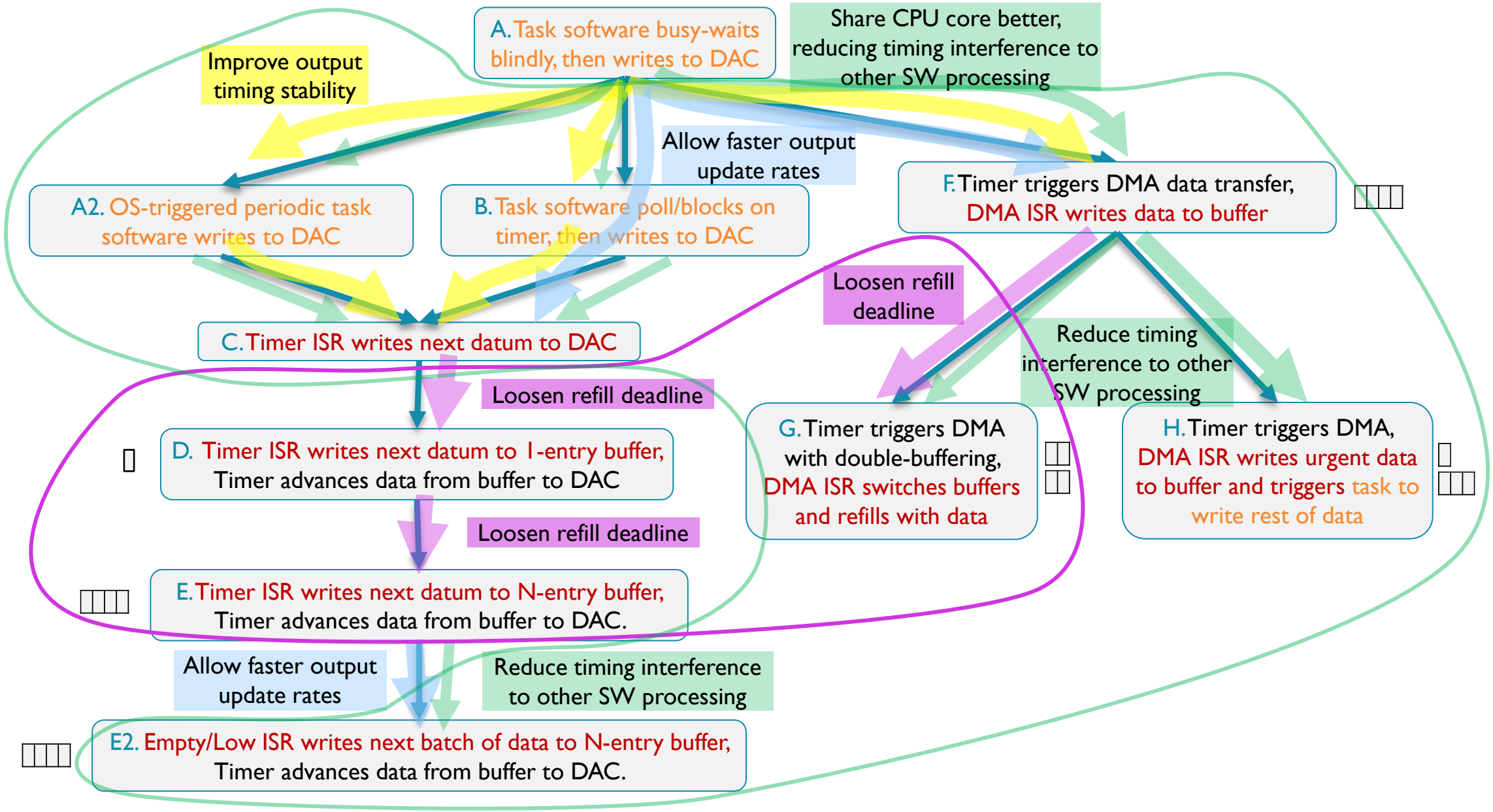
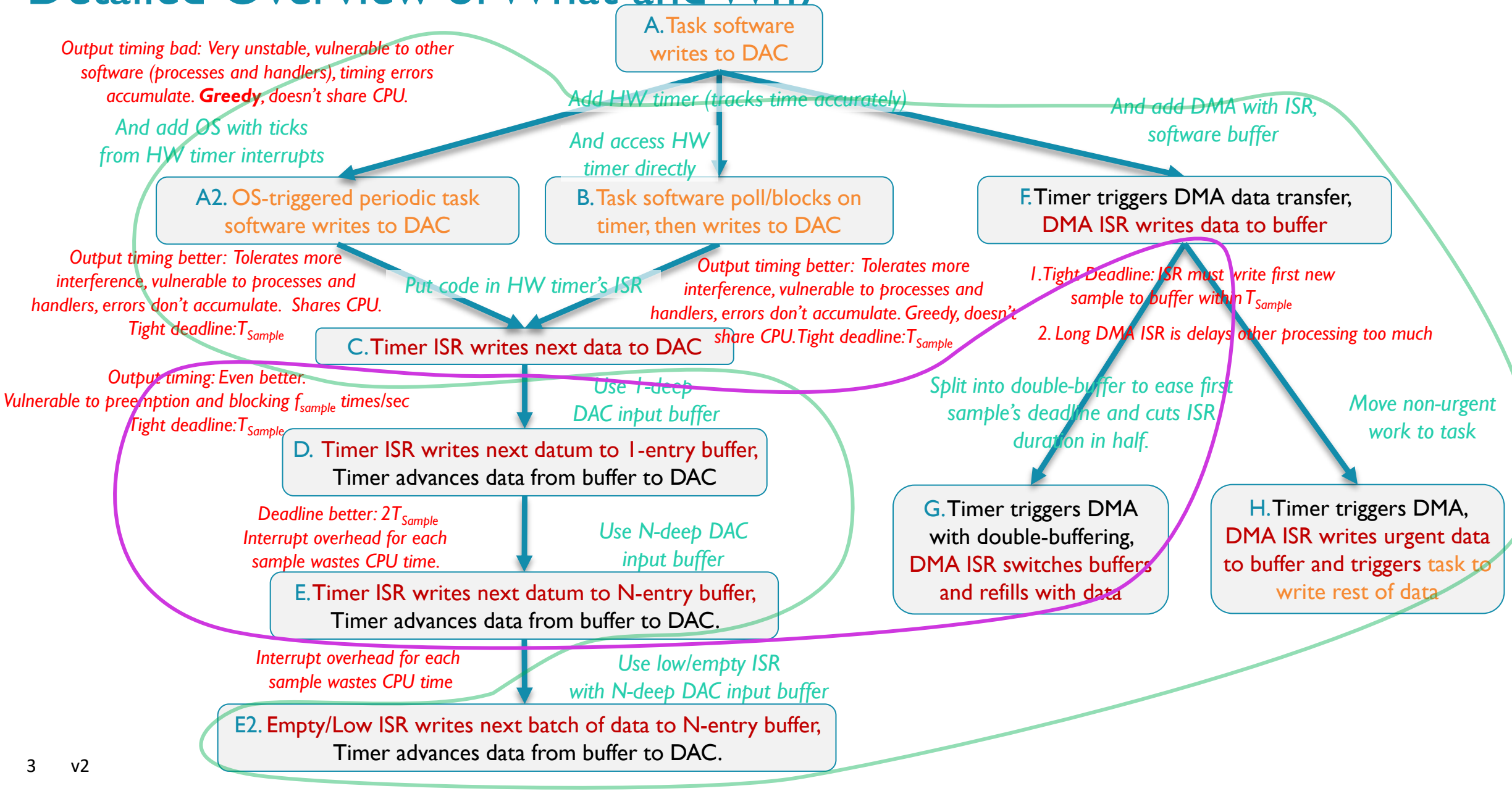
17: WaveGen: Loosening Timing Requirements, Interfering Less with Other Processing

v2

Overview of Waveform Generator Design Evolution: What and Why



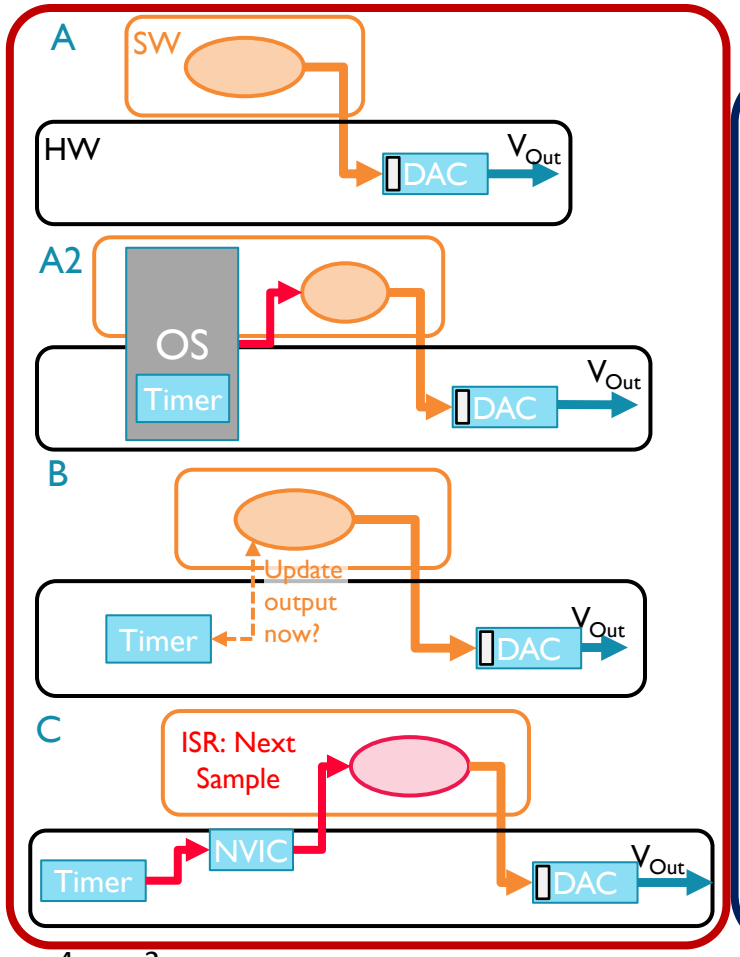
Detailed Overview of What and Why



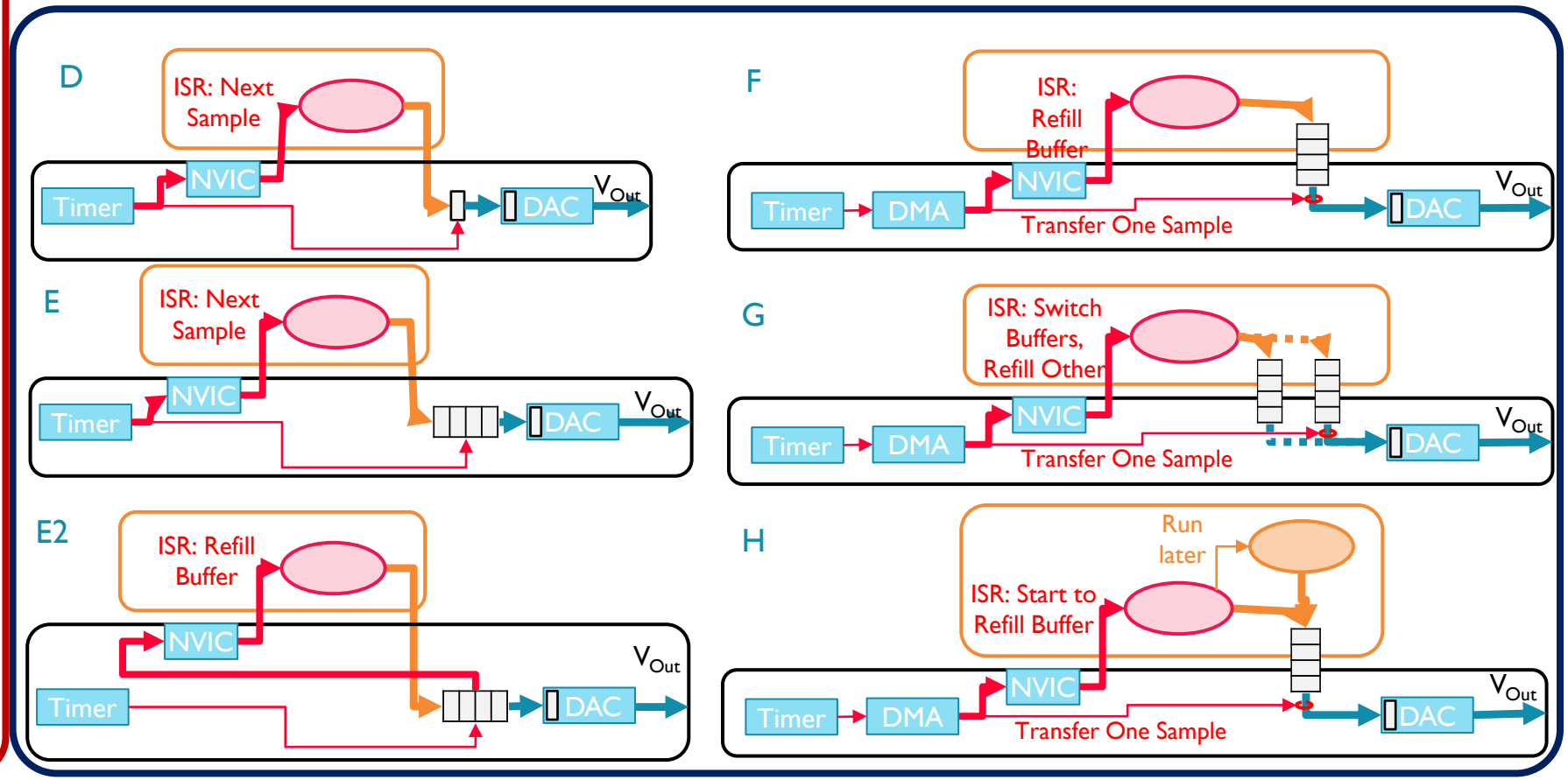
Software and Hardware Components in Design Evolution

***General Trend**: Move operations which need synchronization (update output, compute new value) from **software** to **hardware** to improve stability, performance*

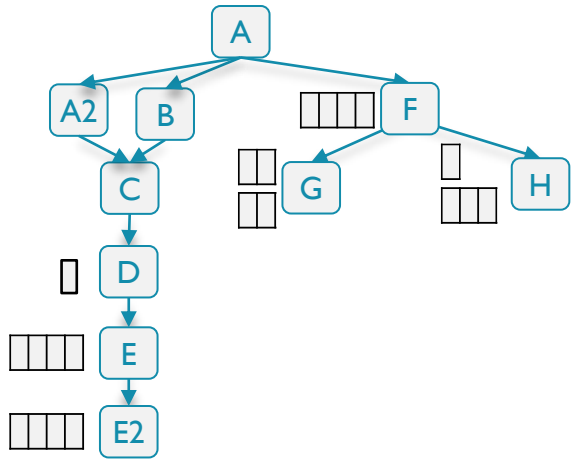
***Synchronous Output: Software write** to hardware **changes output**, so sloppy timing*



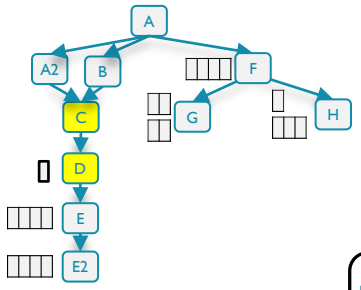
***Asynchronous Output:** Output is updated by hardware, so it is **decoupled** from (asynchronous with) **software execution**, so sloppy software timing affects output timing less*

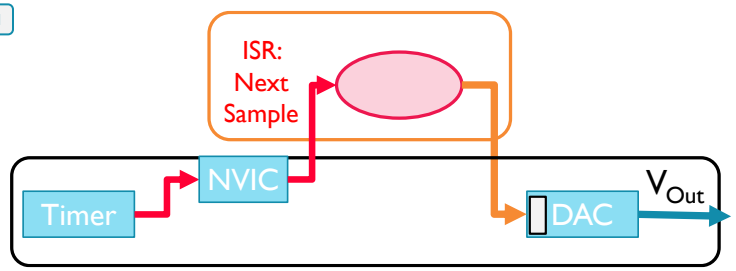


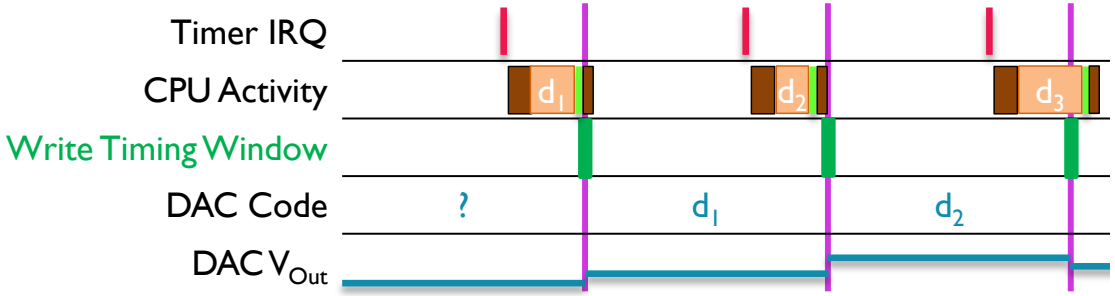
LOOSENING DEADLINES



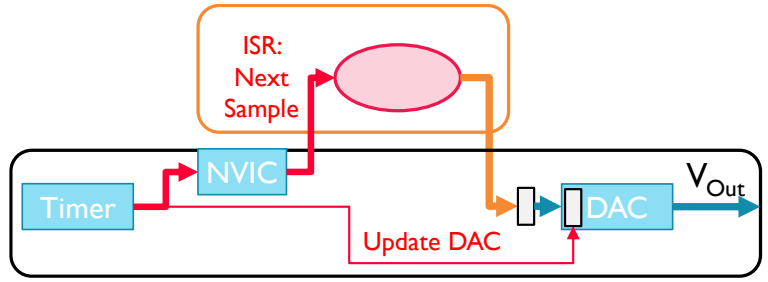
Reminder of Recent Designs C & D

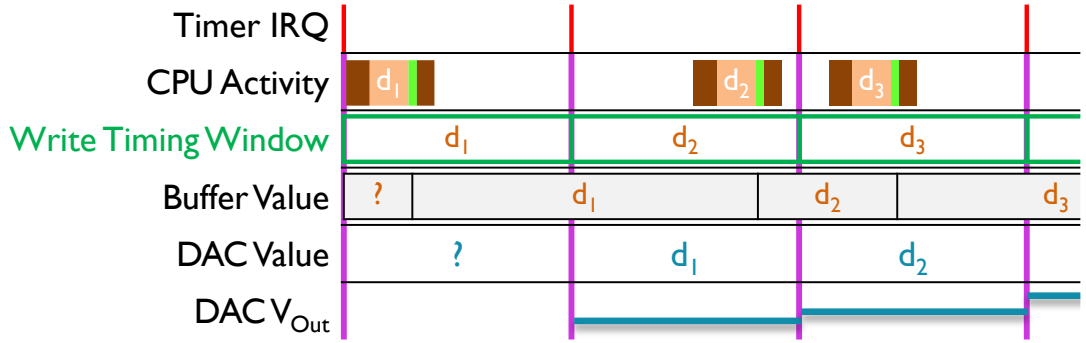






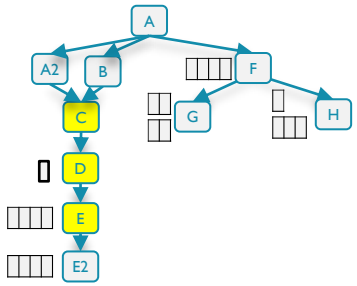
- C. Timer ISR
  - Output update is still synchronous with software, since write in ISR updates DAC
  - Better timing stability because ISR has priority and can preempt other code

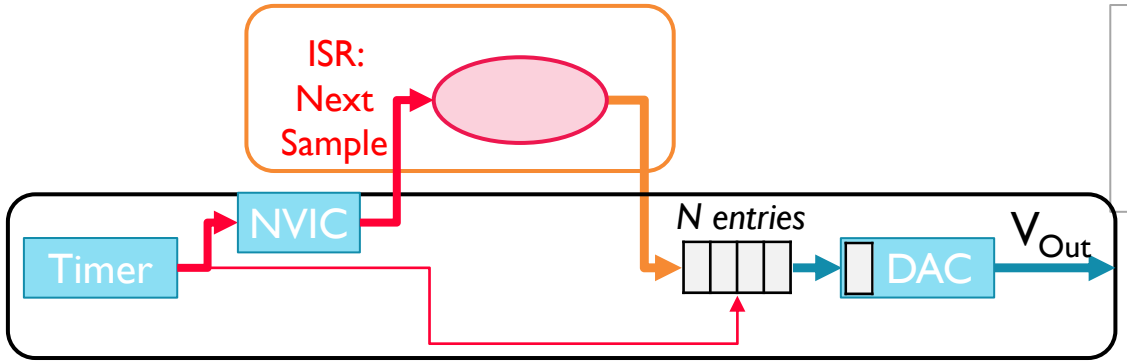




- D. Timer triggers DAC to copy data from single element buffer
  - Output update is now asynchronous with software, since timer signal updates DAC
  - Much better timing stability because of wider write timing window (T_{sample}), so easier to make software meet it
  - Improves timing stability by loosening deadlines. Can extend this approach.

