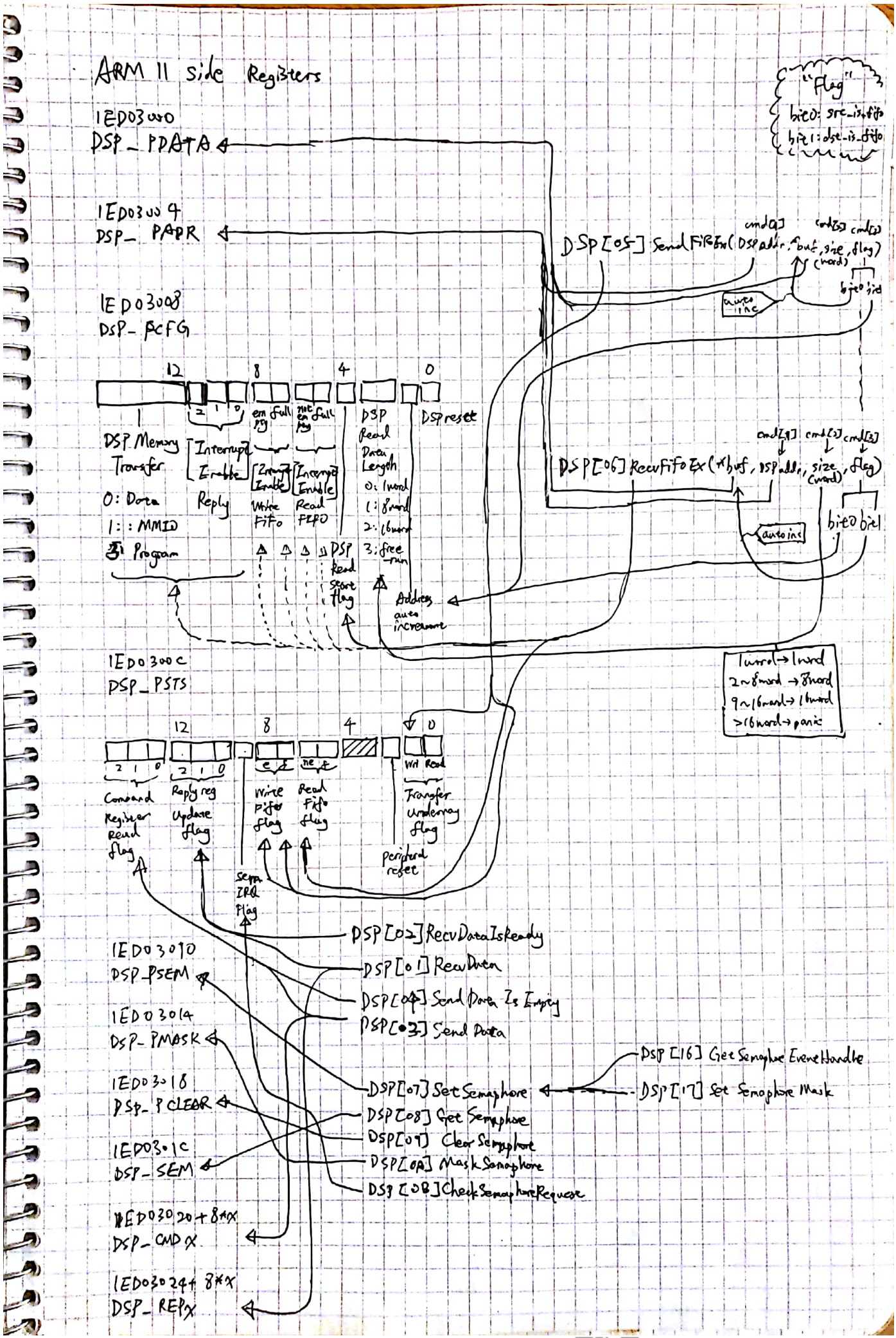
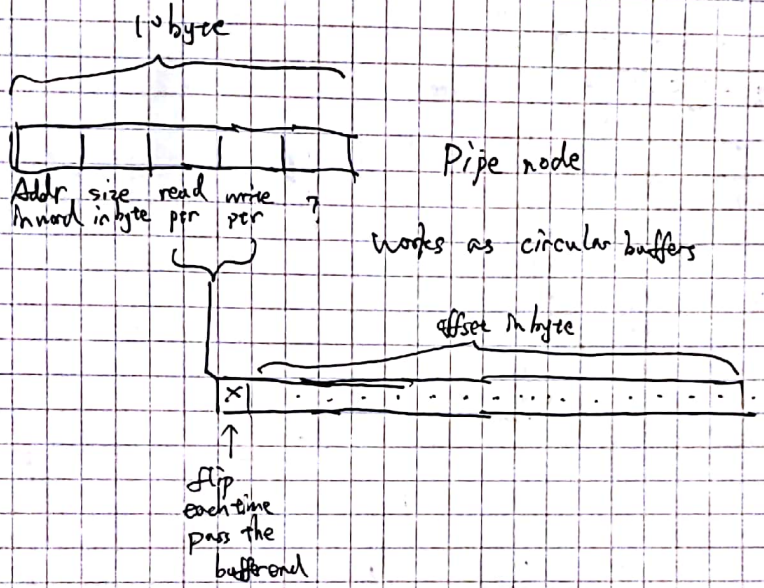
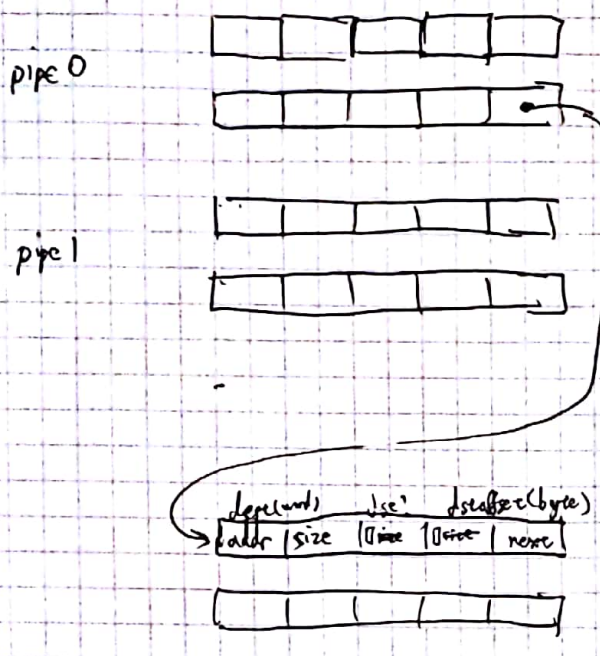
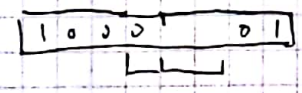
3PS DSP Note

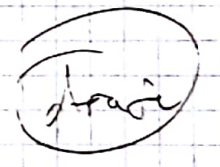

ARM 11 side Registers
1ED03000
DSP_PDATA
1ED03004
DSP_PAPR
1ED03008
DSP_PCFG
"flag"
bit0: src_is_fifo
bit1: dst_is_fifo
DSP[05] SendFifoEx(DSPaddr, buf, size, flag)
(word)
auto inc
bit0 bit1
DSP[06] RecvFifoEx(*buf, DSPaddr, size (word), flag)
auto inc
DSP Memory Transfer
0: Data
1: MMU
Program
Interrupt Enable
Reply
Write FIFO
Read FIFO
DSP Read Data Length
0: 1word
1: 8word
2: 16word
3: free run
DSP reset
DSP read store flag
Address auto increment
1word→1word
2~8word→8word
9~16word→16word
>16word→panic
1ED0300C
DSP_PSTS
Command Register Read flag
Reply reg Update flag
Write FIFO flag
Read FIFO flag
Transfer underway flag
peripheral reset
Semaphore IRQ Flag
DSP[02] RecvDataIsReady
DSP[01] RecvData
DSP[04] SendData Is Empty
DSP[03] SendData
1ED03010
DSP_PSEM
1ED03014
DSP_PMASK
1ED03018
DSP_PCLEAR
1ED0301C
DSP_SEM
DSP[16] GetSemaphoreEventHandle
DSP[07] SetSemaphore
DSP[17] SetSemaphoreMask
DSP[08] GetSemaphore
DSP[09] ClearSemaphore
DSP[0A] MaskSemaphore
DSP[0B] CheckSemaphoreRequest
1ED03020+8*X
DSP_CMDX
1ED03024+8*X
DSP_REPX



draft





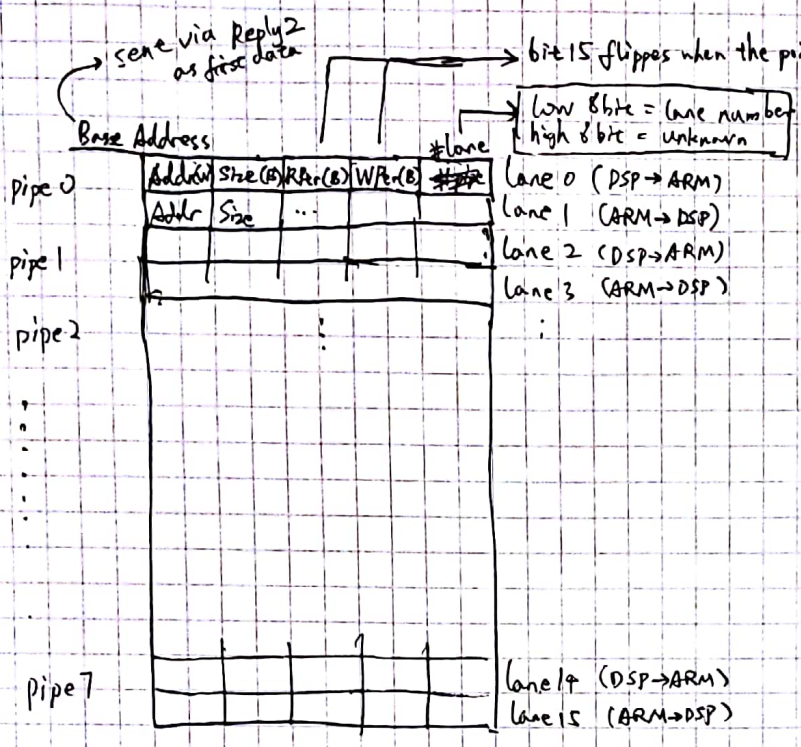


R4+ FFFFFFFC - FF00FFFF < 0000

R4+ FFFFFFFF < FF000000

Pipe protocol

Pipe pores in DSP memory



Each lane is a circular buffer

DSP[0D] WriteProcessPipe: goes to ARM→DSP lanes
if buffer is full, SendData(register=2, data=#lane) is called and then block until all data sent

DSP[0E] ReadPipe: if buffer is empty, SendData(register=2, data=#lane) is called and then block until all recv

