iterating over input bits in a loop 的翻译是否准确

- 从上一个周期发送到输出的 32 位数据（即图 1-2 中的 PREV OUTPUT 模块）。发送到输出寄存器的数据将在一个时钟周期后出现在这个寄存器中。这使得比特传输控制器可以在多个时钟周期中“记住”数据：即使某个比特没有发送到输出 FIFO，只要是输出到了输出寄存器，下一个时钟周期仍然可以通过选择这个寄存器来访问该比特。通过再次从这个寄存器中选择比特进行输出，可以实现比特的长期记忆，使它们在随后的周期中出现在此寄存器中。
- 由 LUT RAM 输出的 8、16 或 32 位数据，该数据的地址由上一周期输出数据的最高有效位决定。LUT RAM 是比特传输控制器的一部分，它首先接收到某个地址，然后在 LUT RAM 中查找该地址对应的数据，最后输出该数据。比特传输控制器初始化时可填充 LUT RAM，但比特传输控制器不能修改 LUT RAM 。可根据需求将 LUT RAM 配置成以下模式进行初始化：
  - 512 x 32 位字
  - 1024 x 16 位字
  - 2048 x 8 位字

  LUT RAM 可以用于数据的转换，例如用于某传感器的校准曲线，也可用于实现数学函数：将地址输入拆分成两个或两个以上的输入变量，数据输出作为函数的输出；存储器应加载每个输入值对应的函数输出。
- 2 x 16 位计数器提供的 32 位数据（即图 1-2 中的 CTR A/B 模块）。该寄存器包含两个 16 位计数器，分别命名为计数器 A 和 B，可通过控制路径指令对它们进行加载、递增和递减等操作。
- 计数器 B 和输入 FIFO 数据之间的比较器中包含的 30 位数据（属于图 1-2 中的 AUX 模块）。比较结果可用于输出或程序流程，例如在解码游程编码流 (run-length encoded stream) 或生成 PWM 信号时非常有用。
- 2 位数据（属于图 1-2 中的 AUX 模块），一位固定为高电平，一位固定为低电平。使用场景：某协议需要输出一个始终为高电平或低电平的位，不受输入数据影响。

请注意，某些数据源无法同时访问。具体而言，指令无法同时从输入 FIFO 源寄存器的高 32 位以及计数器寄存器中获取数据。当从 LUT 读取数据时，对于宽度为 N 的 LUT，输入 FIFO 寄存器的最高 N 位和计数器寄存器的最高 N 位不可用。

1.3.2 控制路径

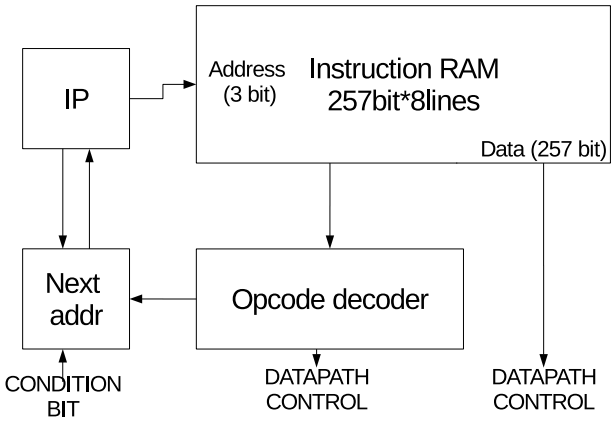


图 1-3. 比特传输控制器控制路径图

比特传输控制器的行为由存储在程序 RAM 中的程序控制。程序 RAM 最多可以存储 8 条指令，每条指令为 257 位。这些指令可配置该指令运行时钟周期内的数据路径，以及控制程序流程的操作码。

比特传输控制器的程序和微处理器的程序类似，每个周期都会执行指令存储器中的下一条指令，除非操作码明确指示需要跳转到另一个位置。除此之外，操作码还可以影响计数器寄存器。

在配置数据路径的指令位中，一些位用于配置 32 个多路复用器，另一些位用于配置计数器位或 LUT 输出是否可用。指令还可以将多路复用器设置为相对模式，在这种模式下，从 DMA FIFO 获取的比特的位置将被偏移计数器 B 位。

1.4 功能描述

比特传输控制器是一种灵活的设备，其行为可通过加载到控制器内部的程序来控制。此外，一些配置寄存器也可以全局配置控制器的一些行为。因此，比特传输控制器程序包括 257 位 x 8 的指令 RAM 以及各种配置寄存器的设置，下文将对二者进行介绍。

1.4.1 指令

表 1-1 为 257 位指令的结构。

表 1-1. 指令结构

位	域	位	域
0 - 6	CTL_MUX_SEL_0	133 - 139	CTL_MUX_SEL_19
7 - 13	CTL_MUX_SEL_1	140 - 146	CTL_MUX_SEL_20
14 - 20	CTL_MUX_SEL_2	147 - 153	CTL_MUX_SEL_21
21 - 27	CTL_MUX_SEL_3	154 - 160	CTL_MUX_SEL_22
28 - 34	CTL_MUX_SEL_4	161 - 167	CTL_MUX_SEL_23
35 - 41	CTL_MUX_SEL_5	168 - 174	CTL_MUX_SEL_24
42 - 48	CTL_MUX_SEL_6	175 - 181	CTL_MUX_SEL_25
49 - 55	CTL_MUX_SEL_7	182 - 188	CTL_MUX_SEL_26
56 - 62	CTL_MUX_SEL_8	189 - 195	CTL_MUX_SEL_27
63 - 69	CTL_MUX_SEL_9	196 - 202	CTL_MUX_SEL_28
70 - 76	CTL_MUX_SEL_10	203 - 209	CTL_MUX_SEL_29
77 - 83	CTL_MUX_SEL_11	210 - 216	CTL_MUX_SEL_30
84 - 90	CTL_MUX_SEL_12	217 - 223	CTL_MUX_SEL_31
91 - 97	CTL_MUX_SEL_13	224 - 249	OPCODE
98 - 104	CTL_MUX_SEL_14	250 - 251	CTL_READ_IN
105 - 111	CTL_MUX_SEL_15	252 - 253	CTL_WR_OUT
112 - 118	CTL_MUX_SEL_16	254	CTL_MUX_REL
119 - 125	CTL_MUX_SEL_17	255	CTL_CTR_SRC_SEL
126 - 132	CTL_MUX_SEL_18	256	CTL_LUT_SRC_SEL

下文介绍指令各域的含义。

- CTL_MUX_SEL_n：包含 7 位，选择输出寄存器中第 n 位的来源。详情参见表 1-3。

  yiying: add reference later

- OPCODE：操作码，该字段的位格式请参见表 1-4。
- CTL_READ_IN：在指令结束时选择从输入 FIFO 读取的位数。
  - 0：不读取任何数据
  - 1：读取 8 位数据

  - 2：读取 16 位数据
  - 3：读取 32 位数据

- CTL_WR_OUT：选择从输出寄存器写入到输出 FIFO 的位数：
  - 0：不写入任何数据
  - 1：写入 8 位数据
  - 2：写入 16 位数据
  - 3：写入 32 位数据

