MediaTek 2020
The New VPU iDMA Driver

v1.0

About iDMA

- Single-channel integrated DMA engine
  - Supports data movement between
    - External memory (system DRAM) and data RAM (local SRAM, DMEM)
    - Data RAM to data RAM (local SRAM, DMEM)
- All iDMA operations are controlled by VPU through
  - iDMA control registers
  - DMA descriptors, which are data structures residing in data RAM
    - Descriptors will be processed in a linear order
- The transfer granularity is byte
  - iDMA can automatically shift and align the data accordingly

Limitation of iDMA

- iDMA does not support:
  - External-to-external memory transfers (system DRAM to system DRAM)
  - Data movement to/from the I/O port, registers, cache, or other non-memory devices
- Data transfers should not cross the data RAM boundary
- The source region and destination region should not overlap due to limited buffering capability
  - May cause data corruption
- Each descriptor is expected to describe a range that entirely fits in an Memory Protection Unit (MPU) entry
  - Data transfers should not cross the MPU region
  - A descriptor should be aligned to a 32-bit boundary (for JUMP?)
- iDMA only supports Little Endian configuration

About iDMA descriptors

- Descriptors are stored in the data RAM (local SRAM)
  - A special 1-word (32-bit) JUMP command can be placed between descriptors to alter the execution flow
- There are two types of descriptors:
  - A 1D transfer is described by a 128-bit (16-byte) descriptor
  - A 2D transfer is described by a 256-bit (32-byte) descriptor

```
/* 1D descriptor structure */
struct idma_desc_struct {
  uint32_t  control;
  void*     src;
  void*     dst;
  uint32_t  size; // number of bytes
};
```

```
/* 2D descriptor structure */
struct idma_2d_desc_struct {
  uint32_t  control;
  void*     src;
  void*     dst;
  uint32_t  size; // bytes in a row
  uint32_t  src_pitch;
  uint32_t  dst_pitch;
  uint32_t  nrows;
  uint32_t  dummy;
};
```

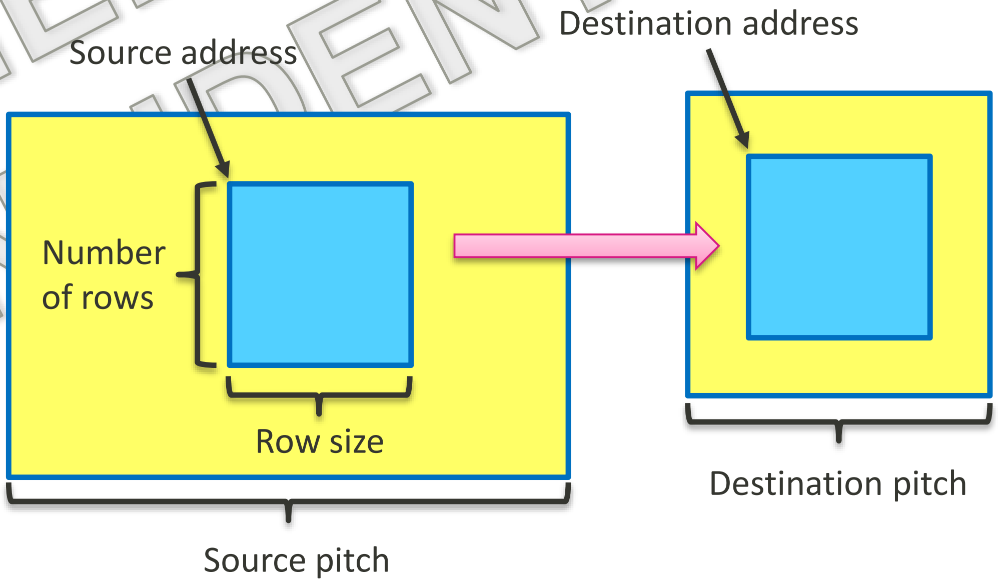
MTK iDMA driver only supports 2D!