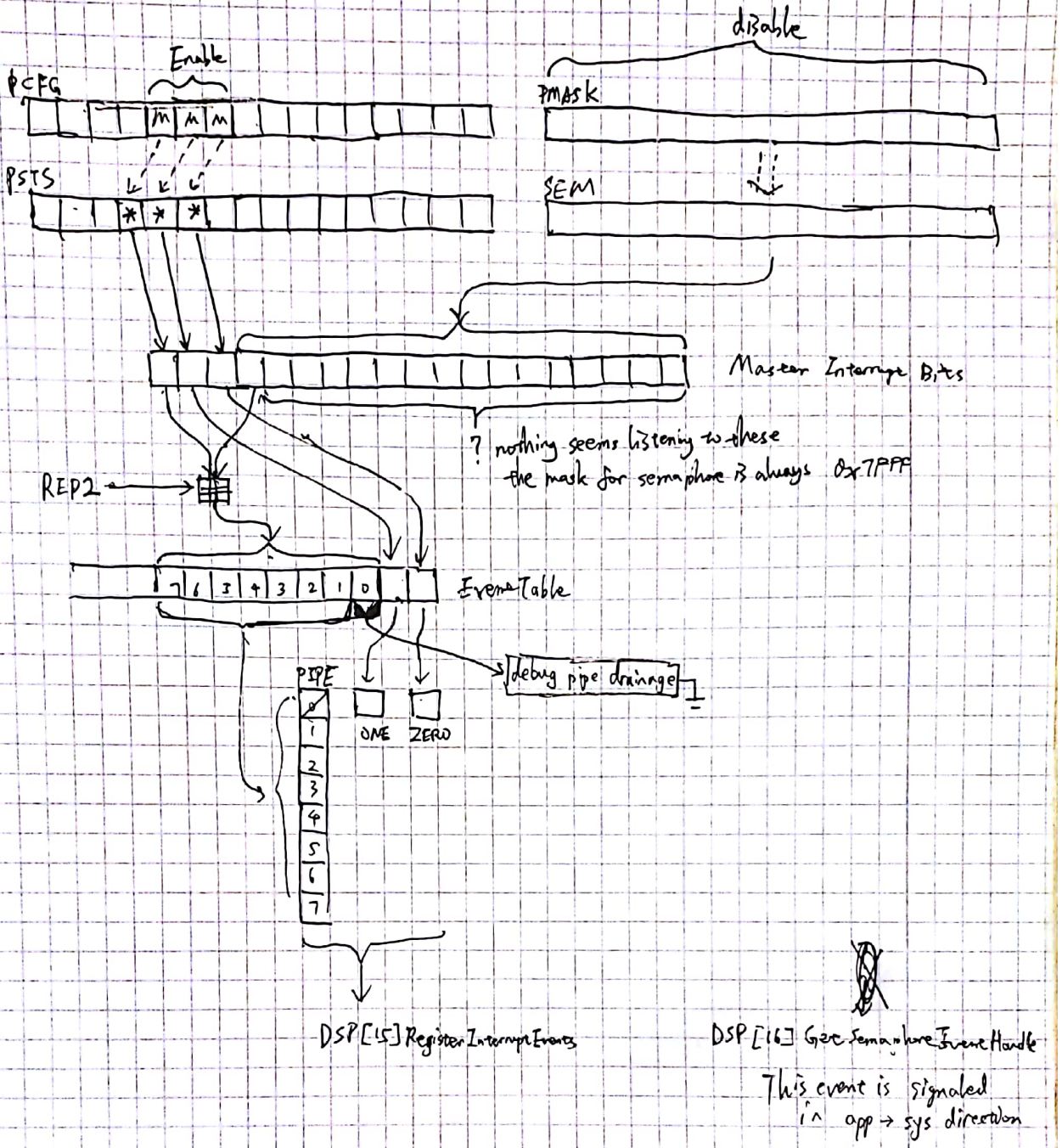
DSP[0F] GetPipeReadableSize: Gets the effective difference WPtr - RPtr

dsp[10] ReadPipeIfPossible: combination of [0F] + [0E]. non-blocking

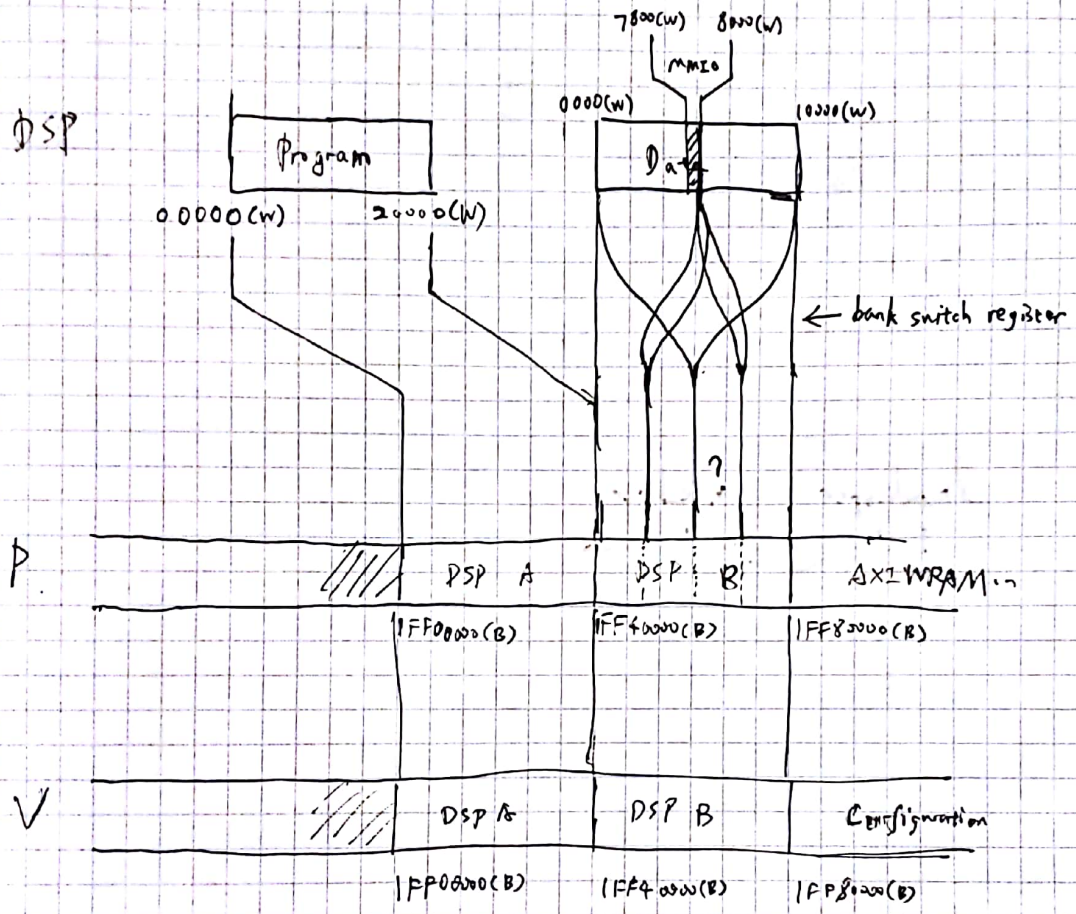
All pipe functions build on byte FIFO, which builds on word FIFO
DSP[05,06]

ReadPipe functions accept "peer" to specify one of the two lanes. But read on ARM→DSP lane would cause strange error?!
[0E, 0F, 10]

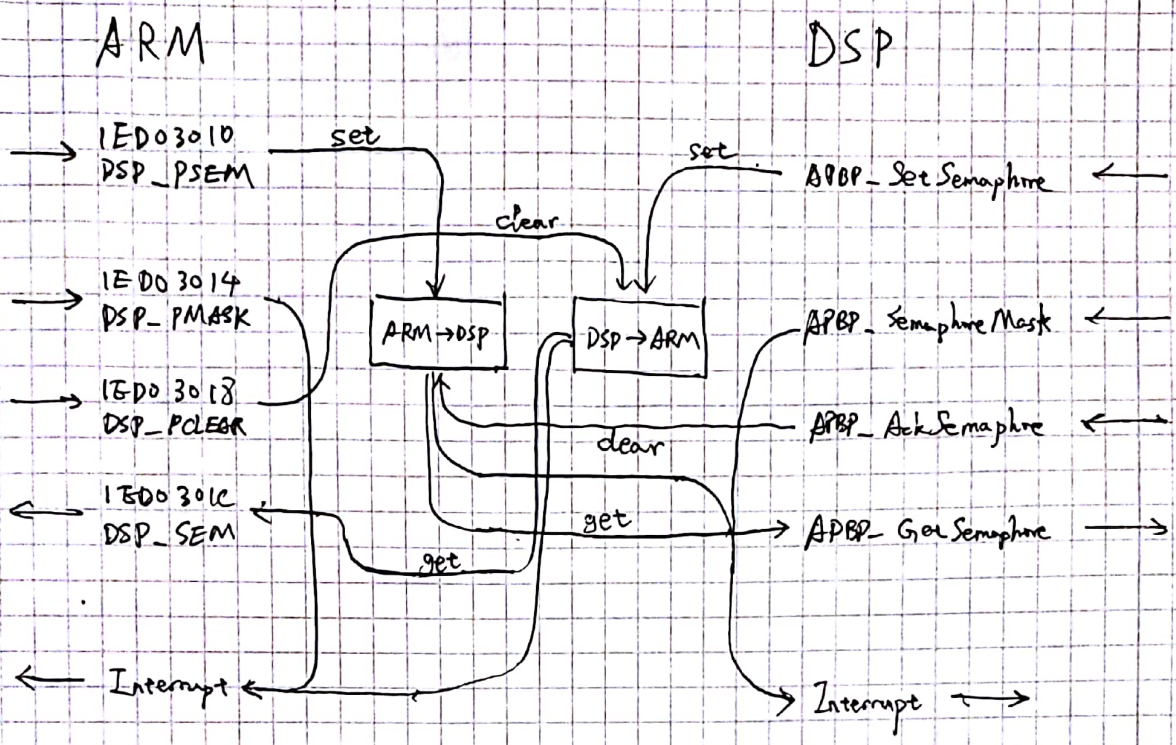
Event:


Enable
disable
PCFG
PMASK
PSTS
SEM
Mascar Interrupt Bits
? nothing seems listening to these
the mask for semaphore is always 0x7FFF
REP2
Event Table
debug pipe drainage
PIPE
ONE
ZERO
DSP[15] Register Interrupt Events
DSP[16] Get Semaphore Event Handle
This event is signaled
^ app → sys direction

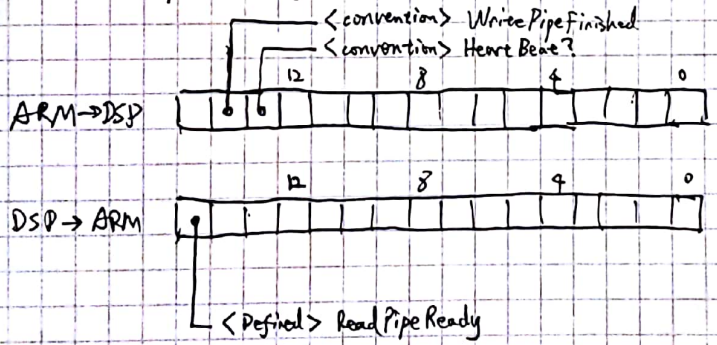
Memory Mapping



Semaphore



Semaphore usage:



Unload sequence: (DSP[12])

- Unregister handler for interrupt bits 0x8000 (pipe interrupt)
- Send (0x8000) to reg2 and RecvPara from reg2 (ignore value)
- ResetDSP - PowerOffDSP
- remove component node from linked list (???)
- CDC_DSP_7(0), CDC_DSP_8(1)

