NC STATE UNIVERSITY

Improving Output Timing Stability

V2

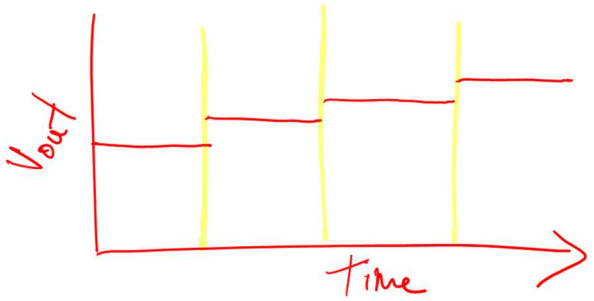
NC STATE UNIVERSITY

Topics Covered

- Output timing requirements
- Hardware and software methods to improve timing stability

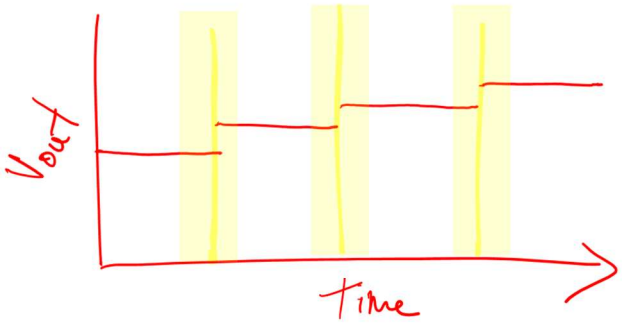
Overall Challenge: Update Analog Output Signal at Correct Times

- Some outputs need to be updated at **specific times**
  - Audio signal reconstruction: change output every T_{Sample}
  - For control system, **periodic** output updates make design & analysis easier



- Some control systems require output update to be **synchronized** with other events in system
  - To push someone on a swing, when do you extend your arms?
  - Examples: Anything **switched-mode**
    - Switched-mode power converters and drivers
    - Brushless DC motor drivers.

- "Specific time" = timing window
  - System performance falls if output changes before or after window
  - Harder to design system which meets timing requirements as window narrows

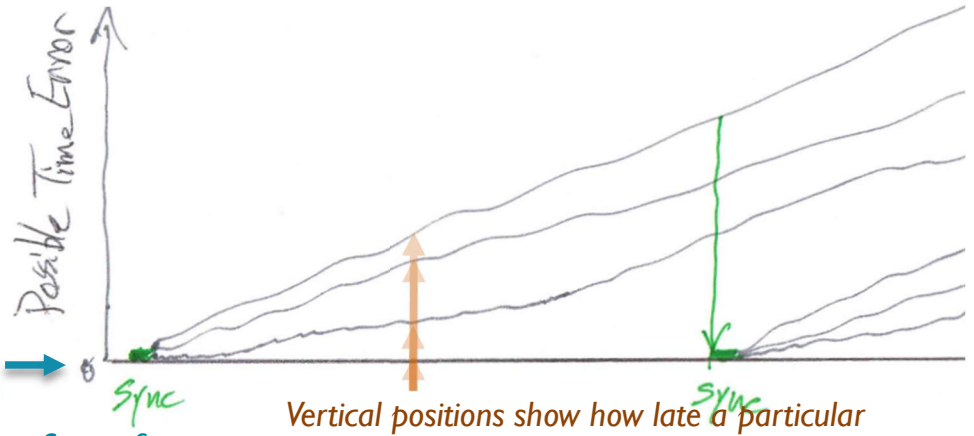


Precise, Stable Timing for Software May Be Difficult

- Currently no feasible programming language support to define timing. Need hardware, OS support.
- Translation of program source code **specification** to object code **implementation** disconnects developer from timing more
  - Source code is translated to executable machine code: compilation, assembly, linking
  - Can measure & observe machine code, not source code

- **When** a specific instruction executes relative to **latest synchronization** depends on
  - Control flow path taken to get there: conditionals, loops, etc.
  - Often other factors

On horizontal axis (time error of 0), each instruction always starts at same time relative to sync operation



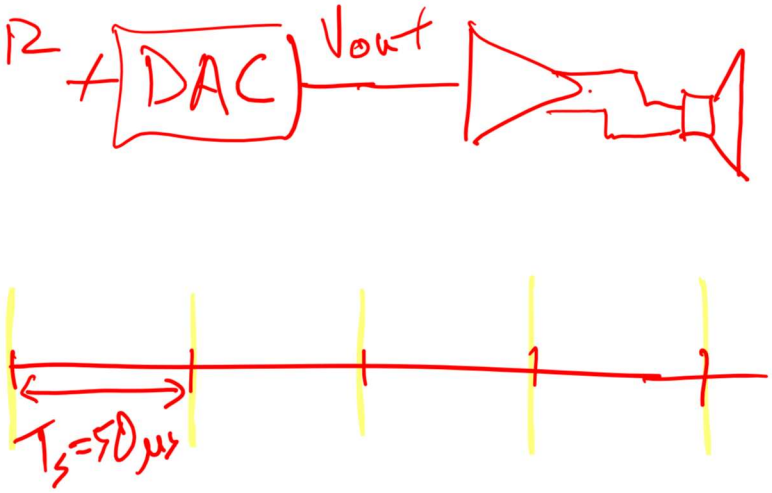
Start of process: First Instruction

Vertical positions show how late a particular instruction might execute over multiple process runs, relative to its earliest time.

- **Sharing CPU** with other processes and handlers (scheduling) can delay or preempt code which generates output signals

Waveform Generator Application

- Use hardware (and software) to help stabilize output timing of waveform generator
  - Want to update DAC output every 50 us for a 20 kHz update rate
  - DAC signal amplified to drive speaker
- Timing analysis approach - Vulnerabilities?
  - What kinds of events and over what time periods can affect the output update time?
- Solutions
  - Use hardware to help (or even replace) software doing synchronization, scheduling, or work.
    - Synchronization: determining when to update output
    - Scheduling: selecting code to run
    - Work: updating output
  - Buffer data to loosen (simplify) software timing requirements

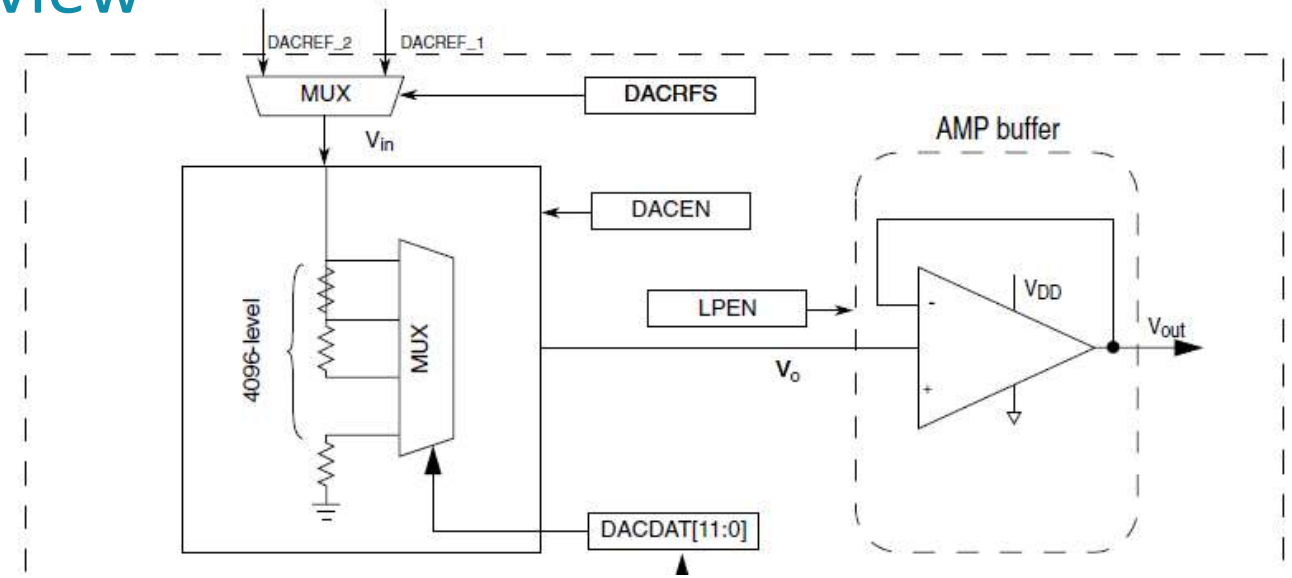


Example Code from ESF

```
void Play_Tone(unsigned int period, unsigned int
num_cycles, unsigned wave_type) {
 unsigned step, out_data;

 while (num_cycles>0) {
  num_cycles--;
  for (step = 0; step < NUM_STEPS; step++) {
   switch (wave_type) {
    case SQUARE:
     if (step < NUM_STEPS/2)
      out_data = 0;
     else
      out_data = MAX_DAC_CODE;
     break;
    case RAMP:
     out_data = (step*MAX_DAC_CODE)/NUM_STEPS;
     break;
    case SINE:
     out_data = SineTable[step];
     break;
   default:
     break;
   }
   DAC0->DAT[0].DATH = DAC_DATH_DATA1(out_data >> 8);
   DAC0->DAT[0].DATL = DAC_DATL_DATA0(out_data);
   Delay_us(period/NUM_STEPS);
   }
  }
 }
```

