Application Report
ZHCA537 – July 2013

在 KeyStone 器件上实现高效的 LTE 下行控制信道基带发射

范伟 Multicore DSP / FAE

摘要

LTE 下行控制信道分为 PCFICH、PHICH 和 PDCCH 三类，PDCCH 是其中处理复杂度最高的。和下行数据信道 PDSCH 相比，下行控制信道承载的净荷较少、占用的 OFDM 符号数较少、传输模式也仅限于发射分集，理应占用更少的 DSP 核处理资源。但如下两个因素导致用户的实现可能消耗可观的 DSP 核资源：PHICH/PDCCH 的物理资源映射规则比 PDSCH 更复杂、颗粒度更小，如果在每个下行子帧按照协议描述实时完成该映射所涉及的所有计算，消耗的核资源将非常可观；BCP 用户手册对 PDCCH 的描述较少，用户不易自行补全所有细节并产生高效方案，导致用户可能退而采用全软方案。另外，小基站应用通常对全系统功耗和成本有很高的要求，需要尽可能降低处理负载。本文给出了将实时计算量降至最低的物理资源映射实现方法，以及用 BCP 实现 PDCCH 比特级处理的方案细节，并提供了全面的硬件实测负载。

文档历史

版本	日期	作者	注释
1.0	2013 年 5 月	范伟	初始发布版本。
1.1	2013 年 6 月 27 日	范伟	格式调整，规范化器件型号。

目录

1 引言

LTE（Long Term Evolution）是由 3GPP 组织制定的 3G 演进标准，在物理层采用 OFDM 和 MIMO 技术。LTE 分为 FDD 和 TDD 两种双工模式。目前，LTE-FDD 在 20MHz 频谱带宽下的实际速率大约能达到下行 100Mbps、上行 50Mbps。LTE-TDD（国内通常称为 TD-LTE）的实际速率会随上、下行子帧的配比关系而变化。

[1][2][3][4]是主要的几个 LTE 物理层协议文本。[1]描述了上、下行发射机从星座点调制到基带信号上变频之间的处理步骤，通常称为符号级处理。[2]描述了星座点调制之前的处理步骤，通常称为比特级处理。[3]描述了各种物理层过程。[4]描述了各种物理层测量。

LTE 的上行信道包括用来传输数据和物理层随路控制信令的 PUSCH，专门用来传输物理层控制信令的 PUCCH，以及用于随机接入的 PRACH。下行信道包括用来传输数据的 PDSCH，用来传输各种物理层控制信令的三类控制信道——PCFICH、PHICH 和 PDCCH。本文描述的正是这三类下行控制信道的发射机基带实现。

TI 推出了一系列用于 LTE 基站基带处理的 SoC（System On Chip）。这些 SoC 基于 TI 的 KeyStone 架构，该架构目前已演进了两代——KeyStone I 和 KeyStone II。KeyStone I 家族基于 40nm 工艺，包括如下基带 SoC 器件型号：

- TCI6616，详细资料参见[5]
- TCI6618，详细资料参见[6]
- TCI6614 和 TCI6612，详细资料参见[7]和[8]
- TMS320C6670，详细资料参见[9]

KeyStone II 家族基于 28nm 工艺，包括如下基带 SoC 器件型号：

- TCI6636K2H，详细资料参见[10]
- TCI6634K2K，详细资料参见[11]
- TCI6638K2K，详细资料参见[12]
- TCI6630K2L，详细资料参见[13]

所有这些器件都具有多模能力，支持 GSM/EDGE、WCDMA、TD-SCDMA、WiMAX、LTE 的单模实现或混模实现。所有这些器件使用的 DSP 核都是 c66x，但个数不同。TCI6614 和 TCI6612 带一颗 ARM Cortex A8，TCI6636K2H 和 TCI6638K2K 带 4 颗 ARM Cortex A15，TCI6630K2L 带 2 颗 A15，它们除支持物理层以外，还支持高层（层 2，层 3）和传输处理。这些器件也可用于基于 OFDM 的无线回传（wireless backhaul），如 LTE relay 站。

本文介绍如何在上述 KeyStone 器件上高效地实现 LTE 下行控制信道的基带发射。注意，TCI6616 不带 BCP 加速器，和 BCP 相关的描述不适合 TCI6616。

和下行数据信道 PDSCH 相比，下行控制信道承载的净荷较少、占用的 OFDM 符号数较少、传输模式也仅限于发射分集，理应占用更少的 DSP 核处理资源。但如下两个因素导致用户实际的实现可能消耗可观的核资源：

- 每类下行信道在层映射/预编码之后、IFFT 之前，都要把星座点符号序列映射到各天线端口的物理资源上，而 PHICH/PDCCH 的物理资源映射规则比 PDSCH 更复杂、颗粒度更小，如果在每个下行子帧按照协议描述实时完成该映射所涉及的所有计算，消耗的核资源将非常可观。
- BCP 用户手册对 PDCCH 的描述较少，用户不易自行补全所有细节并产生高效方案，导致用户可能退而采用全软方案。

另外，小基站应用通常对全系统功耗和成本有很高的要求，需要尽可能地降低处理负载，因而对降低下行控制信道的处理负载有更强的需求。

本文给出了将实时计算量降至最低的物理资源映射实现方法，以及用 BCP 实现 PDCCH 比特级处理的方案细节，并提供了全面的硬件实测负载。

本文中的“符号”默认指的是 OFDM 符号，星座点符号将采用全称以示区别。

2 LTE 下行控制信道简介

LTE 下行控制信道有 PCFICH、PHICH、PDCCH 三类。每个 1ms 子帧包含的符号数等于 14（normal CP）或 12（extended CP），下行控制信道独占前 1 到 3 个符号（当小区带宽<=10 个 RB 时，独占前 2 到 4 个符号），三类下行控制信道共享此区域，实际占用符号数可以逐子帧变化，由 PCFICH 指示。子帧中剩下的符号被 PDSCH 独占。

有些符号上有小区参考信号（CRS），有些没有。每个符号内除去 CRS 后，每 4 个连续 RE 构成一个 REG。对存在 CRS 的符号，每个 RB 包含 2 个 REG；对其它符号，每个 RB 包含 3 个 REG。REG 是所有下行控制信道物理资源映射共同的基本单位。

关于帧结构、CP、RE、CRS、REG 等 LTE 基本概念，参见[1]。

本章的剩余部分将对下行控制信道的处理流程和关键特征做简要描述，以方便后面描述解决方案。完整信息参见下文给出的参考文献。

2.1 PCFICH 信道

PCFICH（Physical Control Format Indicator CHannel）总是位于每个下行子帧的第一个符号，占用 4 个 REG，用来指示该子帧中 PDCCH 占用的符号数。

PCFICH 基带发射机的结构如图 1 所示。层 2 下发的 CFI（Control Format Indicator）有 1、2、3 三种取值，用 2 比特表示；经过码率为 1/16 的块编码得到 32 比特；随后的加扰步骤使用的扰码序列的初始状态为$c_{\mathrm{init}} = (\lfloor n_s/2 \rfloor + 1) \cdot (2N_{\mathrm{ID}}^{\mathrm{cell}} + 1) \cdot 2^9 + N_{\mathrm{ID}}^{\mathrm{cell}}$，其中$\lfloor n_s/2 \rfloor$是子帧号，$N_{\mathrm{ID}}^{\mathrm{cell}}$是物理层小区 ID，扰码序列的长度是 32 比特；调制总是采用 QPSK；最终在每个天线端口上产生 16 个星座点符号，占用 4 个 REG。

图 1: PCFICH 基带发射机

PCFICH 的编码步骤参见[2]的 5.3.4 节，其它步骤参见[1]的 6.7 节。

2.2 PHICH 信道

PHICH（Physical Hybrid-ARQ Indicator CHannel）用于反馈上行 PUSCH 传输块的 CRC 校验结果。PHICH 在 PHICH duration 参数配置成 normal 时总是位于第一个符号，配置成 extended 时位于前 2（TDD 特殊子帧或 MBSFN 子帧）或 3 个符号（其它子帧）。TDD 时，子帧 1 和 6 是特殊子帧，而子帧 0 和 5 一定不会被配置成 MBSFN 子帧。MBSFN 子帧是用来支持广播业务的子帧，非 MBSFN 子帧称为单播子帧。

在 REG 的基础上，协议为 PHICH 定义了 PHICH group 和 PHICH mapping unit 这两个概念。3 个 REG 构成一个 PHICH mapping unit，承载最多 8 个 PHICH。Normal CP 时，一个 PHICH group 对应一个 PHICH mapping unit，承载最多 8 个码分复用的 PHICH，扩频因子为 4；extended CP 时，一对 PHICH group 对应一个 PHICH mapping unit，这两个 PHICH group 分别占用每个 REG 中不同的 RE 对（一个 REG 有两个 RE 对），一个 PHICH group 承载最多 4 个码分复用的 PHICH，扩频因子为 2。