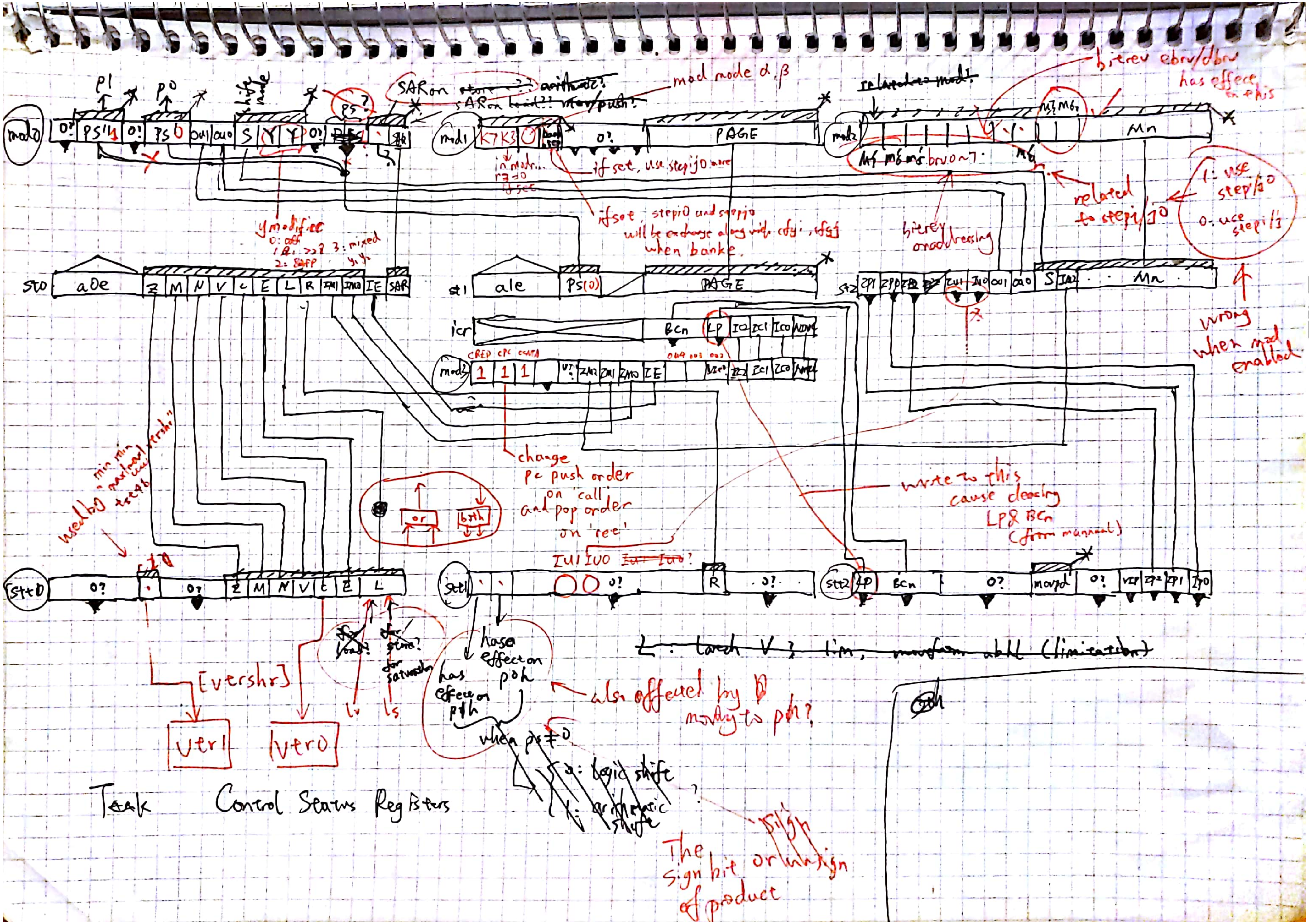
Load sequence: (DSP[11])

- Zero initialize component node
- ~~Clean Up~~
- Clean PipeBufferBase, Create what light event
  Clean Pipe Interrupt Handler, setup DEBUG Pipe Handler
- LoadBinary

+ svcFlushDataCache

- Map Memory

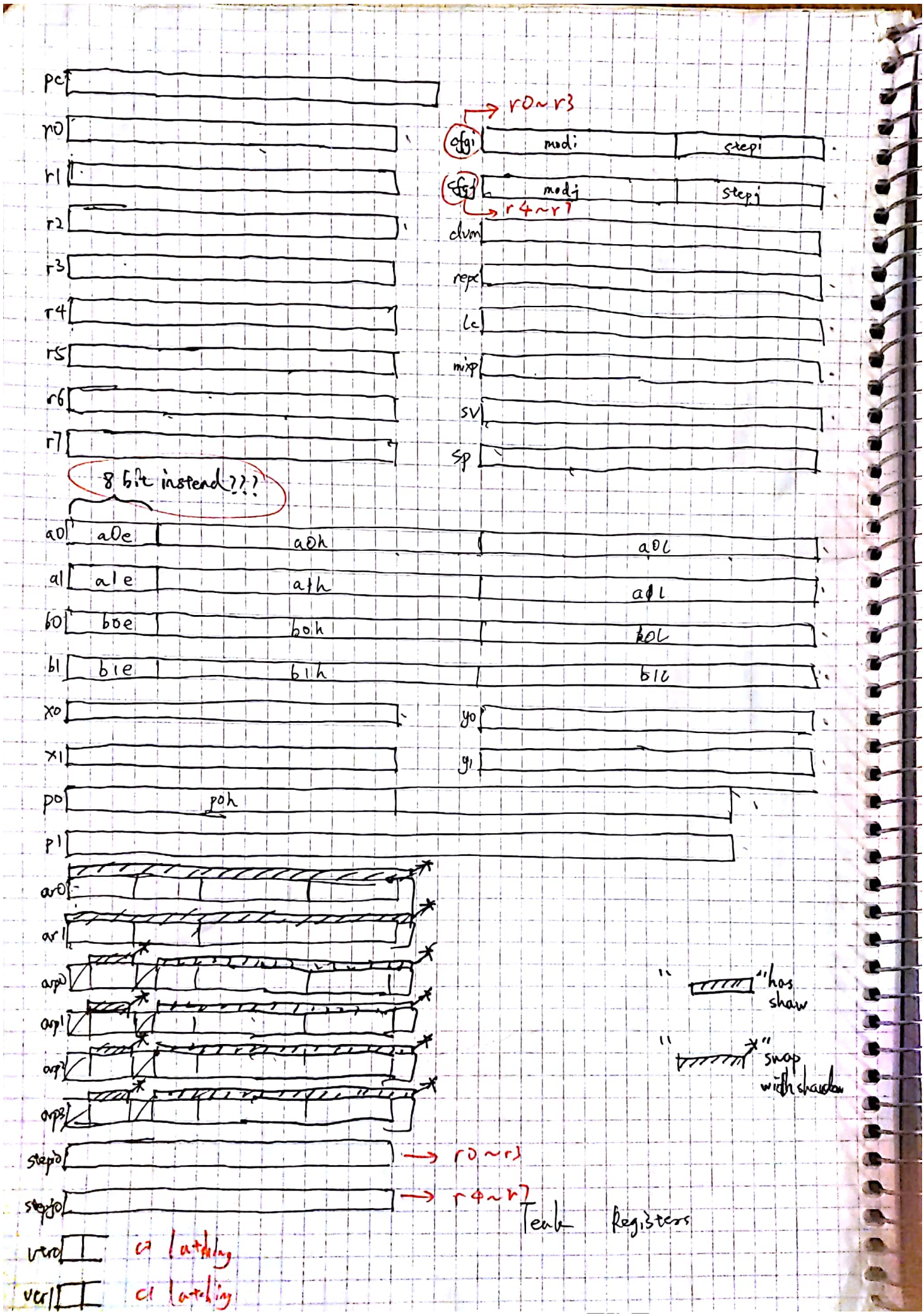
- add component node to linked list (???) - DSPSetup { resee wait for reply (0x1,2) "1" of these
- Set handler for interrupt bits 0x8000 (pipe interrupt) interrupt handler.
- CDC_DSP_7(1), CDC_DSP_8(1)
- Set pipe interrupt handler



Teak Control Status Registers
mod0
mod1
mod2
mod3
st0
st1
st2
stt0
stt1
stt2
icr
PAGE
Mn
Z M N V C E L R IM1 IM0 IE SAR
BCn LP
mod mode α, β
bitrev ebrv/dbrv has effect on this
related to step/j0
1: use step/j0
0: use step/i1
wrong when mod enabled
if set, use step/j0 more
y modifier
0: off
2: SAFF
3: mixed
bitrev addressing
write to this cause clearing LP & BCn (from manual)
change pc push order on 'call' and pop order on 'ret'
has effect on p0h
has effect on p1h
when ps ≠ 0
0: logic shift
1: arithmetic shift
also affected by moving to p0h?
The sign bit or unsign of product
vershr
vtr1
vtr0
lv
ls

THIS

+0 : pNext
+4 : "synthe"
+8
+10
+14
+18 ~(#memory map bits)?
+1C From DSP Binary +0x10F. bit0
+20 Program Memory Enable bit
+24 Data Memory Enable bit
+28 : ~~7 x 8~~ Program Memory Mapping x8
+48 : ~~7 x 8~~ Data Memory Mapping x8 (1 page = 0x800 bytes = 0x200 words)
+68 : Hw Interrupt Mask
+6C : Hw Interrupt Handlers x19
+B8 : Hw Interrupt Handler Param x19
+104 : Hw Interrupt coverage bits x19


pc
r0
r1
r2
r3
r4
r5
r6
r7
r0~r3
cfg0
mod0
step0
cfg1
mod1
step1
r4~r7
dum
rep
lc
mixp
sv
sp
8 bit instead ???
a0
a0e
a0h
a0l
a1
a1e
a1h
a1l
b0
b0e
b0h
b0l
b1
b1e
b1h
b1l
x0
x1
y0
y1
p0
p0h
p1
ar0
ar1
arp0
arp1
arp2
arp3
"has shaw
"swap with shadow
step0
step1
r0~r3
r4~r7
Tenb Registers
ver0
ver1

DSP

MMIO regin: 0x7800 ~0x8000 ?

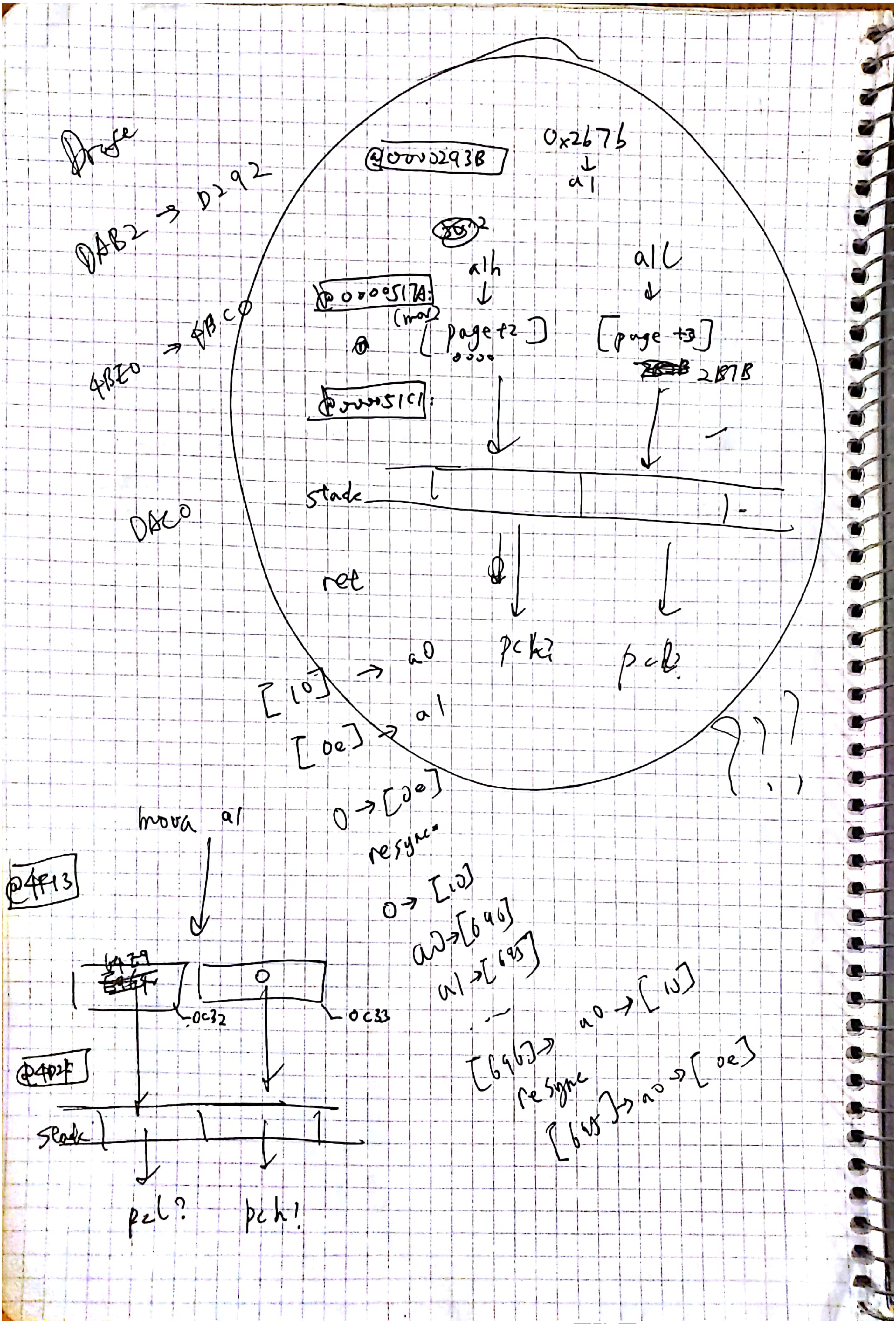
XY:

@data 0150: a.k.a MIO address list?

