

MHSA 加速器系统集成功能验证报告

姓名：黄超凡，张博文，汪子尧
学号：124039910088, 124039910094, 124039910018

上海交通大学微纳电子学系

日期：2025 年 6 月 17 日

目录

1	系统集成实现	2
1.1	加速器 ICB 接口设计	2
1.2	加速器挂载	4
2	加速器软件驱动设计	5
2.1	SDK 设置	5
2.2	主函数设计	6
2.3	VCS 仿真结果	8
3	系统功能验证结果	8

1 系统集成实现

1.1 加速器 ICB 接口设计

加速器适配 ICB 总线的挂载，提供专门的 IMU 单元（Interface Manage Unit）进行接口管理。该单元将 ICB 总线输入的读写请求实时解析为两类信号：面向 SRAM 存储体的统一访存接口（Unified Sram Interface, USI）的访存通道，以及对控制状态寄存器（Control State Register, CSR）的硬件控制通道，CSR 通道包含 start, acc_done, input_base, output_base 信号。

对于 Host 端而言，不必关心加速器内部的具体实现细节，只需要关注加速器的 Memory Map 即可。Memory Map 定义了 Host 端与加速器交互时需要访问的寄存器地址和对应的功能描述。通过 Memory Map，Host 端可以方便地控制加速器的启动、监控加速器的状态以及读写加速器的数据缓冲区。Memory Map 表格如Table 1所示：

表 1: Memory Map

Offset Address	Memory Type	R/W	功能描述
0x00000000-0x00004000	Data Buffer	R/W	统一编址的 SRAM 存储体
0x00004000	Control Register	R/W	bit[0]=1：加速器使能
0x00004004	Status Register	R	bit[0]=1：运算完成
0x00004008	Configuration Register	R/W	输入数据起始地址
0x0000400C	Configuration Register	R/W	输出数据起始地址
0x10000000-0xFFFFFFFF	Reserved	-	保留地址空间

IMU 对 ICB 总线的地址信号进行译码，将其映射到 Memory Map 定义的地址空间。鉴于 USI 访存通道（面向 SRAM 存储体）与 CSR 硬件控制通道的行为逻辑存在显著差异，译码模块需要输出专门的 USRAM 选通信号 (usram_sel)，用以区分对 SRAM 的访问请求。

```
always@(posedge clk)
begin
    if(!rst_n) begin
        usram_addr <= 1'b0;
    end
    else begin
        usram_addr <= {16'b0, icb_cmd_addr[18:3]} - 'ha000;
    end
end

assign usram_sel = (usram_addr >= 'h0000 && usram_addr < 'h4000);
```

该地址转换逻辑的核心在于处理访存粒度的差异。ICB 总线以 32 位字（4 字节）为基本访存单位，而 USI 接口管理的 SRAM 存储体以 64 位（8 字节）为一个存储单元。因此，在计算 SRAM 内部地址时：

- (1) 舍弃 ICB 地址信号的最低 3 位 (icb_cmd_addr[2:0])，以实现地址对齐（将 32 位字地址转换为 64 位单元地址）。

(2) 减去总线地址空间中的加速器基地址偏移量 ('ha000) , 得到相对于 SRAM 存储体起始地址 (0x00000000) 的内部偏移地址 (usram_addr)。

(3) 基于 Memory Map 的定义, 生成有效的 SRAM 选通信号 (usram_sel)。

考虑到 ICB 总线的数据位宽为 32 位, 而 USRAM 存储单元的位宽为 64 位, 因此完成一次完整的 USRAM 单元写入需要两个连续的 32 位 ICB 写操作。IMU 单元实现了将这两个 ICB 写操作合并为单个 64 位 USRAM 写操作的功能 (32 位数据合并)。其核心逻辑如下:

```
// icb to usram interface : merge double-32bit-write to 64bit-write
always@(posedge clk)
begin
    if(!rst_n) begin
        usram_wdata <= 64'h0;
    end
    else begin
        if(icb_write_en & usram_sel) begin
            if(is_low_part) begin
                usram_wdata[31:0] <= usram_wdata[31:0];
                usram_wdata[63:32] <= icb_cmd_wdata;
            end
            else begin
                usram_wdata[31:0] <= icb_cmd_wdata;
                usram_wdata[63:32] <= usram_wdata[63:32];    // hold
            end
        end
        else begin
            usram_wdata <= usram_wdata;    // hold
        end
    end
end

always@(posedge clk)
begin
    if(!rst_n) begin
        usram_write_en <= 1'b0;
    end
    else if(icb_write_en & usram_sel & !is_low_part) begin
        usram_write_en <= 1'b1;    // pull up 1 cycle
    end
    else begin
        usram_write_en <= 1'b0;
    end
end
end
```