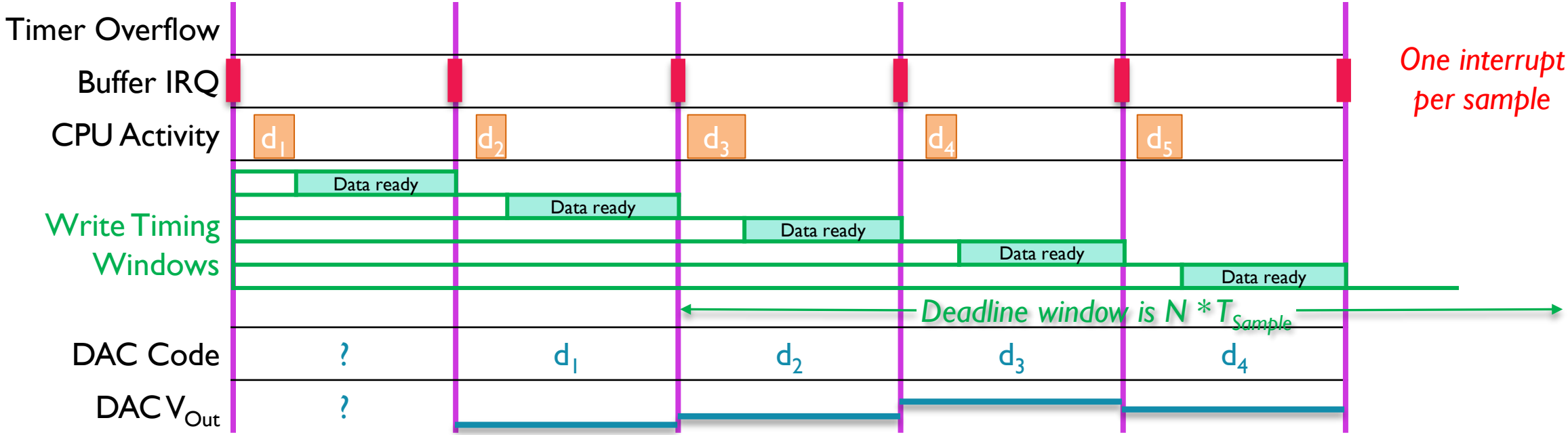
E. Add N-Item Buffer for DAC, Interrupt per Sample



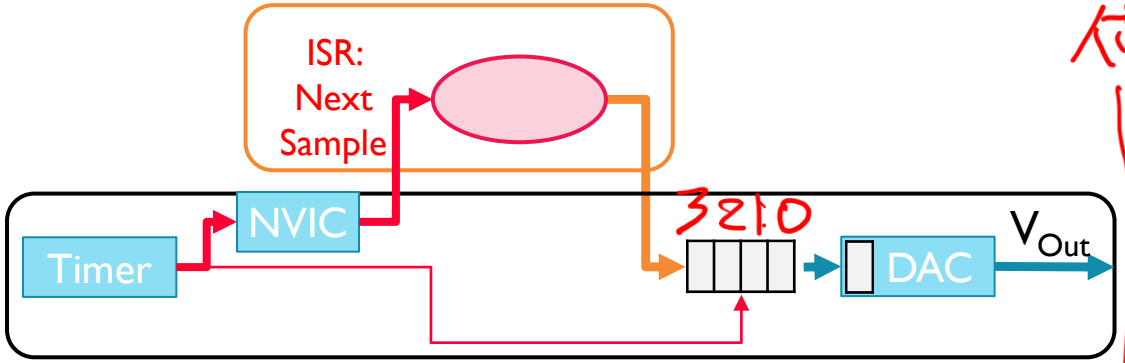


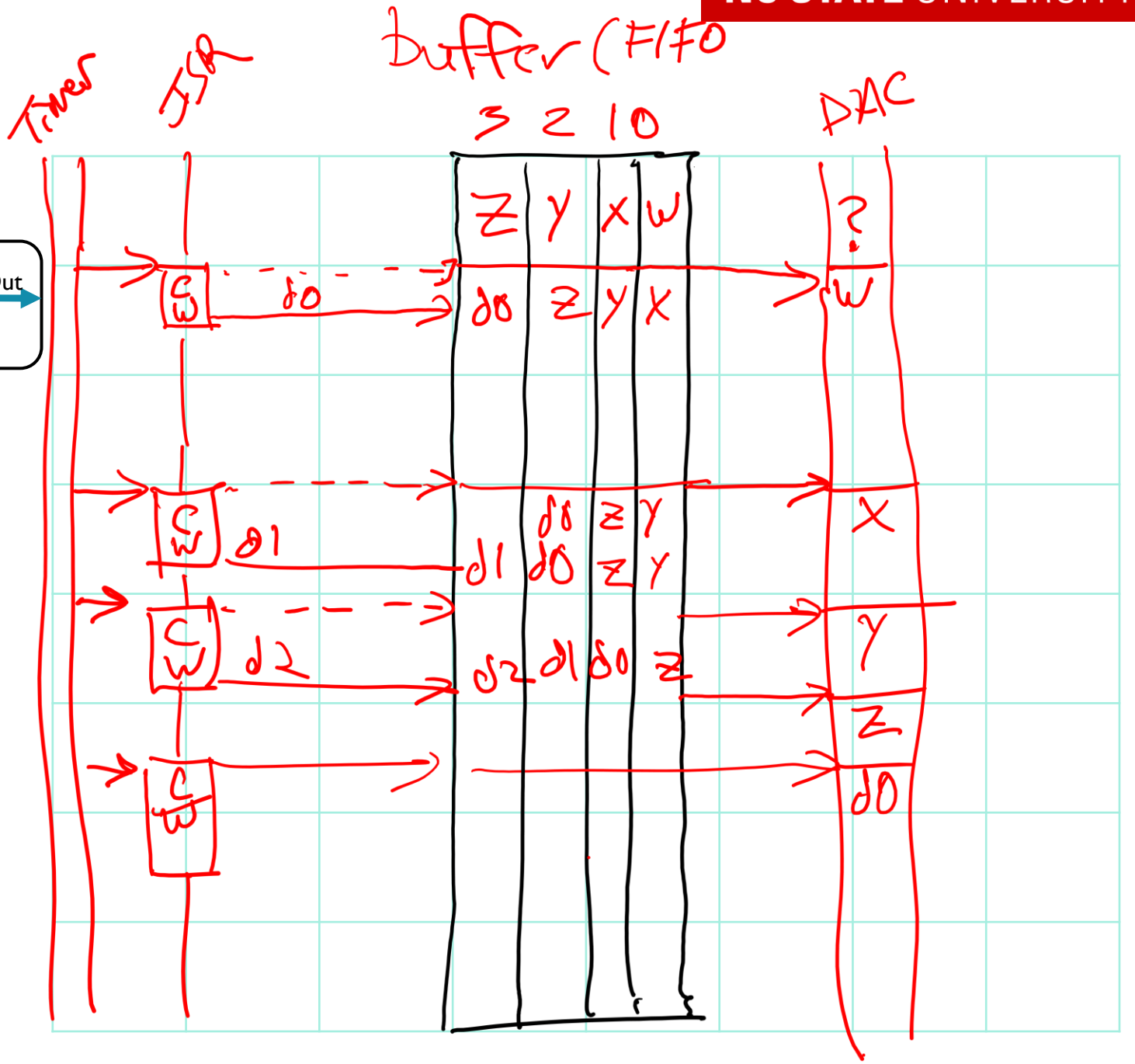

```
Timer ISR {
     compute next sample
     write sample to buffer
}
```

- Extend buffer to multiple-entry queue, aka FIFO (first-in, first-out)
- Note: is part of peripheral hardware design. More likely to be available in higher-performance peripherals.

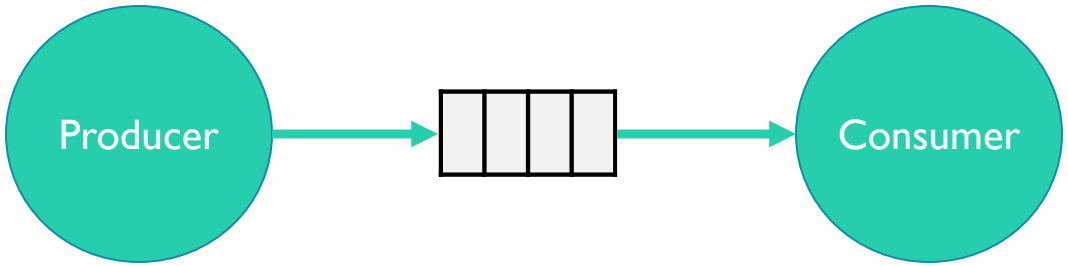


Sequence Diagram



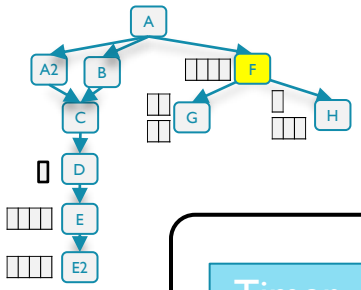


Concepts: Buffering Data for an Asynchronous Output



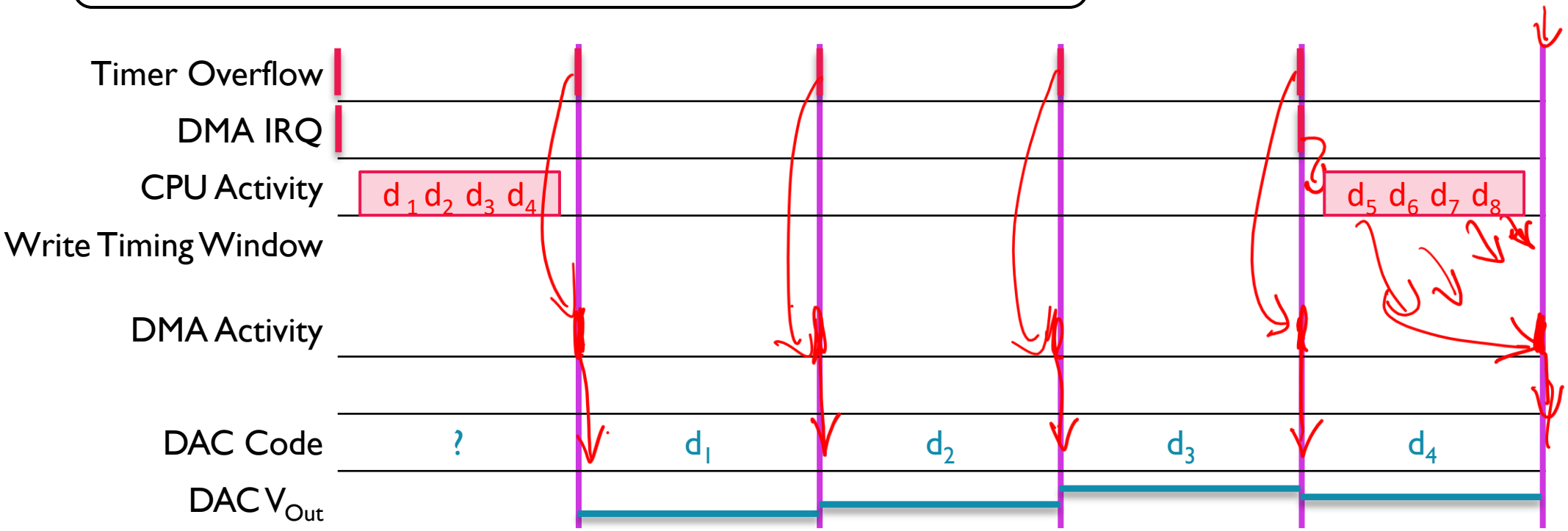
- Is example of **Asynchronous I/O**
  - Output data samples are **generated when ISR runs**
  - DAC output changes **when timer signal triggers update**
- Async I/O requires storing data temporarily
  - From **producer** generating the data (timer ISR)
  - Until **consumer** reads the data for use (timer overflow signal triggers DAC)
- Inherent limitations of buffering
  - Delays when data reaches output signal
  - Buffering and requires pre-computation, so not possible if outputs depend on future unknown data (e.g. control system)
- Producer, consumer must synchronize buffer accesses
  - Producer: don't add data if buffer is full
  - Consumer: don't read data if buffer is empty
  - Both: don't interfere with each other's buffer accesses.
    - Some buffer implementations have critical sections of code which must be protected
- Need deeper buffers as producer, consumer timing characteristics diverge
  - Timing tolerance mismatches:
    - Producer runs once per 100 us, but within 100 us window, so has timing **jitter** of up to 100 us
    - Output allows much less timing jitter (e.g. 10 us)
  - Peak data rate mismatch:
    - Producer's maximum rate > consumer's maximum rate
  - Timing uncertainty from blocking and preemption:
    - Consumer may not get run enough for unknown range of times (jitter for task completion time)

Reminder of Design F. Use Timer-Triggered DMA to Transfer Data

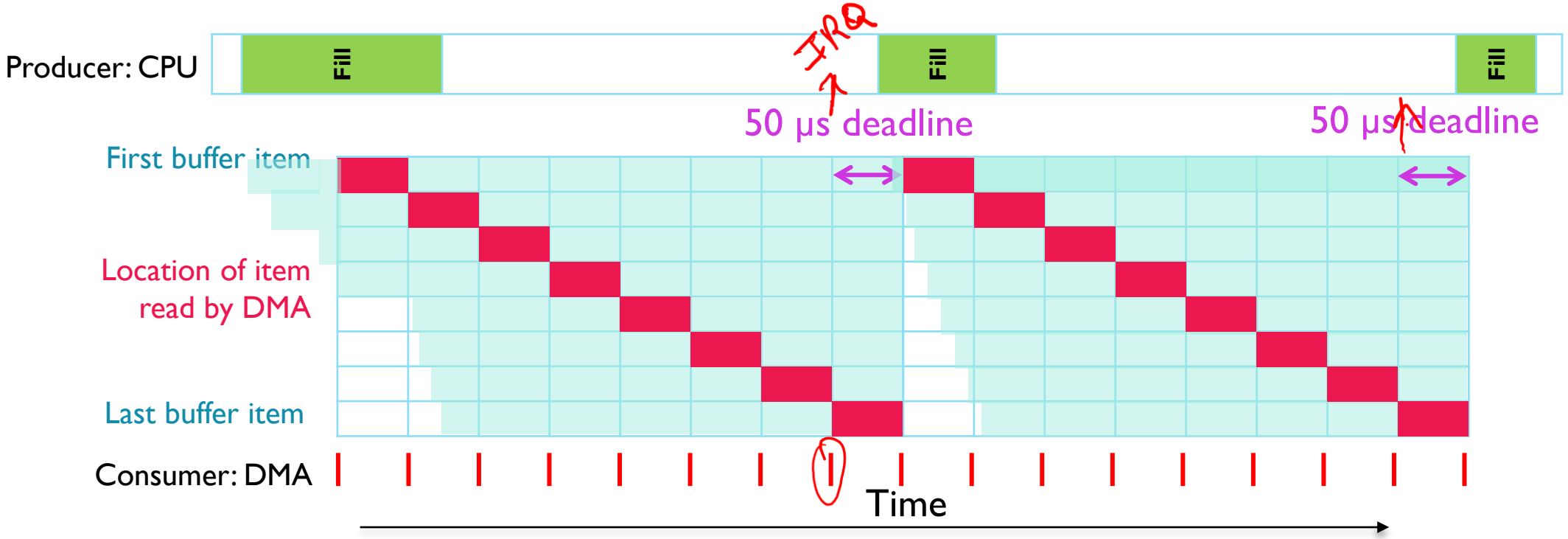



```
DMA ISR {
     Restart DMA channel
     Compute new data and refill buffer
}
```

ISR must save first new data value d_5 before next sample time

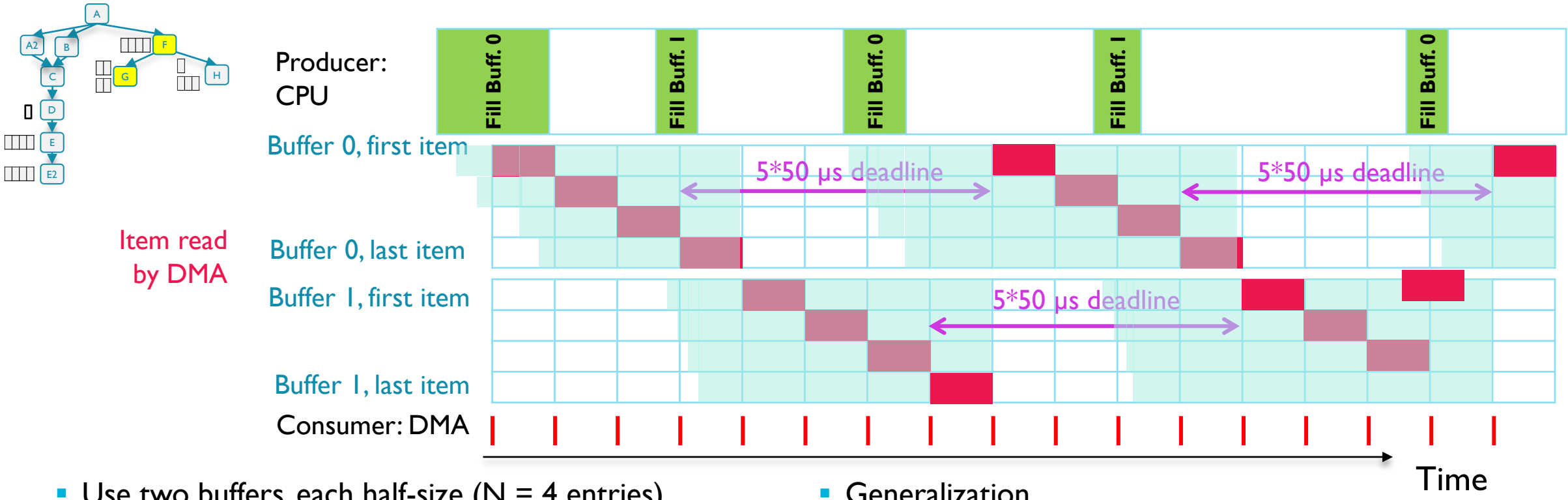


Examining Timing Requirements for Buffer Refill



- Behavior
  - CPU (producer) adds data to fill buffer
  - DMA (consumer) reads data items
  - Data item in buffer is not needed (old, stale) after being read by consumer
  - Producer (Thread_Refill_Sound_Buffer) must stay ahead of consumer (DMA controller)
- Behavior after DMA transfers last data item
  - DMA generates interrupt request
  - ISR contains producer
    - Must refill 1st buffer entry within T_{Sample}, 2nd entry within $2*T_{Sample}$, etc.
  - First deadline may be too tight to meet easily