In MIU

MMIO + 0x110: Double buffer selector register ?! < 0 → 0x8000 ~ 0xFFFF
1 → 0x0000 ~ 0x7FFF

~~MMIO + 0x11E: Double buffer selector register 2 ?!~~ < 0 → 0x8000 ~ 0xFFFF
1 → 0x0000 ~ 0x7FFF



Drofe
DAB2 → D292
4BE0 → 4BC0
DAC0
@00002938
0x2676
a1
@0000517A:
(mar)
a1h
[page+2]
a1L
[page+3]
2B7B
@000051C1:
stack
ret
pch?
pcl?
[10] → a0
[0e] → a1
0 → [0e]
resync
0 → [10]
a0 → [696]
a1 → [695]
a0 → [10]
[696] →
resync
[695] → a0 → [0e]
mova a1
@4F13
0C32
0C33
@402F
stack
pcl?
pch!

[015AE] [015D]
NIU Zeu

~~4bcf interrupt0~~

4bd3 interrupt0

- push all kinds of stuff
- backup 40 words from 0x0028

[MMIO:110] → a0 → [0x0696] ; 0 → [MMIO:110]
[MMIO:10E] → al → [0x0695] ; 0 → [MMIO:10E]

memory bank register

MMIO:110 → [0696]
MMIO:10E → [0695]
then clear MMIO

call [1c030] (a0L = 0xb) — CheckInterrupt(index)

- real MMIO:200 (interrupt request flag)
- right shift according to a0L
- and 1
- return

if (has interrupt) {
  call [0c82]
  call [1c019] (a0L = 0xb) — AckInterrupt(index)
}

- MMIO+202 → al (interrupt Ack)
- call [1c04B]
- switch(...)

↓

[0x0696] → [MMIO+110]
[0x0695] → [MMIO+10E]

↓

~~store the backed~~
restore 0028

4c86 interrupt1

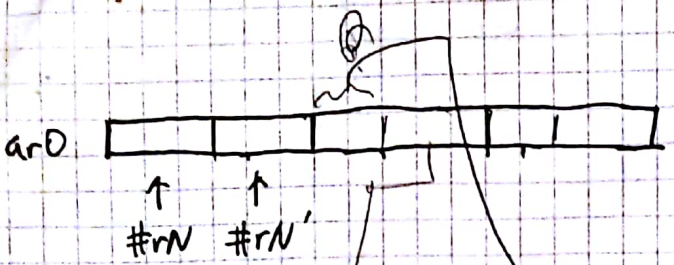
- push
- backup 0028
- backup and reset MMIO+10E, +110
- call ~~1c10Y~~ [1c019] (a0L = 0xe) — AckInterrupt
- call [6046] → r4
- call [6057] ~~cnt~~ → [page:0]
- call [6063] → r0
- call [608d] → b0
- ...

- ~~return~~ apbp flag.bit8 — cmd0?
- return apbp flag.bit12 — cmd1?
- return apbp.flag.bit13 — cmd2?
- return apbpflag.bit9 — semaphore?

al: [| | |] → br [4d5c] (al → r0)

is a loop!

- bit0
- bit1 — [4cff]
- bit2 — pp, return
- bit3 — [6080]