ICB 协议解析的具体细节此处不再赘述。简言之，控制通道通过握手启动读写操作，命令通道则通过握手完成实际数据传输。

1.2 加速器挂载

在 e203_subsys_main.v 文件中加入如下代码：

```
// MHSA Accelerator

wire                                mhsa_icb_cmd_valid;
wire                                mhsa_icb_cmd_ready;
wire                                mhsa_icb_cmd_read;
wire    [31:0]                     mhsa_icb_cmd_addr;
wire    [31:0]                     mhsa_icb_cmd_wdata;
wire    [3:0]                       mhsa_icb_cmd_wmask;

wire                                mhsa_icb_rsp_valid;
wire                                mhsa_icb_rsp_ready;
wire    [31:0]                     mhsa_icb_rsp_rdata;
wire                                mhsa_icb_rsp_err;

icb_mhsa u_icb_mhsa(
    .clk            (hfclk            ),
    .rst_n          (per_rst_n        ),

    // icb slave
    .icb_cmd_valid  (mhsa_icb_cmd_valid),
    .icb_cmd_ready  (mhsa_icb_cmd_ready),
    .icb_cmd_read   (mhsa_icb_cmd_read ),
    .icb_cmd_addr   (mhsa_icb_cmd_addr ),
    .icb_cmd_wdata  (mhsa_icb_cmd_wdata),
    .icb_cmd_wmask  (mhsa_icb_cmd_wmask),

    .icb_rsp_valid  (mhsa_icb_rsp_valid),
    .icb_rsp_ready  (mhsa_icb_rsp_ready),
    .icb_rsp_rdata  (mhsa_icb_rsp_rdata),
    .icb_rsp_err    (mhsa_icb_rsp_err  )
);
```

在 e203_subsys_perips.v 文件中，增加如下端口：

```
output                                mhsa_icb_cmd_valid,
input                                mhsa_icb_cmd_ready,
output                                mhsa_icb_cmd_read,
output    [31:0]                     mhsa_icb_cmd_addr,
output    [31:0]                     mhsa_icb_cmd_wdata,
```

```

output          [3:0]          mhsa_icb_cmd_wmask,

input           mhsa_icb_rsp_valid,
output          mhsa_icb_rsp_ready,
input           [31:0]        mhsa_icb_rsp_rdata,
input           mhsa_icb_rsp_err,

```

E203的私有外设接口地址空间为0x10000000–0x1FFFFFFF,从中划分一块地址空间0x10050000–0x1008FFFF 给 MHSa 加速器。修改 sirv_icb1to16_bus 模块的例化代码:

```

.....
// * MHSa          : 0x1005 0000 -- 0x1008 FFFF
.O15_BASE_ADDR      (32'h1005_0000),
.O15_BASE_REGION_LSB (18)
.....
// * MHSa
.o15_icb_enable      (1'b1),

.o15_icb_cmd_valid   (mhsa_icb_cmd_valid),
.o15_icb_cmd_ready   (mhsa_icb_cmd_ready),
.o15_icb_cmd_addr     (mhsa_icb_cmd_addr),
.o15_icb_cmd_read     (mhsa_icb_cmd_read),
.o15_icb_cmd_wdata    (mhsa_icb_cmd_wdata),
.o15_icb_cmd_wmask    (mhsa_icb_cmd_wmask),
.o15_icb_cmd_lock     (),
.o15_icb_cmd_excl     (),
.o15_icb_cmd_size     (),
.o15_icb_cmd_burst    (),
.o15_icb_cmd_beat     (),

.o15_icb_rsp_valid    (mhsa_icb_rsp_valid),
.o15_icb_rsp_ready    (mhsa_icb_rsp_ready),
.o15_icb_rsp_err      (mhsa_icb_rsp_err),
.o15_icb_rsp_excl_ok  (1'b0),
.o15_icb_rsp_rdata    (mhsa_icb_rsp_rdata),