对 FDD，每个子帧内的 PHICH group 个数$N_{\mathrm{PHICH}}^{\mathrm{group}}$是一样的，其取值是半静态的，normal CP 时$N_{\mathrm{PHICH}}^{\mathrm{group}} = \lceil N_{\mathrm{g}}(N_{\mathrm{RB}}^{\mathrm{DL}}/8) \rceil$，extended CP 时$N_{\mathrm{PHICH}}^{\mathrm{group}} = 2 \cdot \lceil N_{\mathrm{g}}(N_{\mathrm{RB}}^{\mathrm{DL}}/8) \rceil$，其中$N_{\mathrm{RB}}^{\mathrm{DL}}$表示小区的下行 RB 数，$N_{\mathrm{g}} \in \{1/6, 1/2, 1, 2\}$是一个指示小区 PHICH 资源总数的枚举参数。

对 TDD，子帧内的 PHICH group 个数可能不一样，但一个无线帧内的取值模式是半静态的：子帧 i（i=0 到 9）承载的 PHICH group 个数是$m_{\mathrm{i}} \cdot N_{\mathrm{PHICH}}^{\mathrm{group}}$，其中$m_{\mathrm{i}}$的取值如表 1 所示。

表 1: TDD 的m_{i}取值（[1]的 Table 6.9-1）

Uplink-downlink configuration	Subframe number i									
	0	1	2	3	4	5	6	7	8	9
0	2	1	-	-	-	2	1	-	-	-
1	0	1	-	-	1	0	1	-	-	1
2	0	0	-	1	0	0	0	-	1	0
3	1	0	-	-	-	0	0	0	1	1
4	0	0	-	-	0	0	0	0	1	1
5	0	0	-	0	0	0	0	0	1	0
6	1	1	-	-	-	1	1	-	-	1

每个 PHICH 在一个子帧内的位置由其所在的 PHICH group 编号$n_{\mathrm{PHICH}}^{\mathrm{group}}$和组内编号$n_{\mathrm{PHICH}}^{\mathrm{seq}}$标识，两者由相应 PUSCH 的物理资源分配决定，计算公式是$n_{\mathrm{PHICH}}^{\mathrm{group}} = (I_{\mathrm{PRB_RA}}^{\mathrm{lowest_index}} + n_{\mathrm{DMRS}}) \bmod N_{\mathrm{PHICH}}^{\mathrm{group}} + I_{\mathrm{PHICH}} N_{\mathrm{PHICH}}^{\mathrm{group}}$和$n_{\mathrm{PHICH}}^{\mathrm{seq}} = (\lfloor I_{\mathrm{PRB_RA}}^{\mathrm{lowest_index}} / N_{\mathrm{PHICH}}^{\mathrm{group}} \rfloor + n_{\mathrm{DMRS}}) \bmod 2N_{\mathrm{SF}}^{\mathrm{PHICH}}$，其中$I_{\mathrm{PRB_RA}}^{\mathrm{lowest_index}} \in [0, N_{\mathrm{RB}}^{\mathrm{UL}} - 1]$表示 PUSCH 占用的最小上行物理 RB 索引，$n_{\mathrm{DMRS}} \in [0,7]$表示 PUSCH DMRS 的 cyclic shift 索引，$N_{\mathrm{SF}}^{\mathrm{PHICH}}$表示扩频因子。从表 1 可见，对 TDD UL/DL configuration 0 时的子帧 0 和 5，$m_{\mathrm{i}} = 2$，这是因为这些下行子帧需要反馈两个上行子帧中的 PUSCH 传输块。此时，需要一个额外参数I_{PHICH}来表示一个 PHICH 对应的 PUSCH 位于这两个上行子帧中的哪一个。相关协议参见[3]的 9.1.2 节。

PHICH 基带发射机的结构如图 2 所示。层 2 下发的每个 HI（HARQ Indicator）有 0、1 两种取值，用 1 比特表示；经过码率为 1/3 的重复编码得到 3 比特；调制总是采用 BPSK；把每个调制符号扩频到 SF 个符号，SF 等于 4（normal CP）或 2（extended CP）；随后的加扰步骤使用的扰码序列的初始状态和 PCFICH 完全一样，但长度只有 12（normal CP）或 6（extended CP）；如前所述，extended CP 时，两个相邻 PHICH group 映射到同一个 PHICH mapping unit，在每个

REG 内部，这两个 PHICH group 占用不同的 RE 对，这被称为资源 group 对齐；在每个天线端口上产生 12 个星座点符号，占用 3 个 REG，而且同一个 group 的所有 PHICH 信号需累加起来。

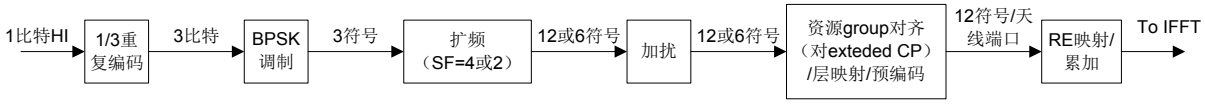

图 2: PHICH 基带发射机

PHICH 的编码步骤参见[2]的 5.3.5 节，其它步骤参见[1]的 6.9 节。注意，[1]的 6.9.3 节在描述 RE 映射时用m'表示 PHICH mapping unit，实际上，$m' = n_{\mathrm{PHICH}}^{\mathrm{group}}$（normal CP）或$m' = n_{\mathrm{PHICH}}^{\mathrm{group}}/2$（extended CP）。

2.3 PDCCH 信道

PDCCH（Physical Downlink Control CHannel）用于承载上/下行调度、上行功控等信令。PDCCH 占用的符号数至少要和 PHICH 一样。表 2 给出了各种情况下一个子帧内 PDCCH 可能占用的符号数。

表 2: 允许 PDCCH 占用的符号数（[1]的 Table 6.7-1，有所简化）

Subframe	Number of OFDM symbols for PDCCH when $N_{\mathrm{RB}}^{\mathrm{DL}} > 10$	Number of OFDM symbols for PDCCH when $N_{\mathrm{RB}}^{\mathrm{DL}} \le 10$
Subframe 1 and 6 for frame structure type 2	1, 2	2
MBSFN subframes on a carrier supporting PDSCH, configured with 1 or 2 cell-specific antenna ports	1, 2	2
MBSFN subframes on a carrier supporting PDSCH, configured with 4 cell-specific antenna ports	2	2
All other cases	1, 2, 3	2, 3, 4

在 REG 的基础上，协议为 PDCCH 定义了 CCE（Control Channel Element）的概念，每个 CCE 占用 9 个 REG。PDCCH 有 4 种格式，每种格式占用不同数量的 CCE，如表 3 所示。

表 3: 支持的 PDCCH 格式（[1]的 Table 6.8.1-1）

PDCCH format	Number of CCEs	Number of resource-element groups	Number of PDCCH bits
0	1	9	72
1	2	18	144
2	4	36	288
3	8	72	576

所有 PDCCH 符号中除去被 PCFICH 和 PHICH 占用的 REG 之外的所有其它 REG 都属于 PDCCH，其数量记为N_{REG}，包含$N_{\mathrm{CCE}} = \lfloor N_{\mathrm{REG}}/9 \rfloor$个 CCE。

PDCCH 基带发射机的结构如图 3 所示。复用之前的步骤为每个 DCI 单独执行：计算 16 位 CRC，并将计算结果和该 DCI 对应的 16 位 RNTI 异或（对用于上行调度的 DCI 格式 0，如果相应 UE 使用发射天线选择，还要异或天线选择掩码来通知 UE 天线选择决策），添加到 DCI 净荷的尾部；信道编码采用尾比特卷积（CC）编码；随后的速率匹配采用和 CC 编码配套的速率匹配过程。

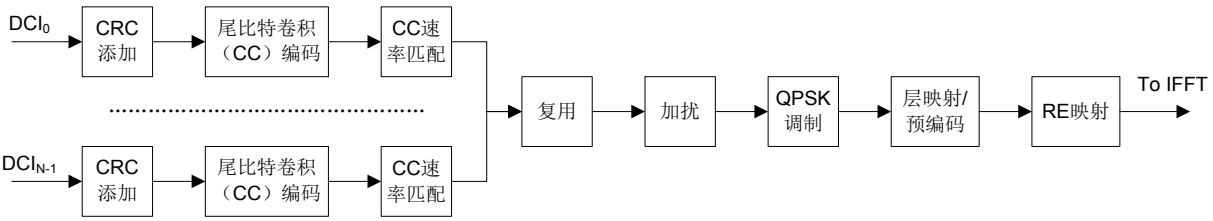

图 3: PDCCH 基带发射机

复用后总的序列长度是$72 \cdot N_{\mathrm{CCE}}$比特，每个 DCI 的速率匹配输出在其中占用连续的一段，长度由该 DCI 的 PDCCH 格式决定，起始位置必须位于由 RNTI 决定的若干搜索空间内，具体位置由层 2 调度器决定并通知层 1。不同 DCI 在复用后序列中的位置不会重叠，但可能有未使用的 CCE 夹杂其间。协议定义未使用位置上的序列元素等于<NIL>，实际系统在这些位置处发射 0 信号。图 4 给出了一个复用的例子。各 DCI 在复用后序列中的位置顺序和层 2 下发的 DCI 顺序可以不一致。