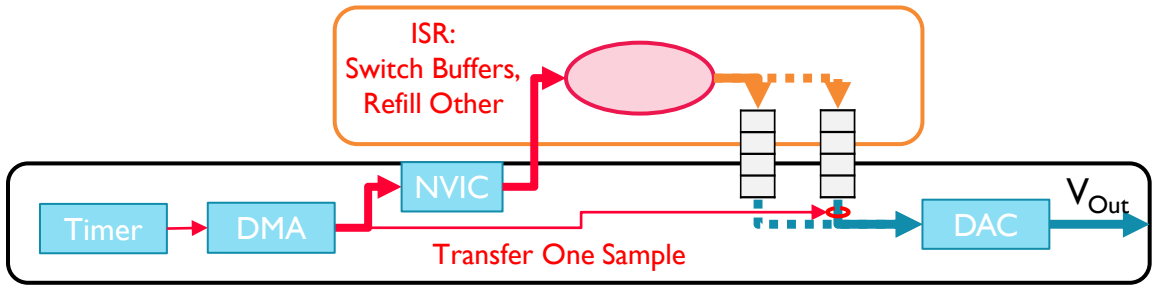
Loosening Timing Requirements with Double-Buffering



- Use two buffers, each half-size (N = 4 entries)
- Initialization
  - Start filling buffer 0
  - Can start playing buffer 0 after it has ≥1 sample
- After buffer 0 is filled, start filling buffer 1
- Operation: After playing last sample from buffer 0,
  - Switch to playing buffer 1
  - Start refilling buffer 0

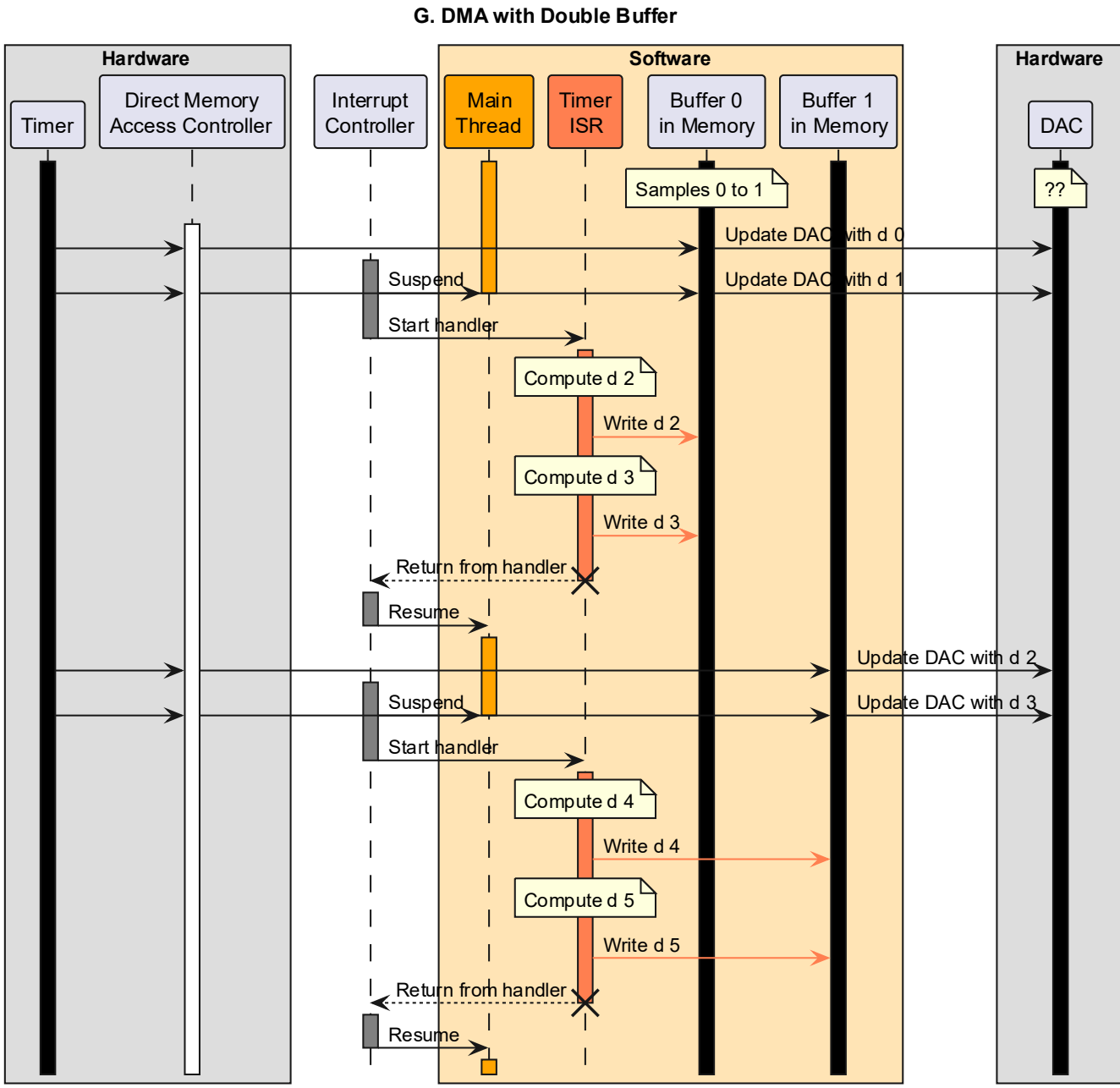
- Generalization
  - After playing last sample from buffer x, switch to playing buffer y, start filling buffer x
- Deadlines
  - Now have two deadlines, one per buffer ☹
  - Much looser deadlines: extended to from T_{Sample} to $(N+1)*T_{Sample}$ ☺

G. DMA with Double Buffering



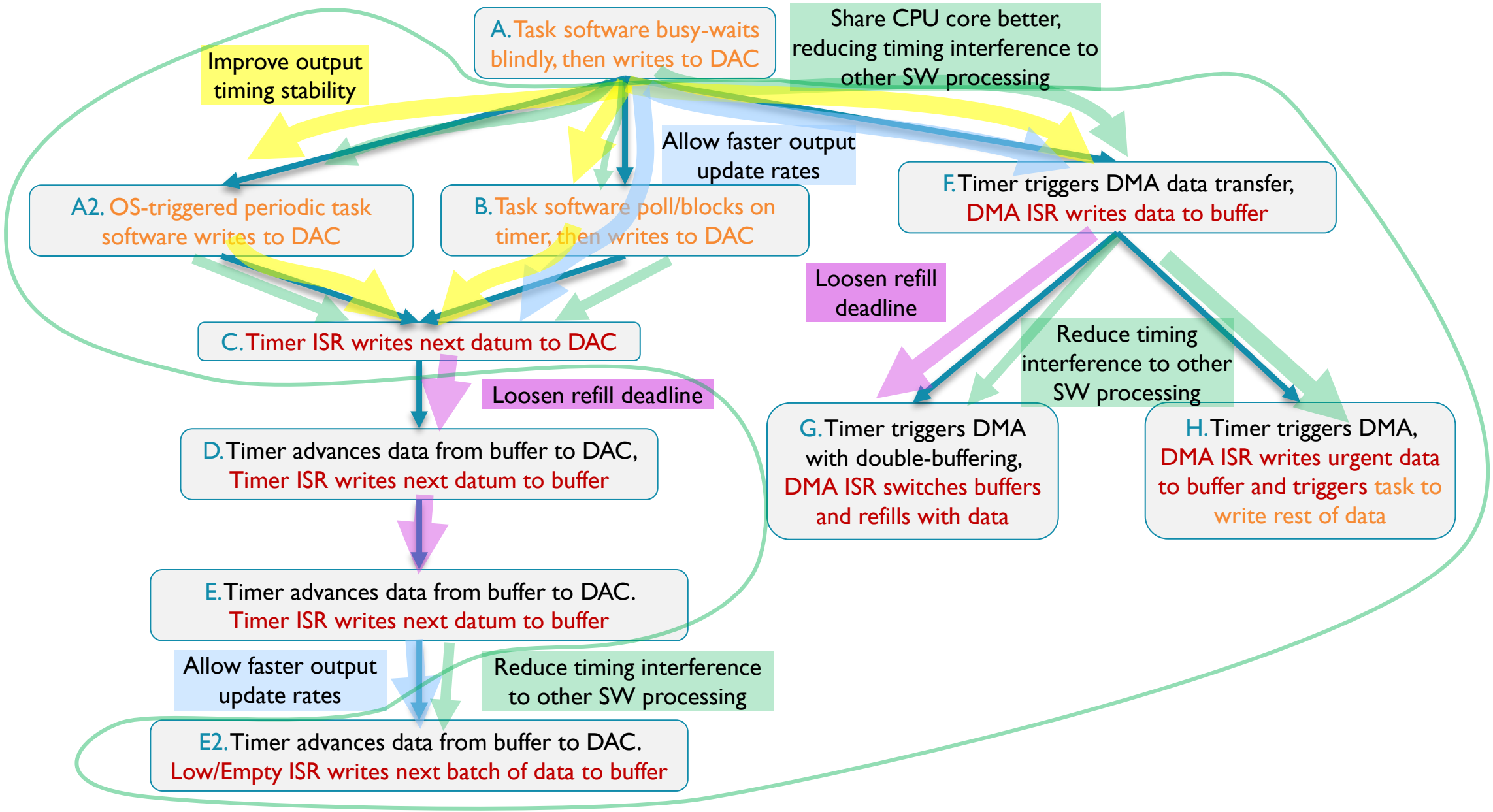

```
int rbn = 0; // reader's buffer number
uint16_t buffer[2][1024];
Initialization code prefills buffer[0]

DMA IRQ Handler {
    old_bn = rbn;
    rbn = 1 – rbn; // Switch buffers
    Restart DMA with buffer[rbn]
    Refill buffer[old_bn]
}
```

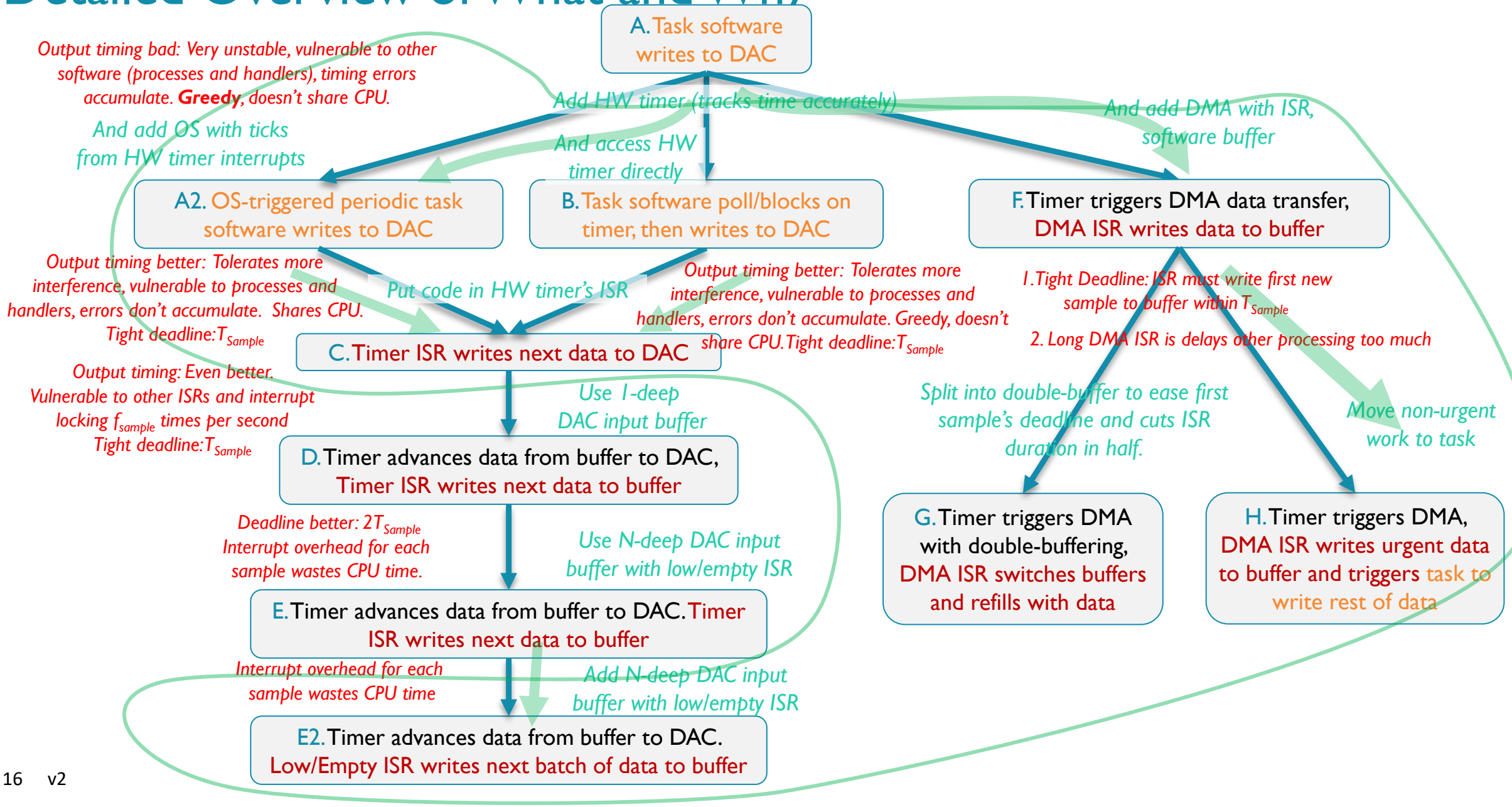


REDUCE TIMING INTERFERENCE FOR OTHER SW PROCESSING (SHARE CPU BETTER)

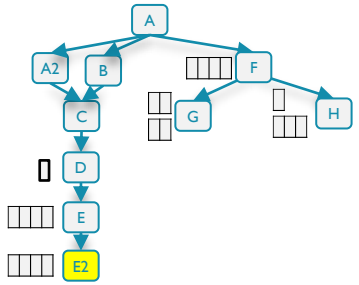
Overview of Waveform Generator Design Evolution: What and Why

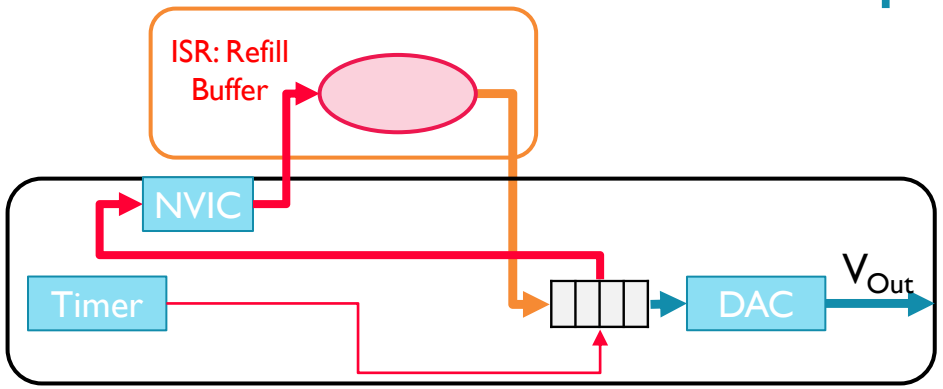


Detailed Overview of What and Why



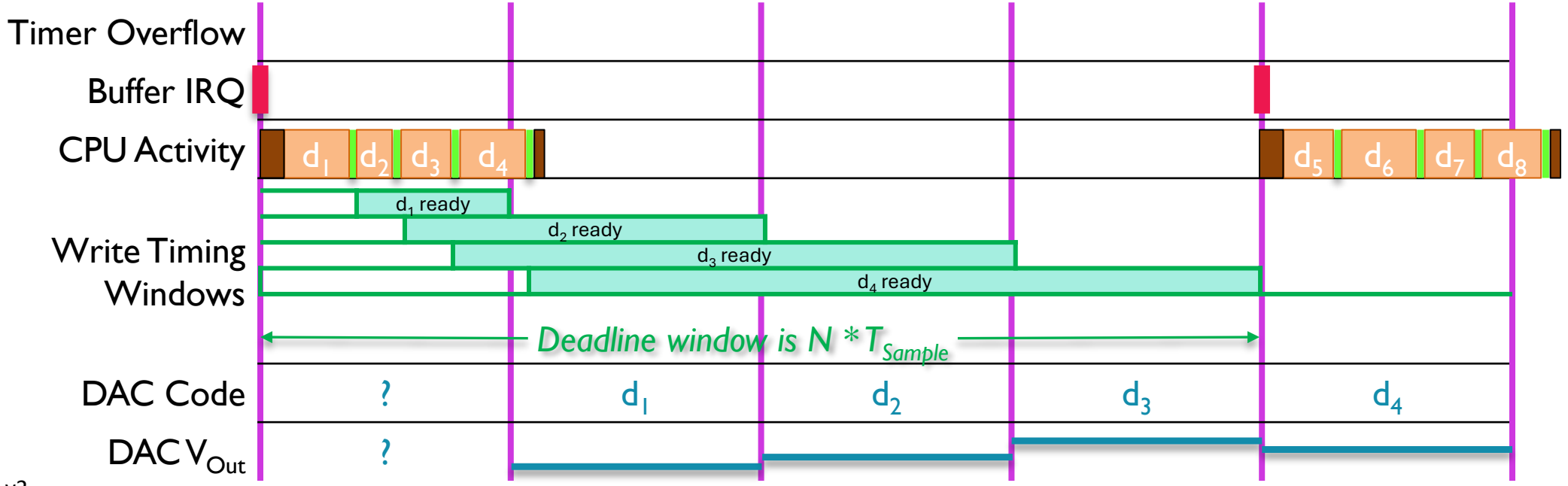
E2. DAC Buffer Generates Interrupt Only When **Buffer is Empty**