Control word of descriptor

Field	Index	Definition	Setting in MTK driver
(D)escriptor	2:0	Type of the descriptor	3’b111: 2D descriptor
Reserved	8:3	Don’t-cares	N/A
Ps	9	MPU privilege for source access	Supervisor-Privilege
Reserved	10	Don’t-cares	N/A
Pd	11	MPU privilege for destination access	Supervisor-Privilege
QoS	15:12	Priority when DMA accesses AXI	High-priority
Reserved	28:16	Don’t-cares	N/A
Twait	29	Trigger Wait	Disable
Trig	30	Trigger	Disable
I	31	Interrupt on successfully completion	Disable

About 2D descriptor

```
/* 2D descriptor structure */
struct idma_2d_desc_struct {
  uint32_t  control;
  void*     src;
  void*     dst;
  uint32_t  size; // bytes in a row
  uint32_t  src_pitch;
  uint32_t  dst_pitch;
  uint32_t  nrows;
  uint32_t  dummy;
};
```



iDMA registers

Register	Supervisor Address	Accessing mode	Description
Settings	0x00110000	Read-write	iDMA setting register
Timeout	0x00110004	Read-write	Timeout threshold
DescStartAdrs	0x00110008	Read-write	Descriptor start address (the lowest two bits are hardwired to 0)
NumDesc	0x0011000c	Read-only	The number of descriptors to process
NumDescIncr	0x00110010	Write-only	Increment to the number of descriptors
Control	0x00110014	Read-write	iDMA control register
Privilege	0x00110018	Read-write	iDMA privilege register
Status	0x00110040	Read-only	iDMA status register
DescCurrAdrs	0x00110044	Read-only	The current descriptor address
DescCurrType	0x00110048	Read-only	The current descriptor type
SrcAdrs	0x0011004c	Read-only	Source address from which iDMA is reading from
DestAdrs	0x00110050	Read-only	Destination address to which iDMA is writing to

iDMA status register

31 18	17 5	4	3	2 0
Error Codes	Reserved	HT		RUN

RUN	Definition
3’b000	IDLE
3’b001	STANDBY
3’b010	BUSY
3’b011	DONE
3’b100	HALT
3’b101	ERROR

Error Codes
[31]: Fetch address error
[30]: Fetch data error
[29]: Read address error
[28]: Read data error
[27]: Write address error
[26]: Write data error
[25]: Timeout
[24]: Trigger Overflow
[23]: NumDesc overflow
[22]: Descriptor: unknown command
[21]: Descriptor: unsupported transfer direction
[20]: Descriptor: bad parameters
[19]: Descriptor: null address
[18]: Descriptor: privilege violation

Error code details

Error Bit	Error Type	Causes
31	Fetch address error	The fetch address is not in the data RAM region, fetch access deny, fetch address overrun-mmu-region, etc.
30	Fetch data error	Uncorrectable fetch data error.
29	Read address error	Read target is changing during a descriptor run., e.g. going from inside of one data RAM to outside, or going from AXI region to non-AXI region. AXI address error, read access deny, read address overrun-mmu-region, etc.
28	Read data error	Uncorrectable read data error, AXI data error, etc.
27	Write address error	Write target is changing during a descriptor run., e.g. going from inside of one data RAM to outside, or going from AXI region to non-AXI region, AXI address error, write access deny, write address overrun-mmu-region, etc.
26	Write data error	Uncorrectable data error during write, AXI data error, etc.
25	Timeout	Descriptor timeout.
24	TriggerOverflow	Asserted when HaveTrigger is set and another TrigIn_iDMA is asserted. In such cases, the trigger overflows.
23	NumDesc overflow	NumDesc overflows.
22	Descriptor: unknown command	Unknown descriptor control command.
21	Descriptor: unsupported transfer direction	Unsupported transfer direction. The supported transfer directions are: AXI to a data RAM, a data RAM to AXI, or a data RAM to a data RAM.
20	Descriptor: bad parameters	Zero or negative number of rows, zero or negative row bytes, or negative source/destination pitch, etc.
19	Descriptor: null address	A NULL address is used in a descriptor.
18	Descriptor: privilege violation	Privilege violation; if user-privilege iDMA attempts to run a supervisor-privilege descriptor.