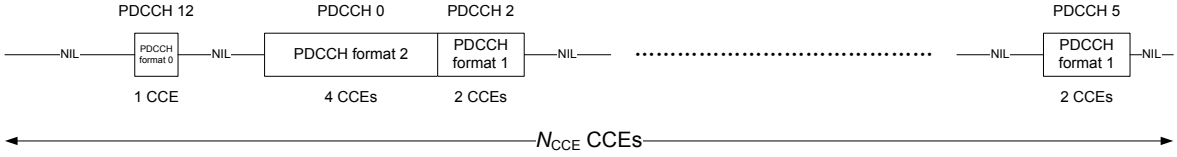

图 4: PDCCH 复用（例子）

随后各步骤对整个复用后序列进行操作。加扰步骤使用的扰码序列的初始状态为$c_{\mathrm{init}} = \lfloor n_{\mathrm{s}}/2 \rfloor \cdot 2^9 + N_{\mathrm{ID}}^{\mathrm{cell}}$。层映射/预编码步骤使用发射分集。RE 映射步骤在每个天线端口上分别执行（对多个天线端口，映射关系总是一样的，但待发射的预编码后的星座点符号序列不同），具体地，把星座点符号序列中每连续 4 个元素组成一个四元组，称为 quad，以 quad 为单元执行子块交织（等同于 CC 速率匹配中的子块交织，只是以 quad，而非比特为操作单位），然后执行先符号、再子载波的二维扫描，把子块交织后的 quad 序列依次映射到扫描到的空闲（没有被 PCFICH 或 PHICH 占用的）REG 中。

PDCCH 的 CRC 添加、信道编码、速率匹配步骤参见[2]的 5.3.3.2 至 5.3.3.4 节，其它步骤参见[1]的 6.8 节。

3 KeyStone 实现方案与负载测量

一种通用的降低实时计算量的策略是：识别对实时操作有影响的静态（系统建立后不再变化）和半静态（可能重配，但很少发生）系统参数，在系统配置或重配时，一旦这些参数中的一部分发生变化，用新参数初始化或更新实时计算所需的中间参数、查找表等数据并保存，这些数据应在内存需求量合理的前提下，尽量提供实时操作所需的直接信息，尽量避免实时操作时对静态、半静态中间数据的重复计算。

具体到 LTE 下行控制信道，可以为每种信道实现一个 Init 函数和一个 Gen 函数，Init 函数在系统配置、重配且相关参数发生变更时调用，Gen 函数为实时操作函数，每子帧或每符号调用一次。实际实现时，可以把三种信道的 Init 函数合并成一个统一的 Init 函数，在其中初始化/更新所有 Gen 函数用到的中间数据。和下行控制信道有关的静态、半静态参数集合如下。当这些参数的一部分发生变化时，Init 函数需要被调用一次。

- $N_{\mathrm{ID}}^{\mathrm{cell}}$，物理层小区 ID
- $N_{\mathrm{RB}}^{\mathrm{DL}}$，小区下行 RB 数
- $N_{\mathrm{RB}}^{\mathrm{UL}}$，小区上行 RB 数
- 帧结构：8 种取值，表示 7 种 TDD UL/DL configuration，加 FDD
- 单播子帧的 CP 类型：normal CP 或 extended CP。单播子帧指的是非 MBSFN 子帧，而 MBSFN 子帧总是使用 extended CP
- P，下行天线端口数
- PHICH duration：normal 或 extended
- N_{g}，PHICH group 数量配置枚举值

本文提供优化代码在 TMS320C6670 EVM 上运行的实测负载。代码和数据都位于 c66x CorePac 的本地 L2 SRAM。所有 KeyStone I/II 器件使用的是相同的 c66x CorePac，有相同的执行效率，所以这些实测负载也适用于其它器件型号。

程序开始执行时数据在 L1D cache 中的就绪状态对 c66x 的执行效率有较为显著的影响。如果程序开始执行时数据已经位于 L1D cache，称为 warm cache 状态；否则，称为 cold cache。C66x 的 L1D 具有 pipeline 多个读 miss 的能力，以隐藏一部分 miss 开销。为了利用此功能提高执行效率，当一个函数需要的一个 buffer 为 cold cache 状态、长度超过若干 cache line（从而能够利用 pipeline 带来的性能好处），所有 buffer 的总长不超过 L1D cache 容量、且不引入 cache line 冲突时，可以在调用该函数前为符合上述要求的 buffer 集合中的每个 buffer 调用如下 touch()函数。该函数用汇编实现，可由 C 代码调用。 Touch()每隔一个 cache line 读一个字节，最大化 L1D 对多个读 miss 的 pipeline。如果使用了 touch()机制，测量负载时，应该把 touch()本身的执行时间也包含在内，而相应的 cache 就绪状态称为 cold cache with touch。Cold cache with touch 并不总是比 cold cache 有更好的执行效率，比如当 buffer 太小（无法发挥 pipeline 的作用）或太大（超过 L1D cache 容量），或者多个 buffer 的内存分布正好会导致 cache line 冲突时。因此，实际测量负载时，应同时测量 cold cache 和 cold cache with touch 两组数据，然后选择较小者。相对数据，代码的 cache 就绪状态对性能的影响通常较小。本文的负载测量对应 warm code cache。关于 c66x cache 的更多信息，请参考[14]和[15]。

```
* ========================================================================== *
*                                                                            *
*   TEXAS INSTRUMENTS, INC.                                                  *
*                                                                            *
*   USAGE                                                                    *
*       This routine is C callable, and has the following C prototype:       *
*                                                                            *
*       void touch                                                           *
*       (                                                                    *
*           void * m,     /* Pointer to vector  */                           *
*           int  num_bytes /* Length of vector in bytes      */             *
*       );                                                                   *
*                                                                            *
* -------------------------------------------------------------------------- *
```

```
*             Copyright (c) 2001 Texas Instruments, Incorporated.           *
*                            All Rights Reserved.                           *
* ========================================================================== *
        .global _touch
        .sect   ".text:_touch"
_touch:
        B       .S2     loop                    ; Pipe up the loop
||      MVK     .S1     128,    A2              ; Step by two cache lines
||      ADDAW   .D2     B4,     31,     B4      ; Round up # of iters

        B       .S2     loop                    ; Pipe up the loop
||      CLR     .S1     A4,     0,  6,  A4      ; Align to cache line
||      MV      .L2X    A4,     B0              ; Twin the pointer

        B       .S1     loop                    ; Pipe up the loop
||      CLR     .S2     B0,     0,  6,  B0      ; Align to cache line
||      MV      .L2X    A2,     B2              ; Twin the stepping constant

        B       .S2     loop                    ; Pipe up the loop
||      SHR     .S1X    B4,     7,      A1      ; Divide by 128 bytes
||      ADDAW   .D2     B0,     17,     B0      ; Offset by one line + one word

   [A1] BDEC    .S1     loop,   A1              ; Step by 128s through array
|| [A1] LDBU    .D1T1   *A4++[A2],      A3      ; Load from [128*i +  0]
|| [A1] LDBU    .D2T2   *B0++[B2],      B4      ; Load from [128*i + 68]
||      SUB     .L1     A1,     7,      A0
loop:
   [A0] BDEC    .S1     loop,   A0              ; Step by 128s through array
|| [A1] LDBU    .D1T1   *A4++[A2],      A3      ; Load from [128*i +  0]
|| [A1] LDBU    .D2T2   *B0++[B2],      B4      ; Load from [128*i + 68]
|| [A1] SUB     .L1     A1,     1,      A1

        BNOP    .S2     B3,     5               ; Get outta here.



* ========================================================================== *
*             Copyright (c) 2000 Texas Instruments, Incorporated.           *
*                            All Rights Reserved.                           *
* ========================================================================== *
```

3.1 PCFICH 信道

PCFICH 要求 Init 函数计算如下中间数据：

- 长度为 32 比特的扰码序列，每子帧一个，共 10 个序列。内存需求为 10*4=40 字节。
- 每个 PCFICH REG 在第一个符号内的 REG 索引，共 4 个索引。内存需求为 4*2=8 字节。
- 第一个符号上一个 REG 的 4 个 RE 相对该 REG 起始子载波的偏移。这是 PHICH/PDCCH 相应需求的子集。