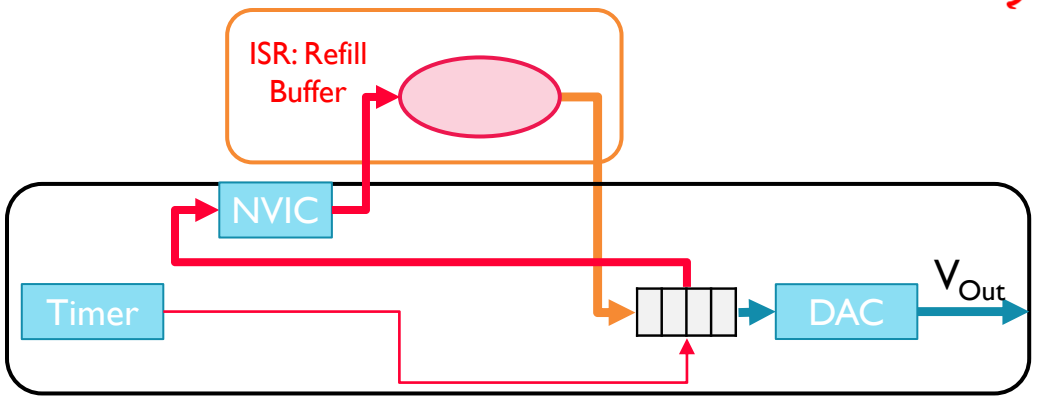


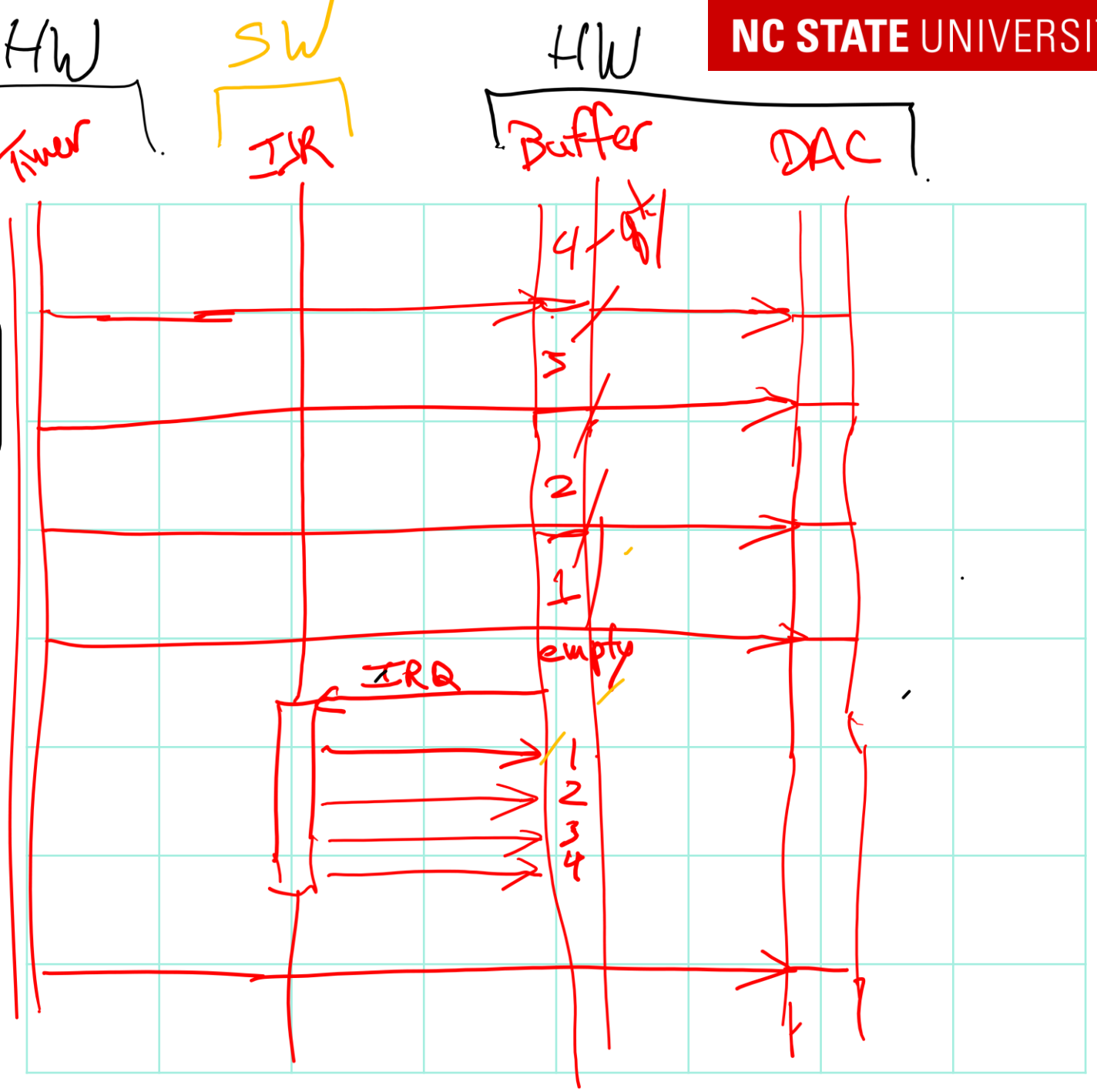

```
DAC Buffer ISR {
  while (buffer not full) {
    compute next sample
    write sample to buffer
  }
}
```

- Reduce timing interference to other processing by generating fewer interrupts. Burst/batch processing
- Divides number of interrupts by buffer size

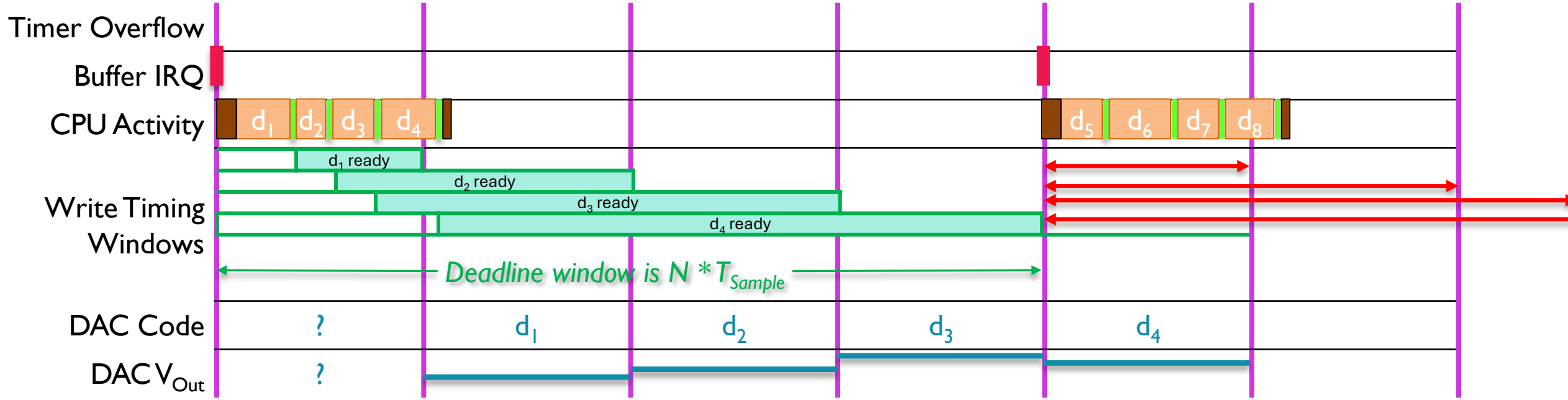


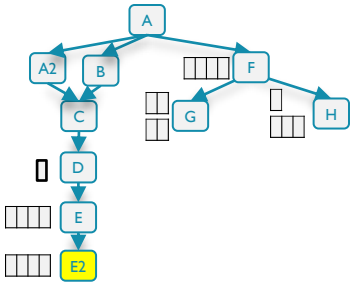
Sequence Diagram





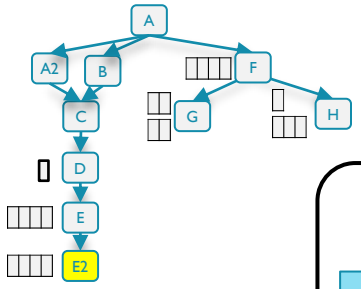
Tighter Timing Requirements for Refilling Buffer

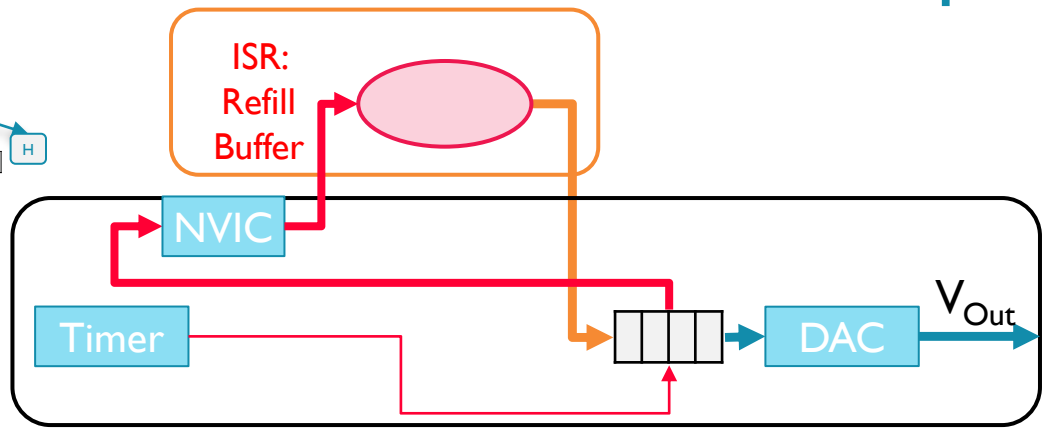




- First sample has deadline of T_{Sample}
- Second sample has deadline of $2*T_{Sample}$
- Third sample has deadline of $3*T_{Sample}$
- Deadline is shorter than window because interrupt occurs late – when **last sample** is read
- Same as problem with DMA design F (solved F with double buffering in G)
- Can DAC Buffer give us an **early warning interrupt** before the buffer is full?

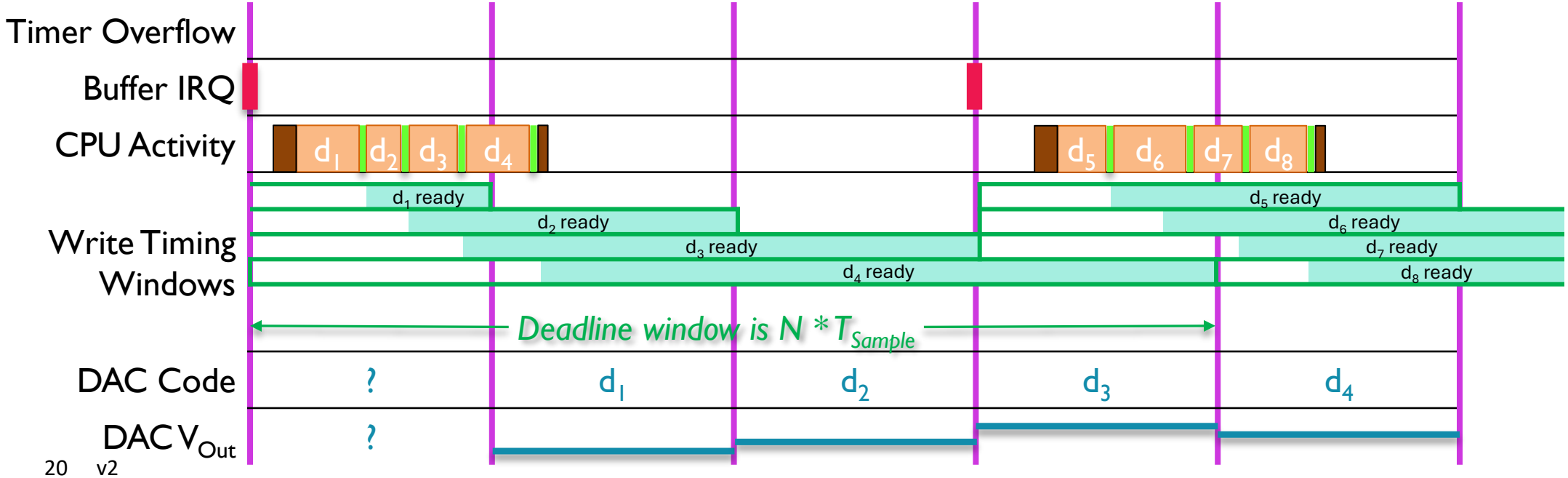
E3. DAC Buffer Generates Interrupt when Buffer is ¾ Empty



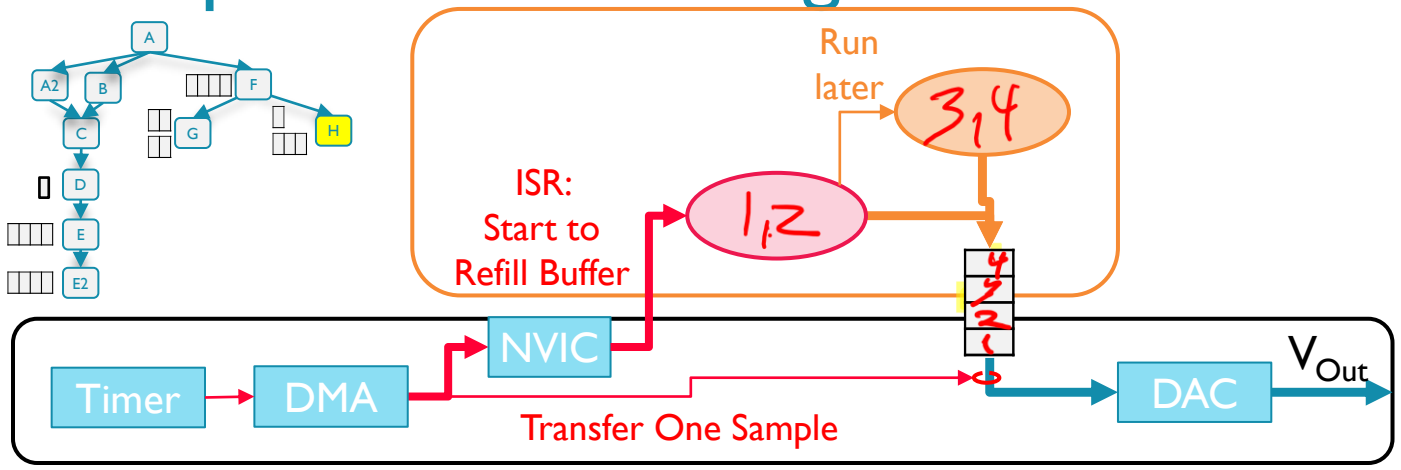



```
DAC Buffer ISR {
  while (buffer not full) {
    compute next sample
    write sample to buffer
  }
}
```

- Interrupt occurs (N/4)*T_{sample} earlier
- Deadline to refill first entry is now (1+N/4)*T_{Sample}
- Make sure that none of the last N/4 samples are overwritten
  - Must synchronize within the ISR to the DMA. **Put on Sync To Do List**



H. Split Work into Urgent and Deferred

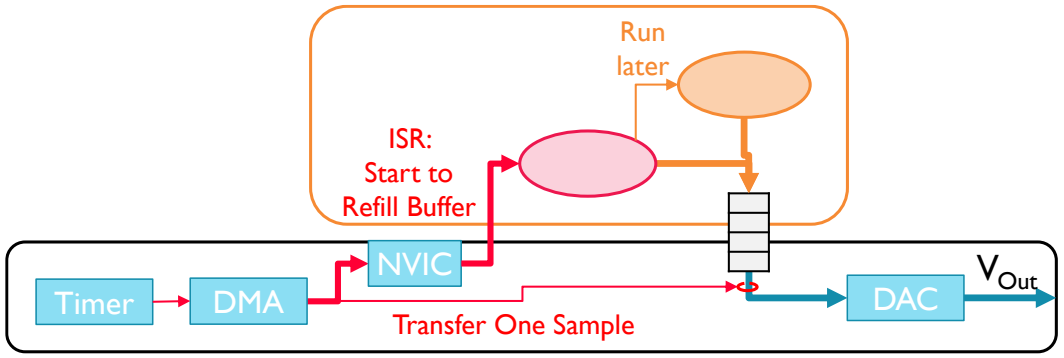


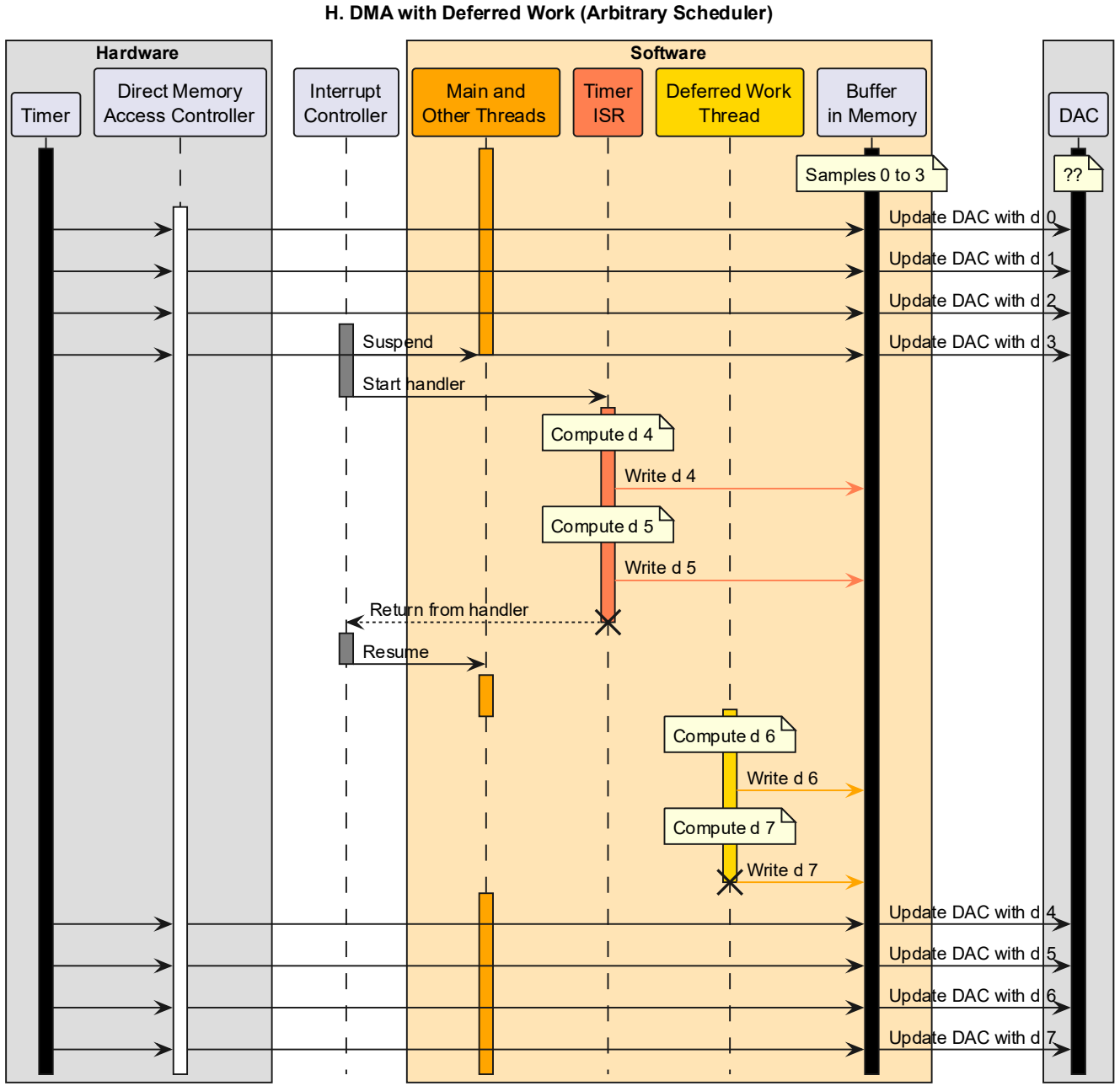

```
ISR {
 do urgent work now
 request deferred proc'g for rest of work
}

thread_Deferred_Work {
  do remaining work
}
```

- When refill buffer ISR runs, it can delay **ISRs** (with same or lower priority) and **all** threads
  - Want to reduce time spent in ISR to improve responsiveness for other software processing
- Observation: Don't need to refill **entire buffer** in ISR
  - Refilling first sample in buffer is highest urgency
  - Lower urgency for second, even lower for third…
- Procrastinate!
  - Change ISR to refill N most urgent samples (e.g. 1 & 2), and request thread to finish the work
  - Defer remaining BUFSIZE – N samples (e.g. 3 & 4) for thread to refill
- Depends on some form of thread scheduler to run the refill thread

Sequence Diagram





SUMMARY

Implementation Analysis

A. Task busy-waits for constant time (blind), then writes to DAC

A2. OS runs task periodically, task writes to DAC

B. Task software poll/blocks on timer, then writes to DAC

C. Timer ISR writes data to DAC

D. Timer advances buffer data to DAC, Timer ISR writes next data to buffer

E. Timer advances buffer data to DAC. Timer ISR writes data to buffer

E2. Timer advances buffer data to DAC. Low/Empty ISR writes next batch of data to buffer