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

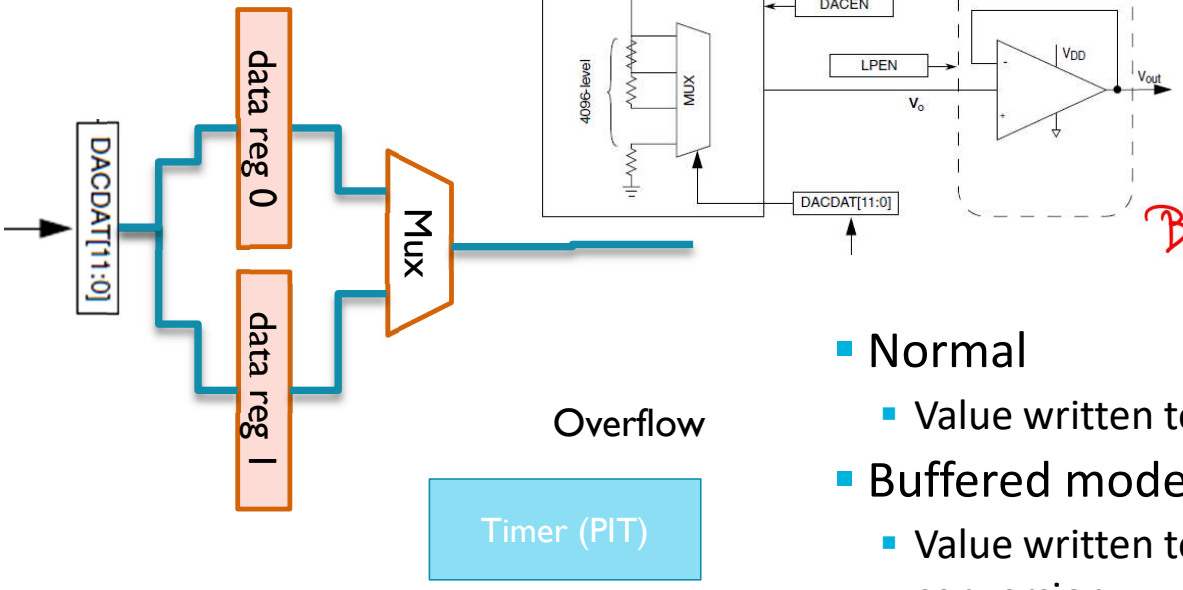
DAC Registers

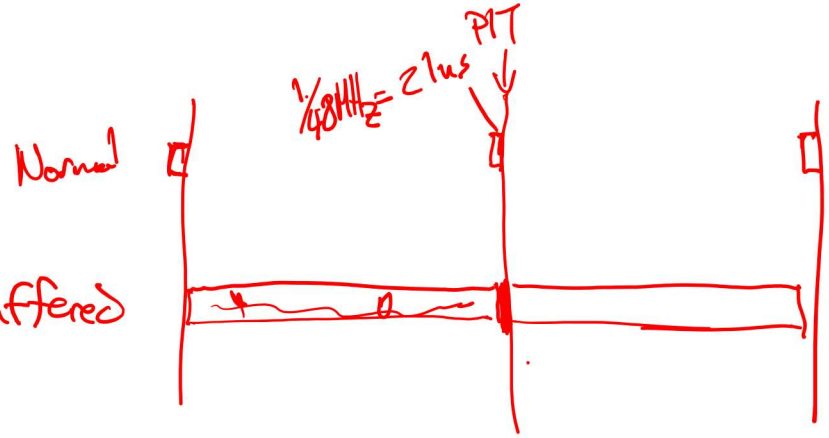
DAC memory map

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 [11:8] in DACx_DATnH

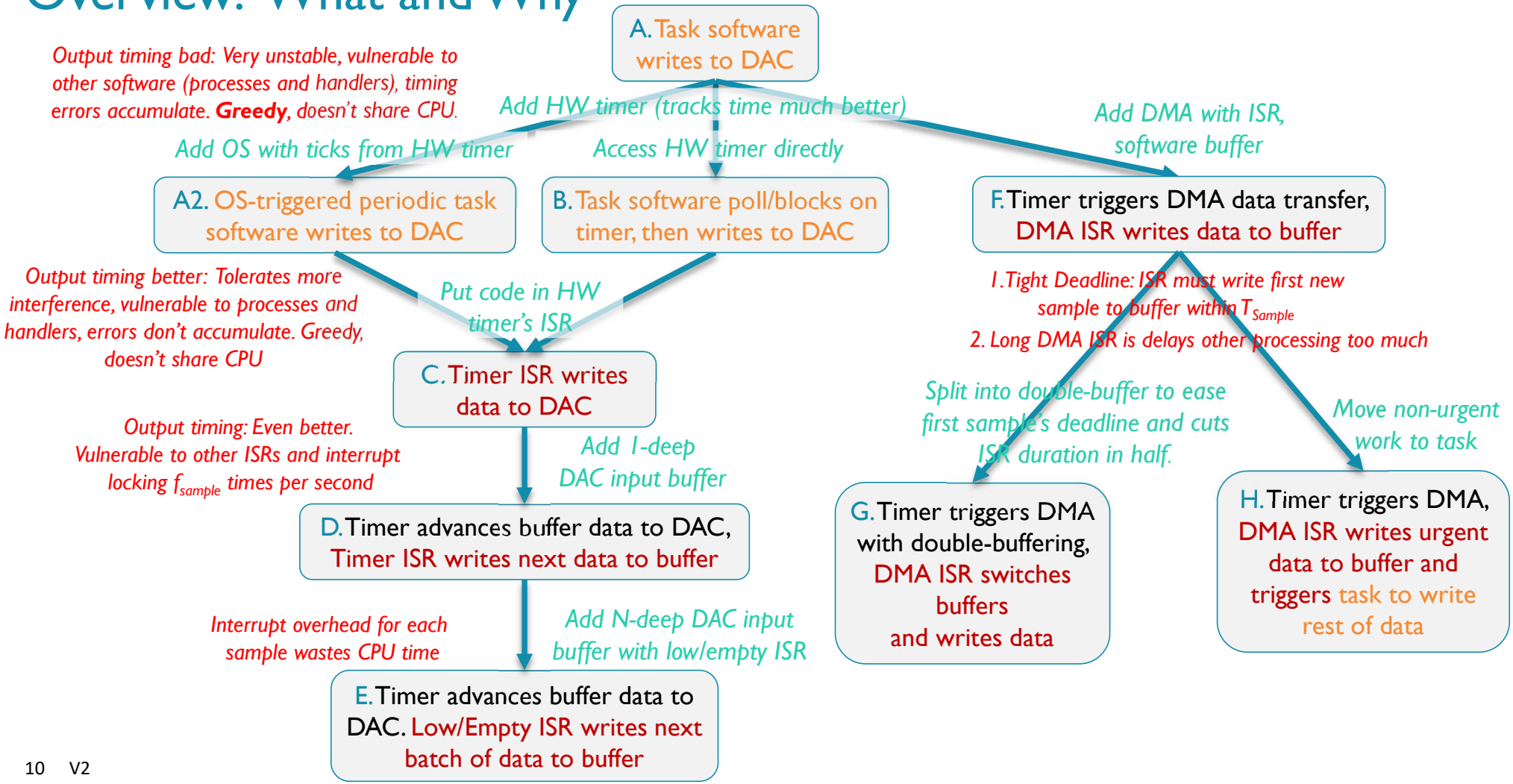
DAC Operating Modes





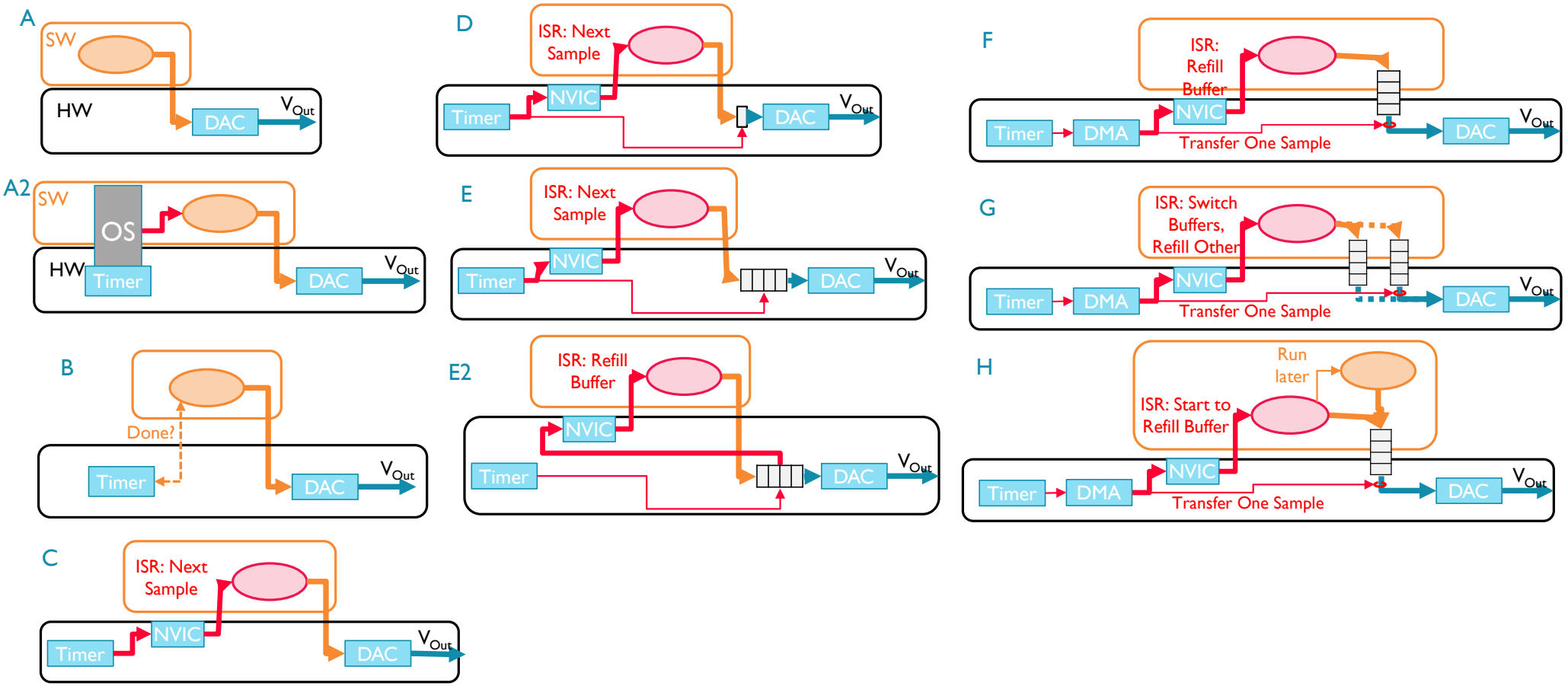
- Normal
  - Value written to DACDAT is converted to voltage immediately
- Buffered mode eases timing requirements
  - Value written to DACDAT is stored in data buffer for later conversion
  - Next data item is sent to DAC when triggered
    - Software Trigger - write to DACSWTRG field in DACx_C0
    - 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