- CTL_MUX_REL、CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL：配置哪些位可以通过 CTL_MUX_SEL 域来选择。
  - CTRL_CTR_SRC_SEL（控制计数器源选择）配置计数器 A 和 B 的位是否可用。
  - CTL_LUT_SRC_SEL（控制 LUT 源选择）配置 LUT RAM 的输出值是否可用。详见表 1-3。
  - 当 CTL_MUX_REL（控制多路复用器相对寻址）为 1 时，比特传输控制器将计数器 A 的值作为值小于或等于 63 的 CTL_MUX_SEL_n 的偏移量。具体行为取决于 CTR_SRC_SEL，详见表 1-2。

    yiying: 确认“比特传输控制器将计数器 A 的值作为值小于或等于 63 的 CTL_MUX_SEL_n 的偏移量。”是否准确

表 1-2. 当 CTL_MUX_REL 为 1 时 CTL_MUX_SEL_n 生效的值

条件	CTL_MUX_SEL 生效的值
CTL_MUX_SEL_n >= 64	CTL_MUX_SEL (unchanged)
CTL_MUX_SEL_n < 64 and CTL_CTR_SRC_SEL = 0	(CTL_MUX_SEL_n + CTRA) & 63
CTL_MUX_SEL_n < 31 and CTL_CTR_SRC_SEL = 1	(CTL_MUX_SEL_n + CTRA) & 31
32 <= CTL_MUX_SEL_n < 64 and CTL_CTR_SRC_SEL = 1	((CTL_MUX_SEL_n + CTRA) & 31) + 32

这些字段中，大部分是数据路径配置项，其中 OPCODE 是影响控制路径的配置项，但 CTL_MUX_REL、CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL 确实可通过调整 IF/IFN 操作码的来源来影响某些操作码的工作方式。

”工作方式”的翻译是否合适

CTL_MUX_SEL_n: 每个 CTL_MUX_SEL_n 域都可用于选择输出寄存器中与之相对应的位的来源。具体选择哪个位在一定程度上取决于 CTL_MUX_REL、CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL 的设置。假设 CTL_MUX_SRC 的值为 N，选择该位的方式如表 1-3 所示。请注意，IF/IFN 操作码的数据源选择也采用相同的方式。

表 1-3. CTL_MUX_SEL 的值

位	描述
0-31	该位来自输入 FIFO 寄存器中的第 N 位。
32-63	该位的来源取决于 CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL 的设置 如果 CTL_CTR_SRC_SEL = 0，该位来源于输入 FIFO 寄存器中的第 N 位。 如果 CTL_CTR_SRC_SEL = 1，该位来源于计数器寄存器中的第 N-32 位。具体来说，如果 N 是 32 到 47，则该位来源于计数器 A 的第 N-32 位；如果 N 是 48 到 63，则该位来源于计数器 B 的第 (N-48) 位。 如果 CTL_LUT_SRC_SEL = 1，根据 LUT 的宽度（8、16 或 32 位），读取最高 8、16 或 32 位将返回来自 LUT 的数据，而不是 CTL_CTR_SRC_SEL 选择的数据。
64	CounterB[7:0] =< last[7:0]
65	CounterB[7:0] > last[7:0]
66	CounterB[7:0] = last[7:0]
67	CounterB[7:0] =< last[15:8]
68	CounterB[7:0] > last[15:8]
69	CounterB[7:0] = last[15:8]
70	CounterB[7:0] =< last[23:16]
71	CounterB[7:0] > last[23:16]
72	CounterB[7:0] = last[23:16]
73	CounterB[7:0] =< last[31:24]
74	CounterB[7:0] > last[31:24]
75	CounterB[7:0] = last[31:24]
76	CounterB[15:8] =< last[7:0]
77	CounterB[15:8] > last[7:0]
78	CounterB[15:8] = last[7:0]
79	CounterB[15:8] =< last[15:8]
80	CounterB[15:8] > last[15:8]
81	CounterB[15:8] = last[15:8]
82	CounterB[15:8] =< last[23:16]
83	CounterB[15:8] > last[23:16]
84	CounterB[15:8] = last[23:16]
85	CounterB[15:8] =< last[31:24]
86	CounterB[15:8] > last[31:24]
87	CounterB[15:8] = last[31:24]
88	CounterB[15:0] =< last[15:0]
89	CounterB[15:0] > last[15:0]
90	CounterB[15:0] = last[15:0]
91	CounterB[15:0] =< last[31:16]
92	CounterB[15:0] > last[31:16]
93	CounterB[15:0] = last[31:16]
94	始终为 0
95	始终为 1
96-127	这些位与上一周期结束时输出寄存器上的位相同

比特传输控制器可识别六种操作码，编码格式如表 1-4 所示。

表 1-4. 指令操作码

Bit	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOOP	1	c	tgt			end_val																ctr_add				
ADD	0	c	h	l	0	0	0	0	0	0	ctr_add															
IF	0	0	tgt			0	0	0	0	1	0	0	0	0	0	0	0	0	0	src						
IFN	0	0	tgt			0	0	0	1	0	0	0	0	0	0	0	0	0	0	src						
LDCTD	0	c	h	l	0	0	0	0	1	1	ctr_set															
LDCTI	0	c	h	l	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

操作码字段的定义如下：

- c：配置使用哪个计数器。
  - 0：计数器 A
  - 1：计数器 B
- end_val, ctr_add, ctr_set：16 位（或 5 位）值。这些字段的含义请参阅下面的指令描述。
- tgt：指令的位置。范围：0 到 7。
- h：如果需要将高 8 位写回计数器，则为 1。
- l：如果需要将低 8 位写回计数器，则为 1。

请注意，如果操作码没有指定 H 或 L，则默认需要将 16 位全部写回，此时 h 和 l 位都会为 1。

下方表格主要介绍这六个操作码如何影响比特传输控制器的状态。

表 1-5. LOOP 操作码

`LOOPc end_val, ctr_add, tgt`	
使用 c 参数设置的计数器进行循环，将 ctr_add 加到计数器中，并循环回到 tgt 位置，直到计数器达到 end_val。当计数器达到 end_val 时，重置计数器并继续执行。	
参数	c：计数器 A 或 B，表示要使用的计数器 tgt：3 位值，表示目标指令地址 end_val：16 位值，用于与计数器进行比较 ctr_add：5 位有符号值，用于加到计数器上

伪代码	`if ((unsigned)ctrC < end_val) begin`
	`  IP <= tgt`
	`  ctr_c <= ctr_c + sign_extend(ctr_add)`
	`end else begin`
	`  IP <= IP + 1`
	`  ctr_c <= 0`
	`end`

表 1-6. ADD 操作码

ADDc[H|L] ctr_add	
将立即数加到计数器上，可以选择只写回高 8 位或低 8 位。	
参数	c：计数器 A 或 B，表示要使用的计数器 ctr_add：16 位值，加到计数器上的值 如果操作码是 ADDcH，则只写回最高 8 位 如果操作码是 ADDcL，则只写回最低 8 位 如果操作码是 ADDc，则 16 位全部都将被写回（h=1，l=1）

伪代码	`tmp <= ctr_c + ctr_add`
	`if (h) begin`
	`  ctr_c[15:8] <= tmp[15:8]`
	`end`
	`if (l) begin`
	`  ctr_c[7:0] <= tmp[7:0]`
	`end`
	`IP <= IP + 1`

说明	NOP（无操作）伪操作码为 “ADDA 0”。

表 1-7. IF 操作码

IF ctl_cond_src, tgt	
如果多路复用器指定的输入源位已设置，则跳转到目标。 yiying: 再确认 source bit 怎么处理	
参数	ctl_cond_src：多路复用器来源，可参考表 CTR_MUX_SRC sources tgt：3 位值，表示目标指令地址

伪代码	`if (ctl_cond_src) begin`
	`  IP <= tgt`
	`end else begin`
	`  IP <= IP + 1`
	`end`

