# Sample

## struct

### struct dwc\_usb3\_device \*usb3\_dev

typedef **struct dwc\_usb3\_device** {

**struct dwc\_usb3\_pcd pcd**;

uint32\_t snpsid;

**dwc\_usb3\_core\_params\_t \*core\_params**;

**dwc\_usb3\_core\_global\_regs\_t \*core\_global\_regs**; // reg : 0xb0400000 + 0xC100

uint32\_t **\*event\_ptr**[1]; //usb3\_dev->event\_buf[0]

uint32\_t **\*event\_buf**[1]; //(0x00000000 + 0x90000)

dwc\_dma\_t **event\_buf\_dma**[1]; //usb3\_dev->event\_buf[0]

uint16\_t total\_fifo\_size;

uint16\_t rx\_fifo\_size;

uint32\_t hwparams0; //reg\_read(core\_global\_regs->ghwparams0)

uint32\_t hwparams1; //reg\_read(core\_global\_regs->ghwparams1)

uint32\_t hwparams2; //reg\_read(core\_global\_regs->ghwparams2)

uint32\_t hwparams3; //reg\_read(core\_global\_regs->ghwparams3)

uint32\_t hwparams4; //reg\_read(core\_global\_regs->ghwparams4)

uint32\_t hwparams5; //reg\_read(core\_global\_regs->ghwparams5)

uint32\_t hwparams6; //reg\_read(core\_global\_regs->ghwparams6)

uint32\_t hwparams7; //reg\_read(core\_global\_regs->ghwparams7)

uint32\_t dcfg; //reg\_read(dev\_global\_regs->dcfg)

unsigned int cmn\_initialized : 1;

unsigned int pcd\_initialized : 1;

unsigned int common\_irq\_installed : 1;

unsigned int sysfs\_initialized : 1;

unsigned int phy\_init\_done : 1;

} dwc\_usb3\_device\_t

### struct dwc\_usb3\_core\_params

typedef **struct dwc\_usb3\_core\_params** {

int burst; //0

int new; //0

int phy; //0

int wakeup; //0

int pwrctl; //0

} dwc\_usb3\_core\_params\_t;

### struct dwc\_usb3\_pcd pcd

typedef **struct dwc\_usb3\_pcd** {

**struct dwc\_usb3\_device \*usb3\_dev**;

int link\_state; //0

ep0state\_e ep0state; //EP0\_UNCONNECTED

unsigned int ep0\_status\_pending : 1;

unsigned int request\_config : 1;

unsigned int remote\_wakeup\_enable : 1;

unsigned int wkup\_rdy : 1; //0

unsigned int eps\_enabled : 1;

unsigned int pad : 3;

unsigned int devnum : 4;

unsigned int request\_pending;

**dwc\_usb3\_pcd\_ep\_t \*ep0**;

**dwc\_usb3\_pcd\_ep\_t \*out\_ep[DWC\_MAX\_EPS - 1]**;

**dwc\_usb3\_pcd\_ep\_t \*in\_ep[DWC\_MAX\_EPS - 1]**;

int test\_mode;

**dwc\_usb3\_dev\_global\_regs\_t \*dev\_global\_regs**; // reg : 0xb0400000 + 0xC700

**dwc\_usb3\_dev\_ep\_regs\_t \*out\_ep\_regs**; // reg : 0xb0400000 + 0xC800

**dwc\_usb3\_dev\_ep\_regs\_t \*in\_ep\_regs**; // reg : 0xb0400000 + 0xC810

uint8\_t speed; //0

uint8\_t **num\_out\_eps**; // GHWPARAMS3 & 0x0003f000 - num\_in\_eps = 4

uint8\_t **num\_in\_eps**; // GHWPARAMS3 & 0x007c0000 – 1 = 4

uint16\_t tx\_fifo\_size[DWC\_MAX\_TX\_FIFOS];

uint16\_t rx\_thr\_en;

uint16\_t iso\_tx\_thr\_en;

uint16\_t non\_iso\_tx\_thr\_en;

uint16\_t rx\_thr\_length;

uint16\_t tx\_thr\_length;

union {

usb\_device\_request\_t req;

uint32\_t d32[2];

} \*setup\_pkt; // (0x00000000 + 0xa0000)

dwc\_dma\_t setup\_pkt\_dma; // pcd->setup\_pkt

uint8\_t \*status\_buf; // (0x00000000 + 0xb0000)

dwc\_dma\_t status\_buf\_dma; // pcd->status\_buf

dwc\_dma\_t dma\_setup\_desc\_addr; // pcd->setup\_desc\_addr

dwc\_usb3\_dma\_desc\_t \*setup\_desc\_addr; // (0x00000000 + 0xc0000)

dwc\_dma\_t dma\_out\_desc\_addr; // pcd->out\_desc\_addr

dwc\_usb3\_dma\_desc\_t \*out\_desc\_addr; // (0x00000000 + 0xd0000)

dwc\_dma\_t dma\_in\_desc\_addr; // pcd->in\_desc\_addr

dwc\_usb3\_dma\_desc\_t \*in\_desc\_addr; // (0x00000000 + 0xd0000)

dwc\_usb3\_pcd\_request\_t \*ep0\_req;

} dwc\_usb3\_pcd\_t;

### struct dwc\_usb3\_pcd\_ep

typedef **struct dwc\_usb3\_pcd\_ep** {

**dwc\_ep\_t dwc\_ep**;

struct { void \*dummy; } usb\_ep;

} dwc\_usb3\_pcd\_ep\_t;