```

至此加速器的挂载结束。

2 加速器软件驱动设计

2.1 SDK 设置

在 hbird-sdk-master/SoC/hbirdv2/Common/Include/hbirdv2.h 文件中,添加如下内容:

```
#define MHS_CFG_BASE          (HBIRD_PERIPH_BASE + 0x50000)
                        /*!< (MHS) Base Address */

#define MHS_CFG_REG(offset)    _REG32(MHS_CFG_BASE, offset)
```

2.2 主函数设计

main.c 内的设计主要有以下几部分。

首先需要定义加速器内各个寄存器或 mem 的偏移地址：

```
#define MEM0_OFFSET          (0x00000)
#define MEM1_OFFSET          (0x08000)
#define MEM2_OFFSET          (0x10000)
#define MEM3_OFFSET          (0x18000)

#define START_OFFSET         (0x20000)
#define DONE_OFFSET          (0x20004)
#define INPUT_BASE_OFFSET    (0x20008)
#define OUTPUT_BASE_OFFSET   (0x2000c)

#define RESULT_OFFSET         (0x06000)
```

然后需要进行阶段 1——数据的随机化写入，在这一步我们使用指针去将 W_O, W_Q, W_K, W_V 四个权重矩阵与输入矩阵写入对应的 mem 内：

```
// Stage 1: data write
volatile uint32_t *mem0_ptr = (uint32_t *) (MHS_CFG_BASE + MEM0_OFFSET);
volatile uint32_t *mem1_ptr = (uint32_t *) (MHS_CFG_BASE + MEM1_OFFSET);
volatile uint32_t *mem2_ptr = (uint32_t *) (MHS_CFG_BASE + MEM2_OFFSET);
volatile uint32_t *mem3_ptr = (uint32_t *) (MHS_CFG_BASE + MEM3_OFFSET);
volatile uint32_t *mem0_start_ptr = mem0_ptr;
volatile uint32_t *mem1_start_ptr = mem1_ptr;
volatile uint32_t *mem2_start_ptr = mem2_ptr;
volatile uint32_t *mem3_start_ptr = mem3_ptr;

for (i = 0; i < 128; i++) {
    for (j = 0; j < 32; j++) {
        *mem0_ptr = rand();
        mem0_ptr++;
    }
}

for (i = 0; i < 128; i++) {
    for (j = 0; j < 32; j++) {
        *mem1_ptr = rand();
        mem1_ptr++;
    }
}
```

```

    }
}
for (i = 0; i < 128; i++) {
    for (j = 0; j < 32; j++) {
        *mem2_ptr = rand();
        mem2_ptr++;
    }
}
for (i = 0; i < 128; i++) {
    for (j = 0; j < 32; j++) {
        *mem3_ptr = rand();
        mem3_ptr++;
    }
}
for (i = 0; i < 128; i++) {
    for (j = 0; j < 8; j++) {
        *mem0_ptr = rand();
        mem0_ptr++;
    }
}
}

```

阶段 2——配置 MHSa 加速器内的寄存器，使 MHSa 加速器开始计算，并等待计算结束。

```

// Stage 2: MHSa compute
volatile uint32_t *start_ptr = (uint32_t *) (MHSa_CFG_BASE + START_OFFSET);
volatile uint32_t *done_ptr = (uint32_t *) (MHSa_CFG_BASE + DONE_OFFSET);

*start_ptr = 1; // Trigger the MHSa compute operation
printf("MHSa compute operation starts.\n");
while (*done_ptr == 0) {
    // Wait for the compute operation to complete
}
/*start_ptr = 0; // Reset the start signal
printf("MHSa compute operation completed.\n");