说明	JMP（始终跳转）伪操作码为 “IF 1,tgt”，使用 AUX 位中始终为 1 的位。

表 1-8. IFN 操作码

IFN ctl_cond_src, tgt	
如果指定的多路复用器源位未设置，则跳转到目标（注：与 IF 操作相同，但条件取反）。	
参数	ctl_cond_src：多路复用器来源，可参考表 CTR_MUX_SRC sources tgt：3 位值，表示目标指令地址

伪代码	`if (ctl_cond_src) begin`
	`  IP <= IP + 1`
	`end else begin`
	`  IP <= tgt`
	`end`

表 1-9. LDCTD 操作码

LDCTDc[H|L] ctr_set	
将立即数直接加载到计数器，可以选择只加载计数器的高 8 位或低 8 位。	
参数	c：计数器 A 或 B，表示要使用的计数器 ctr_set：16 位值，要设置的计数器的值 h：如果操作码是 LDCTDcH，则只设置最高 8 位 l：如果操作码是 LDCTDcL，则只设置最低 8 位 如果操作码是 LDCTDc，则 16 位全部都将被设置（h=1，l=1）

伪代码	`if (h) begin`
	`  ctr_c[15:8] <= ctr_set[15:8]`
	`end`
	`if (l) begin`
	`  ctr_c[7:0] <= ctr_set[7:0]`
	`end`
	`IP <= IP + 1`

表 1-10. LDCTI 操作码

LDCTIc[H|L]	
将多路复用器输出的高 16 位间接加载到计数器，可以选择只加载到计数器的高 8 位或低 8 位。	
参数	c：计数器 A 或 B，表示要使用的计数器 h：如果操作码是 LDCTDcH，则只设置最高 8 位 l：如果操作码是 LDCTDcL，则只设置最低 8 位 如果操作码是 LDCTDc，则 16 位全部都将被设置（h=1，l=1）

伪代码	`if (h) begin`
	`  ctr_c[15:8] <= mux_out[31:24]`
	`end`
	`if (l) begin`
	`  ctr_c[7:0] <= mux_out[23:16]`
	`end`
	`IP <= IP + 1`

1.4.2 配置寄存器

比特传输控制器的配置是通过一组寄存器实现的。这些寄存器可向比特传输控制器写入程序、写入 LUT RAM、配置控制器行为、以及启动和停止控制器的操作，还可将控制器挂载到外设的 DMA 路径。

ESP32-P4 有两个比特传输控制器，一个可以放置在 TX 路径中（数据从存储器发送到外设），另一个可以放置在 RX 路径中（数据从外设发送到存储器）。两个比特传输控制器的配置寄存器是相同的，唯一的区别在于 TX 路径的比特传输控制器寄存器前缀为 BITSCRAMBLER_TX_，而 RX 路径的为 BITSCRAMBLER_RX_。

说明：
为方便阐述，本节仅介绍 TX 比特传输控制器路径，若想了解 RX 路径，只需替换描述中的前缀即可。

当比特传输控制器修改某外设的输入或输出时，需要挂载到该外设的 DMA 路径。为此，需将该外设的对应值

写入 HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG 寄存器的 HP_SYSTEM_BITSCRAMBLER_PERI_TX_SEL 域或 HP_SYSTEM_BITSCRAMBLER_PERI_RX_SEL 域。

比特传输控制器也可以在存储器到存储器模式下运行，但需要将 BITSCRAMBLER_SYS_REG 寄存器中的 BITSCRAMBLER_LOOP_MODE 域置为 1。该模式下，TX 比特传输控制器进入环回模式，而 RX 比特传输控制器则未使用且无法使用。并且比特传输控制器仍然需要连接到外设，但该外设无需配置，而且对应的外设 DMA 路径将无法使用。

主处理器无法直接访问比特传输控制器的程序存储器和 LUT RAM，但可以间接访问，即在一个寄存器中设置地址，然后在另一个寄存器中写入数据。

具体而言，对于 LUT RAM，地址写入 BITSCRAMBLER_TX_LUT_CFG0_REG 的 BITSCRAMBLER_TX_LUT_IDX 域，数据写入或从 BITSCRAMBLER_TX_LUT_CFG1_REG 寄存器读取数据。LUT 的位宽对比特传输控制器可见的，可以使用 BITSCRAMBLER_TX_LUT_MODE 域进行配置。需要注意的是，无论 LUT 在比特传输控制器中配置为多少位宽，主处理器始终以 32 位位宽的字写入存储器。

同样，对于指令存储器，地址写入 BITSCRAMBLER_TX_INST_CFG0_REG，数据写入或从 BITSCRAMBLER_TX_INST_CFG1_REG 中读取。由于比特传输控制器指令为 257 位，因此一个指令需要写入 9 个 32 位字，第 257 位位于第 9 个字的最低有效位。为此，BITSCRAMBLER_TX_INST_CFG0_REG 寄存器被分为两个域，指令的位置写入 BITSCRAMBLER_TX_INST_IDX 域，而指令内的字偏移写入 BITSCRAMBLER_TX_INST_POS 域。

由于比特传输控制器生成的数据可以比输入多也可以比输入少，所以它控制的 DMA 流可以以一种或两种方式结束。第一种情况是接收器（在 RX 比特传输控制器的情况下是存储器，而在 TX 比特传输控制器的情况下是外设）停止接收数据，因为它的配置是只接收一定数量的字节。这种情况比较简单，比特传输控制器因为无法再写入数据而停止。另一种方式是发射器（在 RX 比特传输控制器的情况下是外设，在 TX 比特传输控制器的情况下是存储器）停止发送数据。这种情况更加复杂，因为比特传输控制器可能正在处理需要写入输出的一些数据，也可能不在处理。

为了让比特传输控制器发送仍在处理的数据，ESP32-P4 设计了两种方法来处理输入数据流结束后的一定数量的数据。这两种方法依赖于将 BITSCRAMBLER_RX_TAILING_BITS_REG 设置为某个值 N：

yiying: 句子流待优化

- EOF-on-N-reads（读取 N 字节后结束）：在输入数据流结束后，比特传输控制器从输入读取 N 个虚拟字节。这些字节的值为零。在读取完第 N 个字节后，比特传输控制器停止，DMA 流结束。
- EOF-on-N-writes（写入 N 字节后结束）：在输入数据流结束后，比特传输控制器允许向输出写入 N 个字节。在此期间，任何输入流上的读取都会返回值为零的虚拟字节。在写入完第 N 个字节后，比特传输控制器停止，DMA 流结束。

请按照下表内容配置上述两种模式。

表 1-11. EOF 模式设置

寄存器	EOF-on-N-reads	EOF-on-N-writes
BITSCRAMBLER_TX_RD_DUMMY	1	1
BITSCRAMBLER_TX_FETCH_MODE	0	0
BITSCRAMBLER_TX_EOF_MODE	1	0
BITSCRAMBLER_TX_HALT_MODE	1	1

1.5 配置流程

请注意，下文描述的是 TX 比特传输控制器的配置流程，TX 比特传输控制器位于存储器到外设方向的路径中，或者在环回模式下位于存储器到存储器方向的路径。对于外设到存储器方向的路径，则使用 RX 比特传输控制器。除非另作说明，否则将下文描述中的 TX 替换为 RX 即为 RX 比特传输控制器的配置流程。