### struct dwc\_ep

typedef **struct dwc\_ep** {

**struct dwc\_usb3\_pcd \*pcd**;

const **usb\_endpoint\_descriptor\_t \*desc**; //ep[0]:=NULL; in\_ep[1]= &\_in1\_desc\_; out\_ep[1]=&\_out1\_desc\_

unsigned int stopped : 1; //1

unsigned int disabling : 1;

unsigned int queue\_sof : 1; //0

unsigned int send\_zlp : 1; //0

unsigned int stall\_clear\_flag : 1;

unsigned int three\_stage : 1;

unsigned int isoc\_started : 1;

unsigned int **is\_in** : 1; // ep[0]:=0; in\_ep[1-4]= 1; out\_ep[1-4]= 0

unsigned int active : 1; //0

unsigned int data\_pid\_start : 1;

unsigned int even\_odd\_frame : 1;

uint8\_t **num**; // ep[0]:=0; in\_ep[1-4]= i; out\_ep[1-4]= i

uint8\_t **type;** //DWC\_USB3\_EP\_TYPE\_CONTROL

uint16\_t **maxpacket**; // ep[0]:=64; in\_ep[1-4]= 512; out\_ep[1-4]= 512

uint8\_t intvl;

uint8\_t mult;

uint8\_t maxburst;

uint8\_t num\_streams;

uint8\_t tx\_fifo\_num; // ep[0]:=0; in\_ep[1-4]= i; out\_ep[1-4]= 0

uint8\_t **tri\_out**; //1: xfers out active

uint8\_t **tri\_in**; //1: xfers in active

uint8\_t tri\_out\_initialize; //0

uint8\_t tri\_in\_initialize; //0

uint32\_t condition;

char \*dma\_desc;

dwc\_dma\_t dma\_desc\_dma;

int dma\_desc\_size;

int num\_desc;

int desc\_avail;

int desc\_idx;

}

### struct usb\_endpoint\_descriptor\_t;

typedef struct {

uByte bLength; //7

uByte **bDescriptorType**; //USB\_DT\_ENDPOINT

uByte **bEndpointAddress**; //in: 0x81, out:0x01

uByte **bmAttributes**; //0x02: bulk

uWord **wMaxPacketSize**; //512

uByte bInterval;

} usb\_endpoint\_descriptor\_t;

### struct dwc\_usb3\_pcd\_request

typedef struct dwc\_usb3\_pcd\_request {

dwc\_req\_t dwc\_req;

} dwc\_usb3\_pcd\_request\_t;

### struct dwc\_req

typedef struct dwc\_req {

void \*buf; /\*\*< data buffer \*/

dwc\_dma\_t dma; /\*\*< DMA address of data buffer \*/

dwc\_usb3\_dma\_desc\_t \*trb; /\*\*< TRB for this request \*/

dwc\_dma\_t trb\_dma; /\*\*< DMA address of TRB \*/

uint32\_t length; /\*\*< length of data buffer \*/

uint32\_t actual; /\*\*< actual amount of data transfered \*/

uint32\_t stream; /\*\*< stream # of this request \*/

int flags;

} dwc\_req\_t;

### struct \_ATAPI sAtapi

typedef struct \_ATAPI{

UNION\_CBW uCbw;

UNION\_CSW uCsw;

INQUIRY\_DATA sInquiryData;

READ\_FORMAT\_CAPACITIES\_DATA sReadFormatCapacitiesData;

READ\_CAPACITIES\_DATA sReadCapacitiesData;

MODE\_SENSE\_5A\_DATA sModeSense5AData;

MODE\_SENSE\_1A\_DATA sModeSense1AData;

REQUEST\_SENSE\_DATA sRequestSenseData;

UINT dAtaId[64]; //dAtaId[30] = 0x00008000 = 32k\*512B=16M

UCHAR bMscStage; //means the next stage

}ATAPI,\*PATAPI; //\_\_attribute\_\_ ((unpacked,aligned()))

### struct \_INQUIRY\_DATA

typedef struct \_INQUIRY\_DATA{

UCHAR PeripheralDeviceType; //0x00

UCHAR RMB; //0x80

UCHAR ISO\_ECMA\_ANSI\_version; //0x00

UCHAR ResponseDataFormat; //0x01

UCHAR AdditionalLength; //0x31

UCHAR Reserved0; //0x00

UCHAR Reserved1; //0x00

UCHAR Reserved2; //0x00

UCHAR VendorInformation[8]; //”ACTIONS”

UCHAR ProductIdentification[16]; //”USB2.0 OTG DISK”

UCHAR ProductVersionLevel[4]; //”1.00”

} INQUIRY\_DATA, \* PINQUIRY\_DATA;

### struct \_READ\_FORMAT\_CAPACITIES\_DATA

typedef struct \_READ\_FORMAT\_CAPACITIES\_DATA{

UINT dCapacityListHeader; //0x08000000

UCHAR bCurrentMaxCapacityNumOfBlocks[4]; //sAtapi.dAtaId[30]

UCHAR bCurrentMaxCapacityDescrptCode; //1

UCHAR bCurrentMaxCapacityBlockLength[3]; //0x200 = 512B

UCHAR bFormattableCapacityNumOfBlocks[4]; //sAtapi.dAtaId[30]

UINT dFormattableCapacityBlockLength; //0x00020000 = 128k\*512B=64M

} READ\_FORMAT\_CAPACITIES\_DATA, \* PREAD\_FORMAT\_CAPACITIES\_DATA;

### struct \_READ\_CAPACITIES\_DATA

typedef struct \_READ\_CAPACITIES\_DATA{

UCHAR bLastLogicBlockAddress[4]; //sAtapi.dAtaId[30] - 1

UINT dBlockLength; //0x00020000 = 128k\*512B=64M

} READ\_CAPACITIES\_DATA, \* PREAD\_CAPACITIES\_DATA;

### struct \_MODE\_SENSE\_5A\_DATA

typedef struct \_MODE\_SENSE\_5A\_DATA{

UCHAR bModeParamHeader[8]; //0

MODE\_SENSE\_PAGE sModeSensePage;

} MODE\_SENSE\_5A\_DATA, \* PMODE\_SENSE\_5A\_DATA;

### struct \_MODE\_SENSE\_1A\_DATA

typedef struct \_MODE\_SENSE\_1A\_DATA{

UCHAR bModeParamHeader[4]; //0

MODE\_SENSE\_PAGE sModeSensePage;

} MODE\_SENSE\_1A\_DATA, \* PMODE\_SENSE\_1A\_DATA;

### struct \_MODE\_SENSE\_PAGE

typedef struct \_MODE\_SENSE\_PAGE{

UCHAR bTimerProtectPage[8]; //0x03000600 | TimerProtectPage(0x1c)