0: off+0
1: +1
2: -1
3: +5
4: +2
5: -2
6: +2?
7: -2

0: off+0
1: off+1
2: off-1
3: off-1

default ar0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0

ar1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 0 1 1

R0 R4 +0 +2

R2 R5 -2 +5

arp0 0 0 0 0 0 0 0 0 0 1 0 0 0

arp1 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0

arp2 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0

arp3 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0

+1 +1

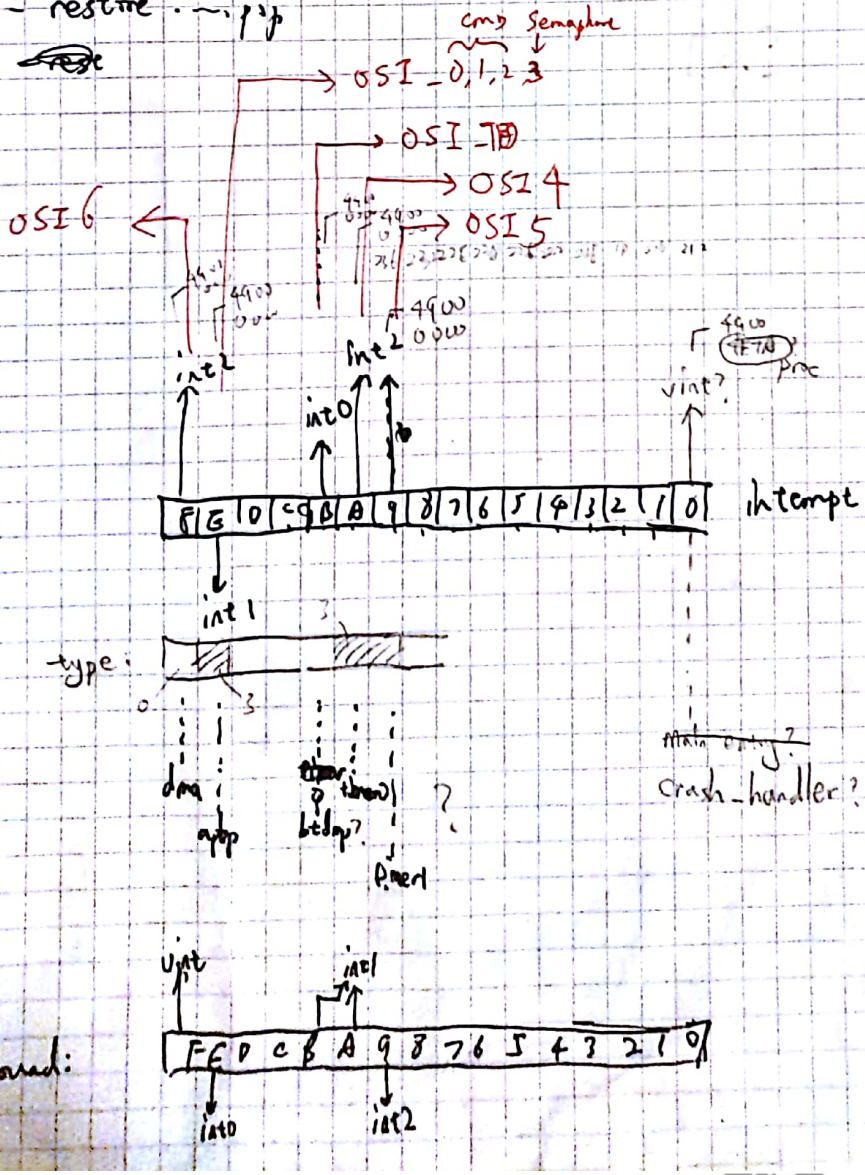
+2 +2

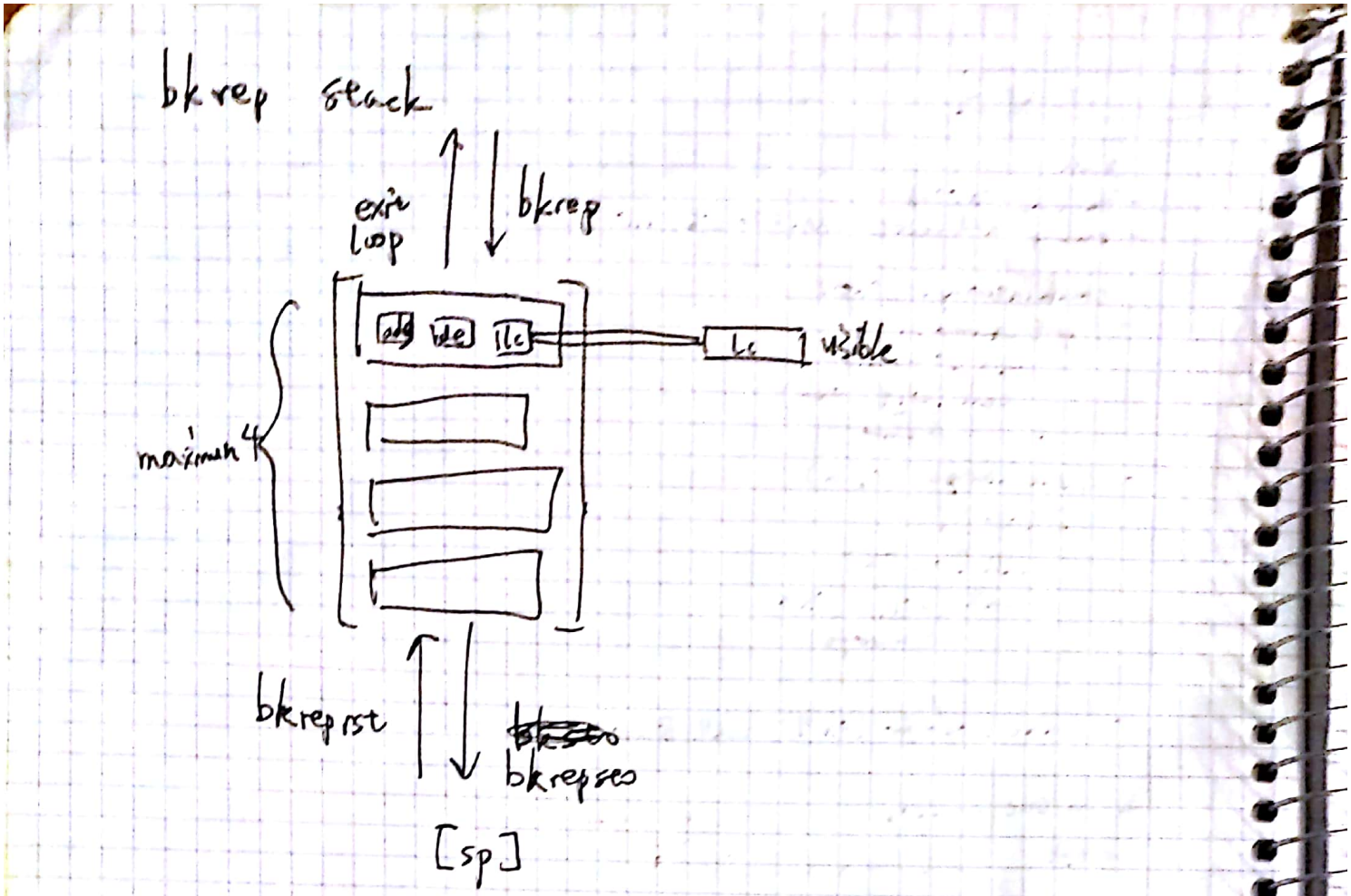
-2 -2

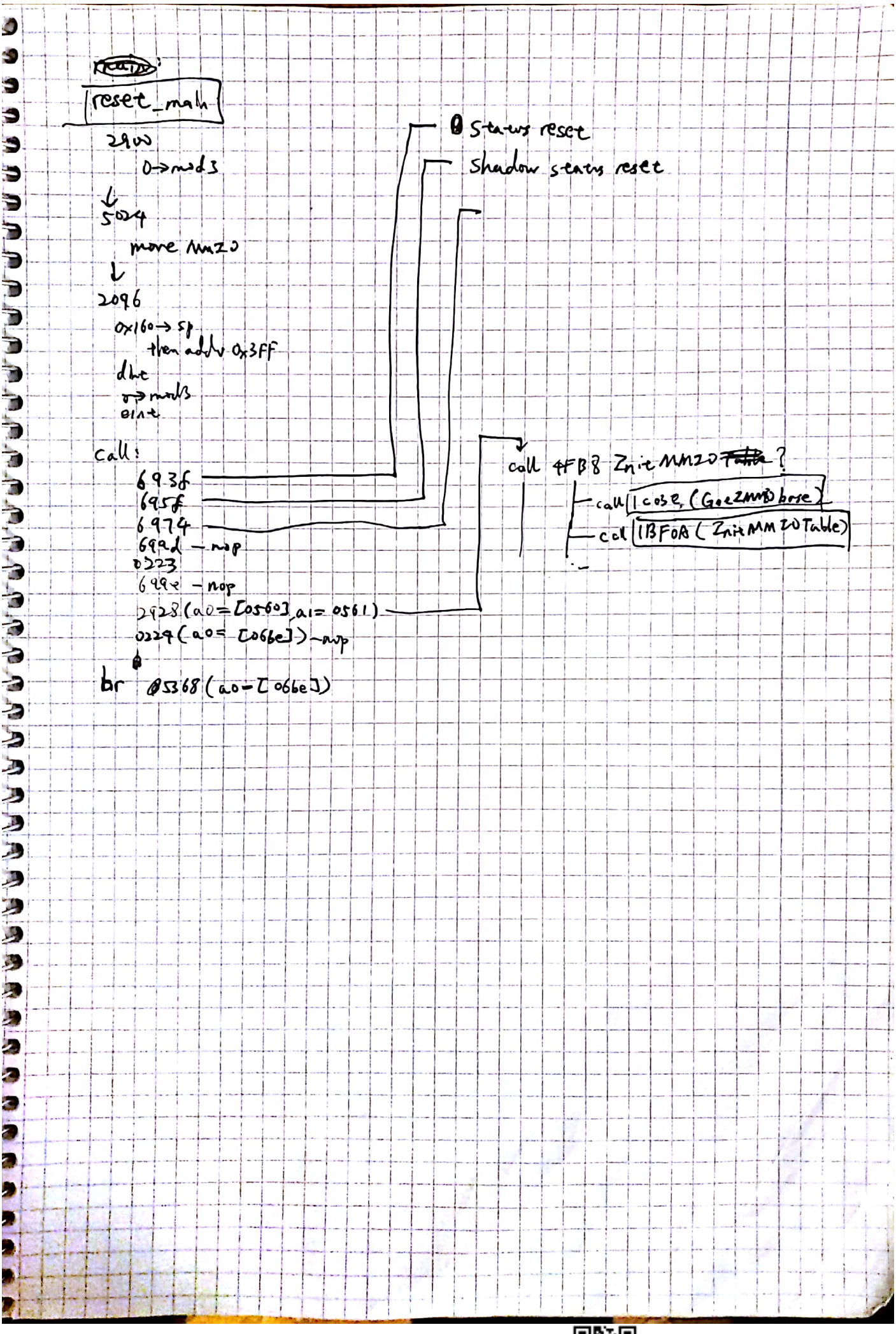
-1 -1

interrupt 2

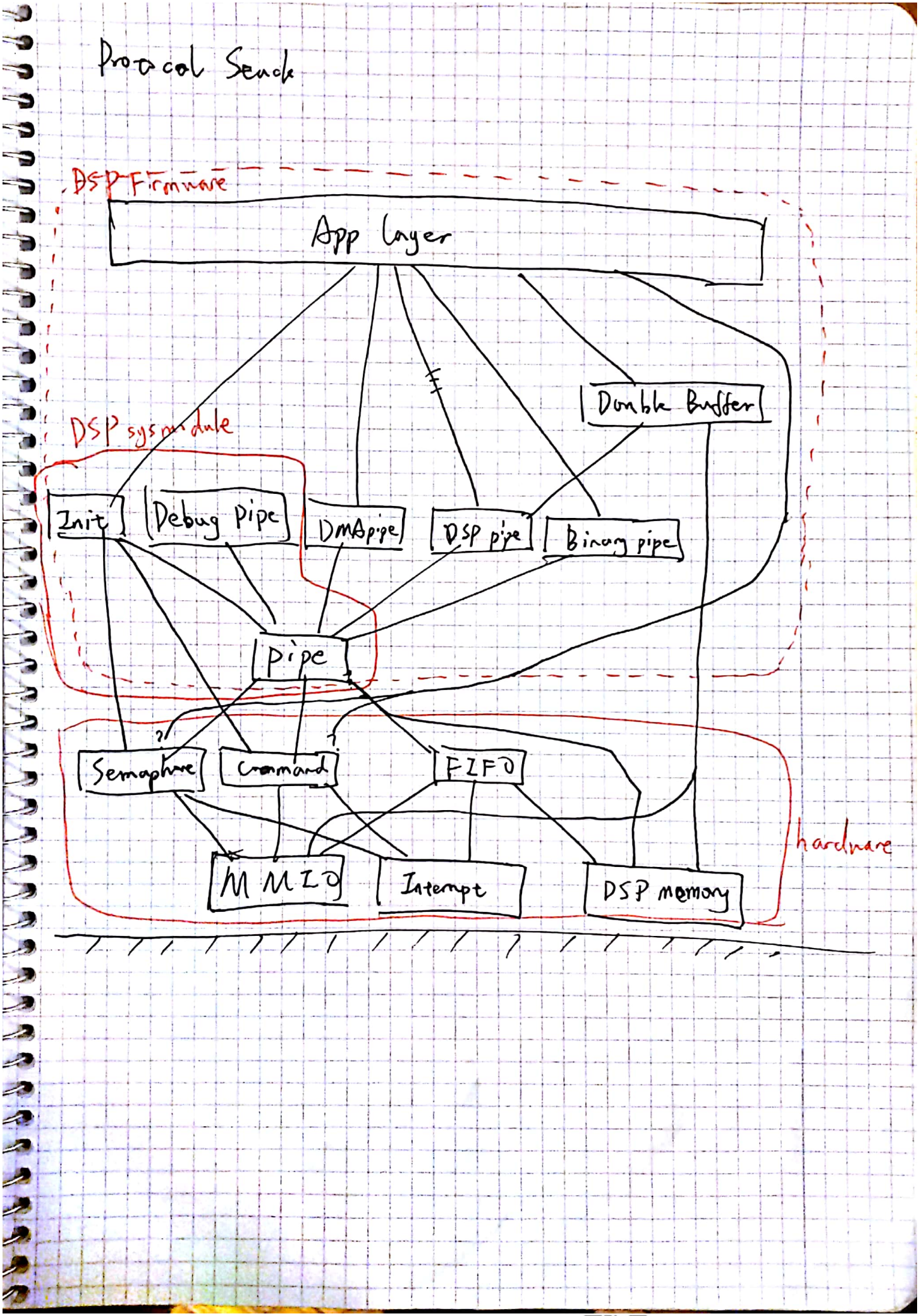
- push
- back up 0x0028
- backup and reset MMIO+ 0x0, +110
- check interrupt (0xf)
- if(checked){
  call [0c72]
  call [c019] (0xf)
  AckInterr
  }
- check interrupt (0xa)
  if(--){
  call [0c52]
  call [c019] (0xa)
  AckInterrupt
  }
- Same block for (0x9), call [0c62]
- restore ... pop






reset_mali
2900
0->mod3
5024
move MM20
2096
0x160->sp
then addw 0x3FF
dbne
0->mod3
exit
call:
6936
6958
6974
699d - nop
0223
699e - nop
2928(a0=[0560],a1=0561)
0229(a0=[066e])-nop
br 05368(a0-[066e])
0 Status reset
Shadow status reset
call 4FB8 Init MM20 Table?
call Icache (Gpe2MM3 base)
call IBFOA (Init MM20 Table)

Protocol Stack



DSP Test suit

Program

0x0000 reset vector br 0x1000

0x1000 main loop

```
while(1){
  run = 0;
  if (start_pending == 1){
      start_pending = 0;
      run = 1
  }
  if stop_pending == 1){
      stop_pending = 0;
      run = 0;
  }
  if (run) ... if (!run) br 0x1000
      sp = reg_loader
      pop...
      sp = stack
      call tester
      sp = reg_saver
      push...
  } while 1 br 1000
  br 0x1000
```

0x1800

0x2000(*) tester

```
...
...
ret br 0x1800
```

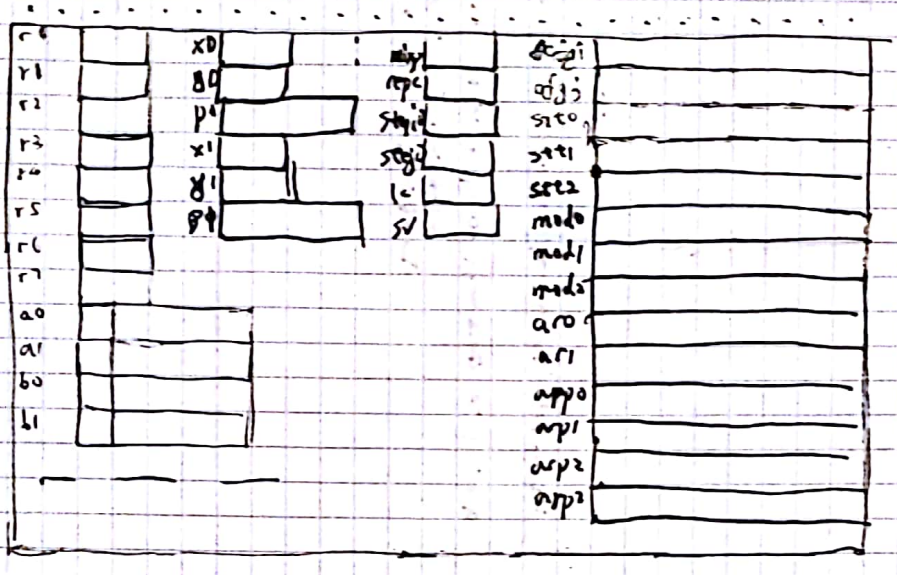
0x3000 interrupt handler

Data:

0x0000 signals: +0 run
+1 start pending
+2 stop pending

0x1000(*) reg_loader

0x3000(*) reg_saver



x0
y0
y1
x1
y1
y1h

mod=4   掩三位, "101" → "000"

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		1	2	3	4											
1		2	3	4	0											
2		3	4	0	6											
3		4	0	6	7											
4		5	6	7	0											
5		6	7	0	1											
6		7	0	1	2											
7		0	1	2	3											
8		9	A	B												
9		A	B	C												
A		B	C	8												
B		C	8	E												
C		8	E	F												
D		E	F	8												
E		F	8	9												
F		8	9	A												

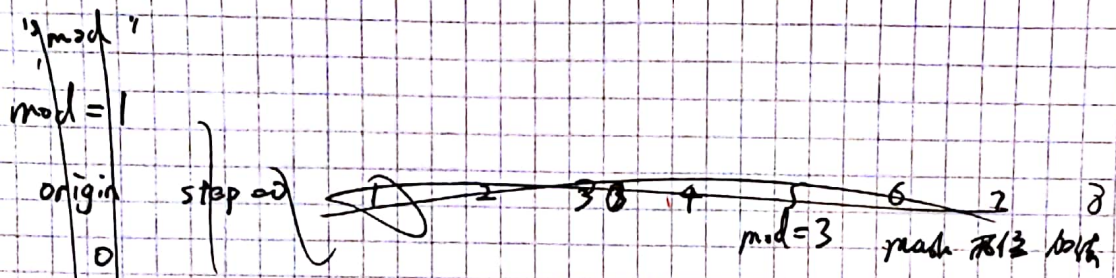
-1	-1	-2	-3
7	4	3	2
0	0	7	6
1	1	0	7
2	2	1	0
3	3	2	1
4	4	3	2
5	5	4	3
6	6	5	4
F	C		A
8	8		
9	9		
A	A		
B	B		
C	C		
D	0		
E	E		

n+5 == mod+1 ?