```

阶段 3——结果输出，我们同样使用指针读取对应地址的数据：

```

// Stage 3: result output
volatile uint32_t *result_ptr = (uint32_t *) (MHSa_CFG_BASE + RESULT_OFFSET);
;
printf("Result matrix is:\n");
for (i = 0; i < 128; i++) {
    for (j = 0; j < 8; j++) {
        printf("%08X ", *result_ptr);
        result_ptr++;
    }
}

```

```

    }
    printf("\n");
}
printf("Test finished.");
return 0;

```

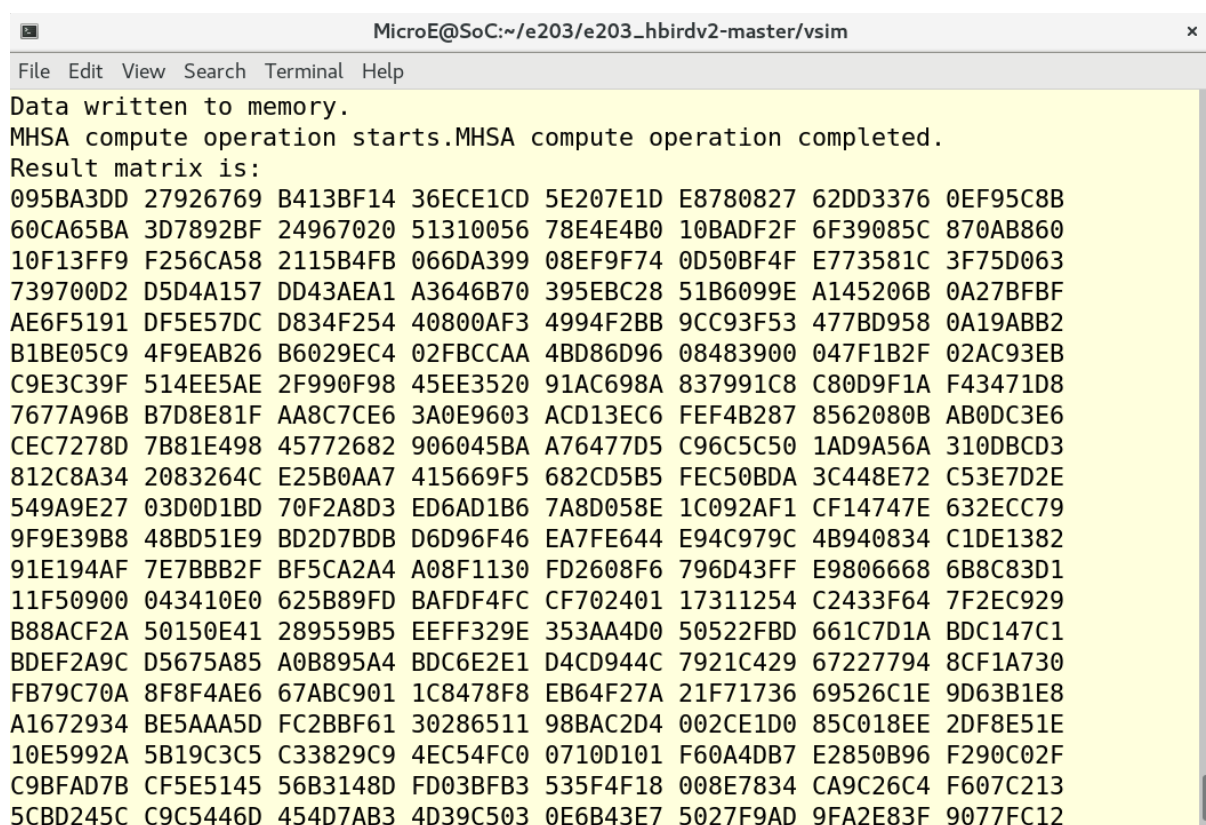
至此软件驱动的主函数设计结束。

2.3 VCS 仿真结果

首先更改 MHSA 文件夹下的 Makefile，更改 TARGET 为 mhsa，然后输入如下指令进行交叉编译：

```
make dasm SOC=hbirdv2 CORE=e203 DOWNLOAD=ilm
```