UCHAR bRwErrRecoveryPage[12]; //0x00000A00 | RwErrRecoveryPage(0x01)

UCHAR bCacheingPage[12]; //0x00010A00 | CacheingPage(0x08)

UCHAR bRemovableBlockPage[12]; //0x00000A00 | RemovableBlockPage(0x1b)

} MODE\_SENSE\_PAGE, \* PMODE\_SENSE\_PAGE;

### struct \_REQUEST\_SENSE\_DATA

### dwc\_usb3\_cil\_callbacks\_t pcd\_callbacks

typedef struct \_REQUEST\_SENSE\_DATA{

UCHAR bErrCode; //0x70

UCHAR bReserved0; //0x0

UCHAR bSenseKey0; //0x0

UCHAR bInformation0; //0x0

UCHAR bInformation1; //0x0

UCHAR bInformation2; //0x0

UCHAR bInformation3; //0x0

UCHAR bAdditionalSenseLength; //0x0

UCHAR bReserved1; //0x0

UCHAR bReserved2; //0x0

UCHAR bReserved3; //0x0

UCHAR bReserved4; //0x0

UCHAR bAdditionalSenseCode; //0x0

UCHAR bAdditionalSenseCodeQualifier; //0x0

UCHAR bFieldReplacebleUnitCode; //0x0

UCHAR bSKSV; //0x0

UCHAR bSenseKeyS0; //0x0

UCHAR bSenseKeyS1; //0x0

} REQUEST\_SENSE\_DATA, \* PREQUEST\_SENSE\_DATA;

static dwc\_usb3\_cil\_callbacks\_t pcd\_callbacks = {

.start = pcd\_start\_cb,

.stop = pcd\_stop\_cb,

.suspend = pcd\_suspend\_cb,

.resume\_wakeup = pcd\_resume\_cb,

.p = 0, // **struct dwc\_usb3\_pcd**

};

## Function

### dwc\_usb3\_mode() //get usb mode

static inline uint32\_t dwc\_usb3\_mode(dwc\_usb3\_device\_t \*usb3\_dev)

{

return dwc\_read\_reg32(&usb3\_dev->core\_global\_regs->**gsts**) & **0x1**; // 0 - Device Mode, 1 - Host Mode

}

### init\_devspd() //clean address

//clean the address

static void init\_devspd(dwc\_usb3\_device\_t \*usb3\_dev)

{

dwc\_modify\_reg32(&usb3\_dev->pcd.dev\_global\_regs->**dcfg**,

DWC\_DCFG\_DEVADDR\_BITS, 0 << DWC\_DCFG\_DEVADDR\_SHIFT);

}

DWC\_DCFG\_DEVADDR\_BITS = 0x0003f8, //Device Address

DWC\_DCFG\_DEVADDR\_SHIFT = 3,

### calc\_num\_in\_eps() //get in\_ep\_nums

GHWPARAMS3 = 0x0514a085

In\_ep\_nums = 0x0514a085 & 0x007c0000 -1 = 00101b -1 = 0x5 – 1 = 0x4;

static uint32\_t calc\_num\_in\_eps(dwc\_usb3\_device\_t \*usb3\_dev)

{

uint32\_t num\_in\_eps = (usb3\_dev->hwparams3 >> DWC\_HWPARAMS3\_NUM\_IN\_EPS\_SHIFT) &

(DWC\_HWPARAMS3\_NUM\_IN\_EPS\_BITS >> DWC\_HWPARAMS3\_NUM\_IN\_EPS\_SHIFT);

return num\_in\_eps - 1;

}

DWC\_HWPARAMS3\_NUM\_IN\_EPS\_BITS = 0x007c0000,

### calc\_num\_out\_eps() //get out\_ep\_nums

GHWPARAMS3 = 0x0514a085

out\_ep\_nums = 0x0514a085 & 0x0003f000 – in\_ep\_nums -1 = 001010b – 0x5 - 1 = 0xa – 0x5 - 1 = 0x4;

static uint32\_t calc\_num\_out\_eps(dwc\_usb3\_device\_t \*usb3\_dev)

{

uint32\_t num\_eps = (usb3\_dev->hwparams3 >> DWC\_HWPARAMS3\_NUM\_EPS\_SHIFT) &

(DWC\_HWPARAMS3\_NUM\_EPS\_BITS >> DWC\_HWPARAMS3\_NUM\_EPS\_SHIFT);

uint32\_t num\_in\_eps = (usb3\_dev->hwparams3 >> DWC\_HWPARAMS3\_NUM\_IN\_EPS\_SHIFT) &

(DWC\_HWPARAMS3\_NUM\_IN\_EPS\_BITS >> DWC\_HWPARAMS3\_NUM\_IN\_EPS\_SHIFT);

return num\_eps - num\_in\_eps - 1;

}

DWC\_HWPARAMS3\_NUM\_EPS\_BITS = 0x0003f000,

### dis\_eventbuf\_intr () //disable Event Buffer interrupt

static void dis\_eventbuf\_intr(dwc\_usb3\_device\_t \*dev, int bufno) // bufno = 0

{

dwc\_modify\_reg32(&dev->core\_global\_regs->geventbuf[bufno].**geventsiz**,

DWC\_EVENTSIZ\_INT\_MSK\_BIT, DWC\_EVENTSIZ\_INT\_MSK\_BIT);

}

DWC\_EVENTSIZ\_INT\_MSK\_BIT = 0x80000000, // Event Interrupt Mask

DWC\_EVENTSIZ\_INT\_MSK\_SHIFT = 31

### ena\_eventbuf\_intr() //enables Event Buffer interrupt

static void ena\_eventbuf\_intr(dwc\_usb3\_device\_t \*dev, int bufno)

{

dwc\_modify\_reg32(&dev->core\_global\_regs->geventbuf[bufno].**geventsiz**,

DWC\_EVENTSIZ\_INT\_MSK\_BIT, 0);

}

DWC\_EVENTSIZ\_INT\_MSK\_BIT = 0x80000000, // Event Interrupt Mask

### dis\_flush\_eventbuf\_intr() //disable and flush event buffer interrupt

static void dis\_flush\_eventbuf\_intr(dwc\_usb3\_device\_t \*dev, int bufno) // bufno = 0