Overview: What and Why

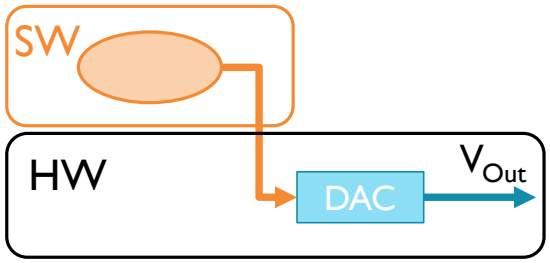


NC STATE UNIVERSITY

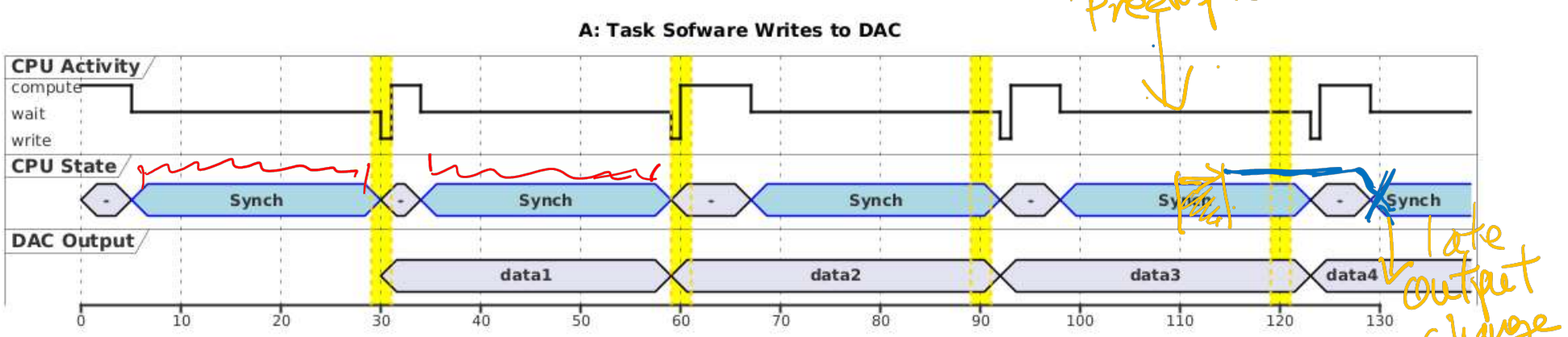
Software and Hardware Components



A. Simple Starter Code

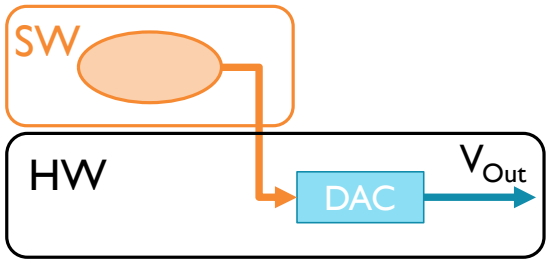


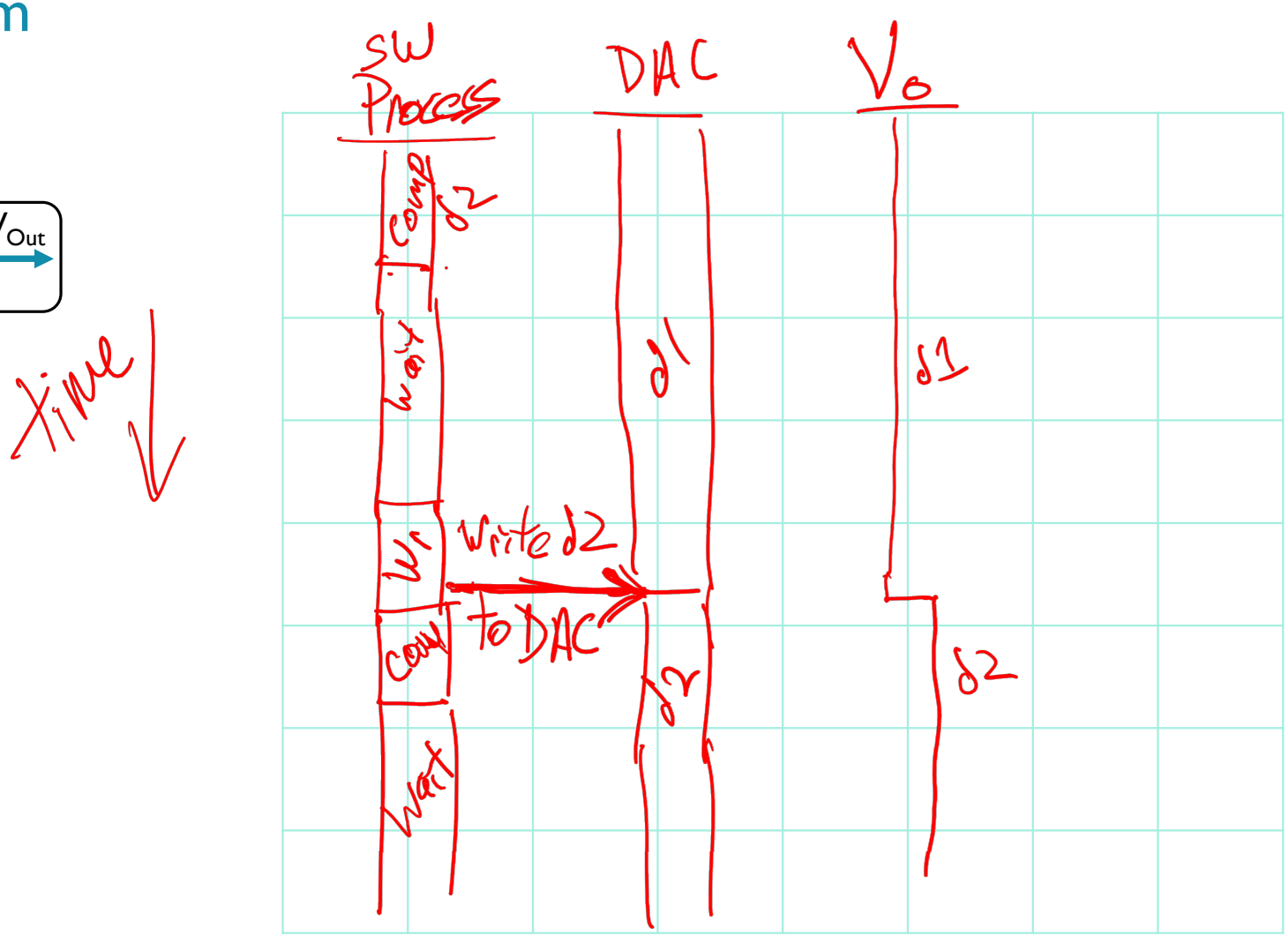

```
while (1) {
    compute data
    // Synchronize: Wait until time for next sample
    for (t = T; t>0; t--); // busy wait loop creates delay
    // Position of following code implicitly schedules it
    write data to DAC
}
```



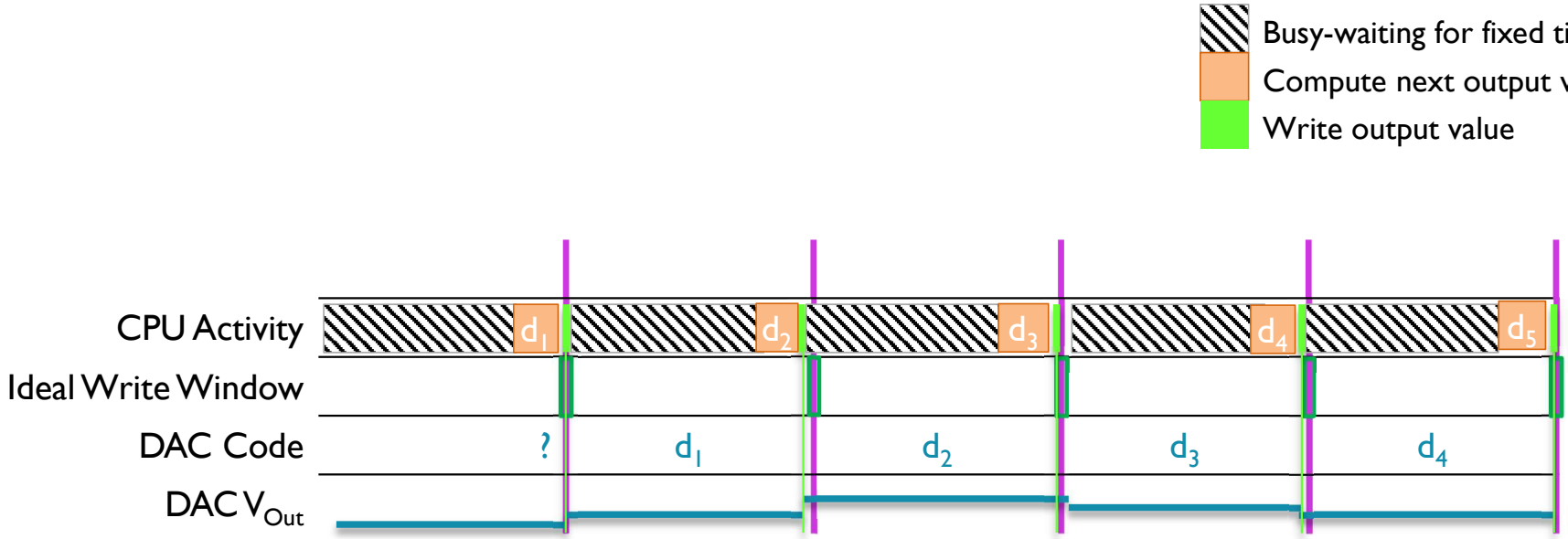
- Timing is unstable. Make T what value?
  - What if computing data takes variable time?
  - Vulnerable to interference by other handlers, processes on CPU
- Using busy-waiting to create time delay is greedy because it doesn't share CPU
- Synchronization and scheduling done completely in software