PCFICH 的 Gen 函数完成从信道编码到 RE 映射的所有操作，最终把 RE 映射结果存入 IFFT 输入缓冲。

PCFICH Gen 函数的硬件实测负载如表 4 所示。

表 4: PCFICH Gen 函数的负载（cache 条件：cold cache）

天线端口数	Cycle 数
1	240
2	310
4	373

3.2 PHICH 信道

本文用术语“PHICH 映射图案”表示：给定系统配置，一个子帧中所有的“PHICH mapping unit 索引m'和 RE 位置之间的映射关系”的集合。给定系统配置，不同子帧有可能使用不同的 PHICH 映射图案，而一个特定子帧也有可能根据是否被配置成 MBSFN 子帧而使用不同的 PHICH 映射图案。如表 5 所示，系统配置中的帧结构、PHICH duration 这两个系统参数决定了系统使用的 PHICH 映射图案的总数，以及这些图案在子帧之间的分布情况。灰色格子表示子帧不存在，黄色格子对应$m_{\mathrm{i}} = 0$的子帧，白色格子对应$m_{\mathrm{i}} = 1$的子帧，红色格子对应$m_{\mathrm{i}} = 2$的子帧。白色和红色的格子包含一或两个形式为“n - Xm”的标识，表示该子帧可以使用哪些 PHICH 映射图案，其中：n 表示图案编号（在给定的系统配置下编码，从 1 开始）；X 表征承载 PHICH 的符号数，可以等于 U（PHICH extended duration 下的单播非特殊子帧，对应 3 个符号）、M（PHICH extended duration 下的 MBSFN 或单播特殊子帧，对应 2 个符号）、A（PHICH normal duration，对应 1 个符号）；m 表示m_{i}，$m_{\mathrm{i}} = 1$时省略 m。

记当前系统配置下的 PHICH 映射图案总数为$N_{\mathrm{pattern}}^{\mathrm{PHICH}}$。从表 5 可见，当 PHICH normal duration 且帧结构不是 TDD UL/DL configuration 0 时，$N_{\mathrm{pattern}}^{\mathrm{PHICH}} = 1$，该唯一图案适用于所有承载 PHICH（$m_{\mathrm{i}} > 0$）的子帧；否则，$N_{\mathrm{pattern}}^{\mathrm{PHICH}} = 2$，一个子帧固定使用两种图案中的某一种，或根据自己是否被配置成了 MBSFN 子帧来决定采用哪一种图案。

表 5: PHICH 映射图案的分布

PHICH normal duration										
Sfm#	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
FDD	1 - A	1 - A	1 - A	1 - A	1 - A	1 - A	1 - A	1 - A	1 - A	1 - A
TDD 0	1 - A2	2 - A	-	-	-	1 - A2	2 - A	-	-	-
TDD 1	-	1 - A	-	-	1 - A	-	1 - A	-	-	1 - A
TDD 2	-	-	-	1 - A	-	-	-	-	1 - A	-
TDD 3	1 - A	-	-	-	-	-	-	-	1 - A	1 - A
TDD 4	-	-	-	-	-	-	-	-	1 - A	1 - A
TDD 5	-	-	-	-	-	-	-	-	1 - A	-
TDD 6	1 - A	1 - A	-	-	-	1 - A	1 - A	-	-	1 – A
PHICH extended duration										
Sfm#	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
FDD	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 – M
TDD 0	1 - U2	2 - M	-	-	-	1 - U2	2 - M	-	-	-
TDD 1	-	2 - M	-	-	1 - U 2 - M	-	2 - M	-	-	1 - U 2 - M
TDD 2	-	-	-	1 - U 2 - M	-	-	-	-	1 - U 2 - M	-
TDD 3	1 - U	-	-	-	-	-	-	-	1 - U	1 - U

									2 - M	2 - M
TDD 4	-	-	-	-	-	-	-	-	1 - U 2 - M	1 - U 2 - M
TDD 5	-	-	-	-	-	-	-	-	1 - U 2 - M	-
TDD 6	1 - U	2 - M	-	-	-	1 - U	2 - M	-	-	1 - U

PHICH 要求 Init 函数计算如下中间数据：

- 长度为 12 比特的扰码序列，每子帧一个，共 10 个序列。注意，PHICH 和 PCFICH 用的是相同的扰码序列，只是 PHICH 所需的序列长度只有 12（normal CP）或 6（extended CP）。
- 子帧内的 PHICH mapping unit 个数，每子帧一个值，共 10 个值。注意，在给定的系统配置下，一个子帧内的 PHICH mapping unit 个数是确定的，即使该子帧有 2 个图案可选。内存需求为 10*2=20 字节。
- 从 PHICH mapping unit 到 REG 的映射关系的集合，具体内容如下。
    - 每个 PHICH 映射图案$p \in \left[1, N_{\text{pattern}}^{\text{PHICH}}\right]$：
        - 每个符号$l \in \left[0, L_{\text{p}} - 1\right]$，其中$L_{\text{p}}$表示图案$p$的 PHICH 符号数，可由表 5 确定（U→3，M→2，A→1）：
            - 符号内用于 PHICH 的 REG 个数，记为$N_{\text{REG}}^{(p,l)}$。最大内存需求为 (3+2)*2=10 字节。
            - 符号内每个 PHICH REG 的符号内 REG 索引，共$N_{\text{REG}}^{(p,l)}$个值，按 PHICH mapping unit 编号m'从小到大的顺序存放。如果一个 PHICH mapping unit 的 3 个 REG 中有超过 1 个位于该符号，则这些（2 或 3 个）REG 按其在 PHICH mapping unit 内的顺序连续存放。
    - 最大内存需求发生在$N_{\text{pattern}}^{\text{PHICH}} = 2$、$N_{\text{RB}}^{\text{DL}} = 100$、$N_{\text{g}} = 2$时：此时有 2 个 PHICH 映射图案，总的内存需求为$2 * \lceil 2 * (100/8) \rceil * 3 * 2 = 300$字节。
- 从 PUSCH 资源分配到 PHICH mapping unit 的映射关系的集合，具体内容如下。
    - 每个可能的上行物理 RB 索引$I_{\text{PRB_RA}}^{\text{lowest_index}} \in \left[0, N_{\text{RB}}^{\text{UL}} - 1\right]$和 DMRS cyclic shift 索引$n_{\text{DMRS}} \in [0,7]$的组合：
        - 对应 PHICH 的组索引$n_{\text{PHICH}}^{\text{group}}$。
        - 对应 PHICH 的组内码道索引$n_{\text{PHICH}}^{\text{seq}}$。
    - 最大内存需求发生在$N_{\text{RB}}^{\text{UL}} = 100$时，为 2*100*8*2=3,200 字节。
- 每个符号上一个 REG 的大小，以及其中的 4 个 RE 相对该 REG 起始子载波的偏移。是 PDCCH 相应需求的子集。

PHICH 的 Gen 函数完成从信道编码到 RE 映射的所有操作，最终把 RE 映射结果存入 IFFT 输入缓冲。在 RE 映射阶段，先根据层 2 下发的 PUSCH 资源分配参数查表得到 PHICH group 索引$n_{\mathrm{PHICH}}^{\mathrm{group}}$，由此得到 PHICH mapping unit 索引m'，然后根据m'查表得到该 PHICH mapping unit 的位于当前符号的 1、2 或 3 个 REG 的符号内索引。对后面这一步查表操作，PHICH mapping unit m'的表索引计算方法为：

- 如果 PHICH 符号数为 1，3 个 REG 都位于当前符号，它们的表索引分别是$3m'$、$3m'+1$、$3m'+2$。
- 如果 PHICH 符号数为 3，只有 1 个 REG 位于当前符号，它的表索引是$m'+l$，其中l表示当前符号索引。
- 如果 PHICH 符号数为 2，先确定该 PHICH mapping unit 位于当前符号的 REG 数：如果当前是符号 0，当$m'/2$是奇数、偶数时，REG 数分别是 2、1；如果当前是符号 1，当$m'/2$是奇数、偶数时，REG 数分别是 1、2。然后计算这 1 或 2 个 REG 的表索引：
    - 如果 REG 数是 1，表索引等于$6\cdot[m'/4]+\mathrm{isodd}(m')+4\cdot(l==1)$（唯一表项）；
    - 如果 REG 数是 2，表索引等于$6\cdot[m'/4]+2\cdot\mathrm{isodd}(m' \bmod 4)+2\cdot(l==0)$（连续 2 个表项）。