{

dis\_eventbuf\_intr(dev, bufno); //disable Event Buffer interrupt

// flushes any pending events from the buffer

cnt = dwc\_read\_reg32(&dev->core\_global\_regs->geventbuf[bufno].**geventcnt**);

dwc\_write\_reg32(&dev->core\_global\_regs->geventbuf[bufno].**geventcnt**, cnt);

}

### dwc\_usb3\_enable\_common\_interrupts() //enable common interrupt

void dwc\_usb3\_enable\_common\_interrupts(dwc\_usb3\_device\_t \*usb3\_dev)

{

dis\_flush\_eventbuf\_intr(usb3\_dev, 0);

ena\_eventbuf\_intr(usb3\_dev, 0);

}

### dwc\_usb3\_enable\_device\_interrupts() //enable device interrupt

void dwc\_usb3\_enable\_device\_interrupts(dwc\_usb3\_device\_t \*usb3\_dev)

{

dwc\_usb3\_enable\_common\_interrupts(usb3\_dev);

dwc\_write\_reg32(&usb3\_dev->pcd.dev\_global\_regs->**devten**,

DWC\_DEVTEN\_DISCONN\_BIT |

DWC\_DEVTEN\_USBRESET\_BIT |

DWC\_DEVTEN\_CONNDONE\_BIT |

DWC\_DEVTEN\_ULST\_CHNG\_EN\_BIT);

}

DWC\_DEVTEN\_DISCONN\_BIT = 0x0001, // Disconnect Detected Event Enable

DWC\_DEVTEN\_USBRESET\_BIT = 0x0002, // USB Reset Enable

DWC\_DEVTEN\_CONNDONE\_BIT = 0x0004, // Connect Done Enable

DWC\_DEVTEN\_ULST\_CHNG\_EN\_BIT = 0x0008, // USB/Link State Change Event Enable

### Set the Run/Stop bit

// Set the Run/Stop bit

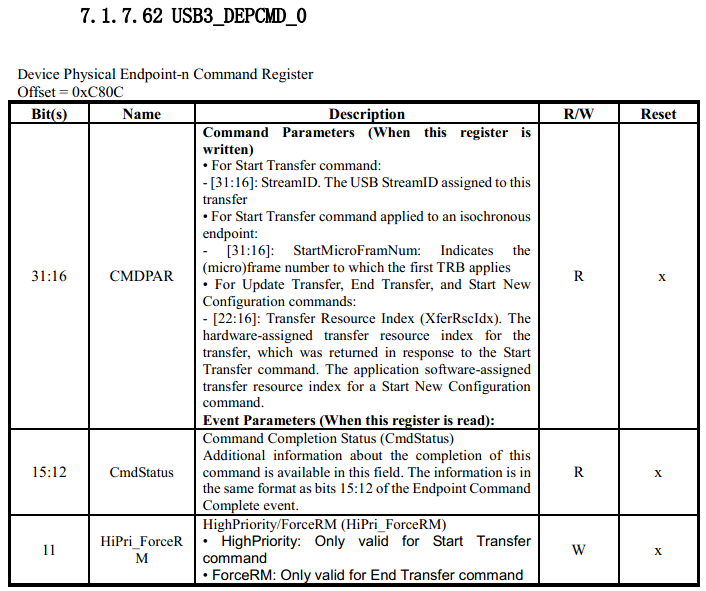
dwc\_modify\_reg32(&pcd->dev\_global\_regs->dctl,

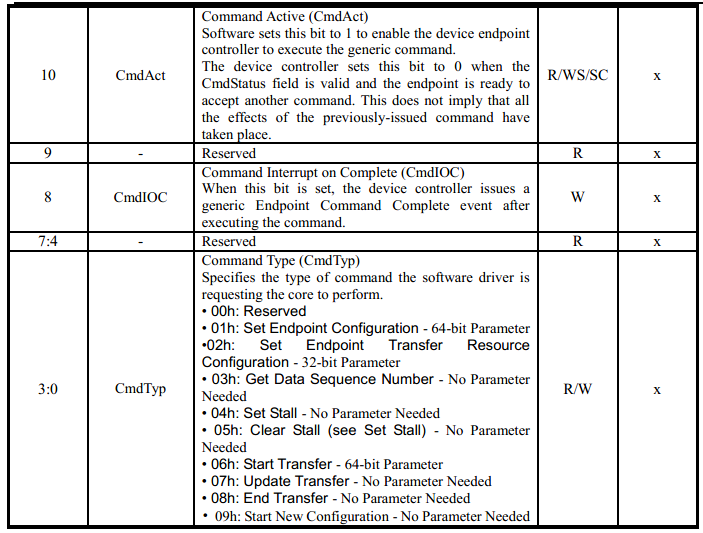
DWC\_DCTL\_RUN\_STOP\_BIT, DWC\_DCTL\_RUN\_STOP\_BIT);

DWC\_DCTL\_RUN\_STOP\_BIT = 0x80000000, // Run/Stop

DWC\_DCTL\_RUN\_STOP\_SHIFT = 31,

## Cmd





Ep0:

DEPCMDPAR2(0) = 0x00000000

DEPCMDPAR1(0) = 0x11042000 //Transfer Descriptor Address (Low)

DEPCMDPAR0(0) = 0x00000000 //Transfer Descriptor Address (High)

DEPCMD(0) = 0x0000000**6** // DWC\_EPCMD\_START\_XFER

Ep\_in:

DEPCMDPAR2(1) = 0x00000000

DEPCMDPAR1(1) = 0x11042000

DEPCMDPAR0(1) = 0x00000000

DEPCMD(1) = 0x000**1**000**6** // DWC\_EPCMD\_START\_XFER | Stream\_ID(1)

Ep\_out:

DEPCMDPAR2(5) = 0x00000000

DEPCMDPAR1(5) = 0x11049010

DEPCMDPAR0(5) = 0x00000000

DEPCMD(5) = 0x000**2**200**6** // DWC\_EPCMD\_START\_XFER | Stream\_ID(2)