Sequence Diagram

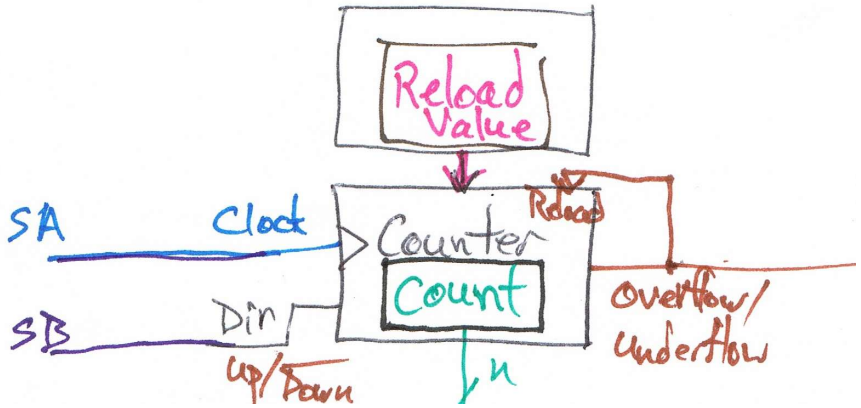


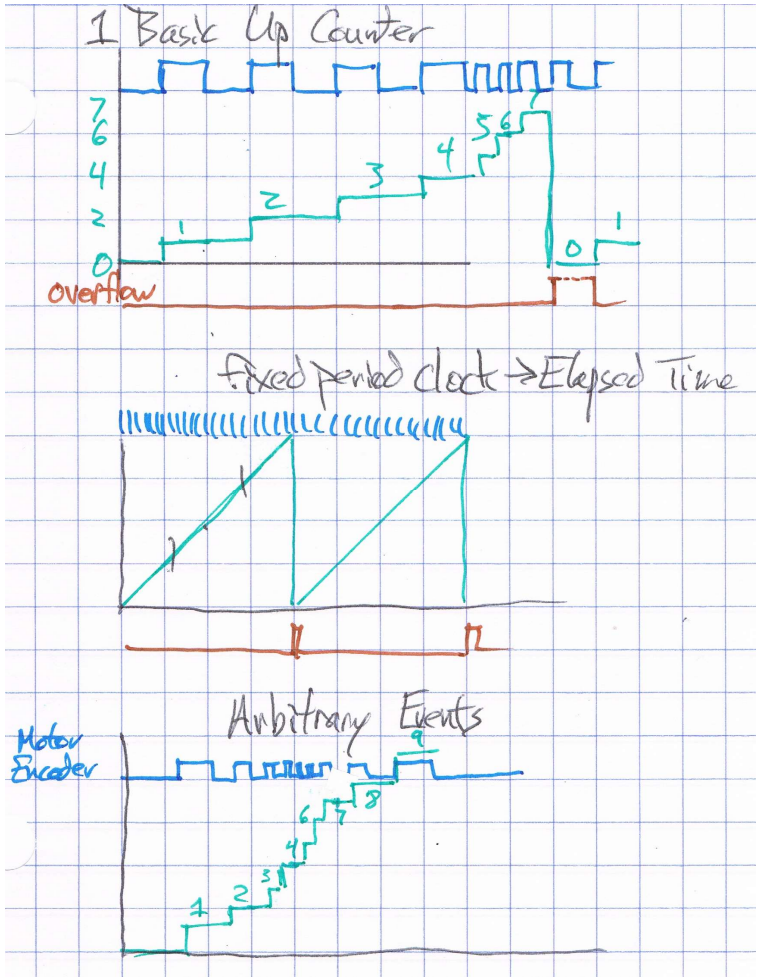


Unbuffered DAC with Busy-Wait Code

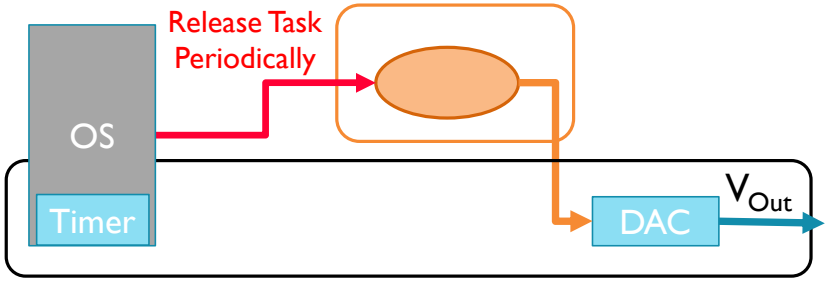


Timer/Counter Concepts





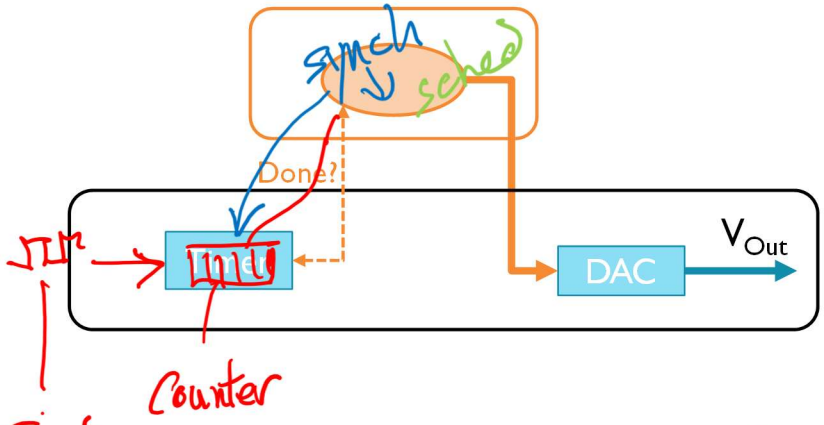
A2: Scheduler Releases WaveGen Periodically




```
void Task_WaveGen (void) {
    compute data
    write data to DAC
}
```


Interrupt entry/exit overhead
Task switching overhead
Compute next output value
Write output value

B. Software Polls Hardware Timer

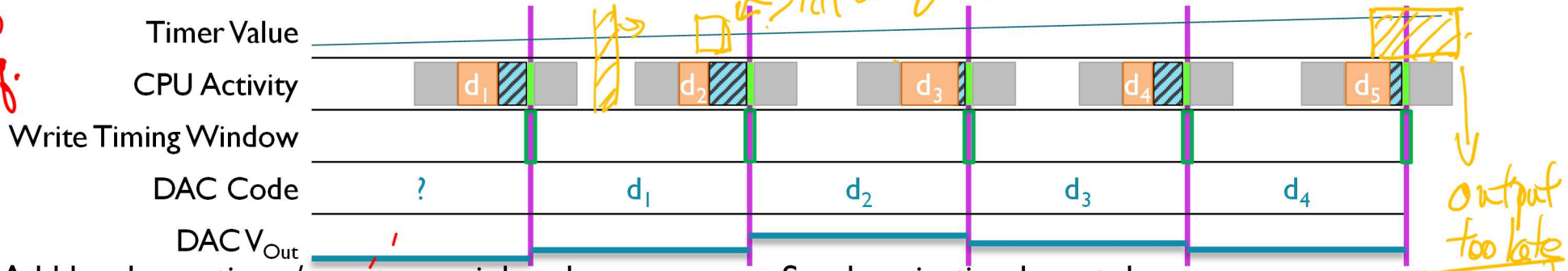



```
TNext = TSample
while (1) {
     compute data
     // Synchronize: Wait until time for next sample
     while (TimerValue < TNext); // busy wait polling loop
     // Position of following code implicitly schedules it
     write data to DAC
     TNext += TSample
}
```

Task switching overhead

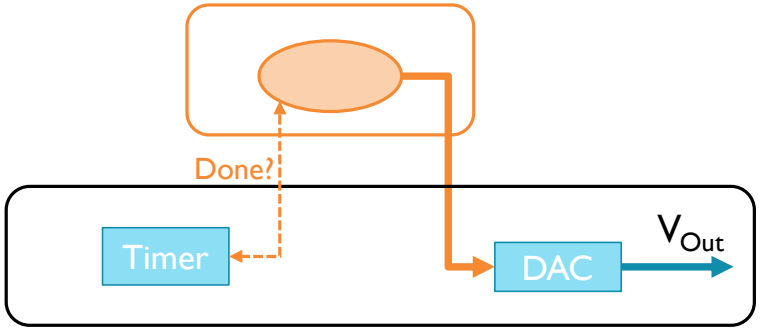
Compute next output value

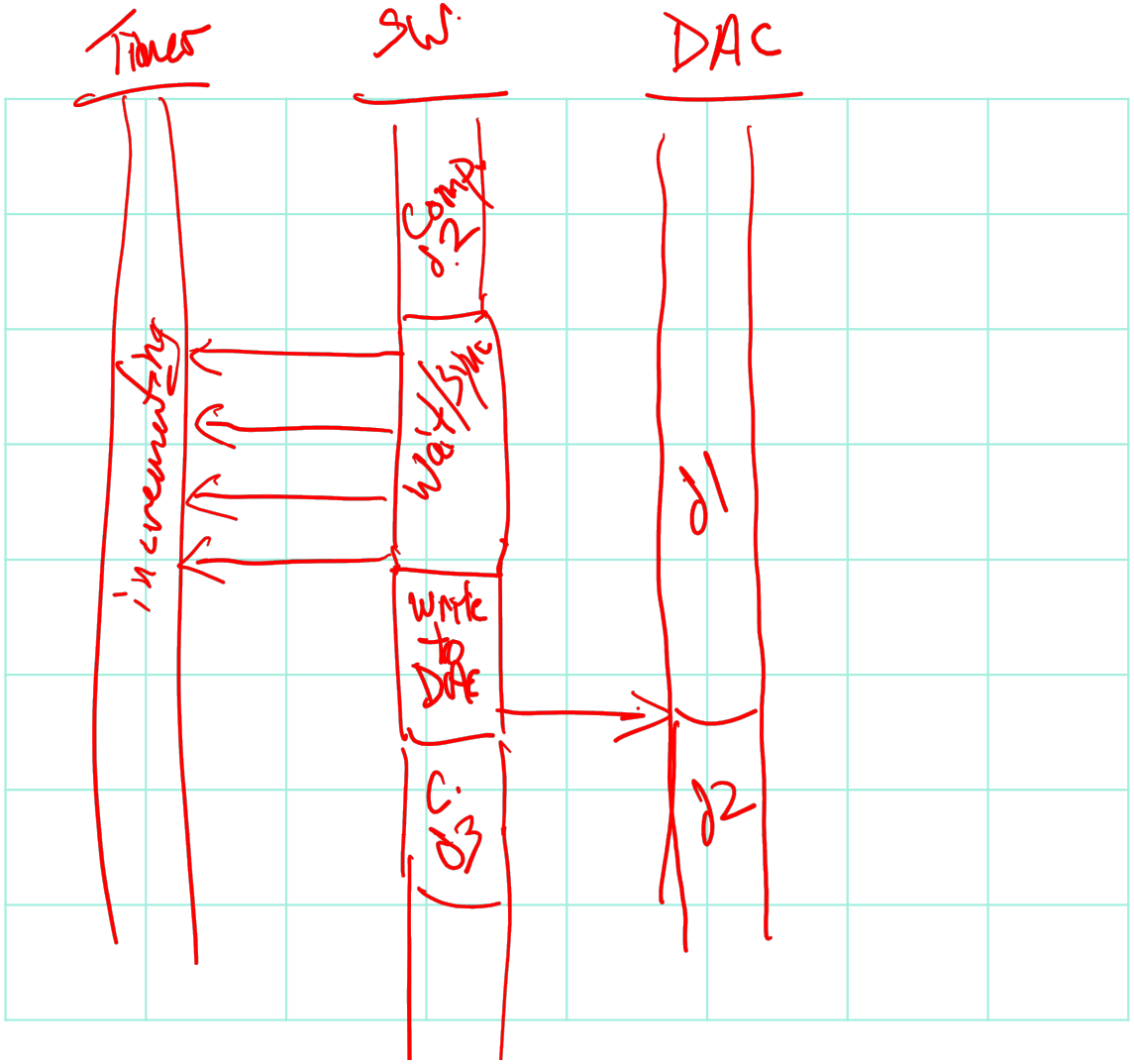
Synchronize by busy-waiting for timer done flag