更改 vsim 文件夹下的 Makefile，更改 TESTCASE 与 TESTNAME 为 MHSA 的名称与路径，在 vcs 工具上进行编译，得到如下所示的控制台输出结果：



```

MicroE@SoC:~/e203/e203_hbirdv2-master/vsim
File Edit View Search Terminal Help
Data written to memory.
MHSA compute operation starts.MHSA compute operation completed.
Result matrix is:
095BA3DD 27926769 B413BF14 36ECE1CD 5E207E1D E8780827 62DD3376 0EF95C8B
60CA65BA 3D7892BF 24967020 51310056 78E4E4B0 10BADF2F 6F39085C 870AB860
10F13FF9 F256CA58 2115B4FB 066DA399 08EF9F74 0D50BF4F E773581C 3F75D063
739700D2 D5D4A157 DD43AEA1 A3646B70 395EBC28 51B6099E A145206B 0A27BFBF
AE6F5191 DF5E57DC D834F254 40800AF3 4994F2BB 9CC93F53 477BD958 0A19ABB2
B1BE05C9 4F9EAB26 B6029EC4 02FBCCAA 4BD86D96 08483900 047F1B2F 02AC93EB
C9E3C39F 514EE5AE 2F990F98 45EE3520 91AC698A 837991C8 C80D9F1A F43471D8
7677A96B B7D8E81F AA8C7CE6 3A0E9603 ACD13EC6 FEF4B287 8562080B AB0DC3E6
CEC7278D 7B81E498 45772682 906045BA A76477D5 C96C5C50 1AD9A56A 310DBCD3
812C8A34 2083264C E25B0AA7 415669F5 682CD5B5 FEC50BDA 3C448E72 C53E7D2E
549A9E27 03D0D1BD 70F2A8D3 ED6AD1B6 7A8D058E 1C092AF1 CF14747E 632ECC79
9F9E39B8 48BD51E9 BD2D7BDB D6D96F46 EA7FE644 E94C979C 4B940834 C1DE1382
91E194AF 7E7BBB2F BF5CA2A4 A08F1130 FD2608F6 796D43FF E9806668 6B8C83D1
11F50900 043410E0 625B89FD BAFDF4FC CF702401 17311254 C2433F64 7F2EC929
B88ACF2A 50150E41 289559B5 EEEF329E 353AA4D0 50522FBD 661C7D1A BDC147C1
BDEF2A9C D5675A85 A0B895A4 BDC6E2E1 D4CD944C 7921C429 67227794 8CF1A730
FB79C70A 8F8F4AE6 67ABC901 1C8478F8 EB64F27A 21F71736 69526C1E 9D63B1E8
A1672934 BE5AAA5D FC2BBF61 30286511 98BAC2D4 002CE1D0 85C018EE 2DF8E51E
10E5992A 5B19C3C5 C33829C9 4EC54FC0 0710D101 F60A4DB7 E2850B96 F290C02F
C9BFAD7B CF5E5145 56B3148D FD03BFB3 535F4F18 008E7834 CA9C26C4 F607C213
5CBD245C C9C5446D 454D7AB3 4D39C503 0E6B43E7 5027F9AD 9FA2E83F 9077FC12

```

图 1: 软件驱动控制台输出（部分）

3 系统功能验证结果

通过日志文件 mhsa.log 可以获取在软件驱动中随机化的 W_O, W_Q, W_K, W_V 权重矩阵与输入矩阵，将权重矩阵与输入矩阵在 SV 验证平台初始化 MEM，进行硬件仿真，获得'.mem' 文件，将 SOC 输出结果与'.mem' 结果进行数据对比完全一致，系统功能验证成功！