→ 0

101

110



X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
3	3	4	5	6	7	8	9	A	B	C	D	E	F			
4	4	5	6	7	8	9	A	B	C	D	E	F				
5	5	6	7	8	9	A	B	C	D	E	F					
6	6	7	8	9	A	B	C	D	E	F						
7	7	8	9	A	B	C	D	E	F							
8	8	9	A	B	C	D	E	F								
9	9	A	B	C	D	E	F									
A	A	B	C	D	E	F										
B	B	C	D	E	F											
C	C	D	E	F												
D	D	E	F													
E	E	F														
F	F															

mod=2 step R和取值

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	0	0	1	2	0	0							
1	1	2	0	0	1	2	0	0	1							
2	2	0	0	1	2	0	0	1	2							
3	3	0	1	2	0	0	1	2	0							
4	4	5	6	4	4	5	6	4	4							
5	5	6	4	4	5	6	4	4	5							
6	6	4	4	5	6	4	4	5	6							
7	7	4	5	6	4	4	5	6	4							
8	8	9	A	8	8	9	A	8	8							
9	9	A	8	8	9	A	8	8	9							
A	A	8	8	9	A	8	8	9	A							
B	B	8	9	A	8	8	9	A	8							
C	C	D	E	C	C	D	E	C	C							
D	D	E	C	C	D	E	C	C	D							
E	E	C	C	D	E	C	C	D	E							
F	F	C	D	E	C	C	D	E	C							

两位掩码加法

掩码内的右"1"变"00"

mod=4 一位掩码加法

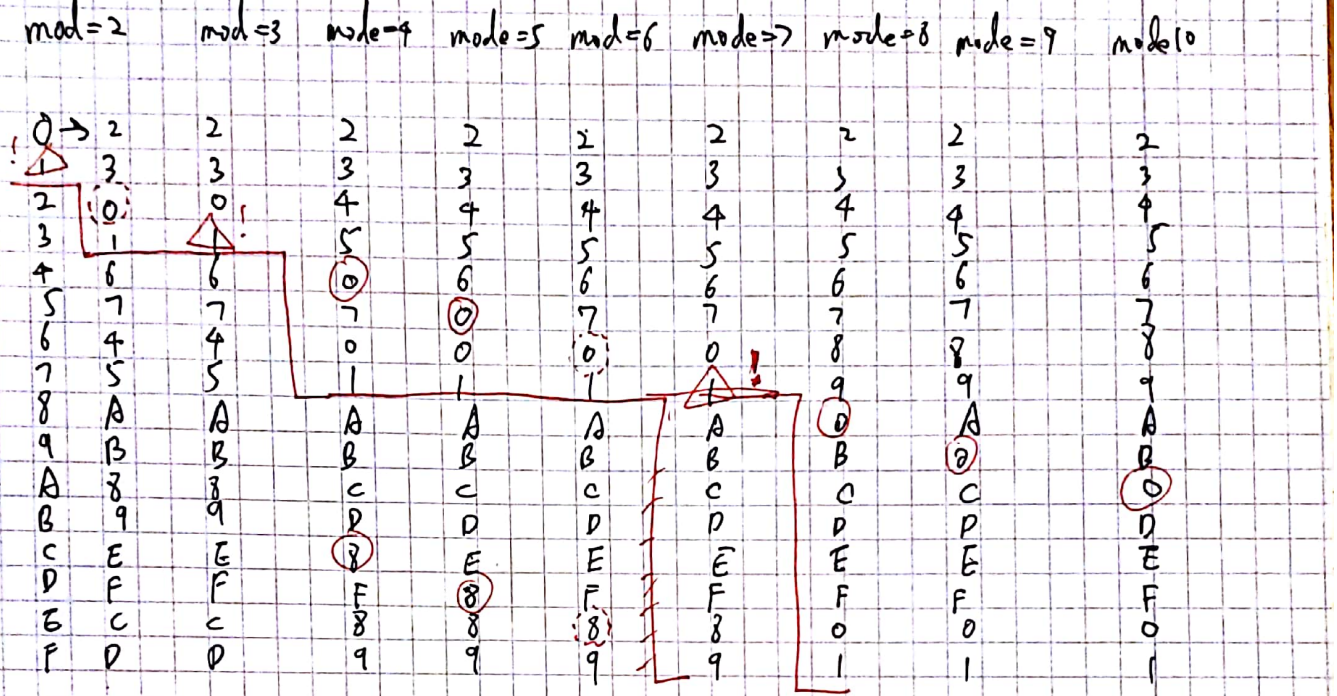
X	0	1	2	3	4	5	6	7	8
0		1	0						
1	1	0	1						
2	2	3	2						
3	3	2	3						
4	4	5	4						
5	5	4	5						
6	6	7	6						
7	7	6	7						
8	8	9	8						
9	9	8	9						
A	A	B	A						
B	B	A	B						
C	C	D	C						
D	D	C	D						
E	E	F	E						
F	F	E	F						

clrr

@2	@0	
0	0	$b_0 b_1$
0	1	$b_0 b_1$
0	2	$b_0 b_1$
0	3	$b_0 b_1$
1	0	$b_0 b_1$
1	1	$b_0 b_1$
1	2	$b_0 b_1$
1	3	$b_0 b_1$
2	0	$a_0 b_0$
2	1	$a_0 b_1$
2	2	$a_0 a_1$
2	3	$a_0 a_1$
3	0	$a_1 b_0$
3	1	$a_1 b_1$
3	2	$a_1 b_0$
3	3	$a_1 b_0$

Astep 6 "+2*"
(new mode)

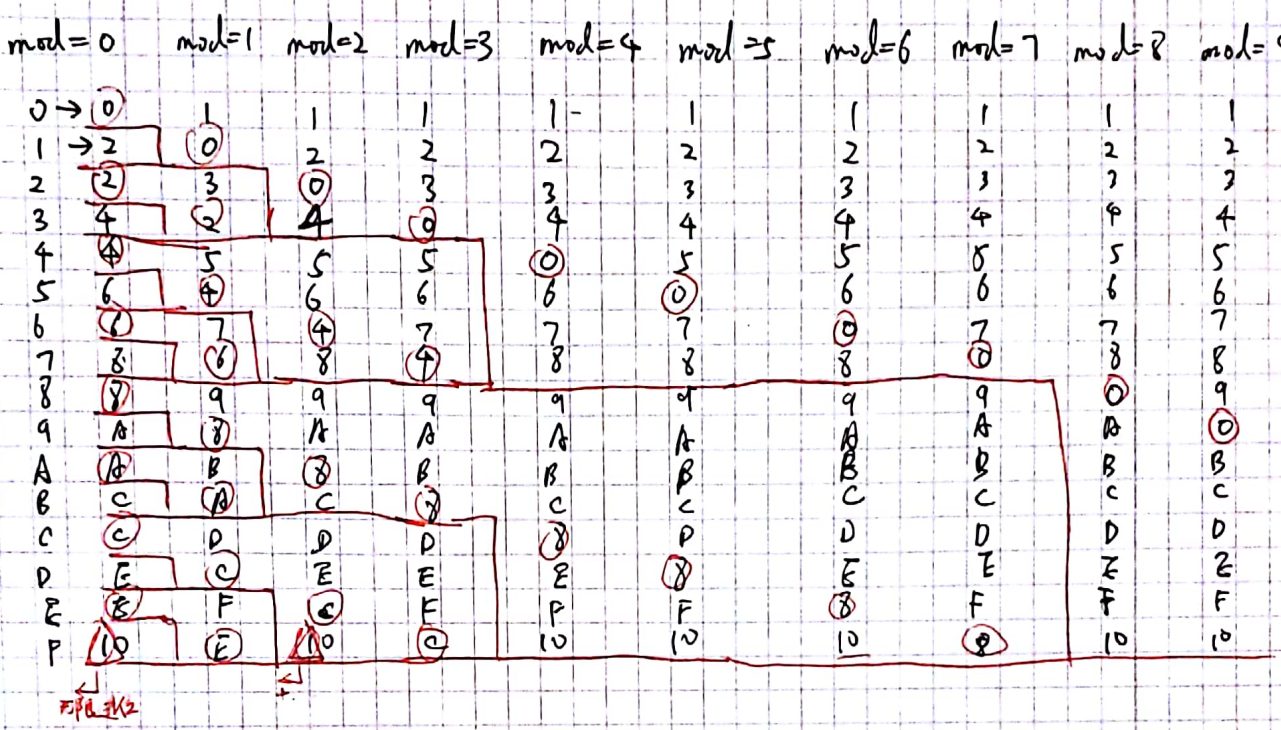
mod=1 : no change



offset+1

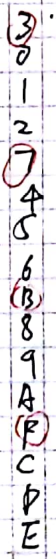
mod=on

Legacy=1/0
brev = 0
step0 = 0/1

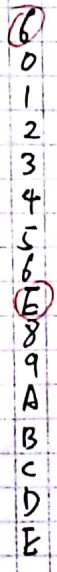


offset-1
mod=0

mod=3



mod=6



nMW:
0.2C2

瞬发连取永远都是3, 但连续使对伤害是1

手动 16, 偶[illegible]

偶? 3 偶? 4, 2

push one:

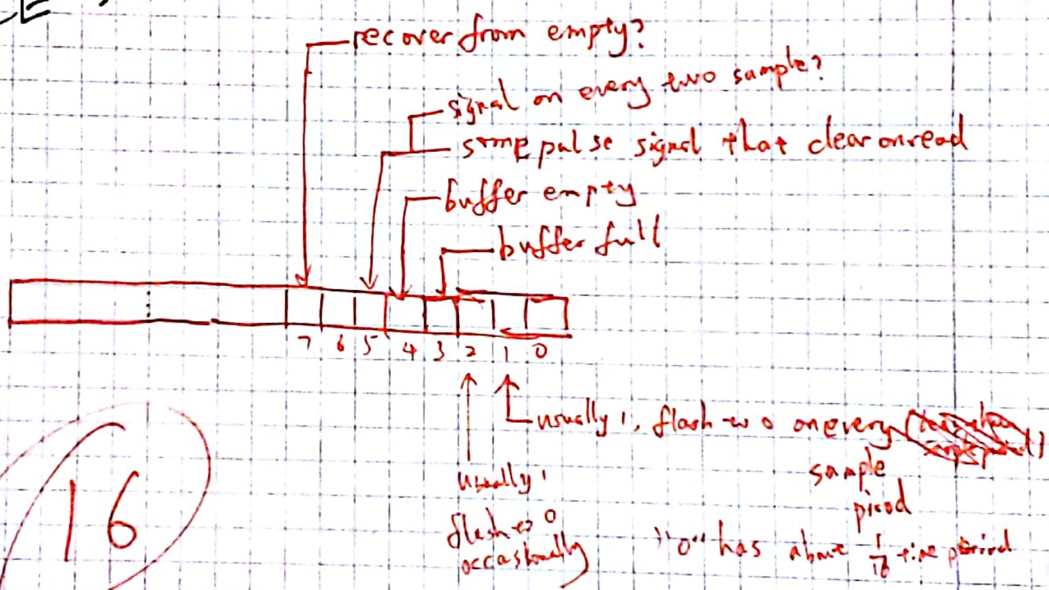
A6

push two: three 4 5, 6 7 8 1

26

push 16:

2E !



Buffer size 16

5E18

1234

04BC

82C6 mov → 82e6

670+1@12 inc a1 → increase dw

9420 (82C2) tstb, bit 4

5000+2@0+125@4

"155"

BTDMP

recieve		dsi	transmit		dsi/3ds
0280	bit9 irq	0000	02A0	bit8 irq	0007
0282		1004	02A2		1004
0284		0004	02A4		0004
0286		0021	02A6		0021
0288		0000	02A8		0000
028A		0000	02AA		0000
028C		0000	02AC		0000
028E			02AE		
0290			02B0		
!					
029E	—enable (8000)		02BE	—enable (8000)	

4100. sample rate divisor? Yes!

Filter? (processor clock = 134 MHz

$\frac{134000}{4100} \approx 32682$ → Matches 32768Hz on 3ds bev?

~~buffer size = 7~~

bug

② 02C0 bit3 recv — bit 3 not set → no recive buffer full?

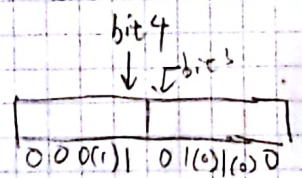
02C2 bit send { bit 4 not set → no send buffer empty? ✓
bit3 set → stop send buffer full! ✓

02C4 recv FIFO?