PHICH Gen 函数的负载依赖于当前子帧需要处理的 HI（Hybrid-ARQ Indicator）个数。我们对不同 HI 个数下的 cycle 数进行测量，然后用最小二乘法拟合出了以 HI 个数为自变量的负载计算公式“cycle 数=a*HI 个数+b”。在 HI 个数相同的情况下，PHICH normal duration 比 PHICH extended duration 在一个符号上处理更多的 REG，因而有更高的单符号负载。为简单起见，这里只给出 PHICH normal duration 情况下的拟合系数，如表 6 所示。拟合时选取了 HI 个数等于 1、10、20、50、100 这 5 个点。FDD 和 TDD 分开测量，TDD 选择了负载最高的$m_{\mathrm{i}}=2$的情况。

表 6: PHICH Gen 函数的负载拟合系数（cache 条件：cold cache）

天线端口数	帧结构	a	b
1	FDD	50.0	557
	TDD 配置 0，子帧 0/5	51.6	639
2	FDD	71.2	763
	TDD 配置 0，子帧 0/5	74.8	883
4	FDD	89.4	791
	TDD 配置 0，子帧 0/5	98.7	822

3.3 PDCCH 信道

PDCCH 信道的处理负载远高于 PCFICH 和 PHICH。下面把 PDCCH 基带发射处理分为比特级和符号级两部分，分别描述。

3.3.1 比特级处理

比特级处理包括从 CRC 添加到 QPSK 调制的步骤。

当系统的每子帧调度用户数较少时，DCI 数目也会较少，此时可用全软方案实现比特级处理。反之，当 DCI 数目较多时，采用基于 BCP 的实现可以节省可观的 c66x 处理资源。特别地，对于一些小基站应用，功耗和成本上的压力可能要求实现较高的处理密度，比如在单个 c66x 上实现一个 2 收 2 发的 20M LTE 载波的物理层。基于 BCP 的方案有助于实现更高的处理密度。

3.3.1.1 基于 c66x 的实现

基于 c66x 的实现要求 Init 函数计算如下中间数据：

- 长度为 31 比特的“扰码序列发生器在产生前面 1600 比特后的中间状态”，每子帧占用一个 32 位字，共 10 个字。根据协议，扰码序列发生器输出的前 1600 个比特总是丢掉不用的。最大内存需求为 10*4=40 字节。

基于 c66x 的实现包括三个实时处理模块，如图 5 所示。每个模块在每个下行子帧中只需调用一次。

- 对复用输出 buffer 的清 0 操作可以由 EDMA 完成。关于 EDMA，请参考[16]。
- 扰码序列生成函数根据当前子帧对应的扰码序列发生器中间状态，计算 PDCCH 扰码序列，长度为$72 \cdot N_{\mathrm{CCE}}$比特。
- PDCCH 比特级主函数对所有层 2 下发的 DCI 逐一处理，完成从 CRC 添加到复用的所有步骤。层 2 下发的信息包括每个 DCI 对应的 CCE 位置，本函数把每个 DCI 的调制后星座点符号序列存放于此，从而完成复用的任务。对一个 DCI 使用的扰码序列片段也由该 CCE 位置确定。

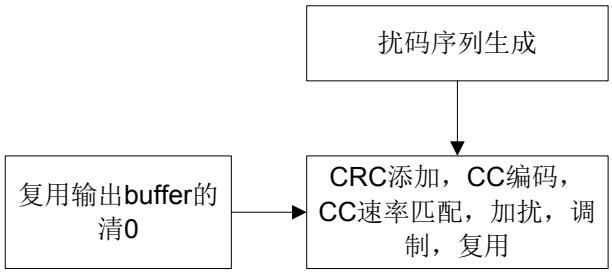

图 5: 全软 PDCCH 比特级实时处理

扰码序列生成函数基于 Init 函数保存的中间状态进行计算，不含对初始 1600 比特的计算。令 cycle 数拟合公式的形式是 a*N+b，N 为扰码序列的比特数（等于$72 \cdot N_{\mathrm{CCE}}$），则拟合系数如表 7 所示。

表 7: PDCCH 扰码序列生成函数的负载拟合系数（cache 条件：cold cache）

a	b
0.25	67

PDCCH 比特级主函数的负载取决于当前子帧内各种格式的 PDCCH 的个数。用 Ni 表示 PDCCH 格式 i（i=0,1,2,3）的个数，cycle 数拟合公式的形式是 a0*N0+a1*N1+a2*N2+a3*N3+b，则拟合系数如表 8 所示。这里采用的测试用例假设每个 DCI 的净荷都是 44 比特，是一个比较适中的取值，且净荷大小对负载的影响不大。

表 8: PDCCH 比特级主函数的负载拟合系数（cache 条件：cold cache）

a0	a1	a2	a3	b
945	1123	1464	2598	2500

3.3.1.2 基于 BCP 的实现

本节描述基于 BCP 的 PDCCH 比特级实现方案。关于 BCP 硬件加速器参考[17]。BCP 对外采用 PktDMA 接口，通过硬件队列获取、交付任务。PktDMA 和硬件队列是 KeyStone 的重要组成部分，相关信息参考[18]。

图 6 描述了基于 BCP 的 PDCCH 比特级处理流程和内存使用。假设系统允许每下行子帧承载的最大 DCI 个数是$N_{\mathrm{DCI}}^{\max}$，则该方案需要为 PDCCH 比特级处理使用$N_{\mathrm{DCI}}^{\max}+2$个 Tx packet 和$N_{\mathrm{DCI}}^{\max}+1$个 Rx packet。Tx/Rx 侧的前$N_{\mathrm{DCI}}^{\max}$个 packet 用于完成图 3 中 DCI 级别的处理，也就是利用 BCP 的 CRC、ENC 和 RM 子模块分别完成 CRC 添加、信道编码和速率匹配，每个 packet 对应一个 DCI。Tx/Rx 侧还各需要一个 packet 用于完成图 3 中的加扰和 QPSK 调制。

考虑到加扰/调制任务和之前的N_{DCI}个 DCI 级任务之间有依赖关系，其中N_{DCI}表示当前子帧实际待发的 DCI 个数，因此需要某种同步机制确保加扰/调制任务执行时，前面所有的 DCI 级任务都已经完成了。本质上，这是要避免加扰/调制任务和前面的 DCI 级任务被 BCP 流水执行，这可以利用 BCP 的 flush/drop 机制实现。该机制是由 BCP 的 TM 子模块，而不是 PktDMA 提供的，需要用到 BCP header。具体方法是：

- 设置最后一个 DCI 级 Tx packet 中的 BCP global header 的 flush 位，使得在所有 DCI 级任务完成且相应的 Rx packet 进入 RxQ 之前，同一个 TxQ 上的下一个任务不会被 BCP 的 TM 子模块（TM 是所有 BCP 任务经过的第一个子模块）接纳。
- Tx 侧使用一个设置了 BCP global header 的 drop 位的 packet，该 packet 位于N_{DCI}个 DCI 级 packet 和 1 个加扰/调制 packet 之间，它的 PktDMA payload（PktDMA payload 由 BCP header 和 BCP payload 组成）长度等于 BCP Tx PktDMA FIFO 的大小，也就是 512 字节，其中 BCP payload 部分可任意取值，因此可以复用其它内存空间来降低内存需求。

Tx 侧的$N_{\mathrm{DCI}}^{\max}+2$个 packet 使用相同的 BCP TxQ。Drop packet 不在 Rx 侧产生输出，所以没有对应的 Rx packet。

加扰操作作用于 PDCCH 复用后的整个序列，所以复用应在 DCI 级操作时完成。所谓复用其实就是把每个 DCI 的速率匹配输出放到整体序列的相应位置，这可以通过配置 Rx 描述符的 buffer pointer 完成。DCI 级操作的 BCP 输出 buffer 中和未使用 CCE 对应的部分（也就是协议定义为<NIL>的部分）不会被 BCP 写，取值可以是任意的。在 RM 子模块的输出中，每比特占用一字节。RM 输出 buffer 在 20M 载波时的最大存储需求是 100*(2+3+3)*4*2*1=6,400 字节。

BCP 的 MOD 子模块用于完成加扰和 QPSK 调制。为了节省内存，这里使用“压缩”输出（图 6 称之为硬调制），也就是每个星座点符号占用一字节，存放该星座点在一个 BCP 预定义常量表中的索引。注意，该常量表不是为每种调制方式单独定义的，而是 BPSK/QPSK/16QAM/64QAM 共用一张大表。QPSK 的 4 个星座点对应的表索引是 2 到 5。MOD 输出 buffer 在 20M 载波时的最大存储需求是 100*(2+3+3)*4*1=3,200 字节。

该方案的一个缺点是：MOD 对整个复用后序列进行加扰/调制，不能自动对<NIL>做清 0 操作。图 6 采用两个分离的步骤来达到对<NIL>清 0 的目的：在启动 BCP 处理之前先启动一个 EDMA 对复

用后星座点符号序列清 0；MOD 完成加扰/调制后，在 c66x 上用一个星座点映射函数对 PDCCH 逐一处理，根据 MOD 输出的常量表索引得到带指定幅度的星座点符号，存入复用序列的相应位置。星座点映射输出 buffer 在 20M 载波时的最大存储需求是 100*(2+3+3)*4*4=12,800 字节。