F. Timer triggers DMA data transfer, DMA ISR writes next batch of data to buffer

G. Timer triggers DMA transfer, DMA ISR switches buffers and writes next batch of data to previous buffer

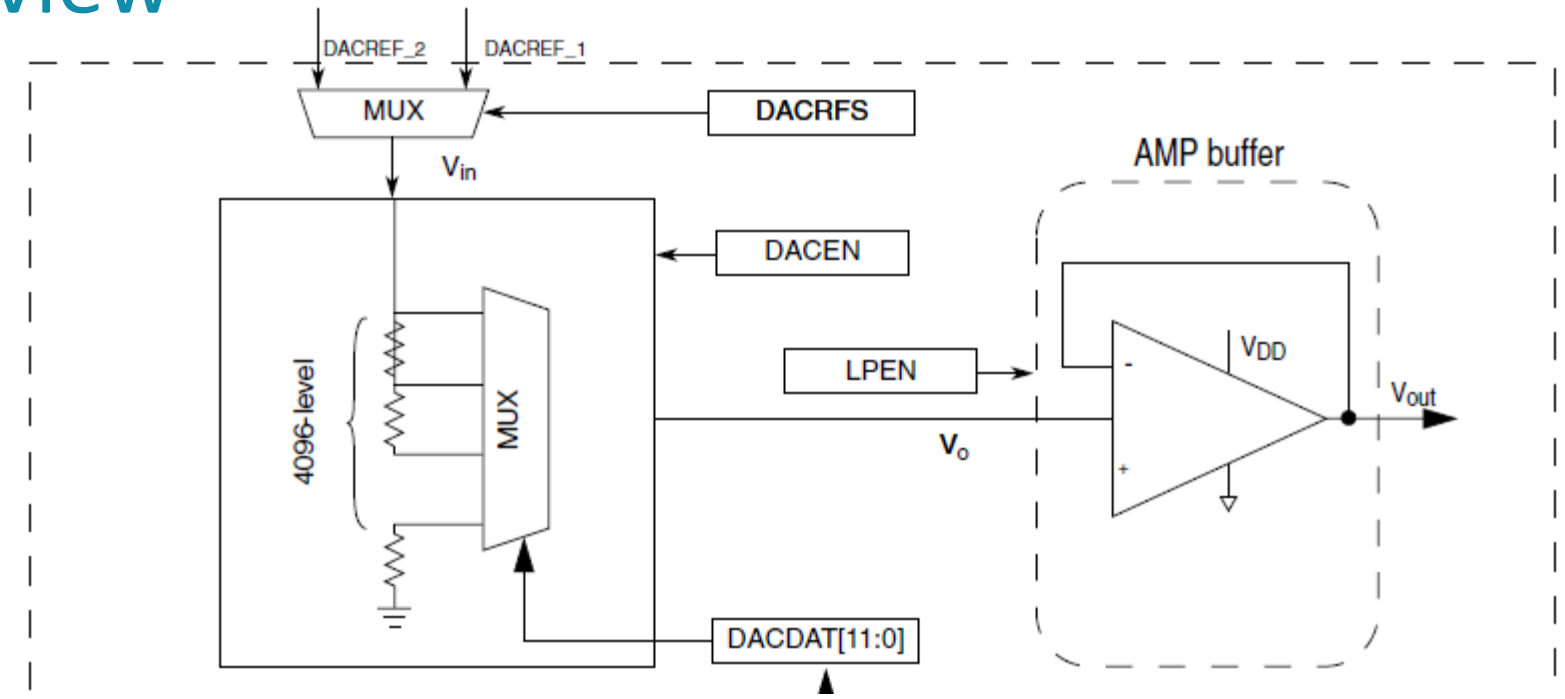
H. Timer triggers DMA transfer, DMA ISR writes urgent data to buffer, triggers task. Task writes rest of data batch

Implementation Comparison

Issue	A. SW -> DAC	B. SW Polls Timer	C. Timer ISR	D. Add single DAC buffer	E. Multi-Entry DAC Buffer	F. DMA Transfer	G. Double Buffering	H. Defer Some Work
CPU Sharing	Greedy	Greedy	Good					
Time Tracking	SW fixed busy-wait delay loop	SW polls HW timer	HW timer					
Resynch on next sample?	No	Yes	Yes					
Vulnerable to other tasks	Yes	Yes	No					
Vulnerable to ISRs, Handlers	Yes	Yes	Only higher-priority					
Width of timing window	T_{CPU}	T_{CPU}	T_{CPU}	T_{sample}				
Relative Deadline: from notification until updating first sample in buffer				T_{sample}				

DIGITAL TO ANALOG CONVERTER

DAC Overview



- Load DACDAT with 12-bit data N
- MUX selects a node from resistor divider network to create
  $V_o = (N+1)*V_{in}/2^{12}$
- V_o is buffered by output amplifier to create V_{out}
  - $V_o = V_{out}$ but V_o is high impedance - can't drive much of a load, so need to buffer it

DAC Registers

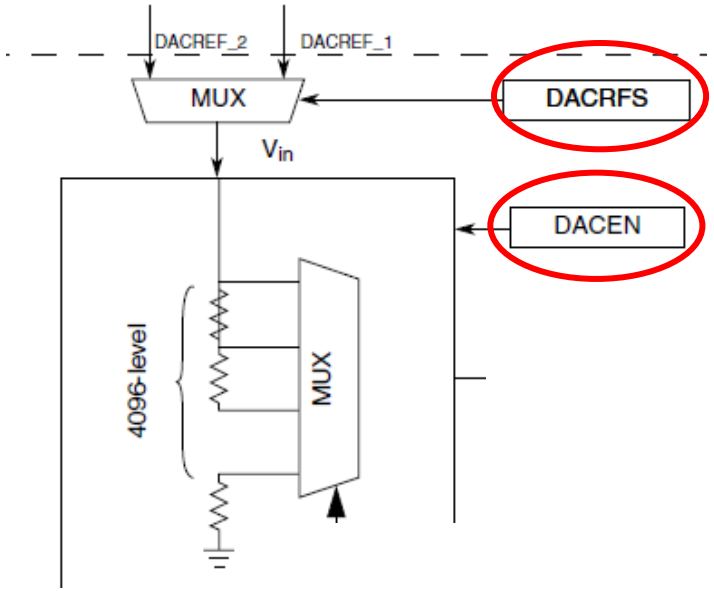
DAC memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
4003_F000	DAC Data Low Register (DAC0_DAT0L)	8	R/W	00h	30.4.1/531
4003_F001	DAC Data High Register (DAC0_DAT0H)	8	R/W	00h	30.4.2/532
4003_F002	DAC Data Low Register (DAC0_DAT1L)	8	R/W	00h	30.4.1/531
4003_F003	DAC Data High Register (DAC0_DAT1H)	8	R/W	00h	30.4.2/532
4003_F020	DAC Status Register (DAC0_SR)	8	R	02h	30.4.3/532
4003_F021	DAC Control Register (DAC0_C0)	8	R/W	00h	30.4.4/533
4003_F022	DAC Control Register 1 (DAC0_C1)	8	R/W	00h	30.4.5/534
4003_F023	DAC Control Register 2 (DAC0_C2)	8	R/W	0Fh	30.4.6/534

- This peripheral's registers are only eight bits long (legacy peripheral).
- DATA[11:0] stored in two registers
  - DATA0: Low byte [7:0] in DACx_DATnL
  - DATA1: High nibble [11:0] in DACx_DATnH

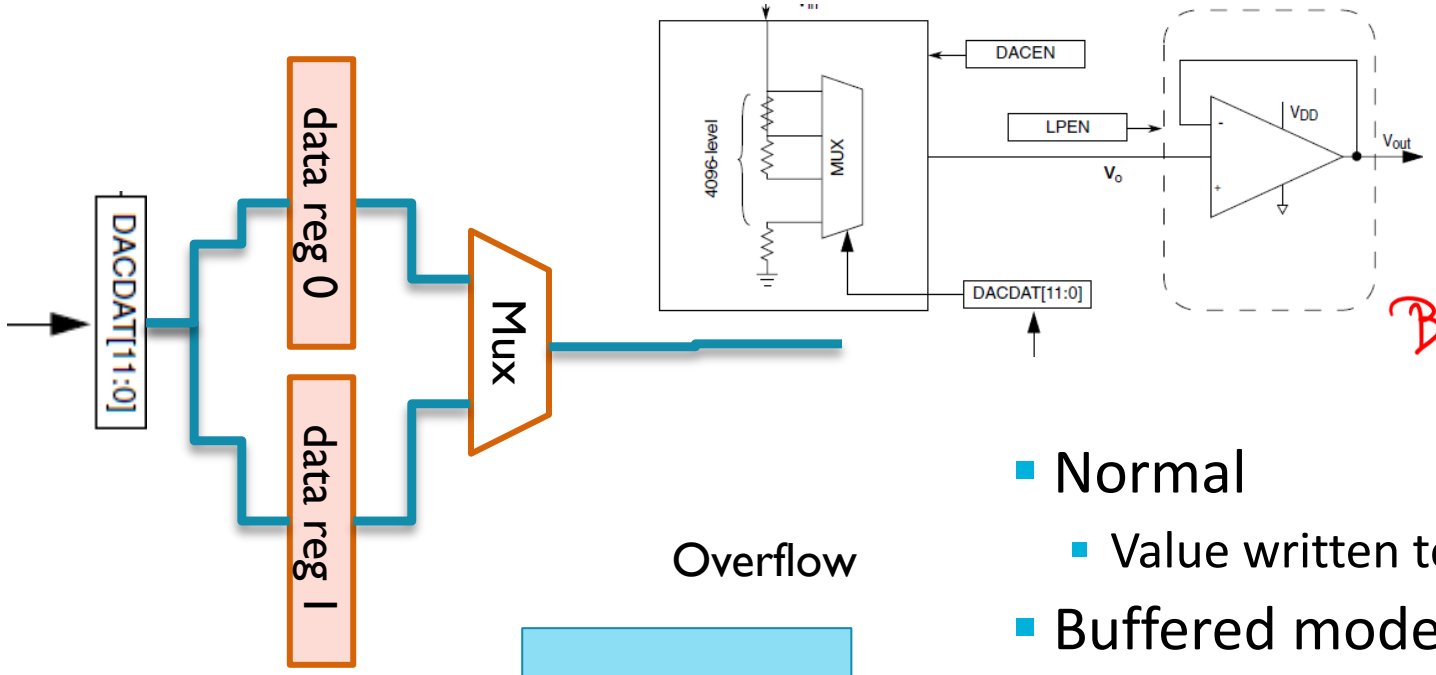
DAC Control Register 0: DACx_C0

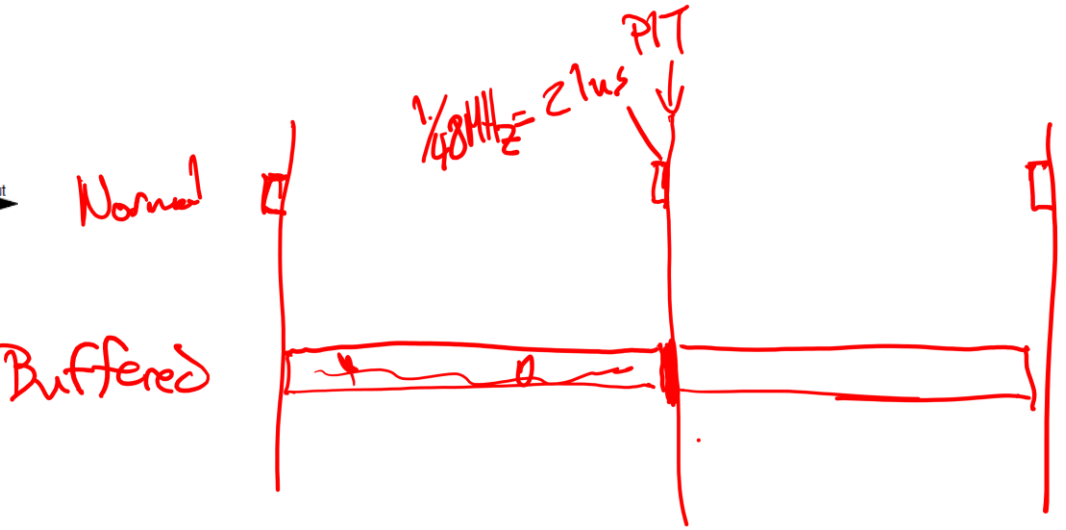
Bit	7	6	5	4	3	2	1	0
Read	DACEN	DACRFS	DACTRGSEL	0	LPEN	0	DACBTIEN	DACBBIEN
Write				DACSWTRG				
Reset	0	0	0	0	0	0	0	0



- DACEN - DAC Enabled when 1
- DACRFS - DAC reference voltage select
  - 0: DACREF_1. Connected to VREFH
  - 1: DACREF_2. Connected to VDDA
- LPEN - low-power mode
  - 0: High-speed mode. Fast (15 us settling time) but uses more power (up to 900 uA supply current)
  - 1: Low-power mode. Slow (100 us settling time) but more power-efficient (up to 250 uA supply current)
- Additional control registers used for buffered mode

DAC Operating Modes



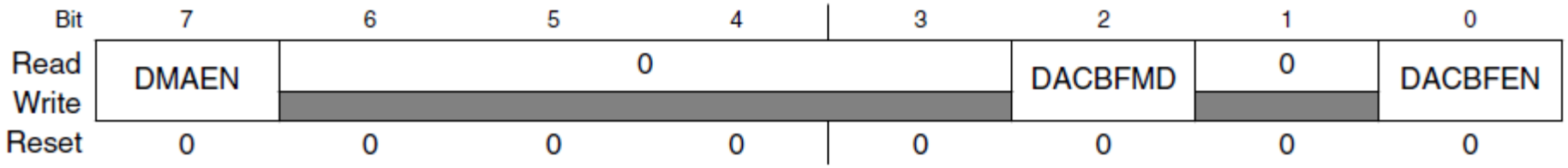


Overflow

Timer (PIT)

- Normal
  - Value written to DACDAT is converted to voltage immediately
- Buffered mode eases timing requirements
  - Value written to DACDAT is stored in data buffer for later conversion
  - Next data item is sent to DAC when triggered
    - Software Trigger - write to DACSWTRG field in DACx_C0
    - Hardware Trigger - from PIT timer peripheral
  - Normal Mode: Circular buffer
  - One-time Scan Mode: Pointer advances, stops at end of buffer
  - Status flags in DACx_SR

DAC Control Register 1: DACx_C1

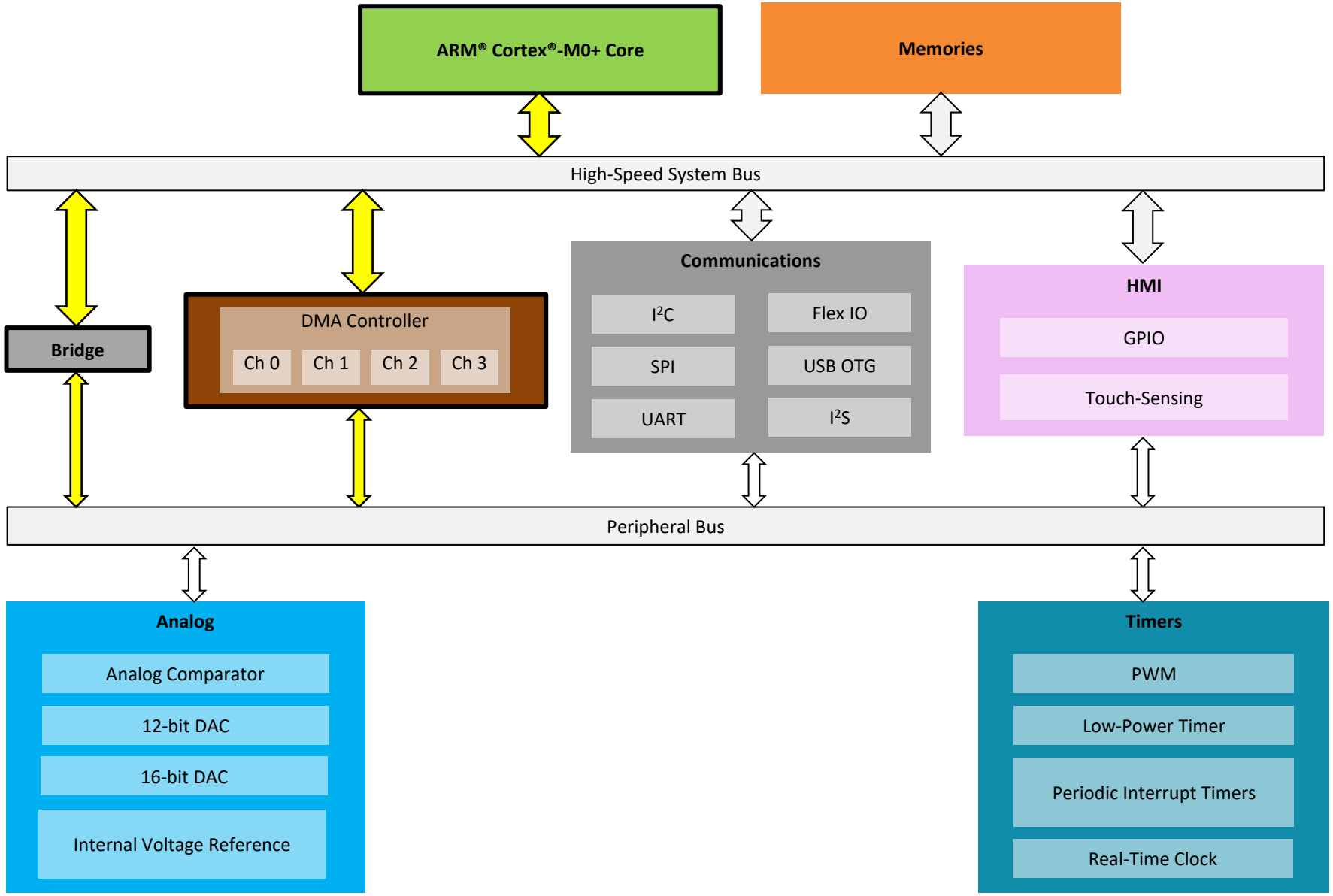


Bit	7	6	5	4	3	2	1	0
Read	DMAEN	0				DACBFMD	0	DACBFEN
Write								
Reset	0	0	0	0	0	0	0	0

- DACBFEN
  - 0: Disable buffer mode
  - 1: Enable buffer mode
- DACBFMD - Buffer mode select
  - 0: Normal mode (circular buffer)
  - 1: One-time scan mode