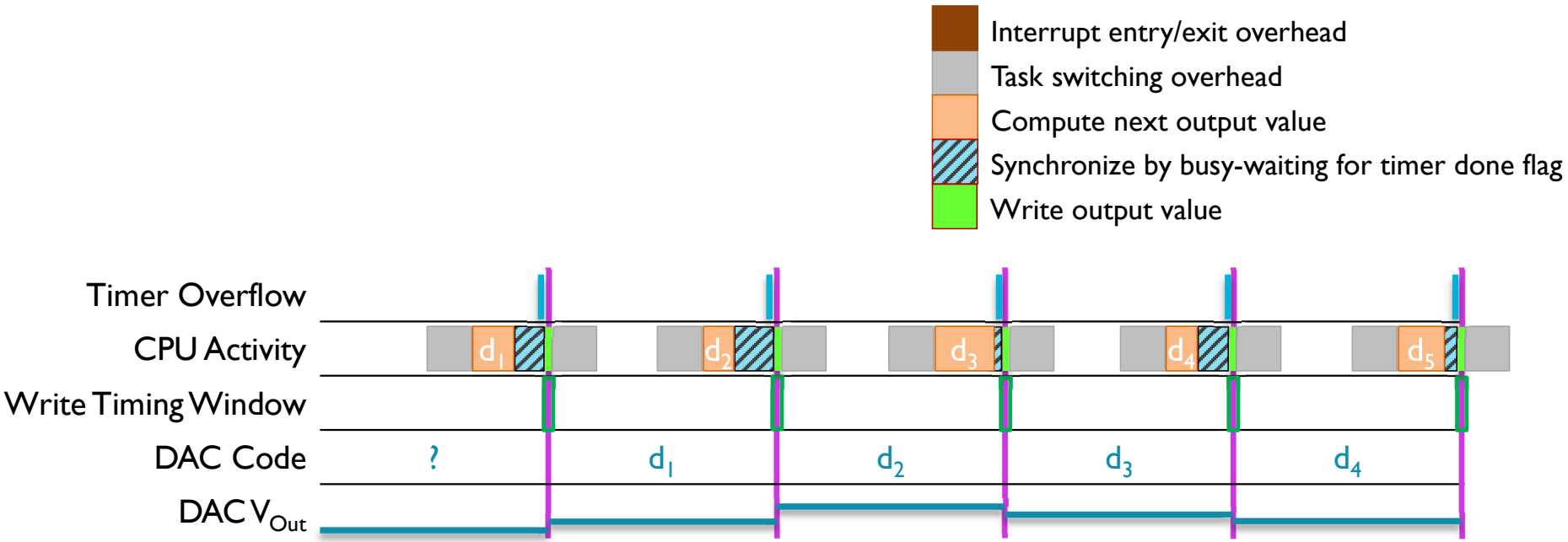
- Add hardware timer/counter peripheral
  - Binary counter (e.g. 16, 32 bits) tracks elapsed time by counting clock pulses. SW can read (e.g. TimerValue)
  - Increments periodically, regardless of SW activity
- Synchronization loop tolerates some timing interference
  - How much? Class discussion activity…
- Synch performed in SW with HW help
- Scheduling performed in SW

Sequence Diagram

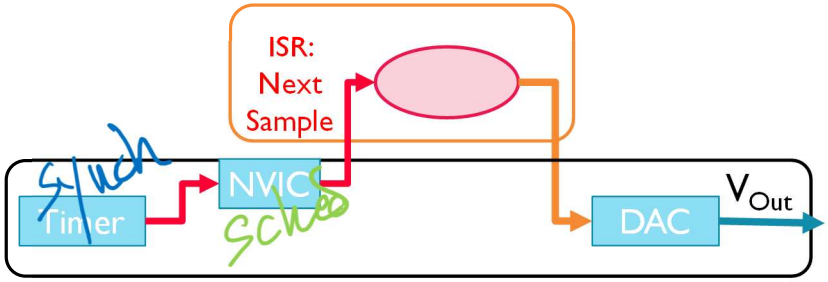




Unbuffered DAC with Timer Polling



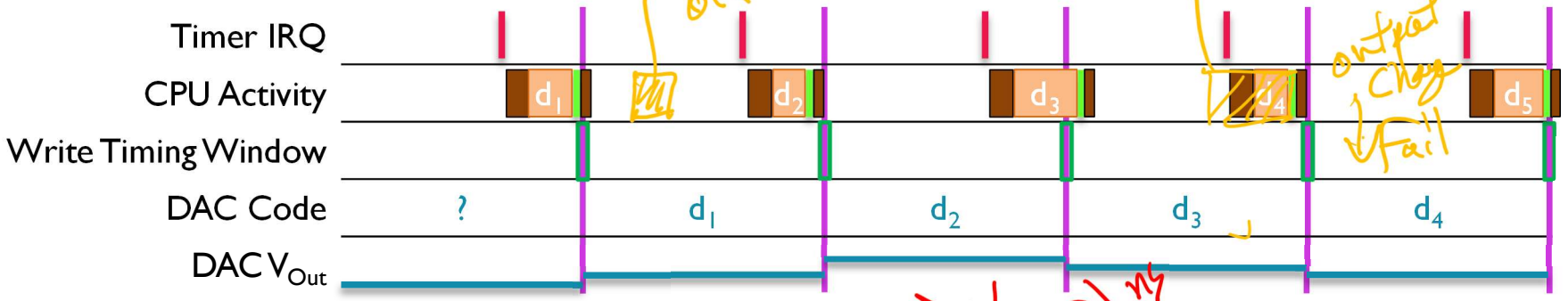
C. Hardware Timer Periodically Triggers Interrupt for DAC Write




```
Interrupt Handler {
  compute data
  write data to DAC
}
```

Higher Pri ISR
Interference
not OK

Interference
OK

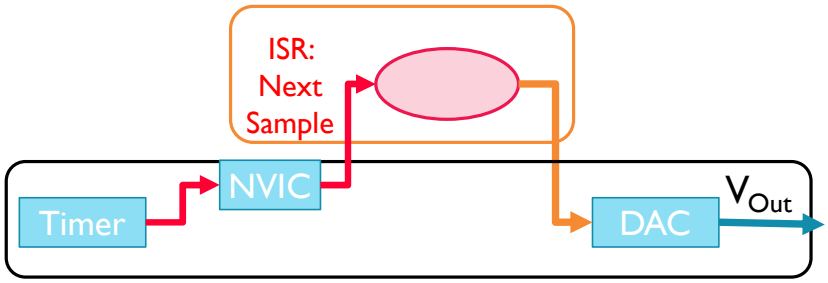


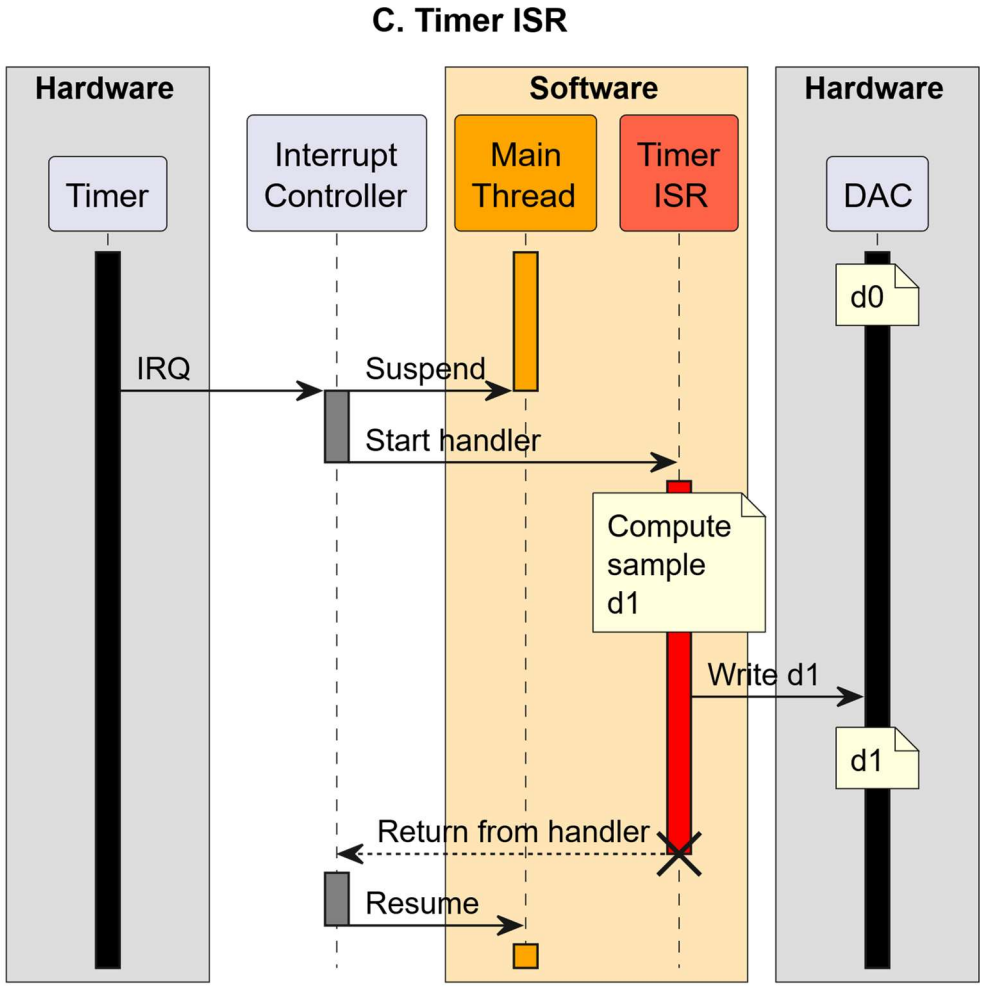
output
chgg
Fail

Timing Window
$T_{cpu} = \frac{1}{48M} \approx 21$ ns

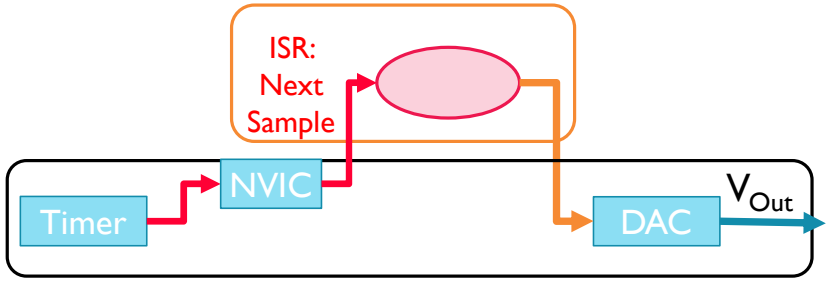
Interrupt entry/exit overhead
Task switching overhead
Compute next output value
Wait for Done. Sync. w/ timer
Write output value