2020 MTK iDMA driver

MTK iDMA driver interface	Brief
init(void)	Reset control data & iDMA HW, and always return MTRUE
uninit(void)	Always return MTRUE
config(void *pSrc, void *pDst, tm_dma_direction direction)	Always return MTRUE
load(void *psrc, void *pdst, uint32_t srcPitchBytes, uint32_t dstPitchBytes, uint32_t numRows, uint32_t numBytesPerRow, uint32_t interruptOnCompletion) The first argument of “xv_pdmaObject pdmaObj” was removed	Add a 2D descriptor to transfer data from DRAM to DMEM, return MTRUE if success, and return MFALSE if descriptor region was full
store(void *psrc, void *pdst, uint32_t srcPitchBytes,uint32_t dstPitchBytes, uint32_t numRows, uint32_t numBytesPerRow, uint32_t interruptOnCompletion) The first argument of “xv_pdmaObject pdmaObj” was removed	Add a 2D descriptor to transfer data from DMEM to DRAM, return MTRUE if success, and return MFALSE if descriptor region was full
copy(void *psrc, void *pdst, uint32_t srcPitchBytes, uint32_t dstPitchBytes, uint32_t numRows, uint32_t numBytesPerRow, uint32_t interruptOnCompletion)	Copy data from psrc to pdst by VPU, and always return MTRUE
start(void) The arguments of “MUINT32 sDesc” and “MUINT32 eDesc” were removed	Enable iDMA, it supports consecutive two “start()”s, and always return MTRUE
stop(void)	Always return MTRUE
stall(void)	Always return MTRUE
waitDone(void)	Return MTRUE if iDMA DONE, return MFALSE if iDMA ERROR, and reset iDMA HW
isDone(void)	Always return MTRUE
align_check(xv_pdmaObject pdmaObj, MUINT8 *psrc,MUINT8 *pdst, MUINT32 srcPitchBytes, MUINT32 dstPitchBytes, MUINT32 numRows, MUINT32 numBytesPerRow, tm_dma_direction direction)	Legacy unused API, removed
get_alignment(_IN_ xv_pdmaObject pdmaObj, _IN_ MUINT32 numBytesPerRow, _OUT_ MUINT32 *rowByteAlign, _OUT_ MUINT32 *sysMemAlign, _OUT_ MUINT32 *SMemAlign)	Legacy unused API, removed

iDMA usage example

```
#include "dmaif.h"
...
int ret;
TMDMA *pDMACtrl = NULL;
...
pDMACtrl = dmaif_getDMACtrl(); 1
pDMACtrl->init(); 2
...
ret = pDMACtrl->load(...); // or store(...) 3
if (ret == MFALSE) {
   // Error handling
}
pDMACtrl->start(); 4-1
...
ret = pDMACtrl->load(...); // or store(...)
if (ret == MFALSE) {
   // Error handling
}
pDMACtrl->start(); 4-2
ret = pDMACtrl->waitDone(); 5
if (ret == MFALSE) {
   // Error handling
}
```

Support consecutive two “start()”s for early start of iDMA transfer!

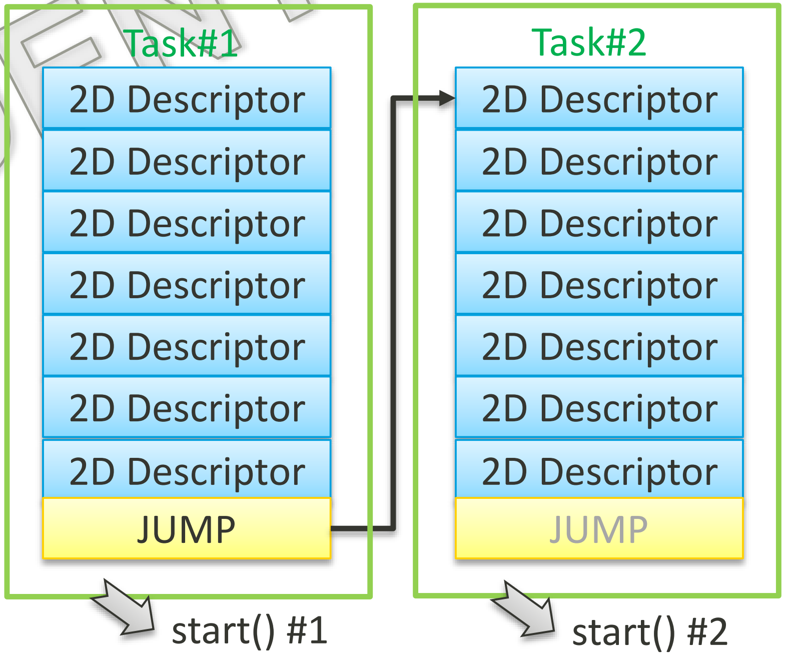
Consecutive starts

- At most 7 load()/store() in each start()
- Support consecutive two start()s
  - Mix up task and fixed buffer mode

```
enum DL_DMA_DESC_IDX {
...
DL_DMA_DESC_IDX_MAX = 16
...
}
#define MULTITASK_NUM 2

static volatile idma_2d_desc_t
task[MULTITASK_NUM][DL_DMA_DESC_IDX_MAX/MU
LTITASK_NUM] __attribute__ ((aligned(4),
section(".dram1_2.data")));
```