02C6 send FIFO ✓

02C8

02CA btdmp_fifo_flush_channel, ~~accept or address~~ test bit2 spinlock



0034
16
2

a0, Addr0,	p+2,+1	DMA+0
a1, Addr1	p+4,+3	DMA+4
b0, size	p+8	DMA+8
b1,	p+10	DMA+A
r7+3 (Addr0)	p+6	DMA+12
r7+4 (Addr1)	p+7	DMA+14
r7+5		DMA+C
r7+6 (Addr0)	p+8	DMA+16
r7+7 (Addr1)	p+9	DMA+18
r7+8	p+0	"7" = Addr0 in FCRAM?
r7+9	p+D	"7" = Addr1 in FCRAM?
r7+A, channel	p+A	
r7+B	p+E	
r7+C	p+11	~~DMA+20~~ ?

p+C: Interrupt save

{

Exchange Interrupt

if ([r7+8] == 7) { //635

} else { ~~635~~ //

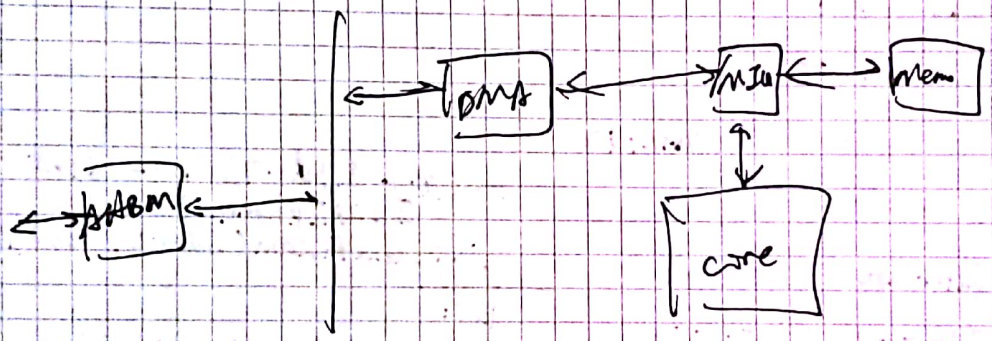
if ([r7+8] != 7) goto 635
D23['channel] = 0x4
if(!CheckFCRAM(Addr0, size)) ~~goto 6166~~ fail

635:
if ([r7+9] != 7) goto 616B
D23[channel] = 0x8
if(!CheckFCRAM(Addr1, size)) ~~goto 6166~~ fail

~~goto 616B~~
~~6166: fail, return~~

616B:
spin wait on AHBM
if(size == 0) end

b0 = [p+5] = 1
if([r7+B] == 0) {
[r7+3] += 1, [r7+4] += 1
[r7+6] += 1 [r7+7] += 1
b0 = [p+5] = 2
}
6187:
if([r7+8] == 7) {
[p+5] <<= 1
[r7+3] <<= 1, [r7+6] <<= 1
}



```
if([r7+9]==7){
    b0<<=1;
    [r7+4]<<=1; [r7+7]<<=1
}

switch DMA channel

[MM20+0DE]@|=0x8000

if( [r7+8]==7 || [r7+9]==7){
  @ 61AB:
   ATBM_channel = 06CC[channel]
q = GetATBM channel
  if (q[2] != 1<<channel){
      [MM20+1DA] &= 0xFF00
      q[2] = 1<<channel
  }
  61C3: [0x6CB] |= 1<<channel
q[0]= [r7+C]|([r7+B]? 0x10 : 0x20)
61D0:
  if ( [r7+8] ==7 )
      q[4]=0x200
  else
      q[4]=0x300
61DA:
  Transfer DMA param
    @ Note: [r+5] → DMA+E
            b0 → DMA+10
```

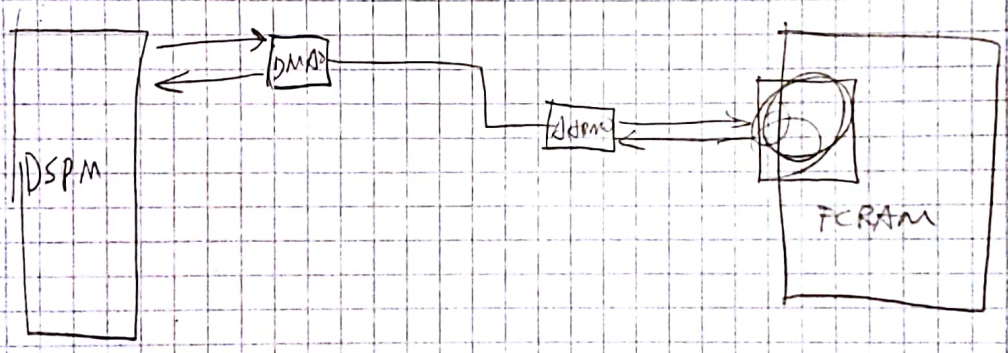
r7+b → 0 → 0x0000
       10 → 0x0200
>> or with Color in FRAM (0,7)
→ @[DMA+5A]

0x0700 → [DMA+5C]

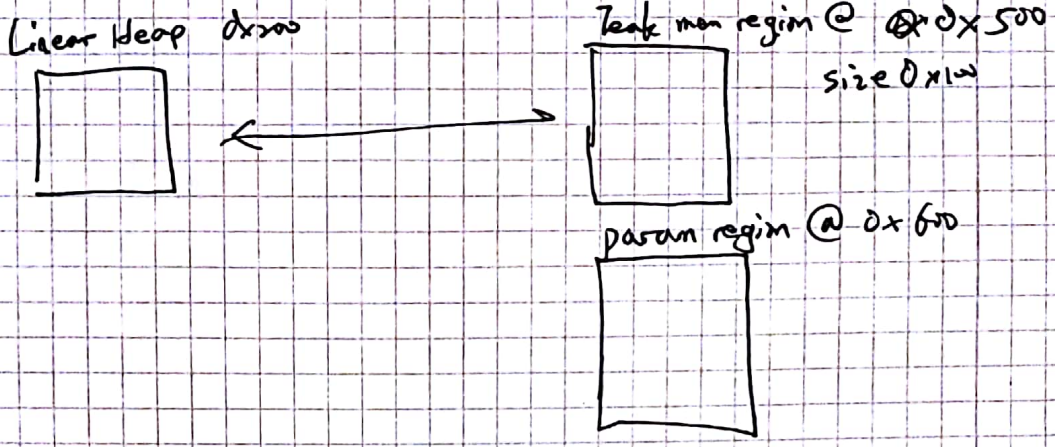
0x40C0 → [DMA+5E]

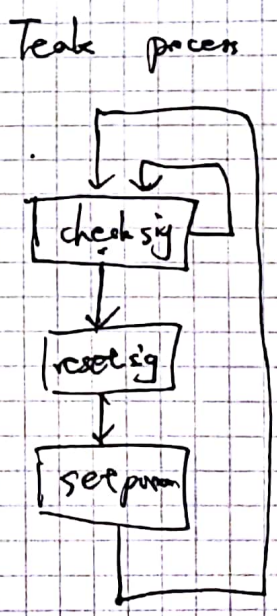
[DmaCopyShare3D]

a0 = channel
a1 = src
b0 = dst
b1 = size
r7+3 = src w1
r7+4 = dst w1
r7+5 = dma+1f
r7+6 = src w2
r7+7 = dst w2
r7+8 = dma+1e



DMA tesser





BTOMP 用户空间

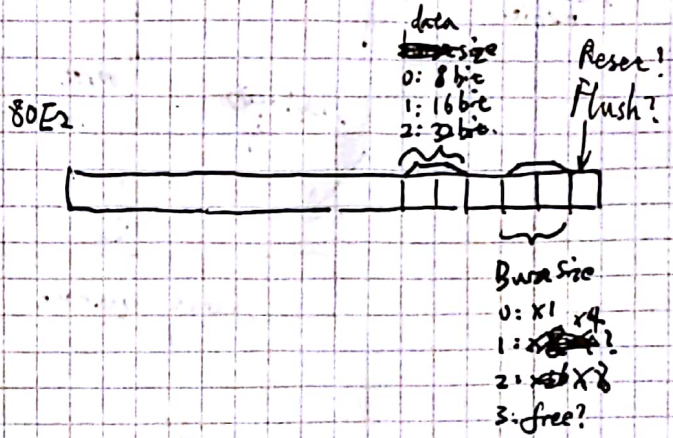
5E18
2345
D4BC } write to FIFO
82C6
0000
9420
5000+2@0+125@4 br -9 nop
D4BC } write to FIFO
82C6
6700+1@12; inc → result: increased to 0x155
9420
5000+2@0+125@4

+1 inc
+1 tscb
+2 br

double: 0x2A1

DMA Registers

AHBM



81BE channel select

81C0 + 81C2 source address lo+hi

81C4 + 81C6 dest address lo+hi

81C8 LO size

81CA L1 size

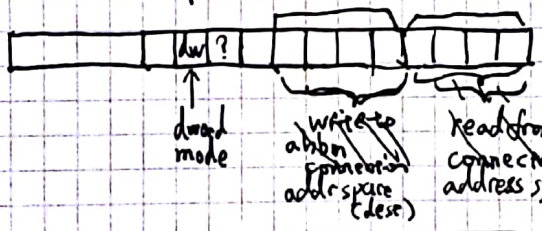
81CC L2 size

81CE, 81D0 L0 step (source, dest)