1. 配置 HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG 寄存器中的 HP_SYSTEM_BITSCRAMBLER_PERI_TX_SEL 域将比特传输控制器挂载到外设。请注意，即使在环回模式下，此步骤也是必须的。
2. 配置 BITSCRAMBLER_TX_CTRL_REG 寄存器中的 BITSCRAMBLER_TX_ENA 域使能比特传输控制器。
3. 根据需要决定是否配置环回模式。如果需要，将 BITSCRAMBLER_SYS_REG 寄存器的 BITSCRAMBLER_LOOP_MODE 域置 1，否则清零。
4. 将程序加载到比特传输控制器中。对于每条指令和指令内的每个 32 位字，设置 BITSCRAMBLER_TX_INST_CFG0_REG 寄存器中的 BITSCRAMBLER_TX_INST_IDX 和 BITSCRAMBLER_TX_INST_POS 域，然后将字写入 BITSCRAMBLER_TX_INST_CFG1_REG 寄存器。注意，第 257 位写入到 BITSCRAMBLER_TX_INST_CFG1_REG 的第 0 位。
5. 如果需要，将 LUT 挂载到比特传输控制器中。与加载程序类似，LUT 以 32 位字加载，首先将字地址写入 BITSCRAMBLER_TX_LUT_CFG0_REG，然后将字写入 BITSCRAMBLER_TX_LUT_CFG1_REG。加载完 LUT 后，通过设置 BITSCRAMBLER_TX_LUT_CFG0_REG 中的 BITSCRAMBLER_TX_LUT_MODE 来配置 LUT 的宽度。
6. 配置比特传输控制器。根据比特传输控制器程序的需求，设置或清除 BITSCRAMBLER_TX_CTRL_REG 寄存器中的相应位，也可将 BITSCRAMBLER_TX_TAILING_BITS_REG 设置为适当的值。
7. 清除 BITSCRAMBLER_TX_CTRL_REG 寄存器的 BITSCRAMBLER_TX_HALT 域，以启动比特传输控制器。
8. 启动 DMA 传输。有关 DMA 子系统的更多信息，请参阅章节 5 *通用 DMA 控制器 [to be added later]*，如果未使用回环模式，请参阅各外设章节。
9. 等待 DMA 传输完成。
10. 设置 BITSCRAMBLER_TX_CTRL_REG 寄存器中的 BITSCRAMBLER_TX_HALT 域停止比特传输控制器。
11. 等待比特传输控制器处于停止状态。当 BITSCRAMBLER_TX_STATE_REG 寄存器中的 BITSCRAMBLER_TX_IN_IDLE 位被置位时，比特传输控制器停止。
12. 在 BITSCRAMBLER_TX_CTRL_REG 寄存器中的 BITSCRAMBLER_TX_FIFO_RST 位先写入 1 然后再写入 0 来复位 FIFO。
13. 比特传输控制器可以使用当前的程序和 LUT 配置开始进行新的传输，从步骤 7 开始即可。

1.6 寄存器列表

本小节的所有地址均为相对于比特传输控制器基地址的地址偏移量（相对地址），具体基地址请见章节 8 *系统和存储器 [to be added later]* 中的表 8。

请查看章节 *寄存器的访问类型*，了解“访问”列缩写的含义。

yiying: I have updated the description of the registers to differentiate the RX core and TX core registers. Could you please review them in case I introduced any mistakes in the technical content while editing. LGTM - J

Name	Description	Address	Access
Control and configuration registers			
BITSCRAMBLER_TX_INST_CFG0_REG	BitScrambler TX core instruction memory address selection register	0x0000	R/W
BITSCRAMBLER_TX_INST_CFG1_REG	BitScrambler TX core instruction memory access register	0x0004	R/W
BITSCRAMBLER_RX_INST_CFG0_REG	BitScrambler RX core instruction memory address selection register	0x0008	R/W
BITSCRAMBLER_RX_INST_CFG1_REG	BitScrambler RX core instruction memory access register	0x000C	R/W
BITSCRAMBLER_TX_LUT_CFG0_REG	BitScrambler TX core LUT memory address selection register	0x0010	R/W
BITSCRAMBLER_TX_LUT_CFG1_REG	BitScrambler TX core LUT memory access register	0x0014	R/W
BITSCRAMBLER_RX_LUT_CFG0_REG	BitScrambler RX core LUT memory address selection register	0x0018	R/W
BITSCRAMBLER_RX_LUT_CFG1_REG	BitScrambler RX core LUT memory access register	0x001C	R/W
Configuration registers			
BITSCRAMBLER_TX_TAILING_BITS_REG	BitScrambler TX core extra data length register	0x0020	R/W
BITSCRAMBLER_RX_TAILING_BITS_REG	BitScrambler RX core extra data length register	0x0024	R/W
BITSCRAMBLER_TX_CTRL_REG	BitScrambler TX core control register	0x0028	varies
BITSCRAMBLER_RX_CTRL_REG	BitScrambler RX core control register	0x002C	varies
BITSCRAMBLER_SYS_REG	Loopback control register	0x00F8	R/W
Status registers			
BITSCRAMBLER_TX_STATE_REG	BitScrambler TX core status register	0x0030	varies
BITSCRAMBLER_RX_STATE_REG	BitScrambler RX core status register	0x0034	varies
Version register			
BITSCRAMBLER_VERSION_REG	Version control register	0x00FC	R/W

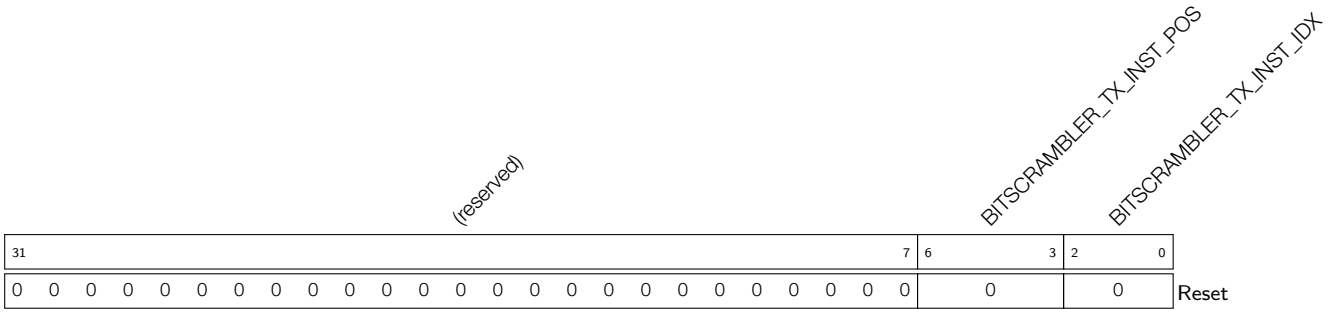
1.7 寄存器

本小节的所有地址均为相对于比特传输控制器基地址的地址偏移量（相对地址），具体基地址请见章节 8 *系统和存储器 [to be added later]* 中的表 8。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

yiying: I have updated the wording of all resgiter descriptions according to the Register Description Conventions. Could you please review them in case I introduced any mistakes in the technical content while editing. LGTM - J

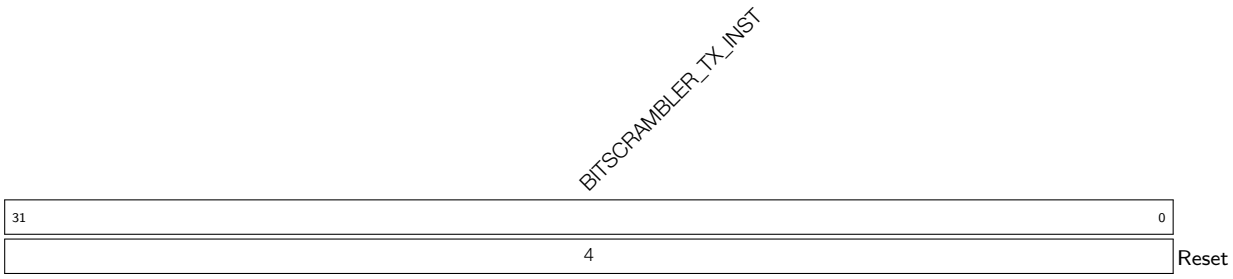
Register 1.1. BITSCRAMBLER_TX_INST_CFG0_REG (0x0000)