16 x 32 Bytes = 512 Bytes
for descriptors @ local memory



Performance gain in sDSP UT

Codebase: alps-mp-q0_mp2-V1_6
Phone load: k6885v1_64_tee_geniezone_svp_userdebug
Setting: Fix Freq. & cores + disable UART log

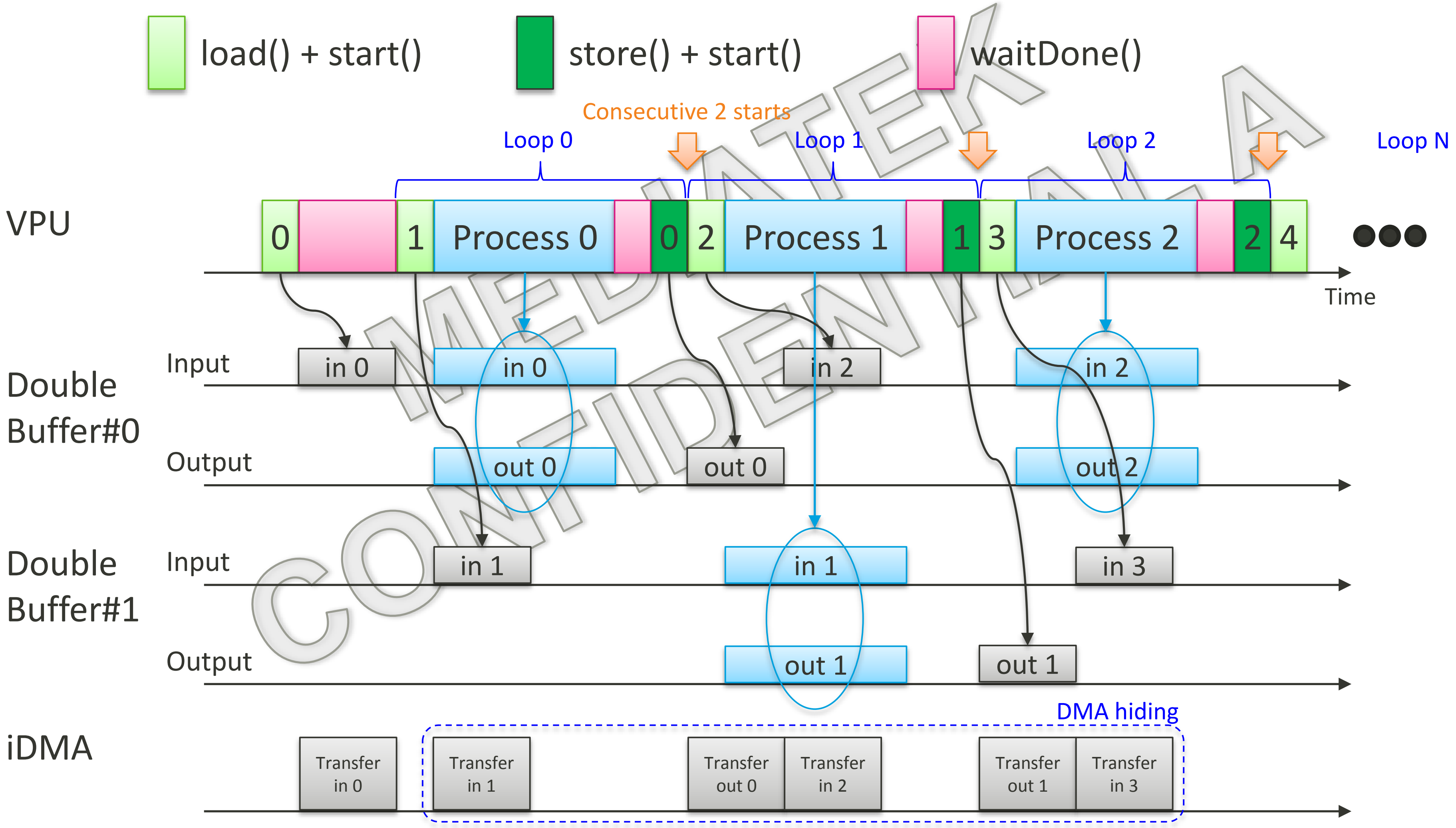
@GZ Log	Legacy	Revised
1	2487	866
2	2491	859
3	2488	859
4	2483	860
5	2484	861
6	2488	859
7	2484	863
8	2488	863
9	2502	861
10	2486	864
Average (us)	2488.1	861.5

About 2.8x speed up!