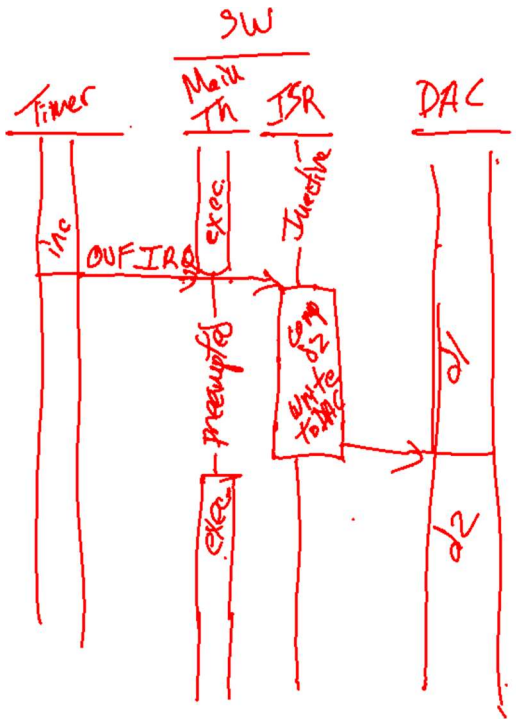
Sequence Diagram

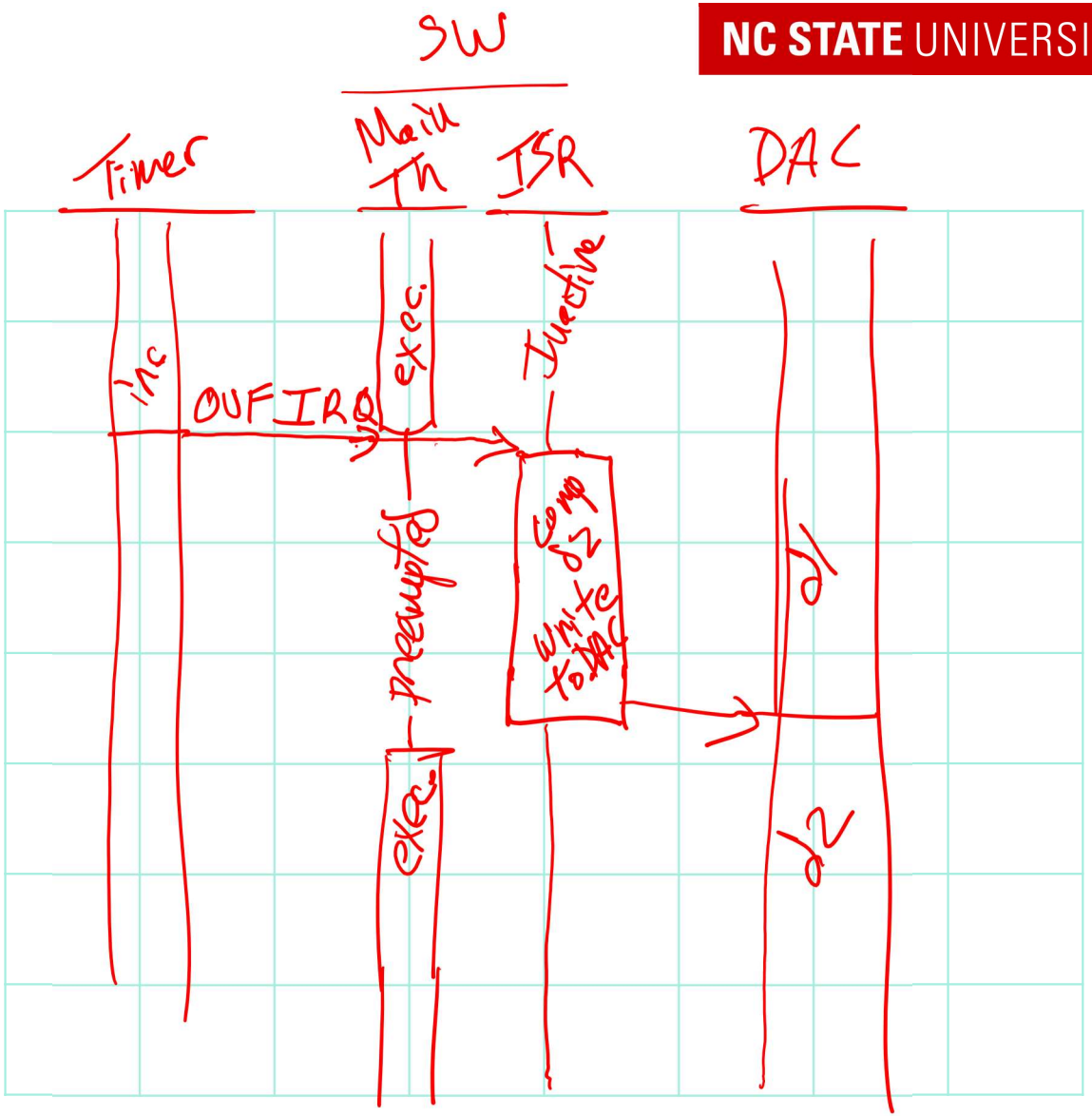




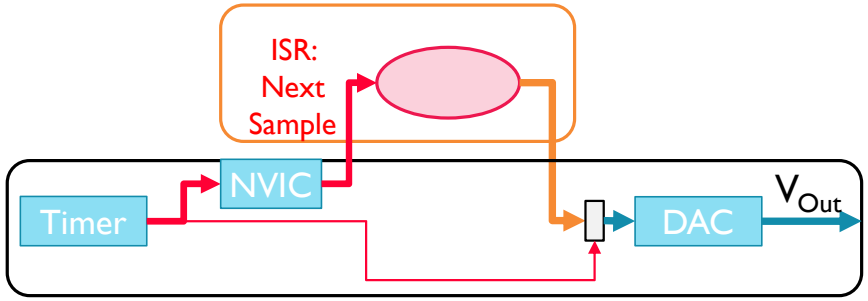
Sequence Diagram





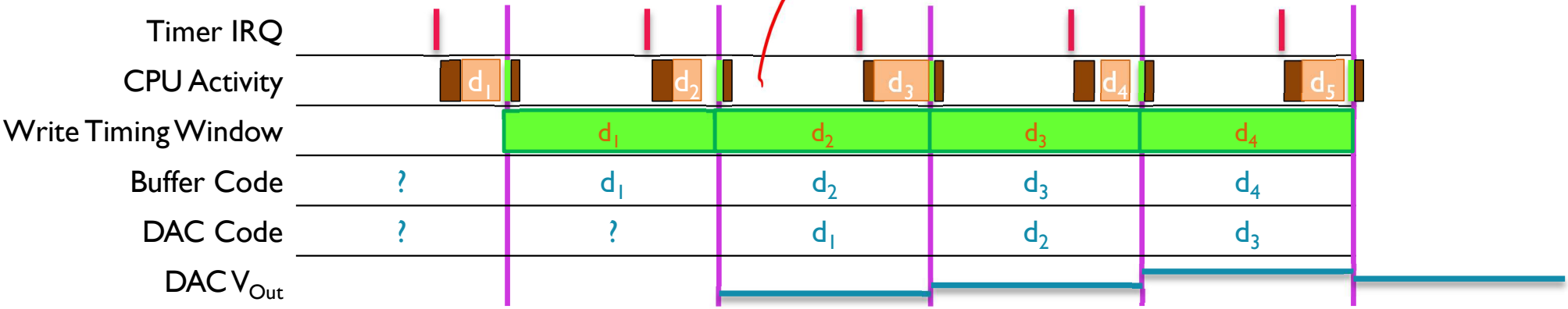


D. Add 1-element Input Data Buffer for DAC

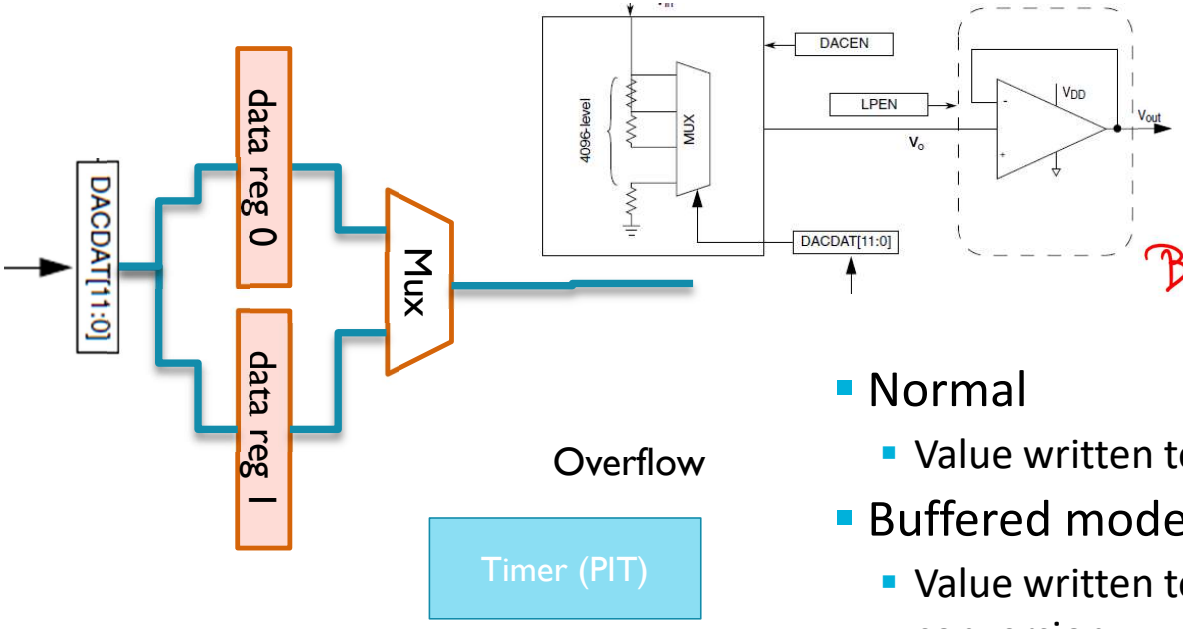


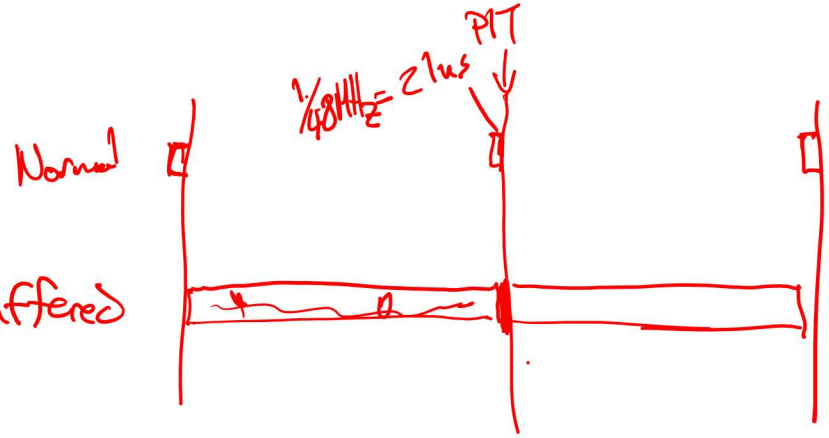

```
(code) {
}
```

Timing window is now $T_{sample} = 50\mu s$



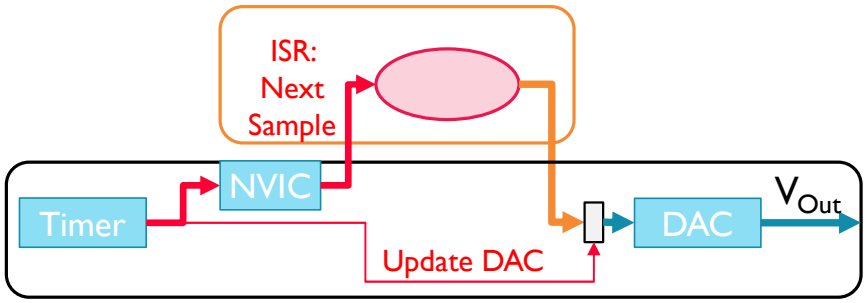
DAC Operating Modes



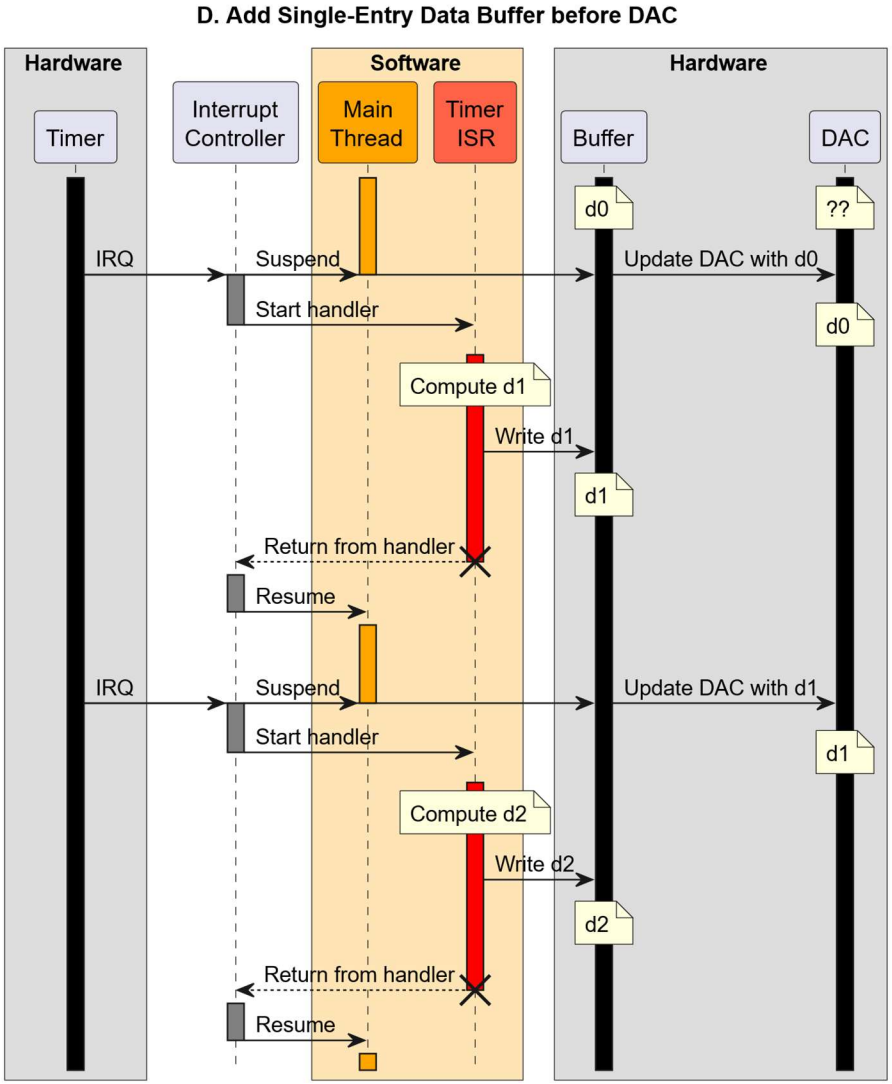


- Normal