### Depcmd[3:0]: Cmd Type

DWC\_EPCMD\_SET\_EP\_CFG = 1, /\*\* @< \*/

DWC\_EPCMD\_SET\_XFER\_CFG = 2, /\*\* @< \*/

DWC\_EPCMD\_CLR\_DATA\_SEQ = 3, /\*\* @< \*/

DWC\_EPCMD\_SET\_STALL = 4, /\*\* @< \*/

DWC\_EPCMD\_CLR\_STALL = 5, /\*\* @< \*/

DWC\_EPCMD\_START\_XFER = 6, /\*\* @< \*/

DWC\_EPCMD\_UPDATE\_XFER = 7, /\*\* @< \*/

DWC\_EPCMD\_END\_XFER = 8, /\*\* @< \*/

DWC\_EPCMD\_START\_NEW\_CFG = 9,

### dwc\_usb3\_dep\_startnewcfg() // DWC\_EPCMD\_START\_NEW\_CFG(9)

XferRscIdx = 0: 即将发送SET\_EP\_CFG命令配置endpoints 0 and 1；

XferRscIdx = 2: 即将发送SET\_EP\_CFG命令配置endpoints > 1；

int dwc\_usb3\_dep\_startnewcfg(dwc\_usb3\_pcd\_t \*pcd, uint32\_t XferRscIdx)

{

/\* Start the command \*/

dwc\_write\_reg32(&ep\_reg->**depcmd**, (**XferRscIdx**<< 16) |**DWC\_EPCMD\_START\_NEW\_CFG** |**DWC\_EPCMD\_ACT\_BIT**);

/\* Wait for command completion \*/

if (!dwc\_usb3\_handshake(pcd->usb3\_dev, &ep\_reg->**depcmd**, **DWC\_EPCMD\_ACT\_BIT**, **0**, 1000)) { }

return 0;

}

DWC\_EPCMD\_START\_NEW\_CFG = 9, // Command Type values: start new cfg

DWC\_EPCMD\_ACT\_BIT = 0x400, // Command Active

### dwc\_usb3\_dep\_cfg() // DWC\_EPCMD\_SET\_EP\_CFG(1)

不同的endpoint需要发送不同参数的START\_NEW\_CFG命令

Ep[0]:

depcfg0 = b[2:1] | b[13:3]

depcfg1 = b[8] |b[10]

Ep\_out[n]:

depcfg0 = b[2:1] | b[13:3] |

depcfg1 = b[26] | b[16] | b[8] |b[9]

Ep\_int[n]:

depcfg0 = b[2:1] | b[13:3] | b[21:17]

depcfg1 = b[26] | b[16] | b[8] |b[9] | b[25]

* DEPCMDPAR0:
  + b[0]: Reserved
  + b[2:1]: Endpoint Type values

DWC\_USB3\_EP\_TYPE\_CONTROL = 0, /\*\* @< \*/

DWC\_USB3\_EP\_TYPE\_ISOC = 1, /\*\* @< \*/

DWC\_USB3\_EP\_TYPE\_BULK = 2, /\*\* @< \*/

DWC\_USB3\_EP\_TYPE\_INTR = 3,

* + b[13:3]: Max Packet Size
    - EP0:

#define DWC\_MAX\_EP0\_SIZE\_SS 512

#define DWC\_MAX\_EP0\_SIZE\_HS 64

* + - EP[n]

#define DWC\_MAX\_PACKET\_SIZE\_SS 1024

#define DWC\_MAX\_PACKET\_SIZE\_HS 512

#define DWC\_MAX\_PACKET\_SIZE\_FS 64

* + b[16:14]: Reserved
  + b[21:17]: FIFO Number
* DEPCMDPAR1
  + b[4-0]: Interrupt number
  + b[7:5]: Reserved
  + b[8]: DWC\_EPCFG1\_XFER\_CMPL\_BIT, // XferComplete : Stream Completed
  + b[9]: DWC\_EPCFG1\_XFER\_IN\_PROG\_BIT, // XferInProgress : Stream In Progress
  + b[10]: DWC\_EPCFG1\_XFER\_NRDY\_BIT, // XferNotReady : Stream Not Ready
  + b[11]: DWC\_EPCFG1\_FIFOXRUN\_BIT, // Rx FIFO Underrun / Tx FIFO Overrun
  + b[12]: DWC\_EPCFG1\_SETUP\_PNDG\_BIT, // Back-to-Back Setup Packets Received
  + b[13]: DWC\_EPCFG1\_EPCMD\_CMPL\_BIT, // Endpoint Command Complete
  + b[16]: DWC\_EPCFG1\_BINTERVAL\_BITS, // Endpoint bInterval
  + b[24]: DWC\_EPCFG1\_STRM\_CAP\_BIT, // Endpoint Stream Capability
  + b[25]: DWC\_EPCFG1\_EP\_DIR\_BIT, // Endpoint Direction
  + b[26]: DWC\_EPCFG1\_EP\_NUM\_BITS, // Endpoint Number

int dwc\_usb3\_dep\_cfg(dwc\_usb3\_pcd\_t \*pcd , dwc\_usb3\_dev\_ep\_regs\_t \*ep\_reg, uint32\_t depcfg0, uint32\_t depcfg1)

{

dwc\_write\_reg32(&ep\_reg->**depcmdpar1**, **depcfg1**); // depcfg1

dwc\_write\_reg32(&ep\_reg->**depcmdpar0**, **depcfg0**); // depcfg0

/\* Start the command \*/

dwc\_write\_reg32(&ep\_reg->**depcmd**, **DWC\_EPCMD\_SET\_EP\_CFG** | **DWC\_EPCMD\_ACT\_BIT**);

/\* Wait for command completion \*/

if (!dwc\_usb3\_handshake(pcd->usb3\_dev, &ep\_reg->**depcmd**, **DWC\_EPCMD\_ACT\_BIT**, **0**, 1000)){}

return 0;