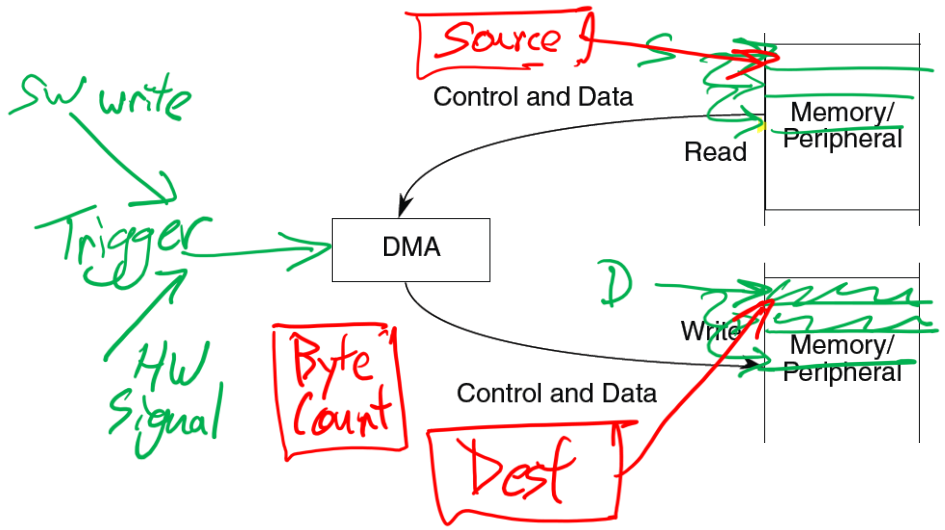
Direct Memory Access (DMA) Controller

DMA Can Read and Write Memory and Peripheral Registers



Basic Concepts

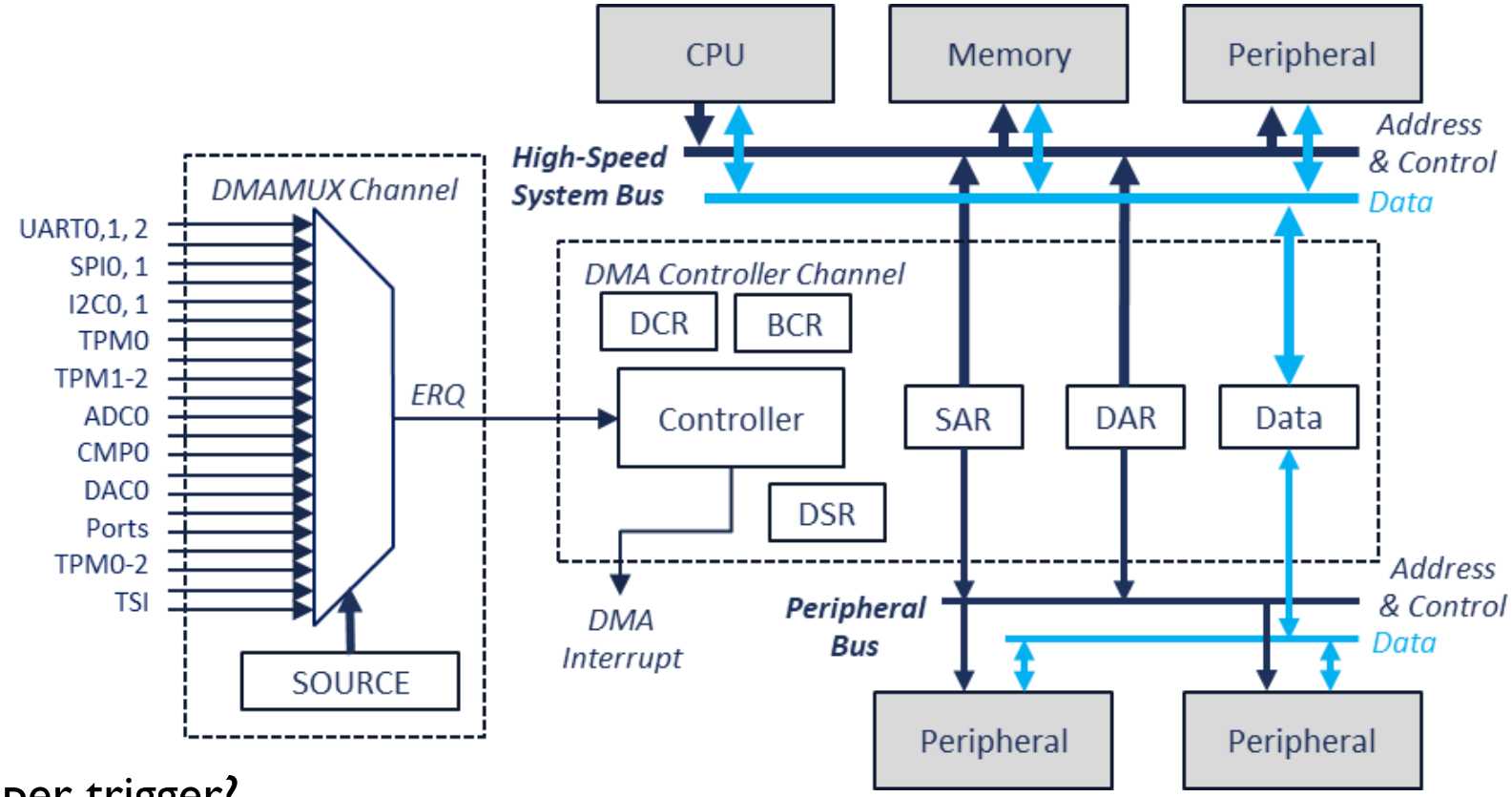
- Memory copy machine built from hardware
  - Reads data from source, writes it to destination
  - Can eliminate ISRs which just copy data (e.g. copy ADC results into buffer)
- Key configurable options
  - What event starts a transfer: software or hardware triggers
  - Source and destination addresses, which can be fixed or change (e.g. increment, decrement)
  - Number of data items to copy
  - Size of data item (1, 2, 4 bytes)
  - More features too: chaining, error handling…
- Sequence of operations
  - Initialization: Configure controller
  - Transfer: Data is copied
  - Termination: Channel indicates transfer has completed (status flag, interrupt request, DMA request)



Address Modification		Data Transfer Behavior
Source	Destination	
fixed	fixed	Write value from fixed source location into fixed destination location
changes	changes	Copy values from source array to destination array
fixed	changes	Write value from fixed source location into array
changes	fixed	Write array contents to fixed destination location

DMA Controller Details

- 4 independent channels
  - Channel 0 has highest priority
- 8-, 16- or 32-bit transfers,
  - Data size can differ between source and destination
- Circular/ring buffer support
  - "Address Modulo:" address wraps around at end of buffer
  - Buffer sizes from 64 B to 256 kB (2^N)
- DMA MUX peripheral selects hardware signal for triggering
- How many data items are transferred per trigger?
  - One: "Cycle stealing"
  - All items: grabs bus
- Hardware acknowledge/done signal

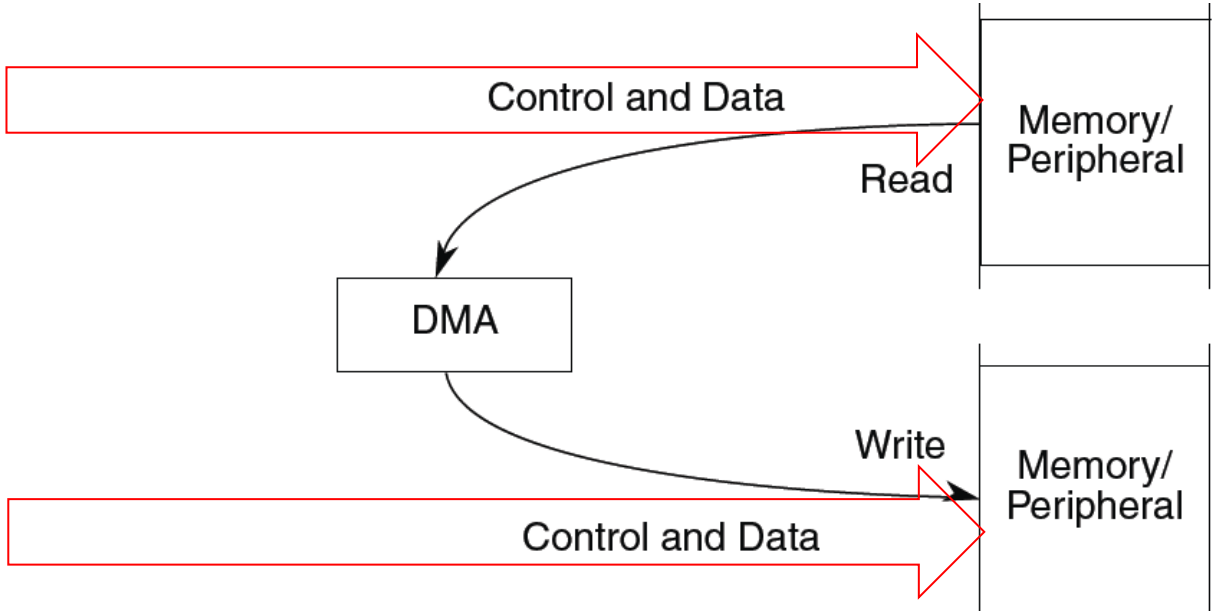


Address Registers

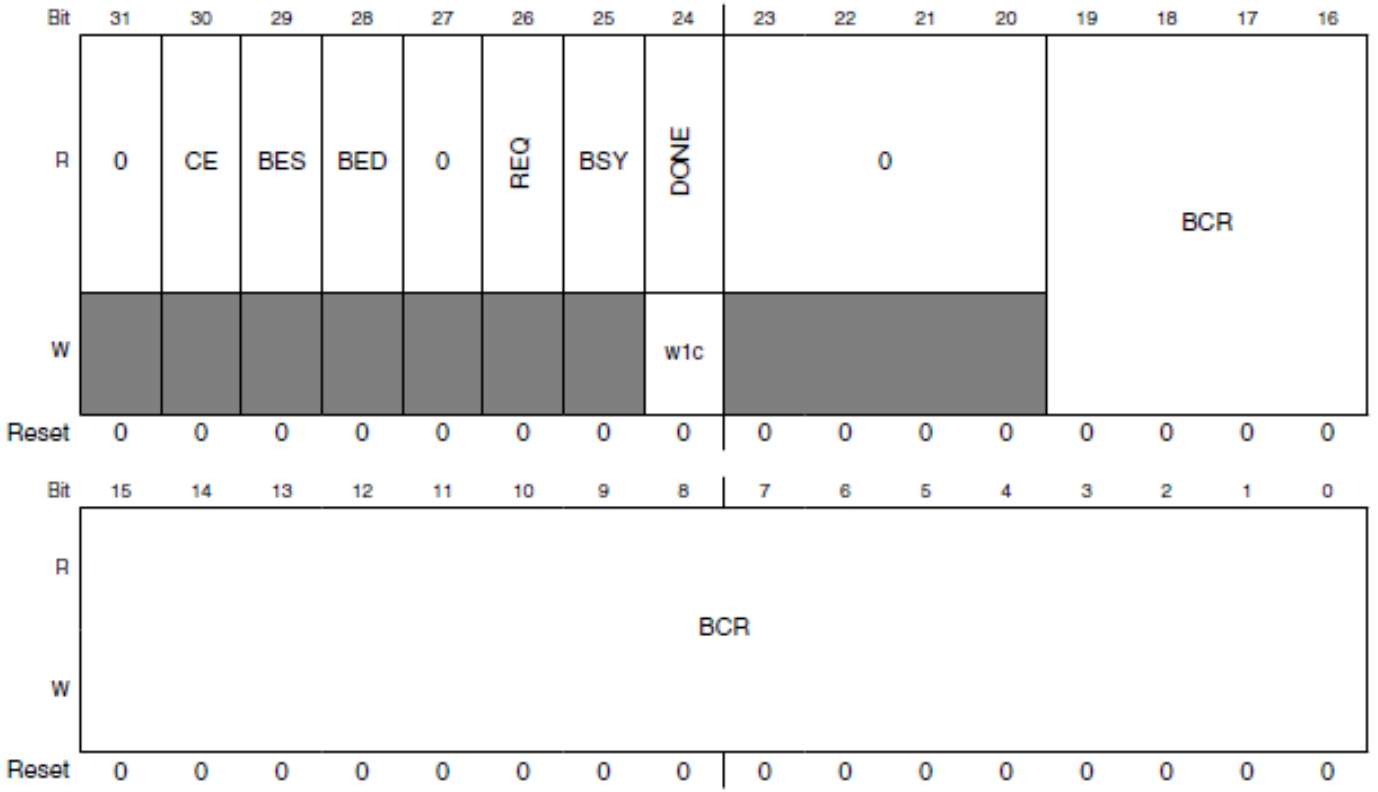
- These registers determine from where the DMA will read its data, and to where it will write that data

- DMA_SARn
  - Source address register,
  - Valid values 0 to 0x000f ffff

- DMA_DARn
  - Destination address register
  - Valid values 0 to 0x000f ffff

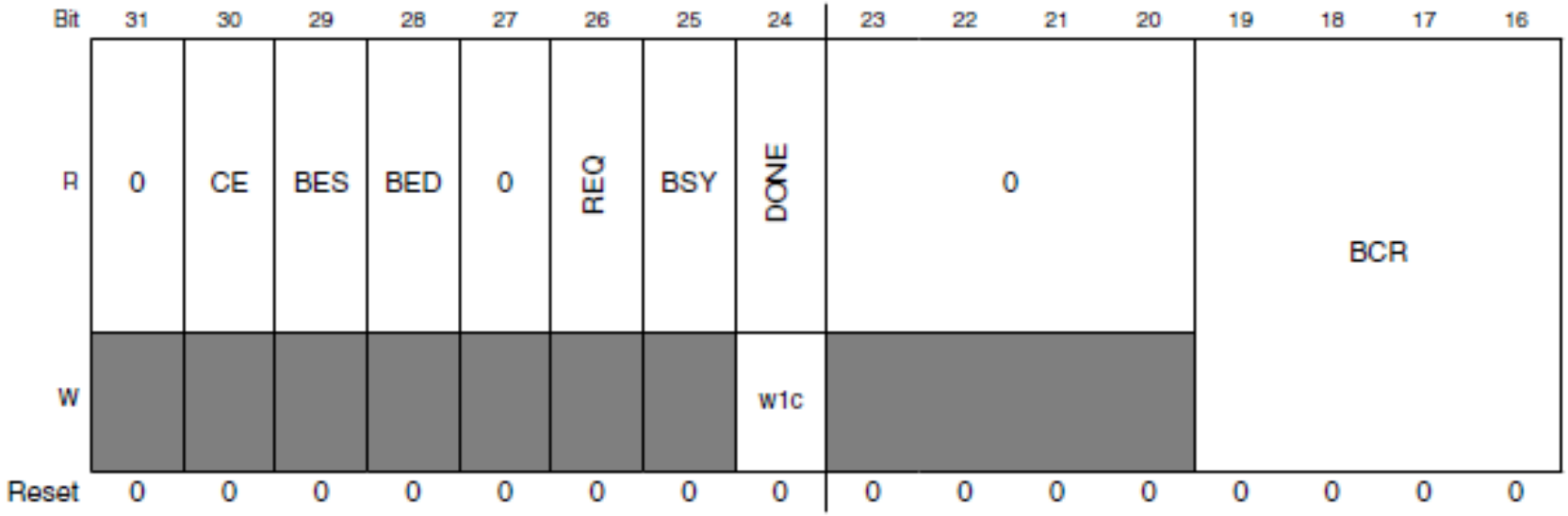


Byte Count Register



- BCR: Bytes remaining to transfer
- Decremented by 1, 2 or 4 after completing write (determined by destination data size)

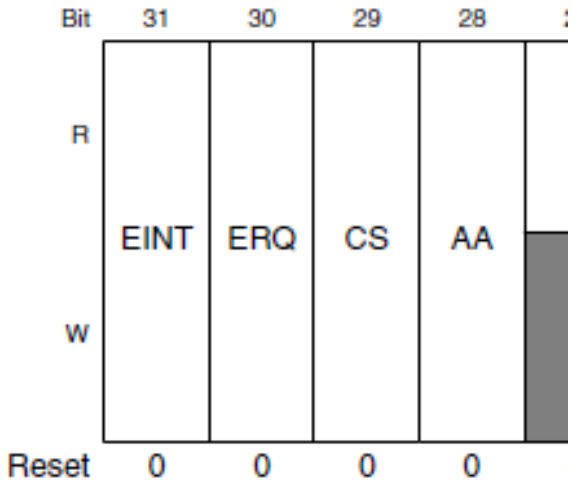
Status Register/Byte Count Register DMA_DSR_BCRn



Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	CE	BES	BED	0	REQ	BSY	DONE	0				BCR			
W								w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

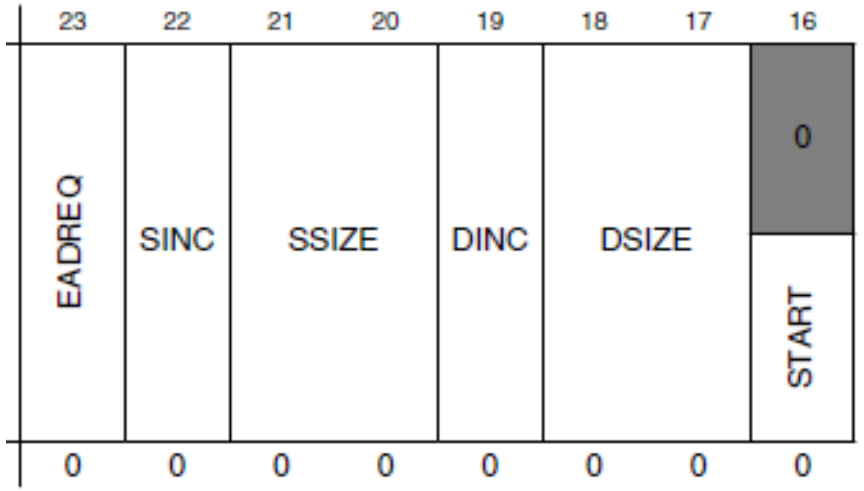
- Status flags: 1 indicates error
  - CE: Configuration error
  - BES: Bus error on source
  - BED: Bus error on destination
  - REQ: A transfer request is pending (more transfers to perform)
  - BSY: DMA channel is busy
  - DONE: Channel transfers have completed or an error occurred. Clear this bit in an ISR.