3064	46310ff2d6b587ea	e522c4b4ec1509f9	4cc3cb4772da5d14	cd4b99ed73f8c36a	07a2a4ede238b0b9	3a7ad6f58753d0b8	f697a8b8310de6df	5b09190ff47ee69a
3072	095ba3dd279267c9	b413bf1436ecec1cd	5e207e1de8780827	62dd33760ef95c8b	60ca65ba3d7892bf	2496702051310056	78e4e4b010badf2f	6f39085c870ab860
3080	10f13ff9f256ca58	2115b4fb066da399	08ef9f740d50bf4f	e773581c3f75a063	739700d2d5d4a157	dd43aea1a3646b70	395ebc2851b6099e	a145206b0a27bfbf
3088	ae6f5191df5e57dc	d834f25440800af3	4994f2bb9cc93f53	477bd9580a19abb2	b1be05c94f9eab26	b6029ec402fbccaa	4bd86d9608483900	047f1b2f02ac93eb
3096	c9e3c39f514ee5ae	2f990f9845ee3520	91ac698a837991c8	c80d9f1af43471d8	7677a96bb7d8e81f	aa8c7ce63a0e9603	acd13ec6fef4b287	8562080bab0dc3e6
3104	cec7278d7b81e498	45772682906045ba	a76477d5c96c5c50	1ad9a56a310dbcd3	812c8a342083264c	e25b0aa7415669f5	682cd5b5fec50bda	3c448e72c53e72de
3112	549e9e2703d0d1bd	70f2a8d3ed6ad1b6	7a8d058e1c092af1	cf14747e632ecc79	9f9e39b848bd51e9	bd2d7b4dbd6d96f46	ea7fe644e94c979c	4b940834c1de1382
3120	91e194af7e7bb2f	bf5ca2a4a08f1130	fd2608f6796d43ff	e98066686b8c83d1	11f50900043410e0	625b89f4dbafdf4fc	cf70240117311254	c2433f647f2ec929
3128	b88acf2a50150e41	289559b5eeff329e	353aa4d050522fbd	661c7d1abdc147c1	bdef2a9cd5675a85	a0b895a4bdc6e2e1	d4cd944c7921c429	672277948cfla730
3136	fb79c70a8f8f4ae6	67abc9011c8478f8	eb64ef27a21f71736	69526c1e9d63b1e8	al672934be5aaa5d	fc2bbf6130286511	98bac2d4002ce1d0	85c018ee2df8e51e
3144	10e5992a5b19c3c5	c33829c94ec54fc0	0710d101f60a4db7	e2850b96f290c02f	c9bfad7bcf5e5145	56b3148df0d3bfb3	535f4f18008e7834	ca9c26c4f607c213
3152	5cbdd245cc9c5446d	454d7ab34d39c503	0e6b43e75027f9ad	9fa2e83f9077fc12	586428582c87572f	36aed84bd42783b8	c8a926b59e916c28	c3d47bb977b6cbb1
3160	f149f65d3559d933	2f553e3e66e0b19c	6867289ab9cf4633	b16b12f94aa28beb	e83c1b90bade4432	da4ac20014c92c01	60aa7b2797028ae5	c3a1e7b467d2eabe
3168	6dafbfe0793257c6	ad8bd14f2755bca3	7f8e1f2c2b48db0b	9d2balaef6915c72	a7f5cd38f568975d	3825be461039122b	0e38ed343a40b7ac	60a0fb4e7837248a
3176	a9ala23fee0de9d9	317a7fbc3aed0786	5180dbf73aa2cbc2	8ce60b34053b812e	350c87c014f35cf9	18189de5a8543d6c	2248571b4ab02033	113e7b273e2d55c7
3184	7e65077c52d2022e	d4e758d3530c3871	de322fddcb4727b1	29c1a217665b6b4d	d3c7f529aac5d572	f8f623480ecc29d2	c538a243ae98c23d	21b53e10b24556de
3192	403f77a1554d066f	6485f2d37191bf6f	25934d6782c0ba5a	213679871d73ef30	3e797ce436e0b561	3bc002d64c3d29db	c8a5b010e1de5a1f	4da4fb74c4e4aff9b