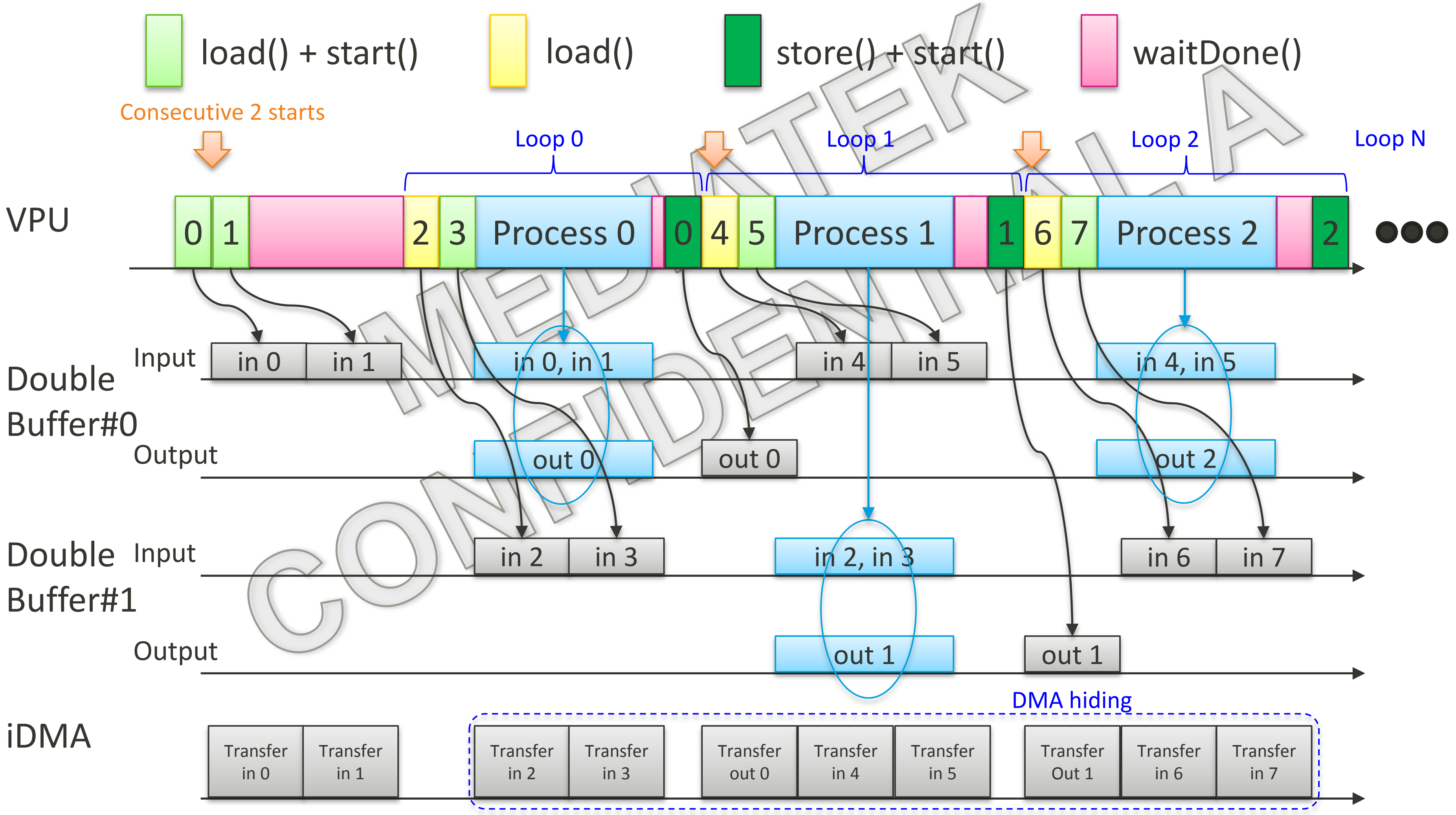
@sDSP	Legacy	Revised
1	1615862	373605
2	1716880	373539
3	1619989	368351
4	1718442	368057
5	1613857	366867
6	1715705	366611
7	1615539	368746
8	1713737	368432
9	1614747	368383
10	1713523	368139
11	1619105	368731
12	1716967	368701
13	1615438	370562
14	1713558	370618
15	1615000	371163
16	1715031	371437
17	1615282	367916
18	1717174	367674
19	1615512	372026
20	1715802	371647
Average (cycles)	1665858	369560.3

About 4.5x speed up!