所有$N_{\mathrm{DCI}}+2$个 BCP Tx packet 的入队操作可以用 EDMA 一次性完成，该 EDMA 可以和对复用 buffer 清 0 的 EDMA 共用一个 EDMA channel，只是配置到不同的 PaRAM set 中，并通过 EDMA 的 link / self-chain 机制自动串行执行。

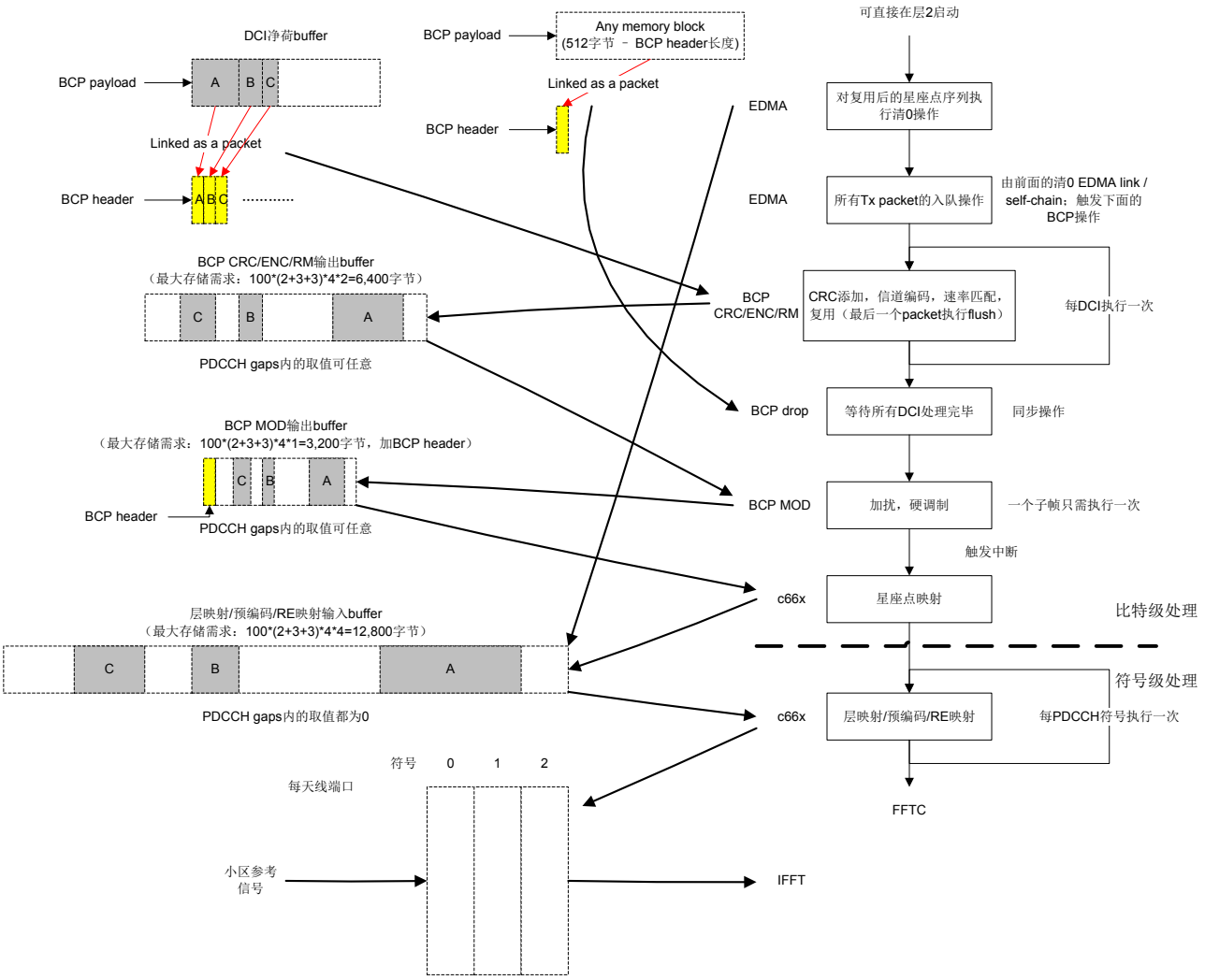

图 6: 基于 BCP 的 PDCCH 比特级处理流程和内存使用

假设层 2 下发的各 DCI 的净荷是连续排列的，它们将成为用于 DCI 级处理的 PktDMA payload 中的 BCP payload 部分。BCP header 部分需要另行配置，这些 BCP header 由 BCP global header 和接下来的针对 TM/CRC/ENC/RM 的子模块配置组成。为了降低实时负载，可以把 BCP header 中的各个配置域分为静态域和动态域两类，静态域只需在系统初始化时配置一次，动态域需要实时修改。为避免对 DCI 净荷或 BCP header 静态域的拷贝，可以为每个 DCI 级 BCP Tx packet 配置 2 个描述符，第一个指向 BCP header，第二个指向 DCI 净荷。

下面给出了 DCI 级处理的 BCP header 配置细节，其中黑体部分表示动态域，其它是静态域。

```
/* BCP 全局配置参数 */

// word 0
PKT_TYPE = 0;
if (last DCI in the TTI) FLUSH = 0;
else FLUSH = 1;
DROP = 0;
HALT = 0;
RADIO_STANDARD = 0;
GHDR_END_PTR = 15;
FLOW_ID = configured flow ID for the per-DCI packets;

// word 1
GLBL_TAG = any;

/* CRC 子模块配置参数 */

// word 0
LOCAL_HDR_LEN = 3;
MOD_ID = 8;
FILLER_BITS = 0;
LTE_ORDER = 1;
DTX_FORMAT = 0;

// word 1
NUM_SCRAMBLE_SYS = 0;

// word 2
if (DCI format 0 with UE Tx ant selection) METHOD2_ID = ((RNTI + UePort) << 16);
else METHOD2_ID = (RNTI << 16);

// word 3
VA_BLK_LEN = DCI payload size;
VA_CRC = 3;    // crc16
VA_BLKS = 1;

/* ENC 子模块配置参数 */

// word 0
LOCAL_HDR_LEN = 1;
MOD_ID = 6;
TURBO_CONV_SEL = 0;
SCR_CRC_ENABLE = 0;
CODE_RATE_FLAG = 0;

// word 1
BLOCK_SIZE_1 = DCI payload size + 16;
NUM_CODE_BLKS_1 = 1;

/* RM 子模块配置参数 */

// word 0
LOCAL_HDR_LEN = 4;
MOD_ID = 5;

// word 1
CHANNEL_TYPE = 1;
```

```
INPUT_BIT_FORMAT = 0;
OUTPUT_BIT_FORMAT = 0;
NUM_FILLER_BITS_F = 0;
RV_START_COLUMN1 = 0;
RV_START_COLUMN2 = 0;

// word 2
PARAM_NCB1_COLUMN = 0;
PARAM_NCB1_ROW = 0;
PARAM_NCB2_COLUMN = 0;
PARAM_NCB2_ROW = 0;

// word 3
NUM_CODE_BLOCKS_C1 = 1;
BLOCK_SIZE_K1 = DCI payload size + 16;

// word 4
NUM_CODE_BLOCKS_CE1 = 1;
BLOCK_SIZE_E1 = 72 * (1 << PDCCH format index);

/* TM 子模块配置参数 */

// word 0
LOCAL_HDR_LEN = 1;
MOD_ID = 0;

// word 1
PS_DATA_SIZE = 0;
INFO_DATA_SIZE = 0;

/* One dummy word to make the whole BCP header come out to a multiple of 128 bits */

Dummy word = 0;
```

用于 MOD 的 Tx packet 只需一个描述符即可，BCP header 和 BCP payload 级联在一起构成 PktDMA payload。该 BCP header 的配置细节如下。

```
/* BCP 全局配置参数 */

// word 0
PKT_TYPE = 0;
FLUSH = 0;
DROP = 0;
HALT = 0;
RADIO_STANDARD = 0;
GHDR_END_PTR = 7;
FLOW_ID = configured flow ID for the MOD packet;

// word 1
GLBL_TAG = any;

/* MOD 子模块配置参数 */

// word 0
LOCAL_HDR_LEN = 2;
MOD_ID = 7;
UVA_VAL = 0;
```

```
// word 1
SH_MOD_SEL = 1;     // Hard modulation (compressed)
SPLIT_MODE_EN = 0;
SCR_EN = 1;
MOD_TYPE_SEL = 1;
CMUX_LN = 0;
Q_FORMAT = 0;
B_TABLE_INDEX = 0;
JACK_BIT = 0;

// word 2
CINIT_P2 = Current c_init;

/* TM 子模块配置参数 */

// word 0
LOCAL_HDR_LEN = 1;
MOD_ID = 0;

// word 1
PS_DATA_SIZE = 0;
INFO_DATA_SIZE = 0;

/* One dummy word to make the whole BCP header come out to a multiple of 128 bits */

Dummy word = 0;
```