}

DWC\_EPCMD\_SET\_EP\_CFG = 1, // Command Type values: set ep cfg

### dwc\_usb3\_dep\_xfercfg() // DWC\_EPCMD\_SET\_XFER\_CFG(2)

int dwc\_usb3\_dep\_xfercfg(dwc\_usb3\_pcd\_t \*pcd, , dwc\_usb3\_dev\_ep\_regs\_t \*ep\_reg, uint32\_t depstrmcfg)

{

//**depstrmcfg** = 1; //**One stream**

dwc\_write\_reg32(&ep\_reg->**depcmdpar0**, **depstrmcfg**);

/\* Start the command \*/

dwc\_write\_reg32(&ep\_reg->**depcmd**,**DWC\_EPCMD\_SET\_XFER\_CFG** | **DWC\_EPCMD\_ACT\_BIT**);

/\* Wait for command completion \*/

if (!dwc\_usb3\_handshake(pcd->usb3\_dev, &ep\_reg->**depcmd**, **DWC\_EPCMD\_ACT\_BIT**, **0,** 1000)){}

return 0;

}

DWC\_EPCMD\_SET\_XFER\_CFG = 2 // Command Type values: set xfer cfg

### dwc\_usb3\_dep\_sstall() // DWC\_EPCMD\_SET\_STALL(4)

设置DWC\_EPCMD\_IOC\_BIT后，当执行CLR\_STALL后会产生中断

int dwc\_usb3\_dep\_sstall(dwc\_usb3\_pcd\_t \*pcd, dwc\_usb3\_dev\_ep\_regs\_t \*ep\_reg)

{

dwc\_write\_reg32(&ep\_reg->**depcmd**,**DWC\_EPCMD\_SET\_STALL** | DWC\_EPCMD\_ACT\_BIT | **DWC\_EPCMD\_IOC\_BIT**);

/\* Wait for command completion \*/

if (!dwc\_usb3\_handshake(pcd->usb3\_dev, &ep\_reg->depcmd, DWC\_EPCMD\_ACT\_BIT, 0, 1000)){}

return 0;

}

DWC\_EPCMD\_SET\_STALL = 4

DWC\_EPCMD\_IOC\_BIT = 0x100, // **Command Interrupt on Complete**

### dwc\_usb3\_dep\_cstall() // DWC\_EPCMD\_CLR\_STALL(5)

int dwc\_usb3\_dep\_cstall(dwc\_usb3\_pcd\_t \*pcd, dwc\_usb3\_dev\_ep\_regs\_t \*ep\_reg)

{

dwc\_write\_reg32(&ep\_reg->**depcmd**,**DWC\_EPCMD\_CLR\_STALL** | DWC\_EPCMD\_ACT\_BIT);

/\* Wait for command completion \*/

if (!dwc\_usb3\_handshake(pcd->usb3\_dev, &ep\_reg->depcmd, DWC\_EPCMD\_ACT\_BIT, 0, 1000)){}

return 0;

}

DWC\_EPCMD\_CLR\_STALL = 5,

### dwc\_usb3\_dep\_startxfer() // DWC\_EPCMD\_START\_XFER(6)

int dwc\_usb3\_dep\_startxfer(dwc\_usb3\_pcd\_t \*pcd, dwc\_usb3\_dev\_ep\_regs\_t \*ep\_reg, dwc\_dma\_t dma\_addr, uint32\_t stream\_or\_uf)

{

dwc\_write\_reg32(&ep\_reg->**depcmdpar1**,**dma\_addr** & 0xffffffffU);

dwc\_write\_reg32(&ep\_reg->**depcmdpar0**, 0);

/\* Start the command \*/

dwc\_write\_reg32(&ep\_reg->**depcmd**, (**stream\_or\_uf** << (16)) | **DWC\_EPCMD\_START\_XFER** | DWC\_EPCMD\_ACT\_BIT);

/\* Wait for command completion \*/

if (!dwc\_usb3\_handshake(pcd->usb3\_dev, &ep\_reg->depcmd, DWC\_EPCMD\_ACT\_BIT, 0, 1000)){}

depcmd = dwc\_read\_reg32(&ep\_reg->**depcmd**);

return (depcmd >> (16)) & (**DWC\_EPCMD\_XFER\_RSRC\_IDX\_BITS** >> (16));

}

DWC\_EPCMD\_START\_XFER = 6,

DWC\_EPCMD\_XFER\_RSRC\_IDX\_BITS = 0x007f0000, // Transfer Resource Index (output)

### dwc\_usb3\_dep\_updatexfer() // DWC\_EPCMD\_UPDATE\_XFER(7)

int dwc\_usb3\_dep\_updatexfer(dwc\_usb3\_pcd\_t \*pcd, dwc\_usb3\_dev\_ep\_regs\_t \*ep\_reg, uint32\_t tri)

{

/\* Start the command \*/

dwc\_write\_reg32(&ep\_reg->**depcmd**,(**tri** << (16)) | **DWC\_EPCMD\_UPDATE\_XFER** | DWC\_EPCMD\_ACT\_BIT);

/\* Wait for command completion \*/

if (!dwc\_usb3\_handshake(pcd->usb3\_dev, &ep\_reg->depcmd, DWC\_EPCMD\_ACT\_BIT, 0, 10000)){}

return 0;

}

DWC\_EPCMD\_UPDATE\_XFER = 7,

### dwc\_usb3\_dep\_endxfer() // DWC\_EPCMD\_END\_XFER(8)

int dwc\_usb3\_dep\_endxfer(dwc\_usb3\_pcd\_t \*pcd, dwc\_usb3\_dev\_ep\_regs\_t \*ep\_reg, uint32\_t tri, void \*condition)

{

/\* Start the command. \*/

dwc\_write\_reg32(&ep\_reg->**depcmd**,

(**tri** << (16)) | **DWC\_EPCMD\_END\_XFER** | DWC\_EPCMD\_ACT\_BIT | **DWC\_EPCMD\_HP\_FRM\_BIT**);

/\* Wait for command completion \*/

if (!dwc\_usb3\_handshake(pcd->usb3\_dev, &ep\_reg->depcmd, DWC\_EPCMD\_ACT\_BIT, 0, 10000)){}

return 0;