DMA Control Register (DMA_DCRn)



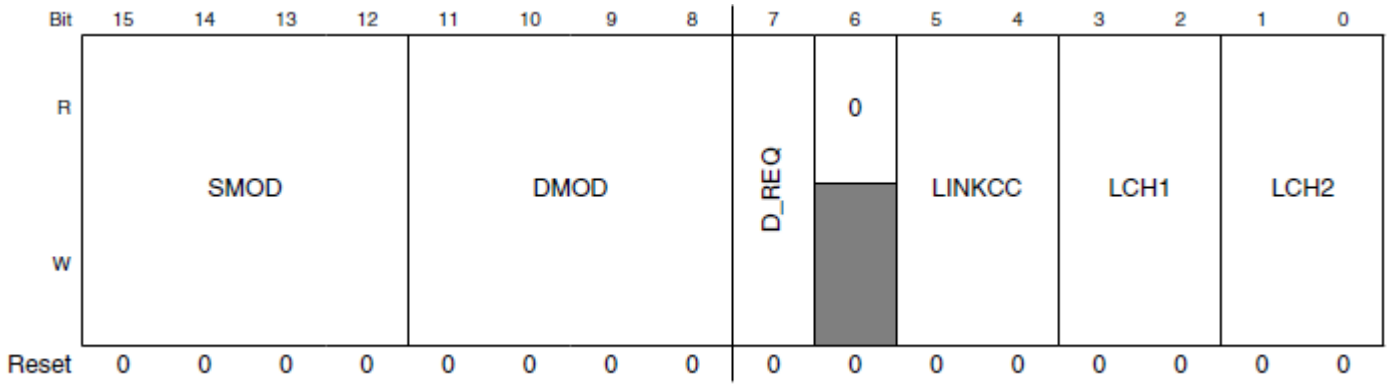
- EINT: Enable interrupt on transfer completion
- ERQ: Enable peripheral request to start transfer
- CS: Cycle steal
  - 0: Greedy - DMA makes continuous transfers until BCR == 0
  - 1: DMA shares bus, performs only one transfer per request
- AA: Autoalign

DMA Control Register (DMA_DCRn)



- EADRQ – Enable asynchronous DMA requests when MCU is in Stop mode
- SINC/DINC – Increment SAR/DAR by 1,2 or 4 based on size of data
- SSIZE/DSIZE – Source/Destination data size.
  - Don't need to match – controller will perform extra reads or writes as needed (e.g. read one word, write two bytes).
  - 00: longword (32 bits)
  - 01: byte (8 bits)
  - 10: word (16 bits)
- START – Write 1 to start transfer

DMA Control Register (DMA_DCRn)



- SMOD, DMOD – Source/Destination address modulo
  - When non-zero, supports circular data buffer – address wraps around after 2^{n+3} bytes (16 bytes to 64 kilobytes)
  - When zero, circular buffer is disabled
- D_REQ: If 1, then when BCR reaches zero, channel will clear ERQ bit, preventing further hardware triggers of channel
- LINKCC: Enables this channel to trigger another channel
  - 00: Disabled
  - 01: Two stages:
    - Link to channel LCH1 after each cycle-steal transfer
    - Link to channel LCH2 after BCR reaches 0
  - 10: Link to channel LCH1 after each cycle-steal transfer
  - 11: Link to channel LCH1 after BCR reaches 0
- LCH1, LCH2: Values 00 to 11 specify linked DMA channel (0 to 3)

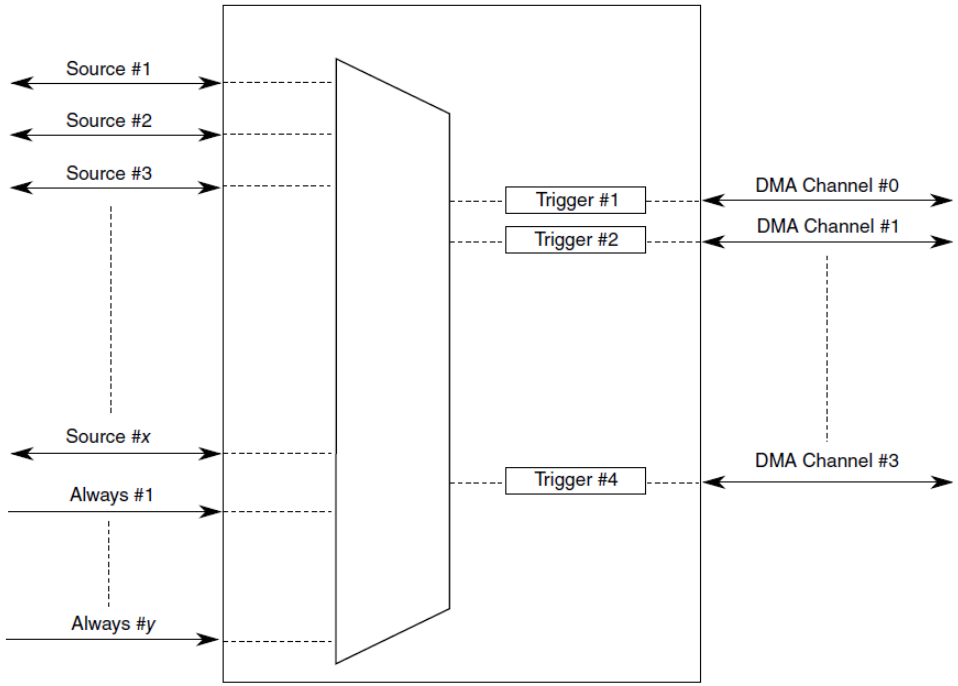
Trigger Sources

- A variety of peripherals can trigger DMA activity
- Trigger sources are chip-specific, so see KL25 Sub-Family Reference Manual (rev. 1), Chapter 3 - Chip Configuration, Section 3.4.8.1 DMA MUX Request Sources

Source #	Module	Description
0	-	Disabled
2-7	UART0,1, 2	Receive, Transmit
16-19	SPI0, 1	Receive, Transmit
22-23	I^2C0, 1	
24-29	TPM0	Channels 0-5
32-35	TPM1-2	Channels 0-1
40	ADC0	
42	CMP0	
45	DAC0	
49-53	Port Control Module	Port A-E
54-56	TPM0-2	Overflow
57	TSI	
60-63	DMAMUX	Always enabled

Triggering DMA Activity Using Peripherals

- Can use trigger events from peripherals to start DMA transfer
- Upon triggering, DMA can perform:
  - One transfer (cycle steal mode)
  - All transfers until BCR == 0 (continuous mode)
- DMA Multiplexer (DMAMUX)
  - Selects which source will trigger a DMA channel
- Each DMA channel n has a configuration register DMAMUX_CHCFGn
  - ENBL: Enable DMA channel
  - TRIG: Enables triggering of DMA channel
  - SOURCE: Selects triggering source



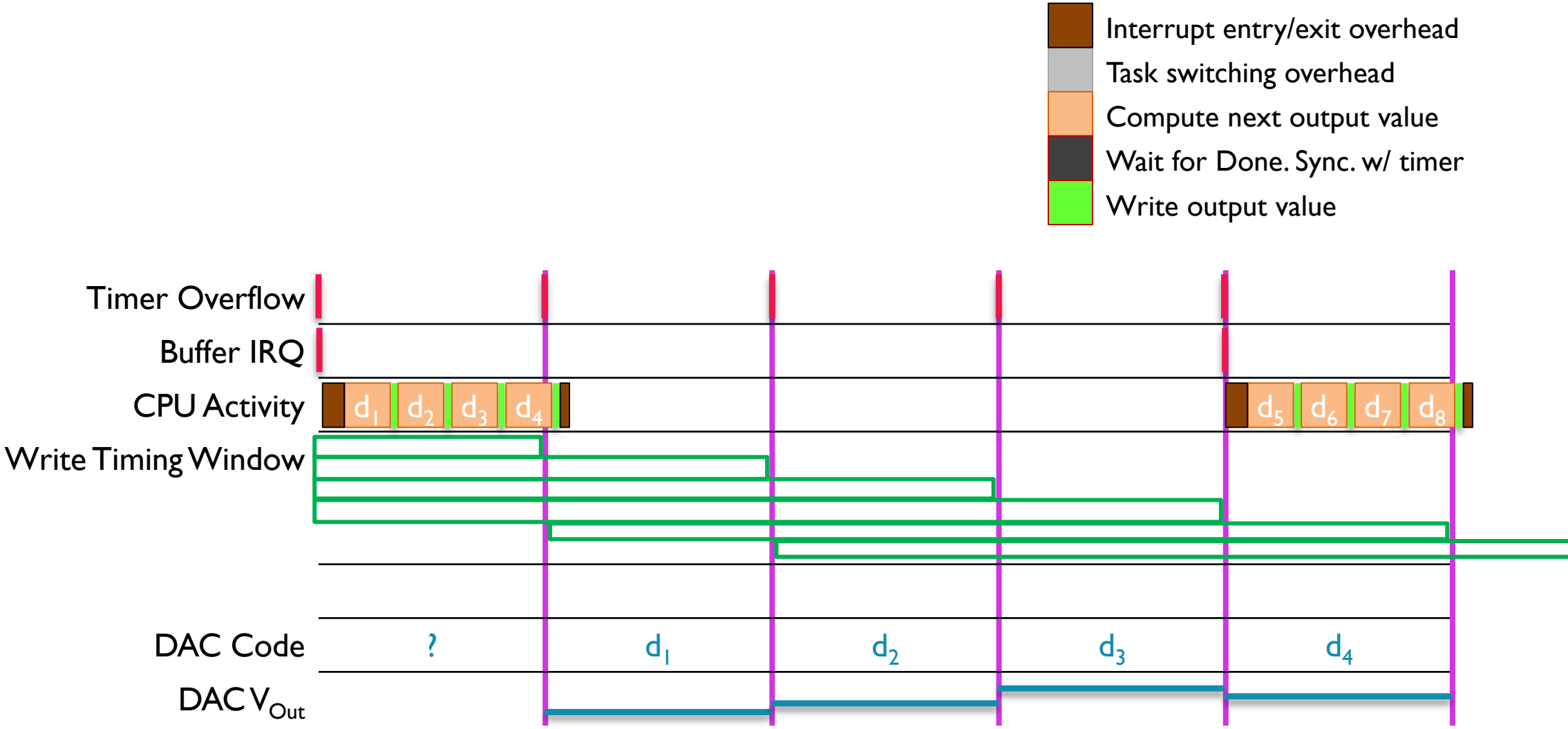
Bit	7	6	5	4	3	2	1	0
Read Write	ENBL	TRIG	SOURCE					
Reset	0	0	0	0	0	0	0	0

DMAMUXx_CHCFGn field descriptions

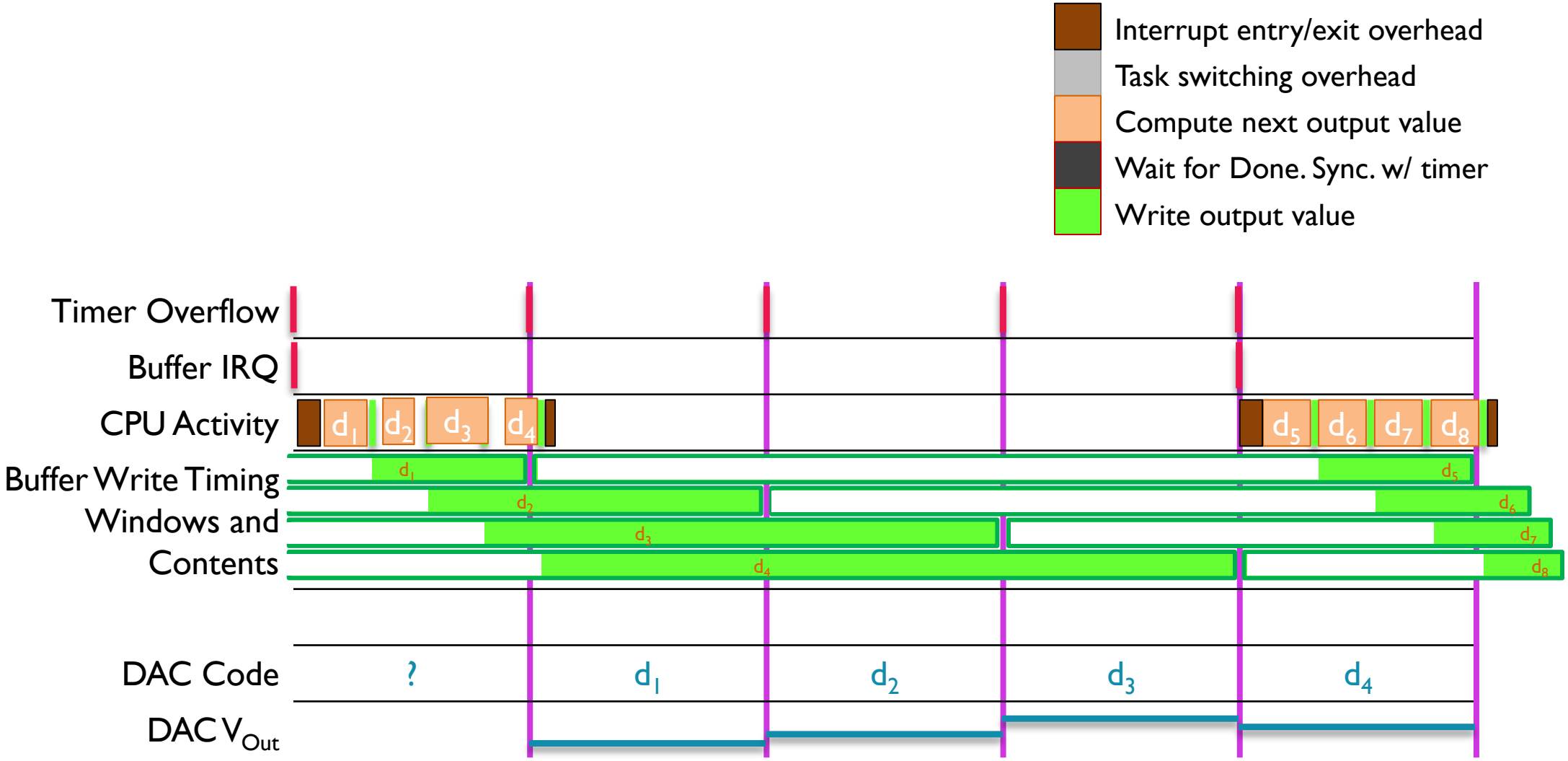
To Do

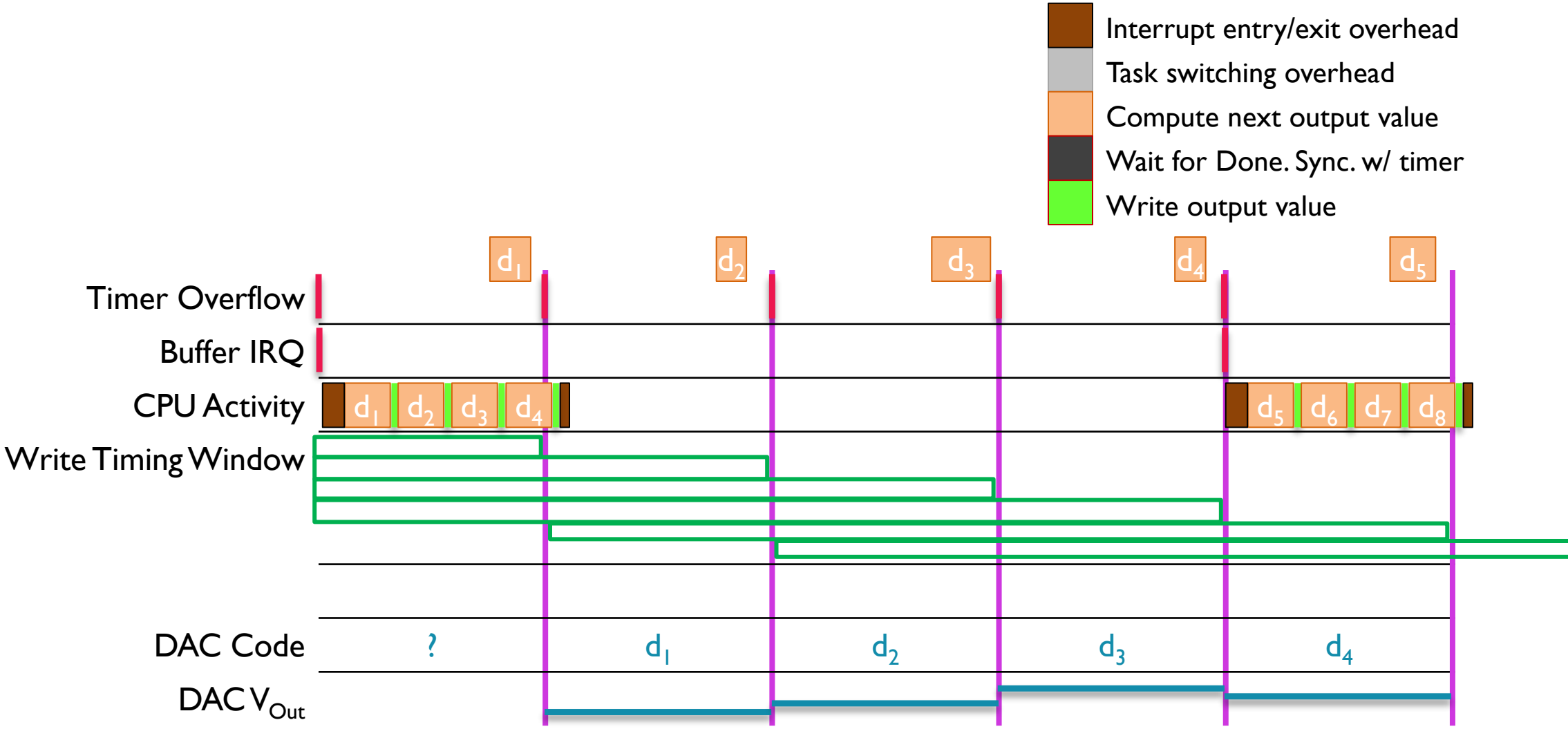
- Review development flow – when does scheduler appear?
- Use same set of compute output data durations (d1-d5)
- Complete diagrams
- When to introduce task switching overhead?
- Complete text notes per slide
- (Convert diagrams from horizontal to vertical HW/SW split?)

Diagrams

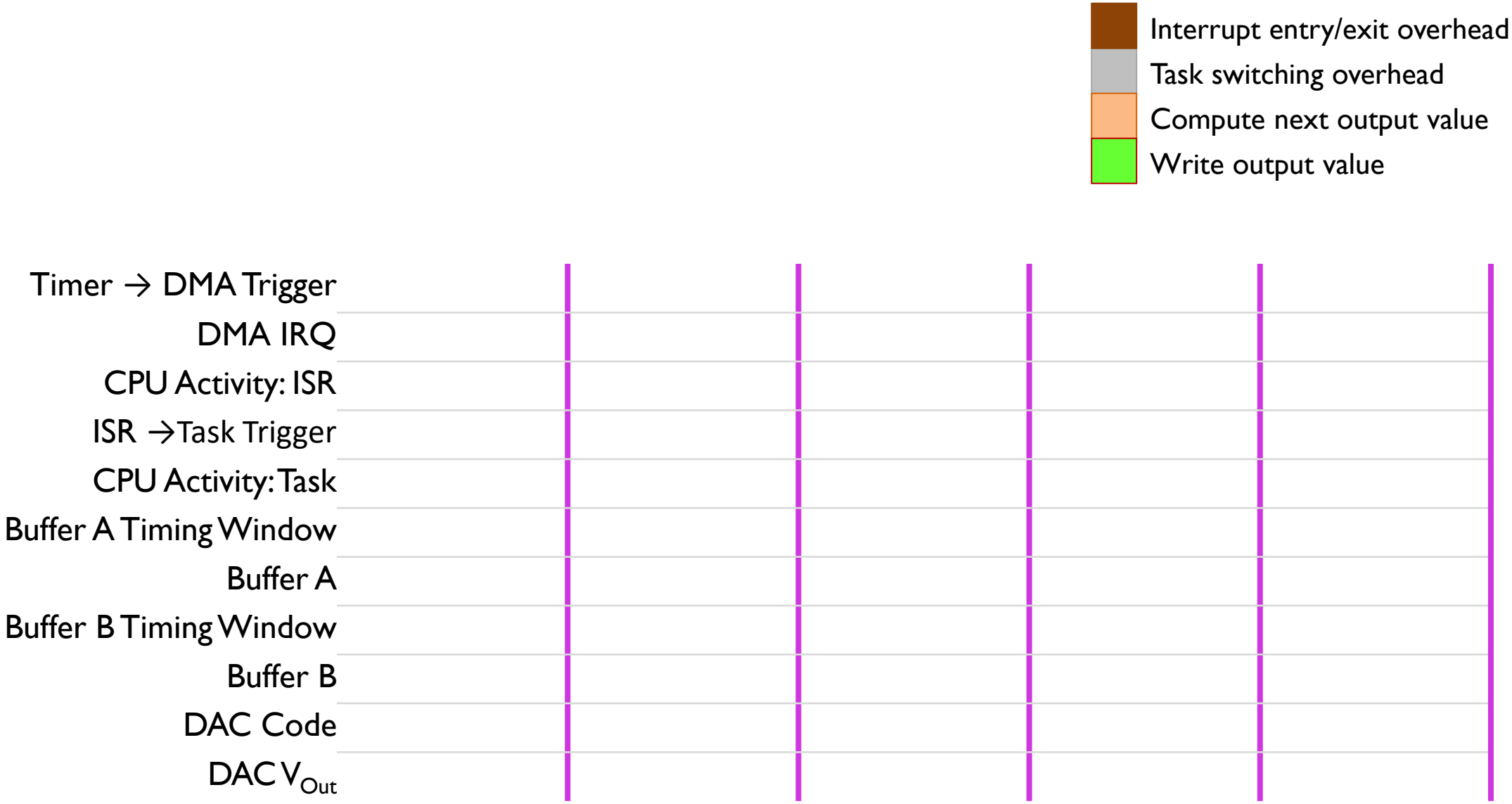

Interrupt entry/exit overhead
Task switching overhead
Compute next output value
Wait for Done. Sync. w/ timer
Write output value
Timer Overflow
Buffer IRQ
CPU Activity
Write Timing Window
DAC Code
DAC V_Out
d1
d2
d3
d4
d5
d6
d7
d8
?
d1
d2
d3
d4

N-Entry DAC Buffer with Empty Interrupt

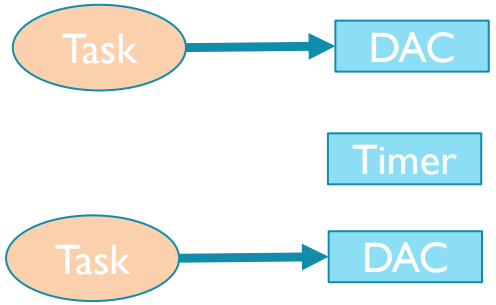


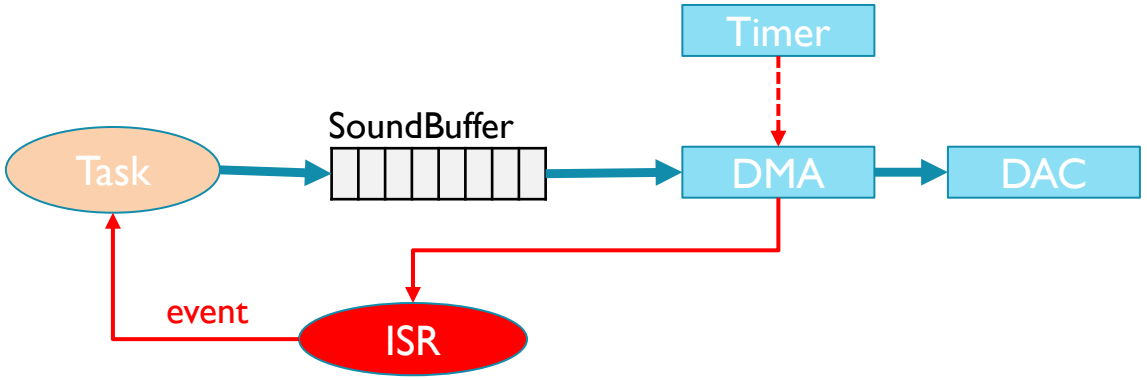

Interrupt entry/exit overhead
Task switching overhead
Compute next output value
Wait for Done. Sync. w/ timer
Write output value
Timer Overflow
Buffer IRQ
CPU Activity
Write Timing Window
DAC Code
DAC V_{Out}
d_1
d_2
d_3
d_4
d_5
d_6
d_7
d_8
?