从整个系统角度看，在下行方向，为简化架构实现，可以将 BCP 作为层 2 核和层 1 核之间的数据（PDSCH）和 DCI（PDCCH）传递接口。此时，可以在层 2 核上跑一个层 1 代理模块，由该代理模块在层 2 核上调用 DCI 驱动 API，完成对 EDMA 和 BCP 的动态配置和触发。对驱动 API 的调用触发 EDMA/BCP 的执行，待全部操作完成后产生一个中断，c66x 收到此中断后调用星座点映射函数完成整个比特级处理。

星座点映射函数的负载取决于当前子帧内各种格式的 PDCCH 个数。用 Ni 表示格式为 i（i=0,1,2,3）的 PDCCH 个数，cycle 数拟合公式的形式是 a0*N0+a1*N1+a2*N2+a3*N3+b，则拟合系数如表 9 所示。

表 9: 星座点映射函数的负载拟合系数（cache 条件：cold cache）

a0	a1	a2	a3	b
67	109	209	411	83

3.3.2 符号级处理

符号级处理包括层映射/预编码和 RE 映射。层映射/预编码比较简单，不会消耗可观的处理资源。但是，PDCCH 的资源映射规则比较复杂，并且占用的是 PHICH/PCFICH 剩下的 REG 资源，和 PHICH/PCFICH 资源映射有直接的耦合，一旦处理不当，可能导致过高的处理资源消耗。

本文用术语“PDCCH 映射图案”表示：给定系统配置和 PDCCH 符号数（也就是 CFI），一个子帧中所有的“PDCCH REG 和块交织之前 quad 之间的一一映射关系”的集合。在特定的系统配置下，对一个特定子帧，PDCCH 映射图案不仅依赖于该子帧是否被配置成了 MBSFN 子帧（通过影响 PHICH 映射图案影响 PDCCH 映射图案），还依赖于这个子帧的当前 CFI。文本把仅由 CFI 不同而导致的多个 PDCCH 映射图案归类成一个“PDCCH 映射类型”。一个特定的 PDCCH 映

射类型对应一组允许的 PDCCH 符号数，其中每个允许的符号数对应该映射类型下的一个映射图案。

如表 10 所示，系统配置中的帧结构、PHICH duration 这两个系统参数决定了系统使用的 PDCCH 映射类型的总数，以及这些类型在子帧之间的分布情况。表中颜色、n - Xm 标识的含义同表 5，但请注意，对黄色的格子，PHICH 不存在，但 PDCCH 存在，因为 PHICH 仅负责反馈 PUSCH，而 PDCCH 还要调度 PDSCH。记当前系统配置下的 PDCCH 映射类型总数为$N_{\text{category}}^{\text{PDCCH}}$，其取值可从表 10 获得，等于对应行中最大的 n 值。

表 10: PDCCH 映射类型的分布

PHICH normal duration										
Sfm#	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
FDD	1 - A	1 - A	1 - A	1 - A	1 - A	1 - A	1 - A	1 - A	1 - A	1 - A
TDD 0	1 - A2	2 - A	-	-	-	1 - A2	2 - A	-	-	-
TDD 1	2 - A0	1 - A	-	-	1 - A	2 - A0	1 - A	-	-	1 - A
TDD 2	2 - A0	2 - A0	-	1 – A	2 - A0	2 - A0	2 - A0	-	1 - A	2 - A0
TDD 3	1 - A	2 - A0	-	-	-	2 - A0	2 - A0	2 - A0	1 - A	1 - A
TDD 4	2 - A0	2 - A0	-	-	2 - A0	2 - A0	2 - A0	2 - A0	1 - A	1 - A
TDD 5	2 - A0	2 - A0	-	2 - A0	2 - A0	2 - A0	2 - A0	2 - A0	1 - A	2 - A0
TDD 6	1 - A	1 - A	-	-	-	1 - A	1 - A	-	-	1 – A
PHICH extended duration										
Sfm#	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
FDD	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 - M	1 - U 2 – M
TDD 0	1 - U2	2 - M	-	-	-	1 - U2	2 - M	-	-	-
TDD 1	2 - U0	3 - M	-	-	1 - U 3 - M	2 - U0	3 - M	-	-	1 - U 3 - M
TDD 2	2 - U0	3 - M0	-	1 - U 4 - M	2 - U0 3 - M0	2 - U0	3 - M0	-	1 - U 4 - M	2 - U0 3 - M0
TDD 3	1 - U	3 - M0	-	-	-	2 - U0	3 - M0	2 - U0 3 - M0	1 - U 4 - M	1 - U 4 - M
TDD 4	2 - U0	3 - M0	-	-	2 - U0 3 - M0	2 - U0	3 - M0	2 - U0 3 - M0	1 - U 4 - M	1 - U 4 - M
TDD 5	2 - U0	3 - M0	-	2 - U0 3 - M0	2 - U0 3 - M0	2 - U0	3 - M0	2 - U0 3 - M0	1 - U 4 - M	2 - U0 3 - M0
TDD 6	1 - U	2 - M	-	-	-	1 - U	2 - M	-	-	1 - U

表 2 给出了不同$N_{\text{RB}}^{\text{DL}}$、子帧类型（TDD 特殊、MBSFN、其它子帧）等配置下 PDCCH 符号数的取值范围，而 PHICH duration 配置又额外地限制了最小 PDCCH 符号数。综合这两个方面，可得各种系统配置、子帧类型情况下 PDCCH 符号数的取值范围，如表 11 所示。

表 11: PDCCH 符号数的取值范围

PHICH duration	**MBSFN 或 TDD 特殊**	**$N_{\text{RB}}^{\text{DL}} > 10$时的取值范围**	**$N_{\text{RB}}^{\text{DL}} \leq 10$时的取值范围**	**表 10 中的类型标识**
Normal	Yes/No	1,2,3	2,3,4	A
Extended	No	3	3,4	U
Extended	Yes	2	2	M

PDCCH 符号级处理要求 Init 函数计算如下中间数据：

- 每个符号上的 PDCCH REG 个数。每种 PDCCH 映射类型一组，每组对应最多 4 个符号，共 16 个值。内存需求为字节 16*2=32 字节。
- 对“块交织前 quad 序列”的 quad 读位置索引数组（子帧级索引），每个 PDCCH 映射图案有专用的 quad 读位置索引数组，具体内容如下。
    - 每个 PDCCH 映射类型$c \in \left[1, N_{\text{category}}^{\text{PDCCH}}\right]$：
        - 每个 PDCCH 映射图案$p \in [1, P_{\text{c}}]$，其中P_{c}表示类型c下的图案数，由表 11 确定：
            - 每个符号$l \in \left[0, L_{\text{c,p}} - 1\right]$，其中$L_{\text{c,p}}$表示类型$c$、图案$p$下的 PDCCH 符号数，由表 11 确定：
                - 映射到该符号内每个 PDCCH REG 的 quad 在块交织前的 quad 序列中的索引，按照 PDCCH REG 的符号内顺序排列。该索引将在实时操作时被用于读索引，读的是调制后星座点符号序列。
    - 最大内存需求发生在$N_{\text{RB}}^{\text{DL}} = 100$、$P$等于 1 或 2、PHICH normal duration、TDD UL/DL configuration 0~5 时：此时有 2 个 PDCCH 映射类型，每个类型包含 3 个图案，总的内存需求为 2*(200+(200+300)+(200+300+300))*2=6,000 字节。
- 对 PDCCH REG 的写位置索引数组（符号级索引），每个 PDCCH 映射类型有专用的 REG 写位置索引数组，该类型下的图案共用此数组（只是 PDCCH 符号数小的图案只会用到此数组的前面一部分），具体内容如下。
    - 每个 PDCCH 映射类型$c \in \left[1, N_{\text{category}}^{\text{PDCCH}}\right]$：
        - 每个符号$l \in [0, L_{\text{c}}^{\max} - 1]$，其中$L_{\text{c}}^{\max}$表示类型$c$下的最大 PDCCH 符号数，由表 11 确定：
            - PDCCH REG 的符号内 REG 索引，从 0 开始取值，从小到大排序，跳过分配给 PCFICH 和 PHICH 的 REG。该索引将在实时操作时被用于写索引。
    - 最大内存需求发生在$N_{\text{RB}}^{\text{DL}} = 100$、$P$等于 1 或 2、PHICH extended duration、TDD UL/DL configuration 2~5 时：此时有 4 个 PDCCH 映射类型，其中 2 个标识为 M，2 个标识为 U；每个类型仅包含 1 个图案；M 类型的图案的 PDCCH 符号数为 2，U 类型为 3；总的内存需求为 2*((200+300)+(200+300+300))*2=5,200 字节。
- 每个符号上一个 REG 的大小，以及其中的 4 个 RE 相对该 REG 起始子载波的偏移。内存需求为 4*(1+4)*1=20 字节。