}

DWC\_EPCMD\_END\_XFER = 8,

DWC\_EPCMD\_HP\_FRM\_BIT = 0x800, // **High Priority / Force RM Bit**

### dwc\_usb3\_ep0\_activate() //ep0 active

void dwc\_usb3\_ep0\_activate(dwc\_usb3\_pcd\_t \*pcd)

{

//Issue "DEPCFG" command to EP0-OUT

ep\_reg = &pcd->**out\_ep\_regs[0]**;

doepcfg0 = DWC\_USB3\_EP\_TYPE\_**CONTROL**<< (1) | DWC\_MAX**\_EP0\_SIZE\_HS**<< (3)**;**

doepcfg1 = DWC\_EPCFG1\_**XFER\_CMPL\_BIT** | DWC\_EPCFG1\_**XFER\_NRDY\_BIT;**

dwc\_usb3\_dep\_startnewcfg(pcd, ep\_reg, 0); // DWC\_EPCMD\_START\_NEW\_CFG

dwc\_usb3\_dep\_cfg(pcd, ep\_reg, doepcfg0, doepcfg1); // DWC\_EPCMD\_SET\_EP\_CFG

dwc\_usb3\_dep\_xfercfg(pcd, ep\_reg, 1); // DWC\_EPCMD\_SET\_XFER\_CFG

//Issue "DEPCFG" command to EP0-IT

ep\_reg = &pcd->**in\_ep\_regs[0]**;

diepcfg0 = DWC\_USB3\_EP\_TYPE\_**CONTROL**<< (1) | DWC\_MAX\_**EP0\_SIZE\_HS** << (3);

diepcfg1 = DWC\_EPCFG1\_**XFER\_CMPL\_BIT** | DWC\_EPCFG1\_**XFER\_NRDY\_BI**T | DWC\_EPCFG1\_**EP\_DIR\_BIT**;

dwc\_usb3\_dep\_cfg(pcd, ep\_reg, diepcfg0, diepcfg1); // DWC\_EPCMD\_SET\_EP\_CFG

dwc\_usb3\_dep\_xfercfg(pcd, ep\_reg, 1); // DWC\_EPCMD\_SET\_XFER\_CFG

}

DWC\_USB3\_EP\_TYPE\_CONTROL = 0,

#define DWC\_MAX\_EP0\_SIZE\_HS 64

DWC\_EPCFG1\_XFER\_CMPL\_BIT = 0x00000100, // Stream Completed

DWC\_EPCFG1\_XFER\_NRDY\_BIT = 0x00000400, // Stream Not Ready

DWC\_EPCFG1\_EP\_DIR\_BIT = 0x02000000, // Endpoint Direction

### dwc\_usb3\_ep\_activate()

void dwc\_usb3\_ep\_activate(dwc\_usb3\_pcd\_t \*pcd, dwc\_usb3\_pcd\_ep\_t \*ep)

{

if (ep->**dwc\_ep.is\_in**) {

ep\_reg = &pcd->**in\_ep\_regs**[ep->dwc\_ep.num];

**ep\_index\_num** = **ep->dwc\_ep.num** \* 2 + 1;

}

else {

ep\_reg = &pcd->**out\_ep\_regs**[ep->dwc\_ep.num];

**ep\_index\_num** = **ep->dwc\_ep.num** \* 2;

}

depcfg0 = ep->**dwc\_ep.type** << (1) | ep->**dwc\_ep.maxpacket** << (3);

if (pcd->usb3\_dev->core\_params->burst)

depcfg0 |= 0 << DWC\_EPCFG0\_BRSTSIZ\_SHIFT;

depcfg1 = ep->**dwc\_ep.num** << (26) | ep->**dwc\_ep.intvl** << (16)

| DWC\_EPCFG1\_**XFER\_CMPL\_**BIT | DWC\_EPCFG1\_**XFER\_IN\_PROG**\_BIT;

if (ep->**dwc\_ep.is\_in**) {

depcfg0 |= ep->**dwc\_ep.tx\_fifo\_num** << (17);

depcfg1 |= DWC\_EPCFG1\_E**P\_DIR\_BIT;**

}

dwc\_usb3\_dep\_cfg(pcd, ep\_reg, depcfg0, depcfg1);

dwc\_usb3\_dep\_xfercfg(pcd, ep\_reg, **1**);

/\* Enable EP in DALEPENA reg \*/

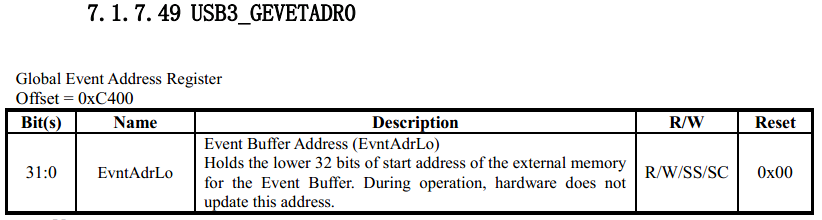
dalepena |= 1 << **ep\_index\_num**;

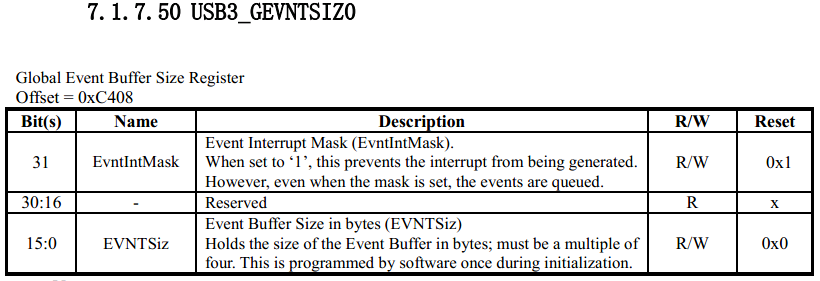
dwc\_write\_reg32(&pcd->dev\_global\_regs->**dalepena**, dalepena);