BITSCRAMBLER_TX_INST_IDX Configures the index of the BitScrambler instruction to be accessed via BITSCRAMBLER_TX_INST_CFG1_REG. (R/W)

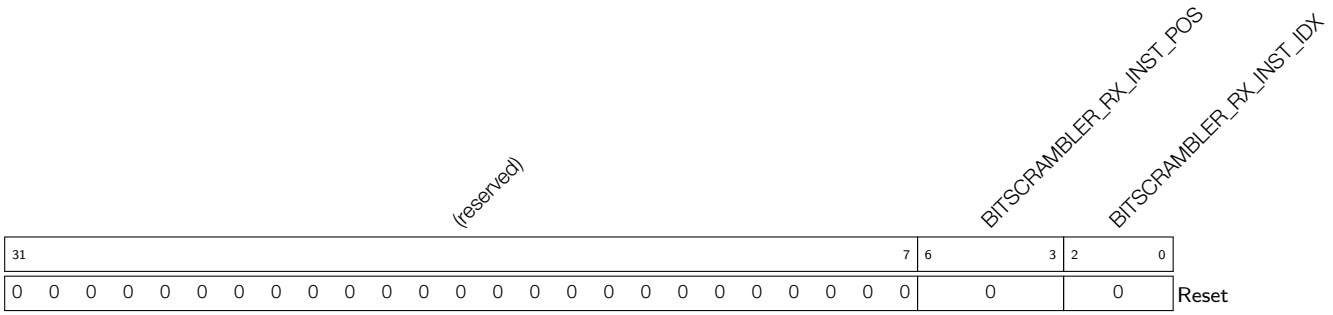
BITSCRAMBLER_TX_INST_POS Configures the offset into the 257-bit BitScrambler instruction (in 32-bit increments) to be accessed via BITSCRAMBLER_TX_INST_CFG1_REG. (R/W)

Register 1.2. BITSCRAMBLER_TX_INST_CFG1_REG (0x0004)



BITSCRAMBLER_TX_INST Contains the instruction to be accessed at the address specified by BITSCRAMBLER_TX_INST_CFG0_REG. (R/W)

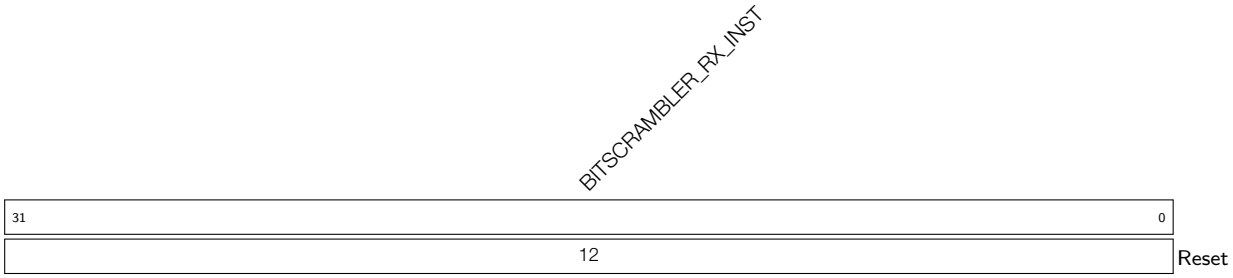
Register 1.3. BITSCRAMBLER_RX_INST_CFG0_REG (0x0008)



BITSCRAMBLER_RX_INST_IDX Configures the index of the BitScrambler instruction to be accessed via BITSCRAMBLER_RX_INST_CFG1_REG. (R/W)

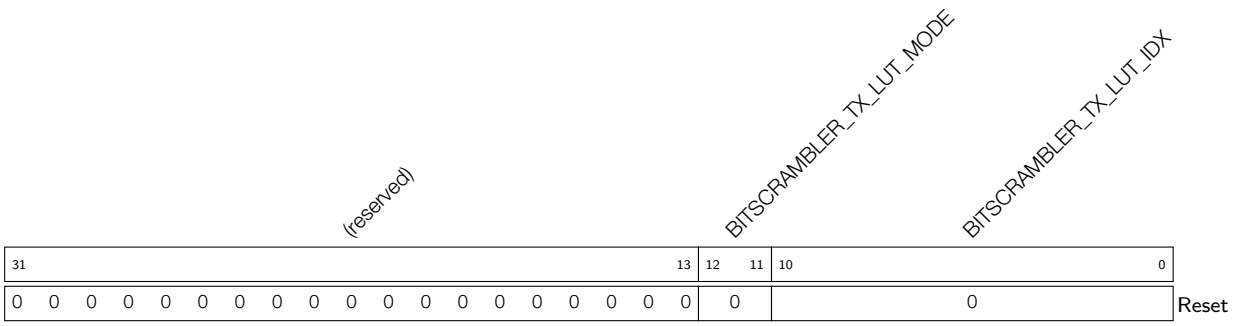
BITSCRAMBLER_RX_INST_POS Configures the offset into the 257-bit BitScrambler instruction (in 32-bit increments) to be accessed via BITSCRAMBLER_RX_INST_CFG1_REG. (R/W)

Register 1.4. BITSCRAMBLER_RX_INST_CFG1_REG (0x000C)



BITSCRAMBLER_RX_INST Contains the instruction to be accessed at the address specified by BITSCRAMBLER_RX_INST_CFG0_REG. (R/W)

Register 1.5. BITSCRAMBLER_TX_LUT_CFG0_REG (0x0010)



BITSCRAMBLER_TX_LUT_IDX Configures the position (in 32-bit increments) into LUT RAM to be accessed via BITSCRAMBLER_TX_LUT_CFG1_REG. (R/W)

BITSCRAMBLER_TX_LUT_MODE Configures the word size of LUT RAM for Bitscrambler programs.