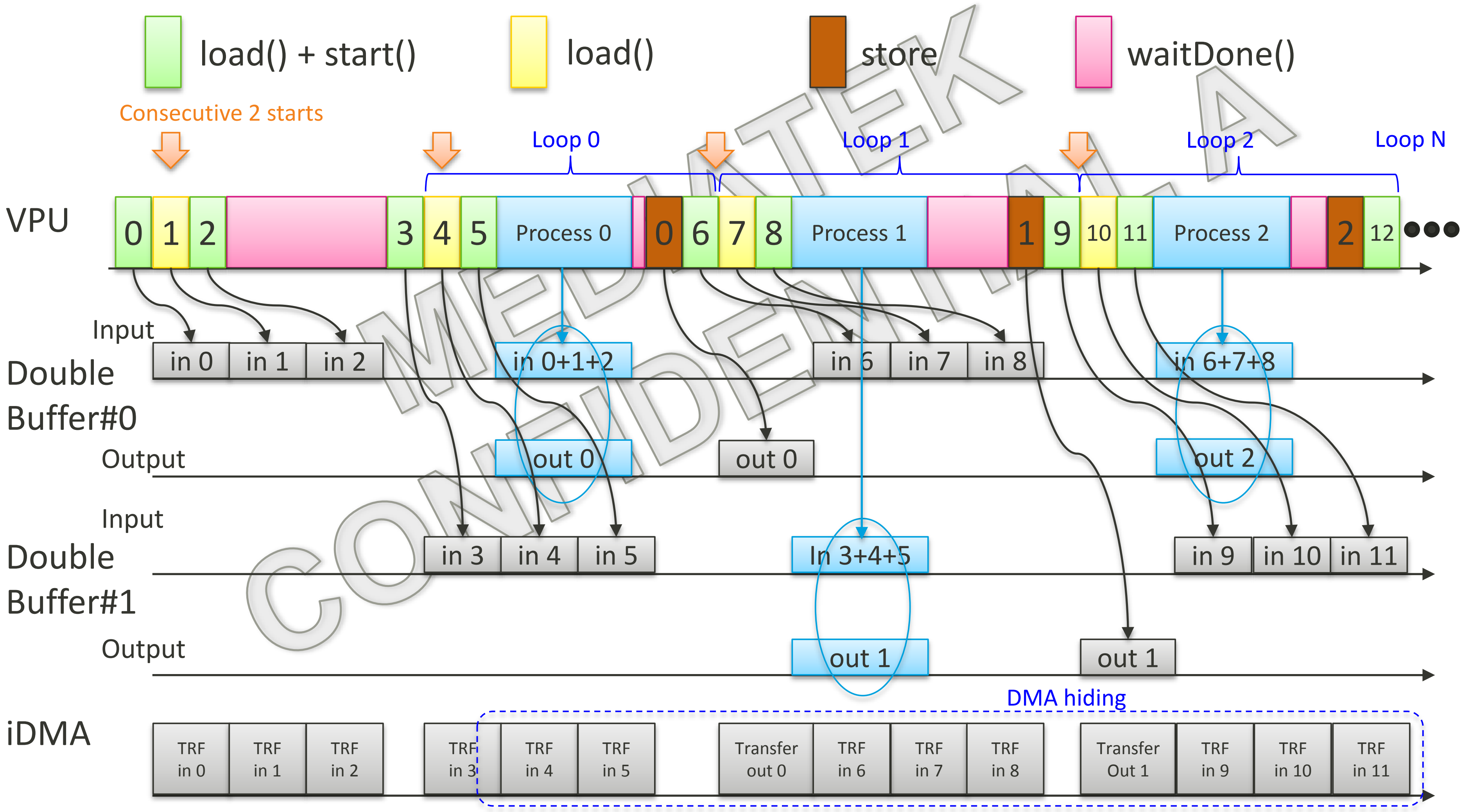
About double buffering (1-in, 1-out)



About double buffering (2-in, 1-out)



About double buffering (3-in, 1-out)



MEDIATEK

everyday genius