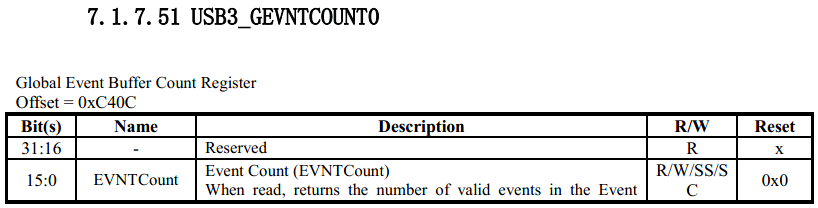
ep->dwc\_ep.stall\_clear\_flag = 0;

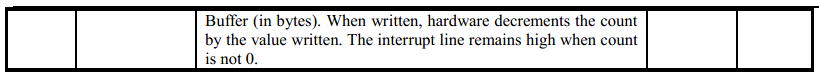
}

## Event









GEVNTADRLO(0) = 0x11040000

GEVNTADRHI(0) = 0x00000000

GEVNTSIZ(0) = 0x00000100

GEVNTCOUNT(0) = 0x00000000

GEVNTADRHI(0) | GEVNTADRLO(0)表示Events存放的地址；

GEVNTSIZ(0) = 0x00000100 \* 4 = 1024B表示该地址长度；每个event占用空间为4B；

GEVNTCOUNT(0)表示当前dma中存放的待处理event集合；

* Event:

Event:b[0] :1= non-endpoint event, 0= endpoint event

* + non-endpoint event:

Event:b[1] :1= device event, 0= core event

* + - device event

Event:b[11-8] :

DWC\_DEVT\_DISCONN = 0,

DWC\_DEVT\_USBRESET = 1,

DWC\_DEVT\_CONNDONE = 2,

DWC\_DEVT\_ULST\_CHNG = 3,

DWC\_DEVT\_WKUP = 4,

DWC\_DEVT\_EOPF = 6,

DWC\_DEVT\_SOF = 7,

DWC\_DEVT\_ERRATICERR = 9,

DWC\_DEVT\_CMD\_CMPL = 10,

DWC\_DEVT\_OVERFLOW = 11,

DWC\_DEVT\_VNDR\_DEV\_TST\_RCVD = 12,

DWC\_DEVT\_INACT\_TIMEOUT\_RCVD = 13,

* + endpoint event

Event:b[5-1] : Out EPs are even, In EPs are odd

Event:b[9-6] :

DWC\_DEPEVT\_XFER\_CMPL = 1,

DWC\_DEPEVT\_XFER\_IN\_PROG = 2,

DWC\_DEPEVT\_XFER\_NRDY = 3,

DWC\_DEPEVT\_FIFOXRUN = 4,

DWC\_DEPEVT\_STRM\_EVT = 6,

DWC\_DEPEVT\_EPCMD\_CMPL = 7,

### init\_eventbuf()

static void init\_eventbuf(dwc\_usb3\_device\_t \*dev)

{

Bufno = 0;

// Up to 32 Event Buffers are supported by the hardware, but we only use 1

usb3\_dev->event\_buf[0] = allocate\_eventbuf(usb3\_dev,DWC\_EVENT\_BUF\_SIZE);

usb3\_dev->event\_buf\_dma[0] = (dwc\_dma\_t)(usb3\_dev->event\_buf[0]);

usb3\_dev->event\_ptr[0] = usb3\_dev->event\_buf[0];

dma\_addr = usb3\_dev->**event\_buf\_dma**[0];

size = DWC\_EVENT\_BUF\_SIZE = 1024\*8; //dwords

dwc\_write\_reg32(&dev->core\_global\_regs->geventbuf[bufno].**geventadr\_lo**, **dma\_addr** & 0xffffffffU);

dwc\_write\_reg32(&dev->core\_global\_regs->geventbuf[bufno].**geventadr\_hi**, 0);

dwc\_write\_reg32(&dev->core\_global\_regs->geventbuf[bufno].**geventsiz**, **size** << 2);

dwc\_read\_reg32(&dev->core\_global\_regs->geventbuf[bufno].**geventsiz**);

dwc\_write\_reg32(&dev->core\_global\_regs->geventbuf[bufno].**geventcnt**, 0);

}

#define DWC\_EVENT\_BUF\_SIZE 1024\*8 // size in dwords

### get\_eventbuf\_count()

//如果**geventcnt** =0xffff 或者**geventcnt** >= DWC\_EVENT\_BUF\_SIZE \* 4 🡪 Bad event count

static int get\_eventbuf\_count(dwc\_usb3\_device\_t \*dev)

{

bufno = 0;

cnt = dwc\_read\_reg32(&dev->core\_global\_regs->geventbuf[bufno].**geventcnt**);

return cnt & DWC\_EVENTCNT\_CNT\_BITS;

}

DWC\_EVENTCNT\_CNT\_BITS = 0x0000ffff,

### get\_eventbuf\_event()

static uint32\_t get\_eventbuf\_event(dwc\_usb3\_device\_t \*dev, int bufno, int size)

{

bufno = 0;

size = DWC\_EVENT\_BUF\_SIZE; // 1024\*8 dwords

uint32\_t **event = \*dev->event\_ptr[bufno]++**;

if (dev->event\_ptr[bufno] >= dev->event\_buf[bufno] + size) dev->event\_ptr[bufno] = dev->event\_buf[bufno];

return event;

}

### update\_eventbuf\_count()

//执行get\_eventbuf\_event()后，需要执行update\_eventbuf\_count()来更新event的值

static void update\_eventbuf\_count(dwc\_usb3\_device\_t \*dev)

{

Bufno = 0;

Cnt = 4; //dword

dwc\_write\_reg32(&dev->core\_global\_regs->geventbuf[bufno].geventcnt, cnt);

}

### device event

#### DWC\_DEVT\_DISCONN

static uint32\_t calc\_num\_in\_eps(dwc\_usb3\_device\_t \*usb3\_dev)

{

uint32\_t num\_in\_eps = (usb3\_dev->hwparams3 >> DWC\_HWPARAMS3\_NUM\_IN\_EPS\_SHIFT) &

(DWC\_HWPARAMS3\_NUM\_IN\_EPS\_BITS >> DWC\_HWPARAMS3\_NUM\_IN\_EPS\_SHIFT);

return num\_in\_eps - 1;

}

# End