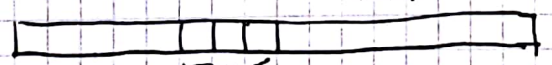
81D2, 81D4 L1 step

81D6, 81D8 L2 step

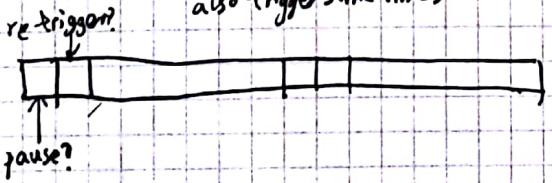
81DA

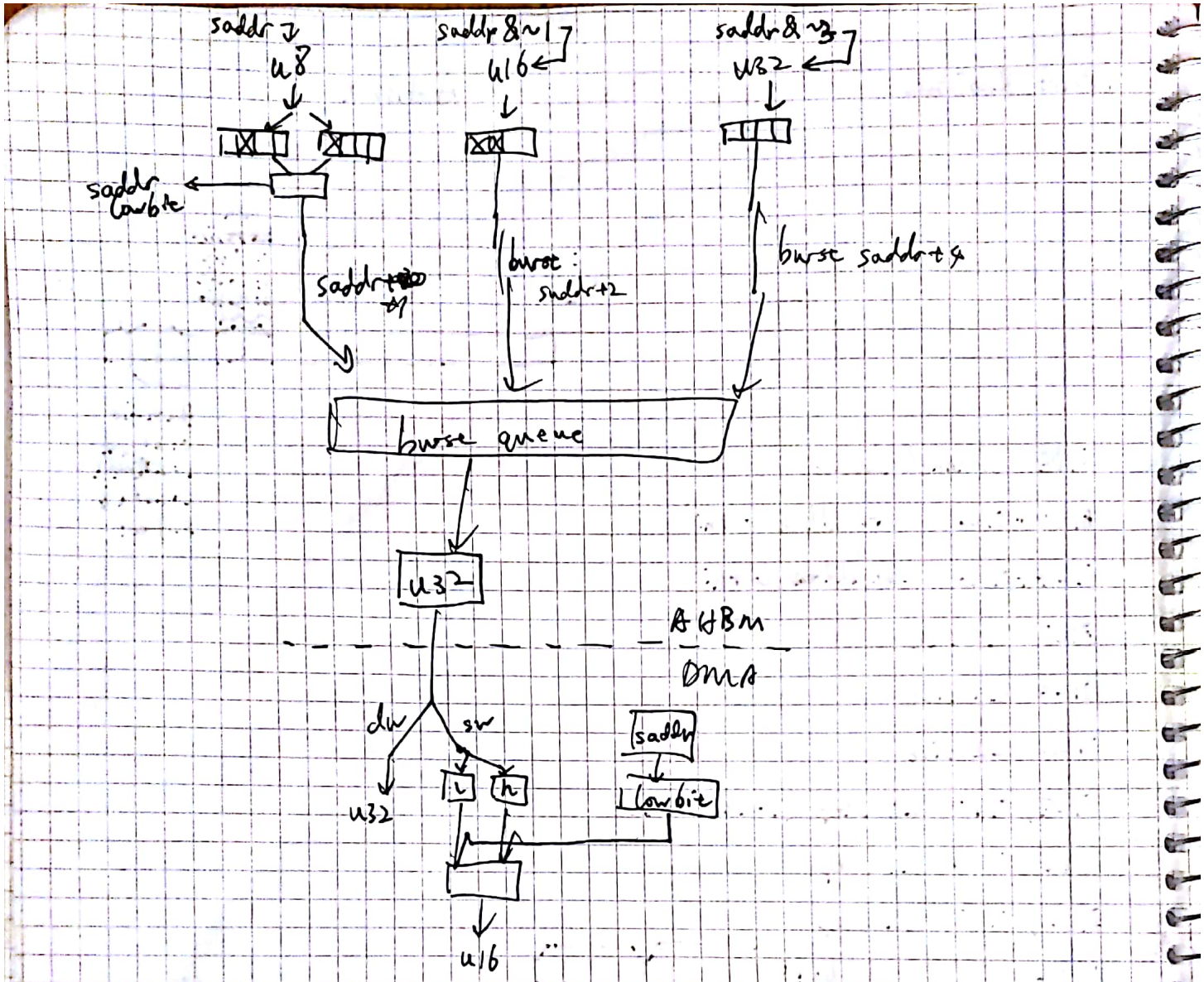


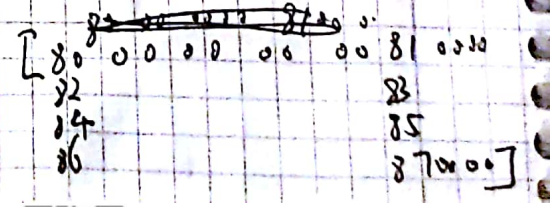
81DC



81DE

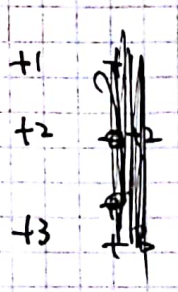


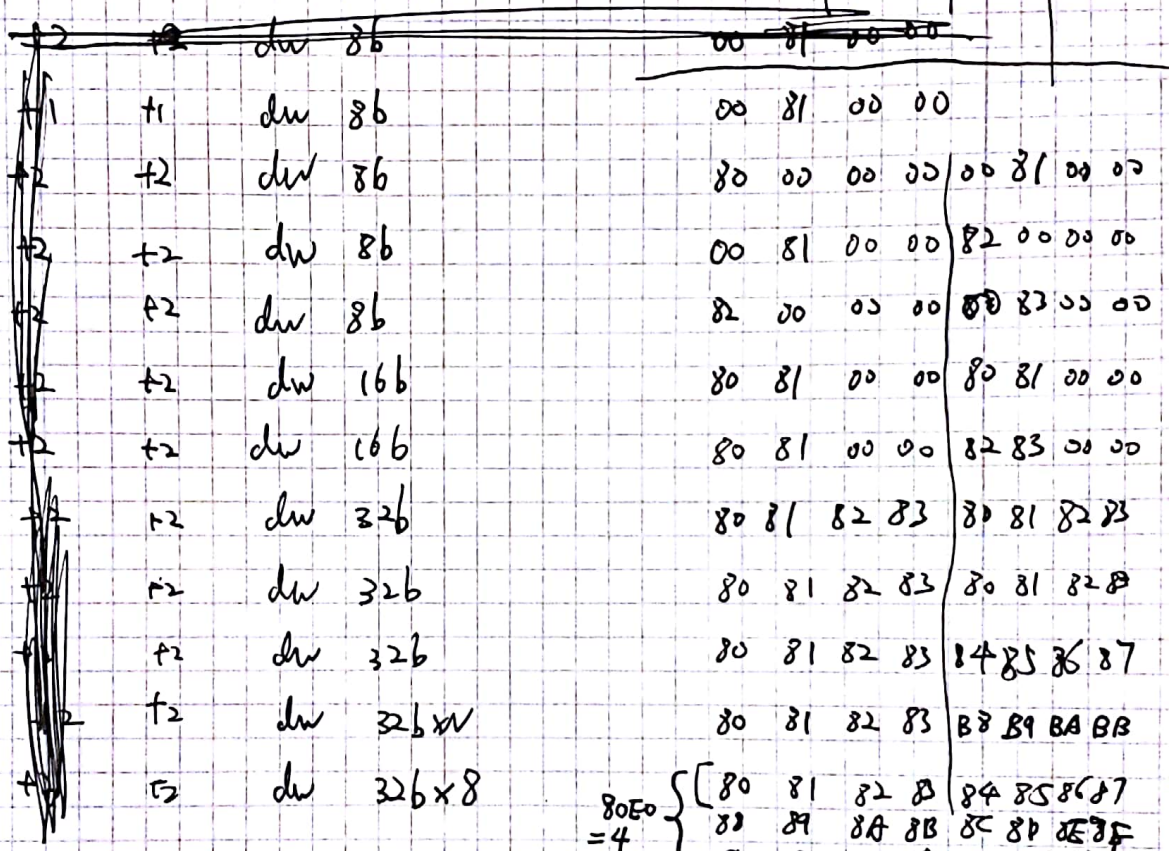

saddr
u8
saddr&~1
u16
saddr&~3
u32
saddr
low bit
saddr+1
burst
saddr+2
burst saddr+4
burst queue
u32
AHBm
DMA
dw
sw
l
h
u32
saddr
low bit
u16



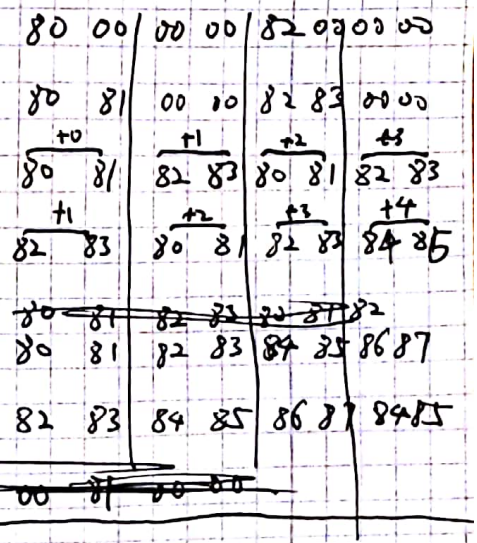
DMA+AHBM 实验 Read From AHBM

src	dst	srcstep	dststep	mode	ahbm	length	ahbm2
+0	+0	+1	+1	sw	8b	4	200
					16b		
					32b		
					32b		



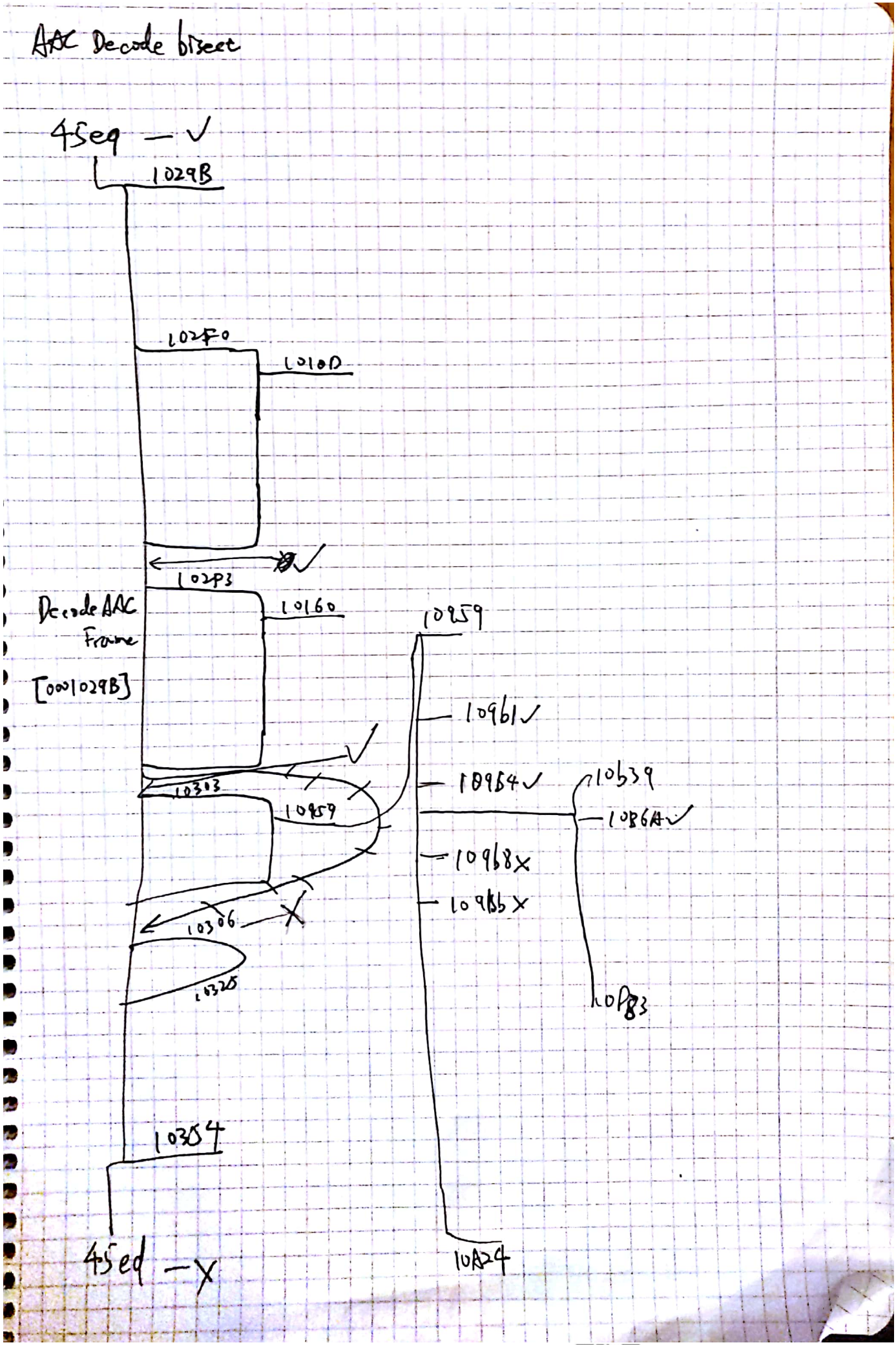


		dw	8b	
+1	+1	dw	8b	
+2	+2	dw	8b	
+1	+2	dw	8b	
+2	+2	dw	16b	
+1	+2	dw	16b	
	+2	dw	32b	
+1/+2	+2	dw	32b	
+3	+2	dw	32b	
	+2	dw	32b×4	
	+2	dw	32b×8	
	+1	sw	32b×8	
	+1	sw	32b×4	
+1	+1	sw	16b×8	
+2	+1	sw	16b×8	
+1	+1	sw	16b×8	
←	+1	dw	8b×8	



Write To AHBM

src	dst	srcstep	dststep	mode	ahbm	length	Result
+0x10	+0	+1	+2	sw	16b	4	20212223 24252627
					8b		20_22_24_26_
			+1		8b		20232427
			+3		8b		20__23__24__27
			+4		8b		20___23___24___26
~~+0x11~~			+1		8b		22252629
+0x10	+1		+1		8b		_21222526
			+1		16b		20232427 (!)
		(repeat)	+2		16b		20212223 24252627
	+1		+1		16b		_2122252627_
	+1		+2		16b		_21_23_25_27
	+0		+3		16b		2021_23__2425_27
			+3		32b		20210000__0000_27000000
			+4		32b		20210000222300002425000026270000
			+2		32b		20210000 24250000
			+1		32b		20230000
	+1		+1		32b		_21000026270000
	+0		+5		32b		20210000_230000__0000___00
	+0		+1	dw	8b		20.23_ _ _ _
		+2	+1	dw	8b		20 27_ _ _
			+2	dw	8b		20 _ 24_
+10/+11			+3	dw	8b		20 _ _ 27
+12			+3	dw	8b		24 _ _ 28
+10			+3	dw	16b		2021_27
			+2	dw	16b		20212425
			+1	dw	16b		2027
			+4	dw	16b		2021 [illegible] _ _ 2425
			+4	dw	32b	8	20212223242526272829[illegible]
			+5	dw	32b	8	20212223_270000_ _[illegible]_ _ _ 00
			+4	dw	32b×8	8	Dead
			+4	dw	32b×8	32	20~5F
		+2	+4	dw	32b×4	16	20~3F
		+2	+8	dw	32b×4	16	20~2F, 10个 30~3F


AAC Decode bisect
45eq — ✓
1029B
102F0
10100
102P3
Decode AAC Frame
[001029B]
10160
10303
10306
10325
10354
45eq — X
10959
10961 ✓
10984 ✓
10968 X
10985 X
10539
10864 ✓
10983
10A24