  - Value written to DACDAT is converted to voltage immediately
- Buffered mode eases timing requirements
  - Value written to DACDAT is stored in data buffer for later conversion
  - Next data item is sent to DAC when triggered
    - Software Trigger - write to DACSWTRG field in DACx_C0
    - 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

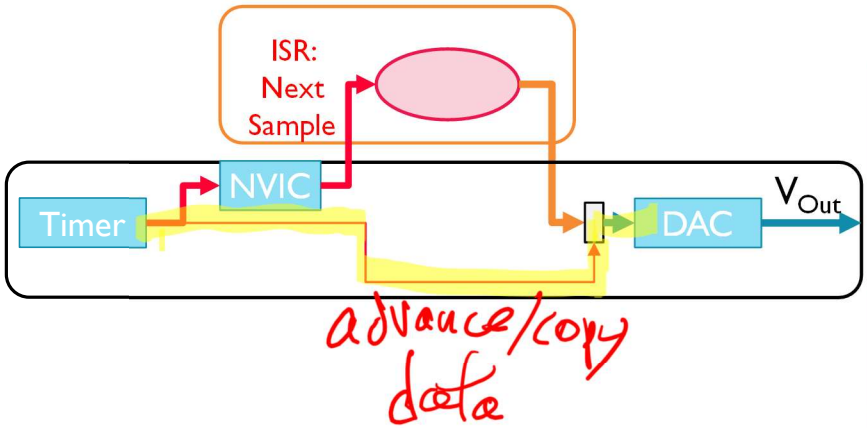
Sequence Diagram

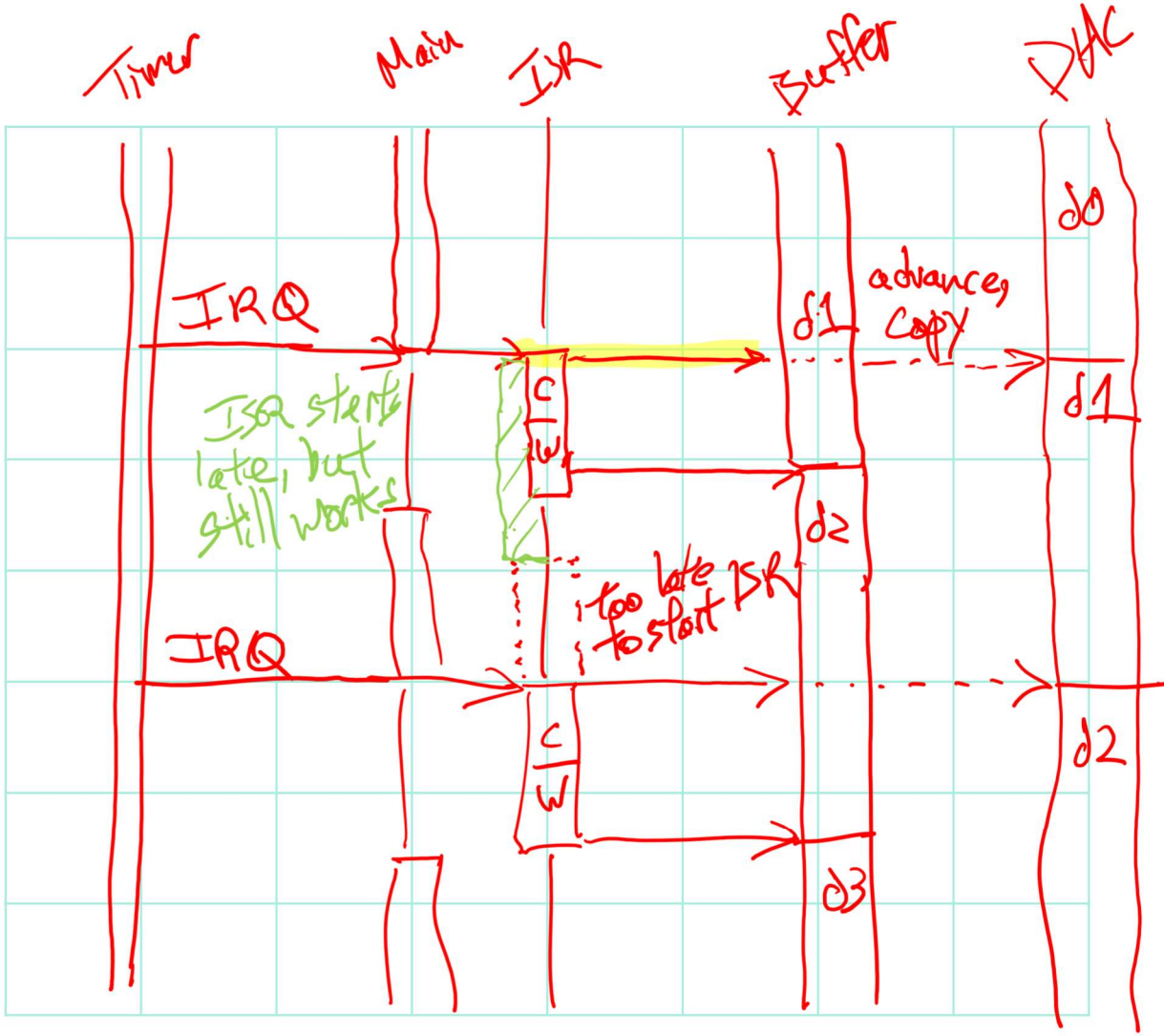


- ISR can start slightly late and still update buffer in time
  - Slack time: Tsample – Tcompute - tbd