Template for Software and Hardware Components and Timing

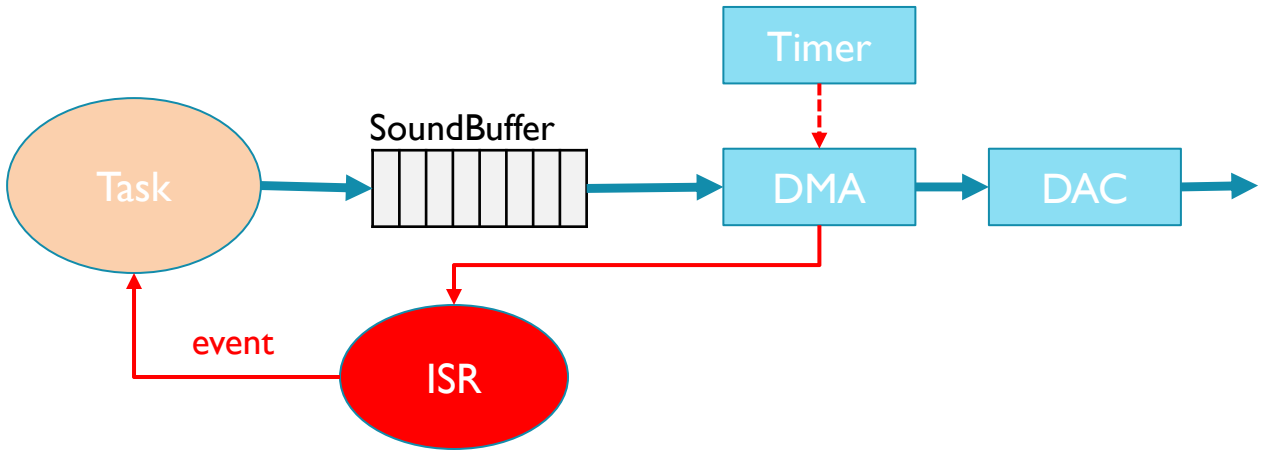


Software and Hardware

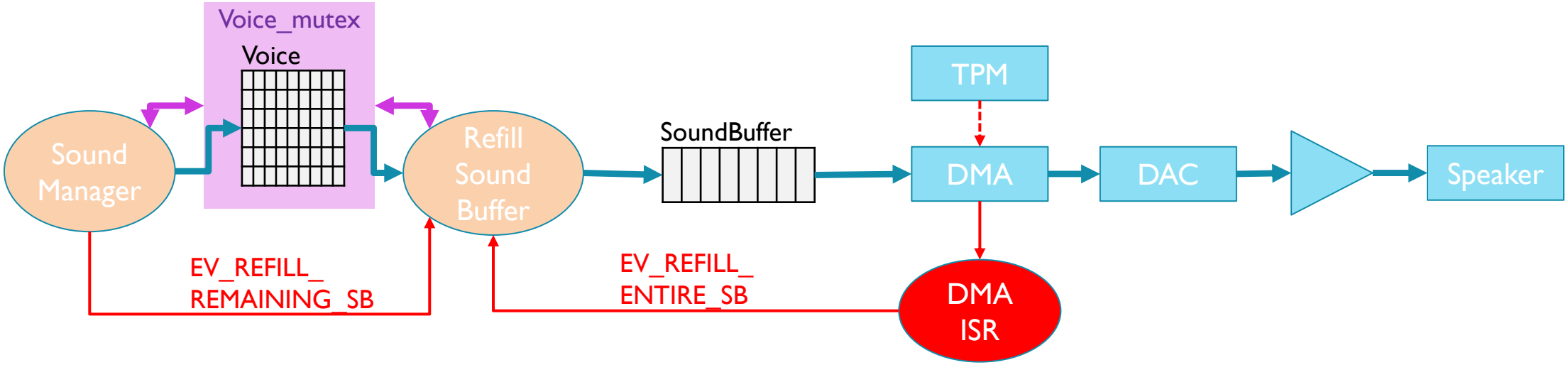




Shield Audio Software and Hardware Architecture

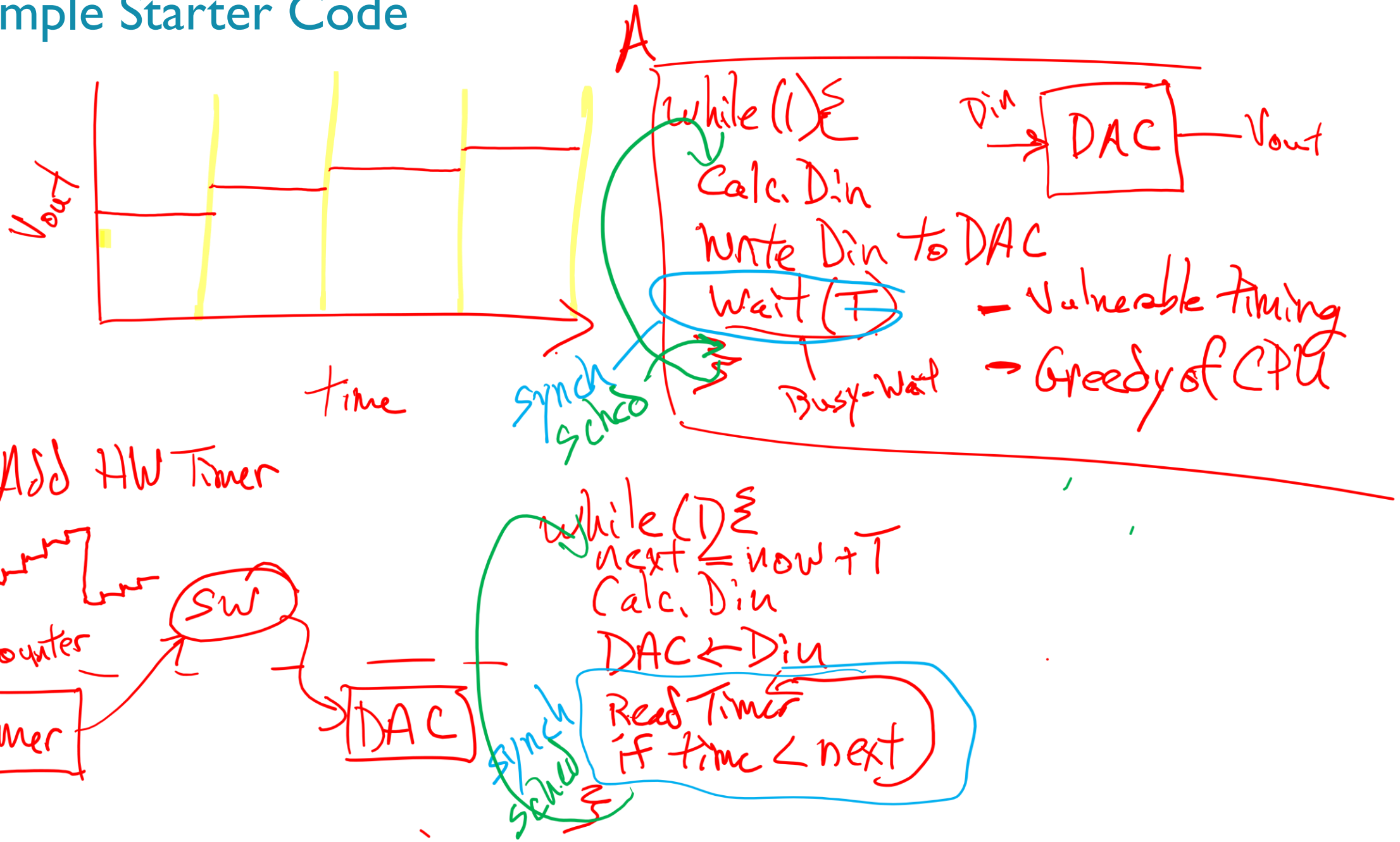


Shield Audio Software and Hardware Architecture

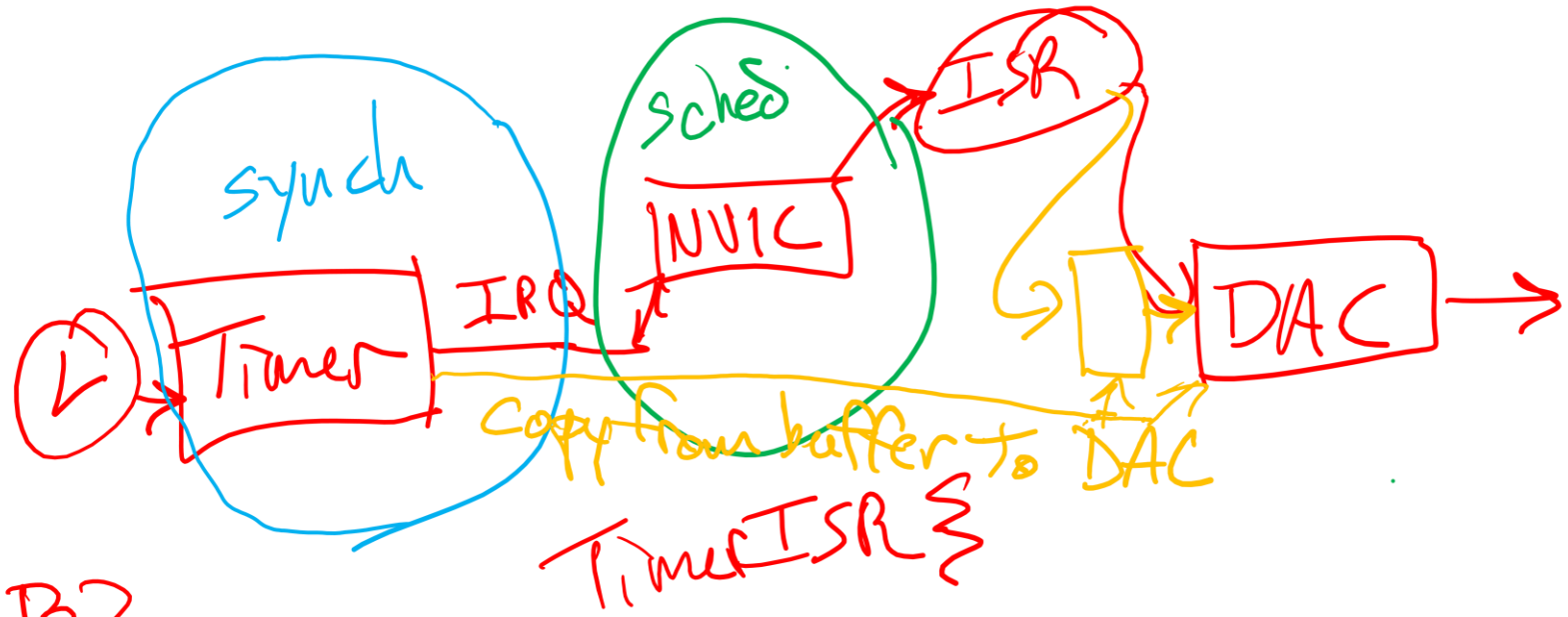


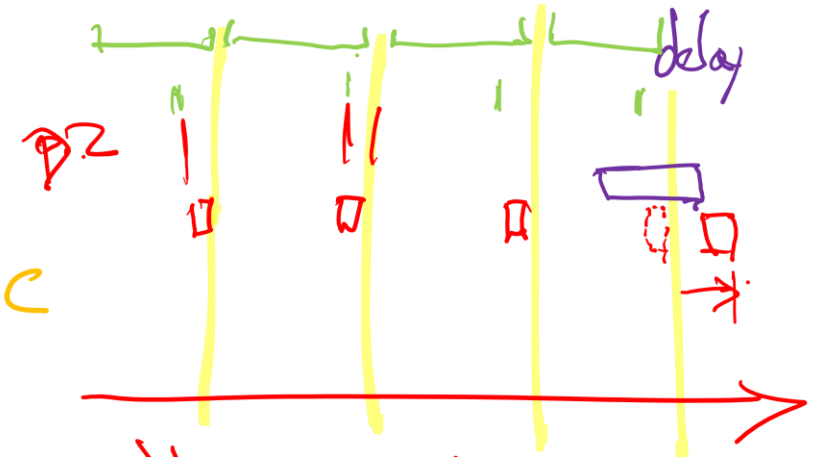
ROUGH DIAGRAMS FROM ECE 560, MONDAY 9/23/24

A. Simple Starter Code



C: Add 1-element buffer





B2

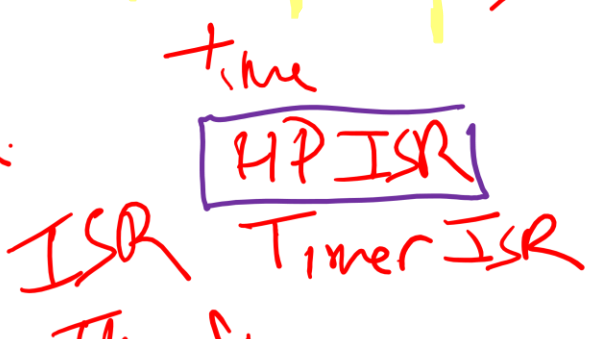
-1. Timing still vuln. to HP ISRs, locking

}

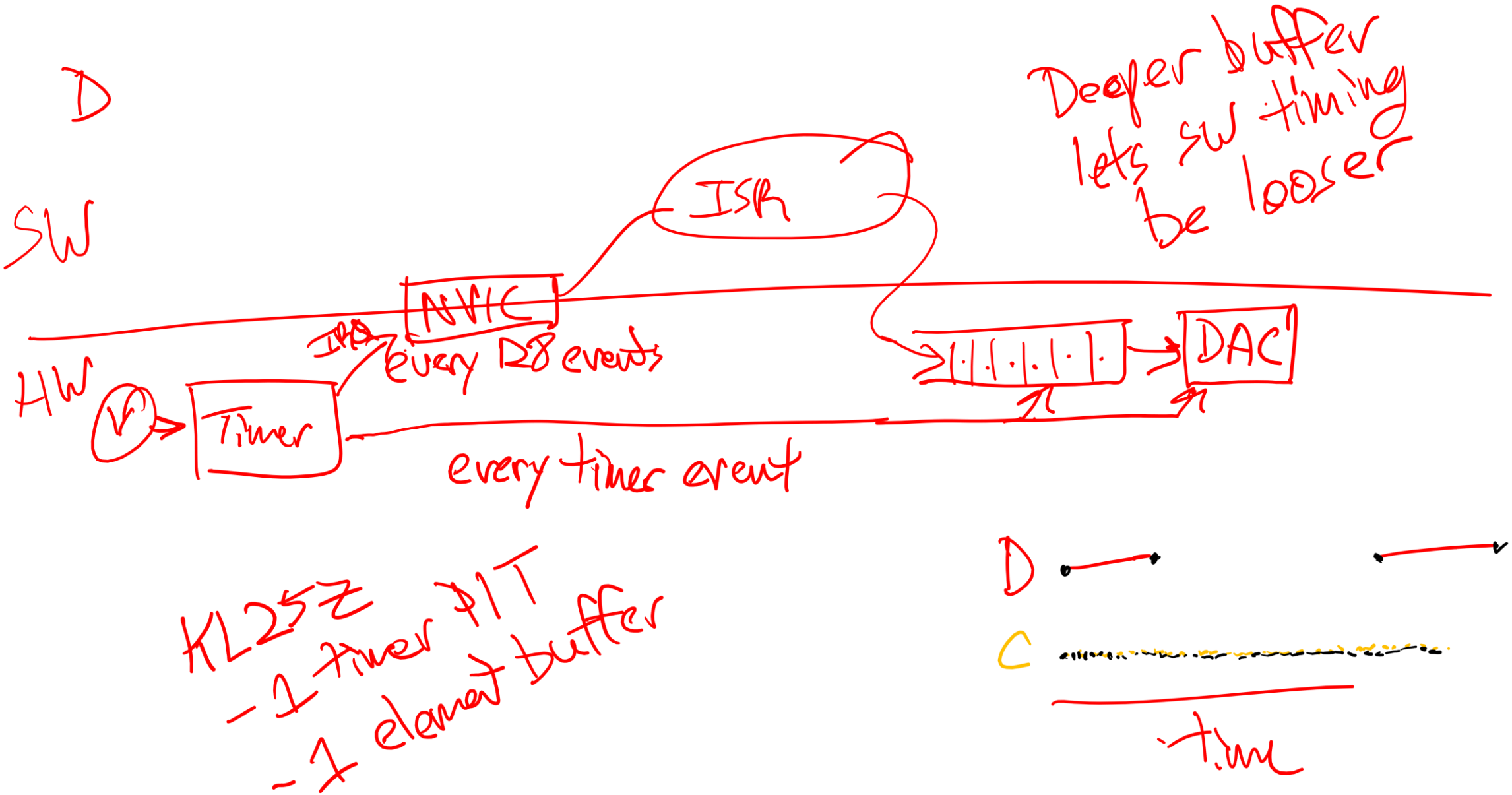
calculate new Din
DAC ← Din

-2. Overhead >15 cycles/interrupt

B2. ISR must start at precise time



C.


D
SW
HW
ISR
NVIC
IRQ
every R8 events
Timer
DAC
every timer event
Deeper buffer
lets sw timing
be looser
KL25Z
- 1 timer PIT
- 1 element buffer
D
C
time