3200	298cd1467e553446	4a7de9baba3ff782	f6e6471a4bf6e51e	793fddf19affbee90	a1738c42f00f1461	bdad0b542990738d	4fd21fd75078b4e0	ebf266fd295756d
3208	063dd0c5feedc12	8179e2526ee0b572	ed4f6c62f1b5a7fe	4437e6cd984a0875	29f540c76996c01c	b0e201a115e116a0	4f559feca10ae0a0	ec5cf301691fd037
3216	b1e25ae49a9e2dc1	7eae421a64a2ba8b	6cf4509d60900594	3a0475ad718ae7b7	57d8e28aa3d0f2d9	82d2e064a23202e5	24cbcd2b612b21a9	4a3d06ad4761e861
3224	2a557fff614081e9	949b3f2f0606e5bb	a66ec8f9f3d2a2be	ecbec106447484d3	2fffef90dbcd5d8e	34a434c45dba51d0	bc594c597728adfc	a3ae1760dfc21e7c
3232	f08aaabd6f083621	1bd1685f80ca144c	901dec9c944d6e9bf	5b0b5600ae104b6d	e6cc812d6a3696d5	ce8f9f073cfcfb882	7901eb308ad77776	27aed470935594d3
3240	708ae2c5e6a06887	fecf29d8947156bf	565cf741c35d8241	9a784eeca3ef46c6	a9694d5b44c1c261	18b0dd7b0847205a	9223e1ced2476ddd	17aafee169f096d2
3248	5c846a68eed00d12	cd30a3876c674daa	365aaf25494019d1	791818bf73add609	342e161e0dd6dd21	80ba9b062e5d2355	91e46b512e286f47	3bfaf7304523bd5d
3256	cd23dd9de9be5a0b	98bccc57c51f766b	f4a84b97eb305744	a3350520da3dffa9	c09b780c60724863	e5a95b1cf4eb9ab9	764c3b1dclb67bcb	291fdcc4246834d
3264	0eeae62dd4fbdcla	83cb87110605df44	eb2339fde15d0ba7	978b4d1e234b869f	2521dfa6b571fc92	6234dbdcdalaf29d	1f8035ae2397c805	4be9b65dcd61d7f
3272	58c5c6a7807a5952	4b670ee8d82ee45d	c5ee8a5f7c217b31	db1d00b981a07af5	f7fc4f7b617ffcd23	b65747a642137768	c2c3e7a2db49defc	8f1fd9f23f1a5add
3280	12aed6d0536a82b7	9e80c4b932d6f1e3	d15969f16faa7c1c	00ebee27b72b7574	828ba94c63ec5e3f	237ef73e762cd0a7	57fae49e50ccccc0f	742d04d54e982d19
3288	bb3575fc7355ef4b	678c4a64eb6f13a3	cb7c6b2a0ca50bf2	d14ff3e649e0636f	b78bd6987ff6b570	d460562501fe2d43	3b11ef7fb8fbb2e7	5463f93098ae51f5
3296	d18895333ba9d5a6	a632e28d4b4fbc5b	19b6b6df52661ce0	09a1947aa9777362	4e292cd984e6182b	9abca24a26720636	167f4221204ac014	8751cfa39735da2b
3304	15140c82ac87bb56	9c961ce1ff18c9d4	bcl17b98802a02a18	ca31c92ec741d317	1215ff125bd6b33a	8239e4d5c21d913a2	5e0e744971566e11	5ff3dc9079a8f10b
3312	4f5949d364583f7d	0139cdde33eb6a61	f83937d168b2dcf7	8257df91be9ee237	45b18000231eeba3	637418a5bde1552f	ba7fea0fc4b611cb	90b69b64392de80c
3320	e79be4561d099e59	1a879326b2e0bee4	d9350f6e6c7da924	d345531e7995747f	11defb64ff07b4b6	f4676c47d49cb94f	ec2cf6e4bd6eeaa53	b952ab949d51b13c
3328	2a4a5559c3291f10	50a39b0eaa80aff7	a7a337a9e9e9484d	626ca7bc03e4573f	689a9aeb51fbbf4f	0c877e589ff534388	4edc980026d8a86	ffbb65dd39b47bfe

图 2: 硬件仿真结果与 SOC 对比 (部分)