PDCCH 符号级 Gen 函数为每个 PDCCH 符号调用一次。该函数首先定位当前 PDCCH 映射类型/图案和当前符号对应的“quad 读位置索引序列”、“REG 写位置索引序列”在各自的整体数组中的位置，然后为该符号上的每个 quad/REG 执行：获取 quad 读位置索引，读取 quad 中的每个星座点符号；完成层映射/预编码；获取 REG 写位置索引，进而获得其中每个 RE 的位置，为每个天线端口写入预编码结果。

PDCCH 符号级 Gen 函数的一组实测负载如表 12 所示。这里以 UL/DL configuration 0 作为 TDD 的代表。当 CFI 大于 1 时，表中负载为多个 PDCCH 符号上的总负载。注意，本函数总是对所有的子帧级 PDCCH quad/REG 进行操作，而不区分是否真的有 PDCCH 承载，因此，负载和 PDCCH 实际承载情况无关。这样做虽然在 PDCCH 轻载时有一些多余的操作，但有利于降低满载时的负载，降低了最差情况下的负载。因为本函数用到的缓冲区（包括输入、输出、Init 计算的中间数据）大小适中，cold catch with touch 相比 cold cache 有明显的性能优势。

表 12: PDCCH 符号级 Gen 函数的负载（cache 条件：cold cache with touch）

	20M, normal CP, Ng = 1/6, non-MBSFN (cold cache with touch)										
	FDD				TDD 0						
	SFN any				SFN 0/5				SFN 1/6		
	PHICH normal			PHICH extended	PHICH normal			PHICH extended	PHICH normal		PHICH extended
	CFI = 1	CFI = 2	CFI = 3	CFI = 3	CFI = 1	CFI = 2	CFI = 3	CFI = 3	CFI = 1	CFI = 2	CFI = 2
P = 1	2150	5386	9857	9129	2335	6297	9742	9230	2385	5887	5526
P = 2	2569	6274	11190	10432	2746	7169	11108	10524	2804	6740	6414
P = 4	3846	8107	14448	14578	4012	8521	15055	14615	4061	8644	8256

4 小结

本文详细描述了在 TI 的 KeyStone 器件上实现高效的 LTE 下行控制信道基带发射的方法，并给出了实测的负载。对符号级处理，关键是要在小区建立或重配时，初始化随后的实时处理要用到的中间数据，利用这些中间数据，实时操作的复杂度得到了极大的简化。对比特级处理，提供了基于 BCP 的实现方案，同时也给出了基于 c66x 方法的负载，供用户对比和选择。

最后，我们通过一个例子来总结一下整个 LTE 下行控制信道处理的 c66x 负载。这个例子使用如下系统参数“TDD UL/DL configuration 0，normal CP，PHICH normal duration，$P = 2$”，假设当前子帧为子帧 1，CFI=3，20 个 HI，20 个格式为 1 的 PDCCH。假设比特级处理采用 BCP 方案，驱动软件的实时开销大致估算为 500+100*nPdcch（nPdcch 表示 PDCCH 个数）。表 13 给出了单项和整体负载。注意，PDCCH 比特级驱动的负载为粗估值，待实测。

表 13: 负载小结（一个例子）

模块		层 1 子帧级 cycle 数
PCFICH Gen		310
PHICH Gen		74.8*20+883=2,379
不用 BCP	PDCCH 扰码生成	0.25*72*88+67=1,651
	PDCCH 比特级主处理	20*1123+2500=24,960
使用 BCP	PDCCH 比特级驱动	100*20+500=2,500 （可在层 1 或层 2 执行）
	PDCCH 星座点映射	109*20+83=2,263
PDCCH 符号级 Gen		11,108
整体		***40,408 （不用 BCP）*** ***18,560 （使用 BCP，驱动由层 1 调用）*** ***16,060 （使用 BCP，驱动由层 2 调用）***

系统配置/重配时要调用 Init 函数。因为不需要实时调用，本文未提供该函数的负载。

参考文献

[1] *3GPP TS-36.211*
[2] *3GPP TS-36.212*
[3] *3GPP TS-36.213*
[4] *3GPP TS-36.214*
[5] http://www.ti.com/product/tms320tci6616
[6] http://www.ti.com/product/tms320tci6618
[7] http://www.ti.com/product/tms320tci6614
[8] http://www.ti.com/product/tms320tci6612
[9] http://www.ti.com/product/tms320c6670
[10] http://www.ti.com/product/tci6636k2h
[11] http://www.ti.com/product/tci6634k2k
[12] http://www.ti.com/product/tci6638k2k
[13] http://www.ti.com/product/tci6630k2l
[14] *TMS320C66x DSP Cache User Guide* (SPRUGY8)
[15] *TMS320C66x DSP CorePac User Guide* (SPRUGW0)
[16] *Enhanced Direct Memory Access (EDMA3) Controller User Guide* (SPRUGS5)
[17] *Bit Rate Coprocessor (BCP) User Guide* (SPRUGZ1)
[18] *Multicore Navigator User Guide* (SPRUGR9)

重要声明

德州仪器(TI) 及其下属子公司有权根据 JESD46 最新标准, 对所提供的产品和服务进行更正、修改、增强、改进或其它更改， 并有权根据 JESD48 最新标准中止提供任何产品和服务。客户在下订单前应获取最新的相关信息, 并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的组件的性能符合产品销售时 TI 半导体产品销售条件与条款的适用规范。仅在 TI 保证的范围内，且 TI 认为 有必要时才会使用测试或其它质量控制技术。除非适用法律做出了硬性规定，否则没有必要对每种组件的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用 TI 组件的产品和应用自行负责。为尽量减小与客户产品和应 用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何 TI 专利权、版权、屏蔽作品权或其它与使用了 TI 组件或服务的组合设备、机器或流程相关的 TI 知识产权中授予 的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息，不能构成从 TI 获得使用这些产品或服 务的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是 TI 的专利权或其它 知识产权方面的许可。

对于 TI 的产品手册或数据表中 TI 信息的重要部分，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况 下才允许进行复制。TI 对此类篡改过的文件不承担任何责任或义务。复制第三方的信息可能需要服从额外的限制条件。

在转售 TI 组件或服务时，如果对该组件或服务参数的陈述与 TI 标明的参数相比存在差异或虚假成分，则会失去相关 TI 组件 或服务的所有明示或暗示授权，且这是不正当的、欺诈性商业行为。TI 对任何此类虚假陈述均不承担任何责任或义务。

客户认可并同意，尽管任何应用相关信息或支持仍可能由 TI 提供，但他们将独力负责满足与其产品及在其应用中使用 TI 产品 相关的所有法律、法规和安全相关要求。客户声明并同意，他们具备制定与实施安全措施所需的全部专业技术和知识，可预见 故障的危险后果、监测故障及其后果、降低有可能造成人身伤害的故障的发生机率并采取适当的补救措施。客户将全额赔偿因 在此类安全关键应用中使用任何 TI 组件而对 TI 及其代理造成的任何损失。

在某些场合中，为了推进安全相关应用有可能对 TI 组件进行特别的促销。TI 的目标是利用此类组件帮助客户设计和创立其特 有的可满足适用的功能安全性标准和要求的终端产品解决方案。尽管如此，此类组件仍然服从这些条款。

TI 组件未获得用于 FDA Class III（或类似的生命攸关医疗设备）的授权许可，除非各方授权官员已经达成了专门管控此类使 用的特别协议。

只有那些 TI 特别注明属于军用等级或“增强型塑料”的 TI 组件才是设计或专门用于军事/航空应用或环境的。购买者认可并同 意，对并非指定面向军事或航空航天用途的 TI 组件进行军事或航空航天方面的应用，其风险由客户单独承担，并且由客户独 力负责满足与此类使用相关的所有法律和法规要求。

TI 已明确指定符合 ISO/TS16949 要求的产品，这些产品主要用于汽车。在任何情况下，因使用非指定产品而无法达到 ISO/TS16949 要求，TI不承担任何责任。

	产品		应用
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	www.ti.com.cn/dataconverters	消费电子	www.ti.com/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com/energy
DSP - 数字信号处理器	www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	www.ti.com.cn/microcontrollers		
RFID 系统	www.ti.com.cn/rfidsys		
OMAP应用处理器	www.ti.com/omap		
无线连通性	www.ti.com.cn/wirelessconnectivity	德州仪器在线技术支持社区	www.deyisupport.com

邮寄地址： 上海市浦东新区世纪大道 1568 号，中建大厦 32 楼 邮政编码： 200122