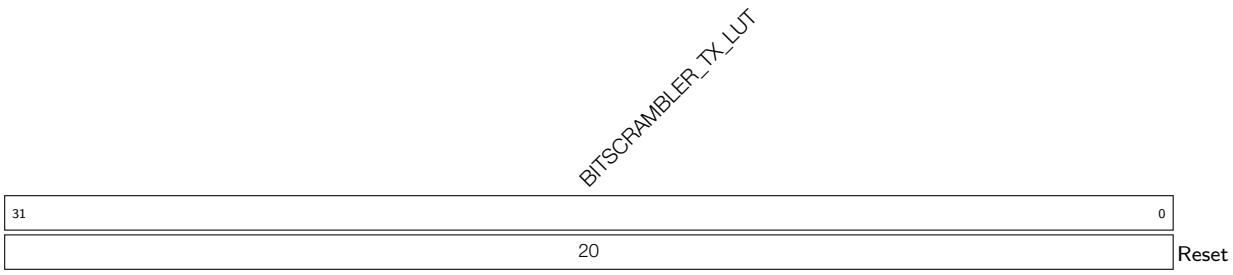
0: 1 byte

1: 2 bytes

2: 4 bytes

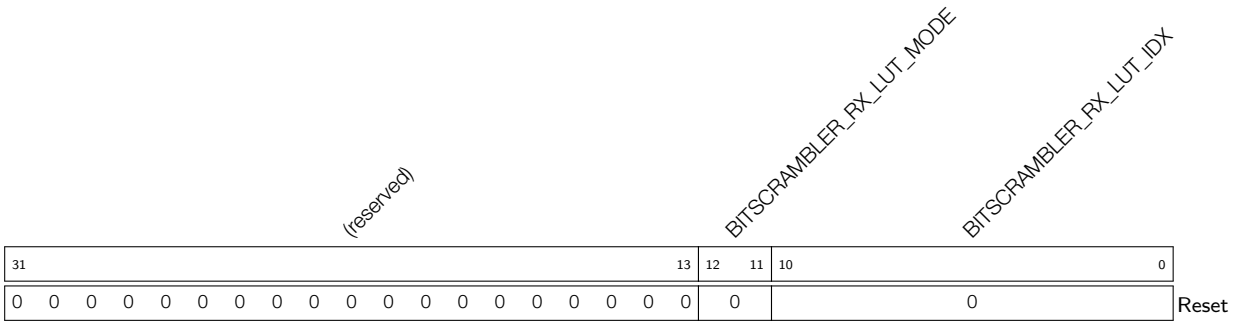
3: Reserved(R/W)

Register 1.6. BITSCRAMBLER_TX_LUT_CFG1_REG (0x0014)



BITSCRAMBLER_TX_LUT Contains the LUT entry to be accessed at the position specified by BITSCRAMBLER_TX_LUT_CFG0_REG. (R/W)

Register 1.7. BITSCRAMBLER_RX_LUT_CFG0_REG (0x0018)



BITSCRAMBLER_RX_LUT_IDX Configures the position (in 32-bit increments) into LUT RAM to be accessed via BITSCRAMBLER_RX_LUT_CFG1_REG. (R/W)

BITSCRAMBLER_RX_LUT_MODE Configures the word size of LUT RAM for BitScrambler programs.

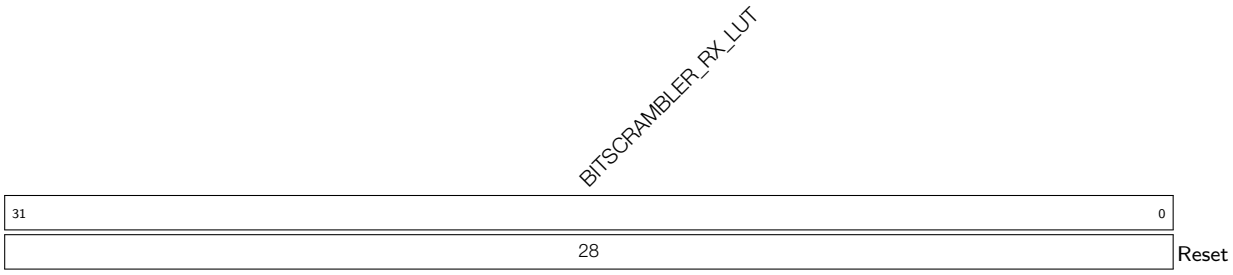
0: 1 byte

1: 2 bytes

2: 4 bytes

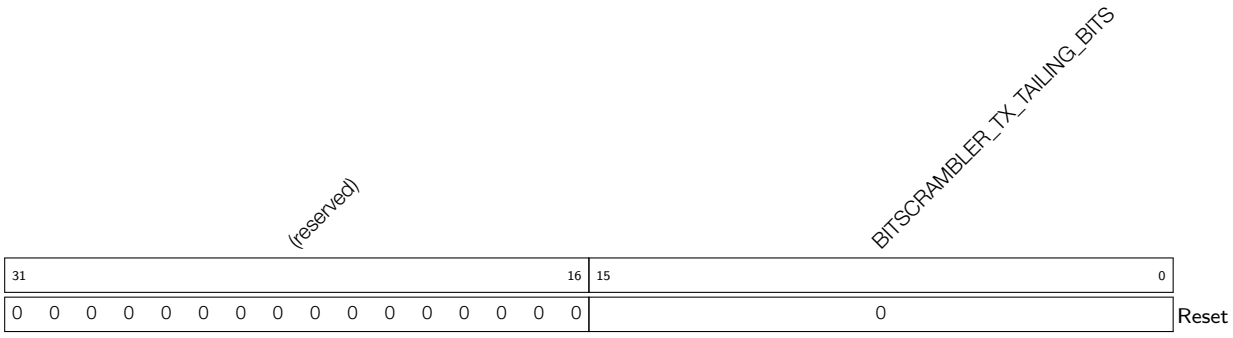
3: Reserved(R/W)

Register 1.8. BITSCRAMBLER_RX_LUT_CFG1_REG (0x001C)



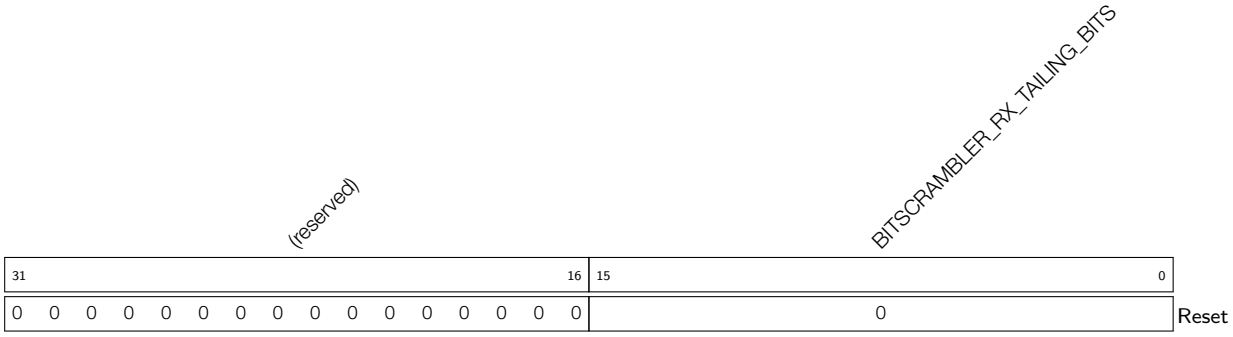
BITSCRAMBLER_RX_LUT Contains the LUT entry to be accessed at the position specified by BITSCRAMBLER_RX_LUT_CFG0_REG. (R/W)

Register 1.9. BITSCRAMBLER_TX_TAILING_BITS_REG (0x0020)



BITSCRAMBLER_TX_TAILING_BITS Configures the length of extra data after getting EOF. Measurement unit: bit. (R/W)

Register 1.10. BITSCRAMBLER_RX_TAILING_BITS_REG (0x0024)

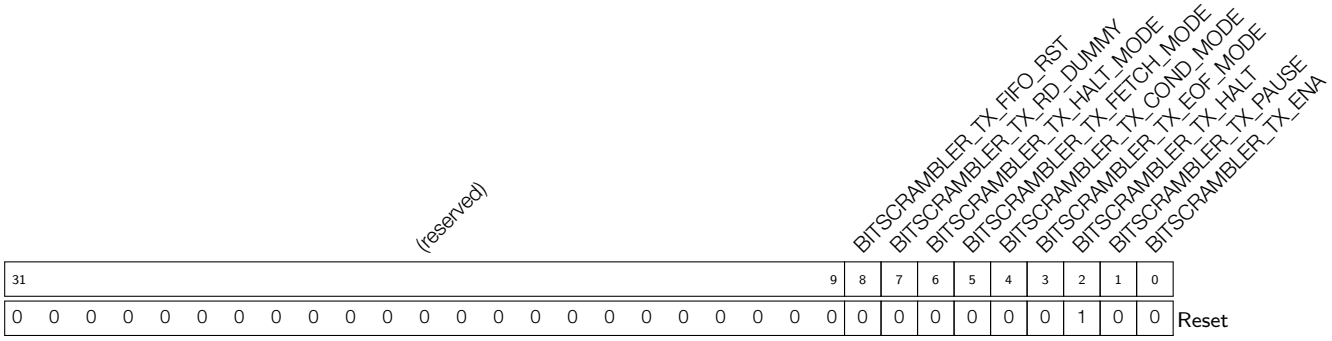


BITSCRAMBLER_RX_TAILING_BITS Configures the length of extra data after getting EOF. Measurement unit: bit.

yiying: the field has exactly the same description as BITSCRAMBLER_TX_TAILING_BITS. Could you please differentiate them?
J: I could, but these registers follow the same implicit convention as the others, i.e. the _RX_ version is for the RX BitScrambler, the _TX_ version is for the TX one. These two registers otherwise have the same functionality, hence the same description.
yiying: By "differentiate" I mean to explicitly describe whether it is for _RX_ version or for the _TX_ version, such as "Configures the length of extra data after getting EOF for the RX BitScrambler".

(R/W)

Register 1.11. BITSCRAMBLER_TX_CTRL_REG (0x0028)



BITSCRAMBLER_TX_ENA Configures whether to enable BitScrambler TX core.

0: Disable

1: Enable

(R/W)

BITSCRAMBLER_TX_PAUSE Configures whether to pause BitScrambler TX core. A paused core can be un-paused to resume execution.

0: Not pause

1: Pause

(R/W)

BITSCRAMBLER_TX_HALT Configures whether to halt BitScrambler TX core. Halting the BitScrambler TX core will flush the FIFOs and cannot be resumed; a FIFO reset and a restart is needed.

0: Not halt

1: Halt

yiying: what's the difference between this field and the pause field BITSCRAMBLER_TX_PAUSE
J: Added.

(R/W)

BITSCRAMBLER_TX_EOF_MODE Configures BitScrambler TX core EOF signal generating mode. This is combined with BITSCRAMBLER_TX_TAILING_BITS for use.

0: Data written to the output FIFO are counted to generate delayed EOF

By Morris: what is "count"? do you mean "continuous"?
J: I mean the verb 'to count'. Lemme rephrase

1: Data read from the input FIFO are counted to generate delayed EOF

(R/W)

BITSCRAMBLER_TX_COND_MODE Configures BitScrambler TX LOOP instruction condition mode.

0: Use the less than operator to get the condition

1: Use not equal operator to get the condition

(R/W)

BITSCRAMBLER_TX_FETCH_MODE Configures BitScrambler TX core fetch instruction mode.

0: Prefetch by reset

1: Fetch by instructions

(R/W)

BITSCRAMBLER_TX_HALT_MODE Configures BitScrambler TX core halt mode when BITSCRAMBLER_TX_HALT is set

{

Register 1.11. BITSCRAMBLER_TX_CTRL_REG (0x0028)

Continued from the previous page...

BITSCRAMBLER_TX_RD_DUMMY Configures BitScrambler TX core read data mode when EOF received.

0: Wait read data

1: Ignore read data

(R/W)

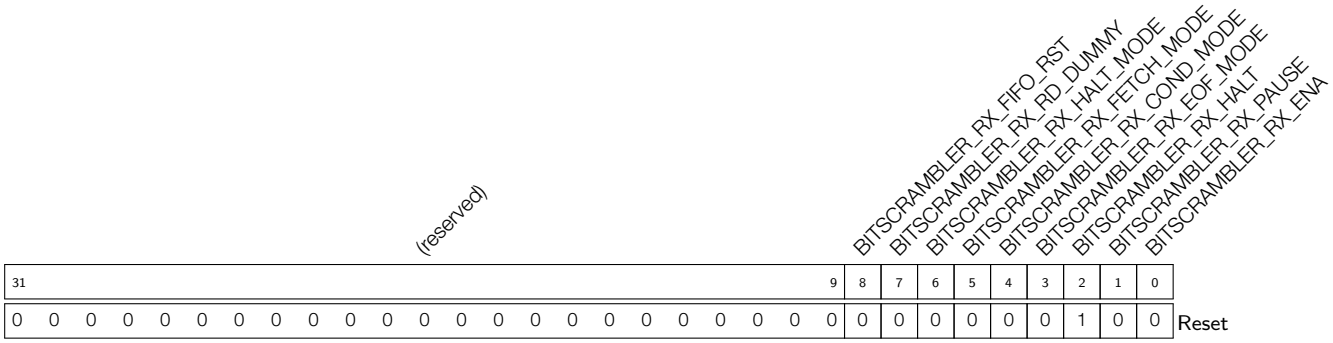
BITSCRAMBLER_TX_FIFO_RST Configures whether to reset BitScrambler TX FIFO.

0: Not reset

1: Reset

(WT)

Register 1.12. BITSCRAMBLER_RX_CTRL_REG (0x002C)



BITSCRAMBLER_RX_ENA Configures whether to enable BitScrambler RX core.

0: Disable

1: Enable

(R/W)

BITSCRAMBLER_RX_PAUSE Configures whether to pause BitScrambler RX core. A paused core can be un-paused to resume execution.

0: Not pause

1: Pause

(R/W)

BITSCRAMBLER_RX_HALT Configures whether to halt BitScrambler RX core. Halting the BitScrambler RX core will flush the FIFOs and cannot be resumed; a FIFO reset and a restart is needed.

0: Not halt

1: Halt

(R/W)

BITSCRAMBLER_RX_EOF_MODE Configures BitScrambler RX core EOF signal generating mode. This is combined with BITSCRAMBLER_RX_TAILING_BITS for use.

0: Data written to the output FIFO are counted to generate delayed EOF

1: Data read from the input FIFO are counted to generate delayed EOF

(R/W)

BITSCRAMBLER_RX_COND_MODE Configures BitScrambler RX LOOP instruction condition mode.

0: Use the less than operator to get the condition

1: Use not equal operator to get the condition

(R/W)

BITSCRAMBLER_RX_FETCH_MODE Configures BitScrambler RX core fetch instruction mode

0: Prefetch by reset

1: Fetch by instructions

(R/W)

BITSCRAMBLER_RX_HALT_MODE Configures BitScrambler RX core halt mode when BITSCRAMBLER_RX_HALT is set

0: Wait for write data back done

1: Ignore write data back

(R/W)

Continued on the next page...

Register 1.12. BITSCRAMBLER_RX_CTRL_REG (0x002C)

Continued from the previous page...

BITSCRAMBLER_RX_RD_DUMMY Configures BitScrambler RX core read data mode when EOF received.

0: Wait read data

1: Ignore read data

(R/W)

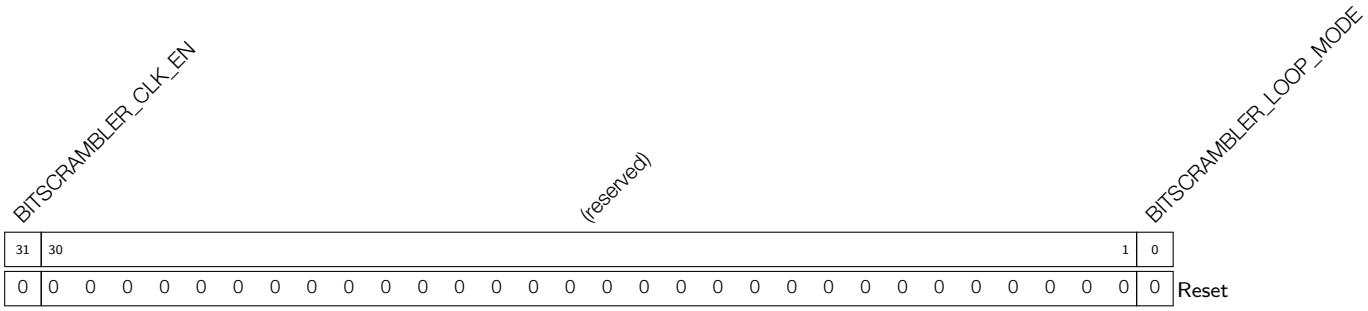
BITSCRAMBLER_RX_FIFO_RST Configures whether to reset BitScrambler RX FIFO.

0: Not reset

1: Reset

(WT)

Register 1.13. BITSCRAMBLER_SYS_REG (0x00F8)



BITSCRAMBLER_LOOP_MODE Configures whether to enable BitScrambler TX to DMA RX loop-back mode

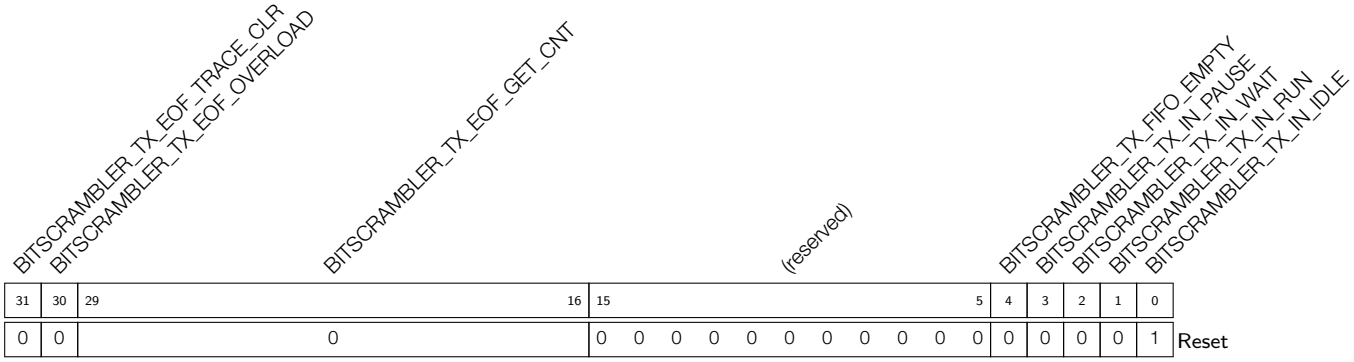
0: Disable

1: Enable

(R/W)

BITSCRAMBLER_CLK_EN Reserved. (R/W)

Register 1.14. BITSCRAMBLER_TX_STATE_REG (0x0030)



BITSCRAMBLER_TX_IN_IDLE Represents whether BitScrambler TX core is halted.

0: Not halted

1: Halted

(RO)

BITSCRAMBLER_TX_IN_RUN Represents whether BitScrambler TX core is running.

0: Not running

1: Running

(RO)

BITSCRAMBLER_TX_IN_WAIT Represents whether BitScrambler TX core is waiting for write back done.

0: Not waiting

1: Waiting

(RO)

BITSCRAMBLER_TX_IN_PAUSE Represents whether BitScrambler TX core is paused.

0: Not paused

1: Paused

(RO)

BITSCRAMBLER_TX_FIFO_EMPTY Represents whether BitScrambler TX FIFO is empty

0: Not empty

1: Empty

(RO)

BITSCRAMBLER_TX_EOF_GET_CNT Represents byte count of BitScrambler TX core after EOF is received. (RO)

BITSCRAMBLER_TX_EOF_OVERLOAD Represents whether BitScrambler TX core tries to process more than one EOF.

0: Not try to process more than one EOF

1: Try to process more than one EOF

(RO)

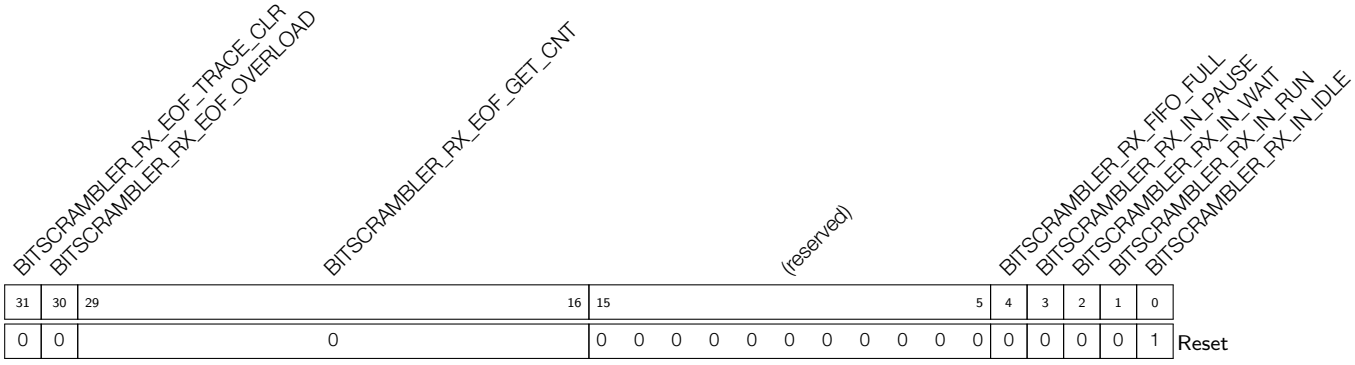
BITSCRAMBLER_TX_EOF_TRACE_CLR Configures whether to clear BITSCRAMBLER_TX_EOF_OVERLOAD and BITSCRAMBLER_TX_EOF_GET_CNT.

0: Not clear

1: Clear

(WT)

Register 1.15. BITSCRAMBLER_RX_STATE_REG (0x0034)



BITSCRAMBLER_RX_IN_IDLE Represents whether BitScrambler TX core is halted.

0: Not halted

1: Halted

(RO)

BITSCRAMBLER_RX_IN_RUN Represents whether BitScrambler TX core is running.

0: Not running

1: Running

(RO)

BITSCRAMBLER_RX_IN_WAIT Represents whether BitScrambler TX core is waiting for write back done.

0: Not waiting

1: Waiting

(RO)

BITSCRAMBLER_RX_IN_PAUSE Represents whether BitScrambler TX core is paused.

0: Not paused

1: Paused

(RO)

BITSCRAMBLER_RX_FIFO_FULL Represents whether BitScrambler RX FIFO is full.

0: Not full

1: Full

(RO)

BITSCRAMBLER_RX_EOF_GET_CNT Represents byte count of BitScrambler TX core after EOF is received. (RO)

BITSCRAMBLER_RX_EOF_OVERLOAD Represents whether BitScrambler TX core tries to process more than one EOF.

0: Not try to process more than one EOF

1: Try to process more than one EOF

(RO)

BITSCRAMBLER_RX_EOF_TRACE_CLR Configures whether to clear BITSCRAMBLER_RX_EOF_OVERLOAD and BITSCRAMBLER_RX_EOF_GET_CNT.

0: Not clear

1: Clear

(WT)

Register 1.16. BITSCRAMBLER_VERSION_REG (0x00FC)

BITSCRAMBLER_BITSCRAMBLER_VER Version contorl register. (R/W)