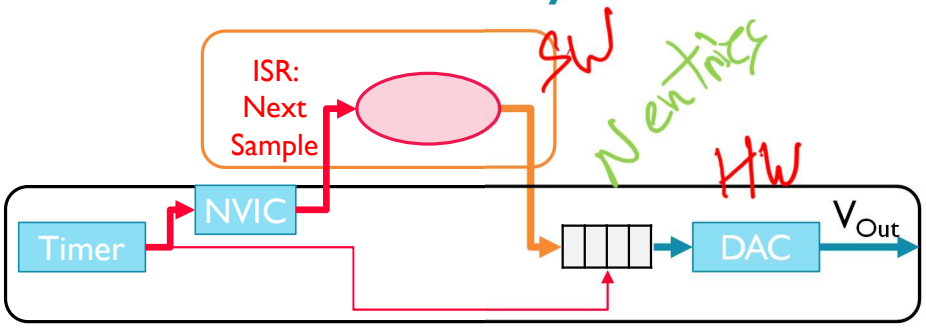


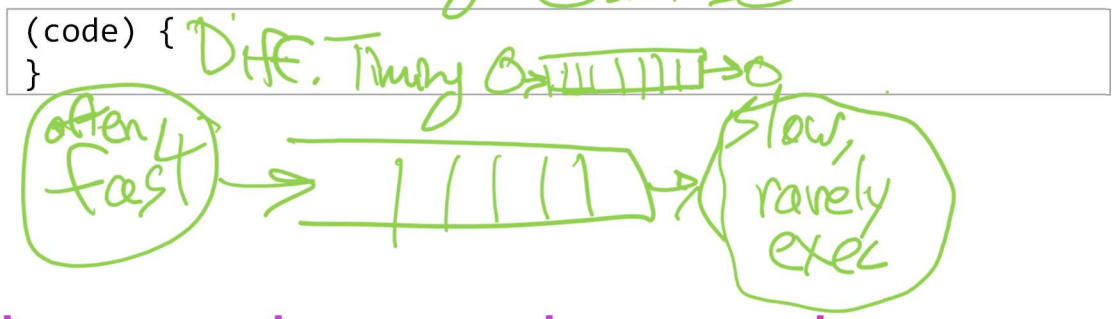
Sequence Diagram

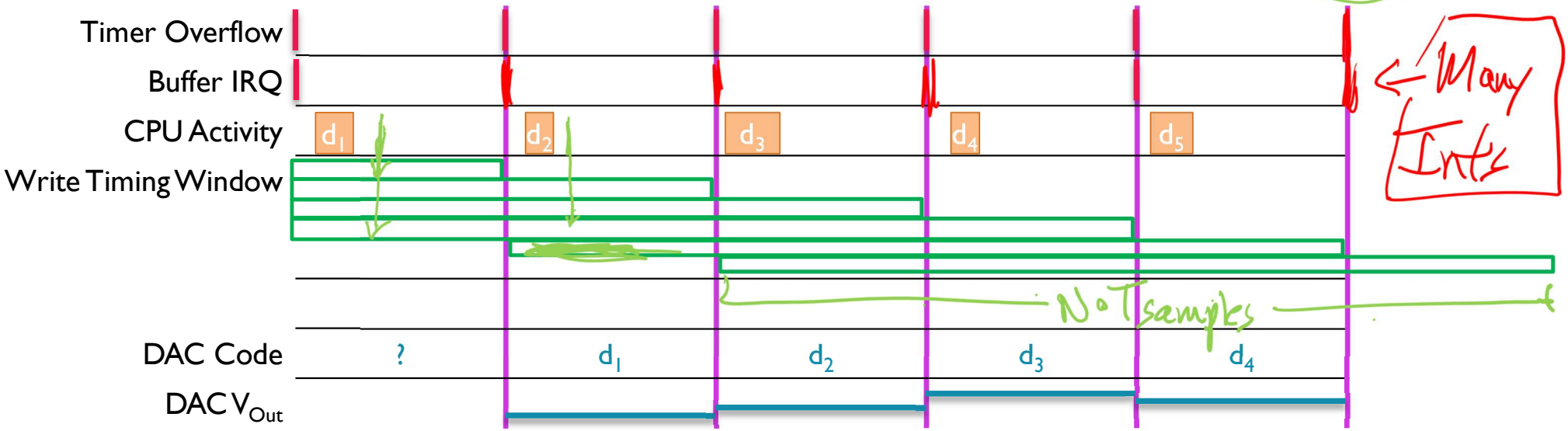




E. Add Multi-Entry Buffer for DAC

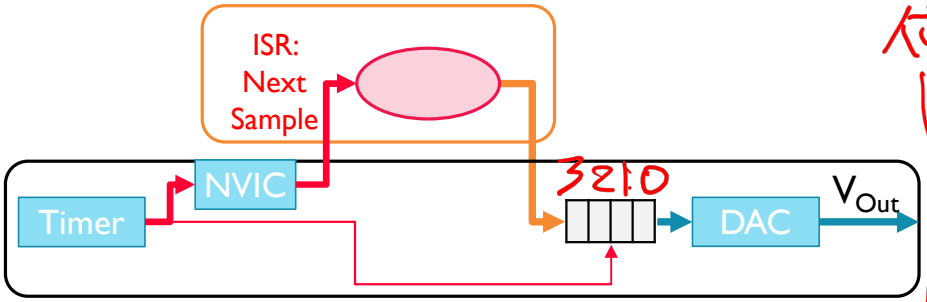


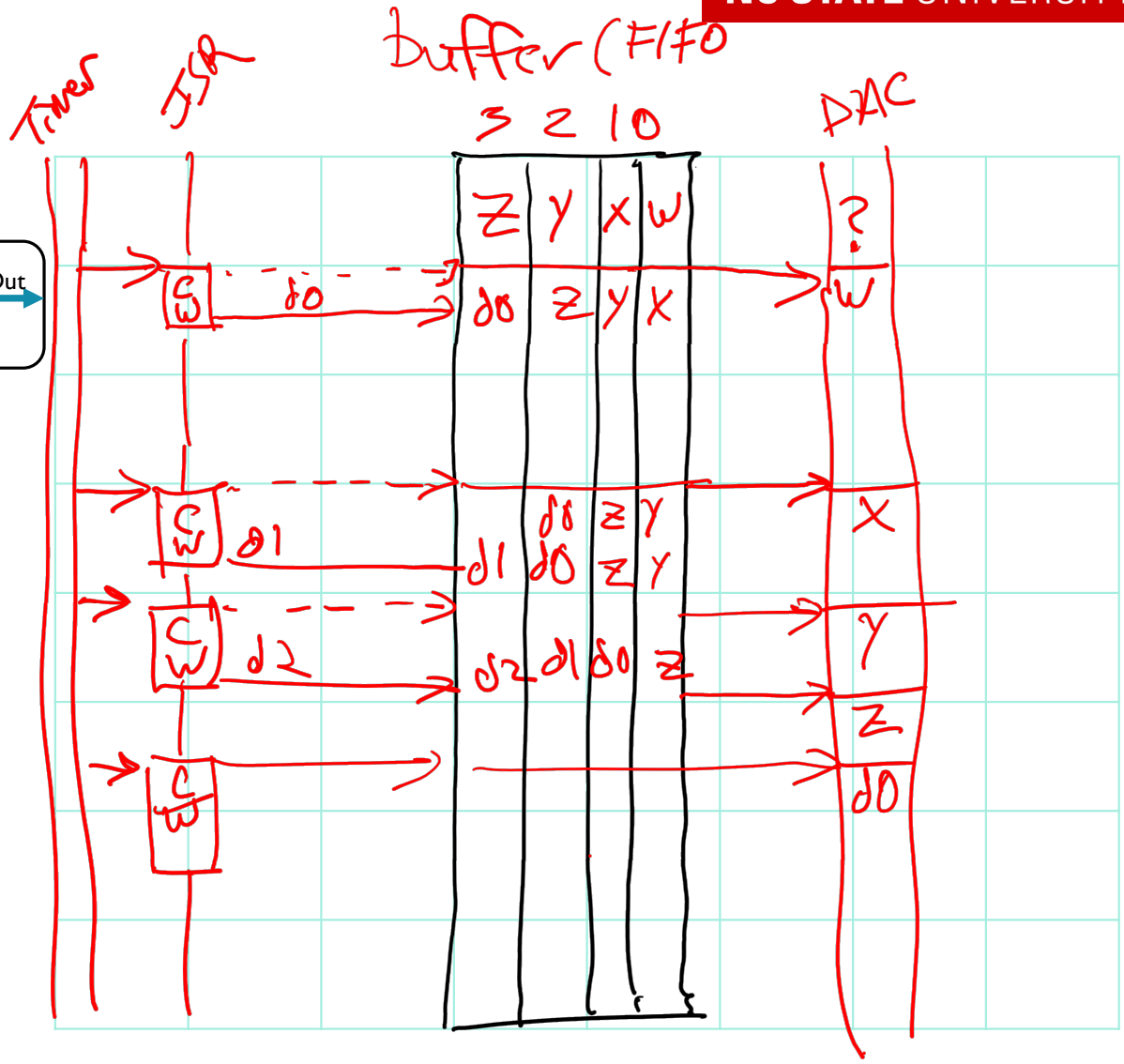




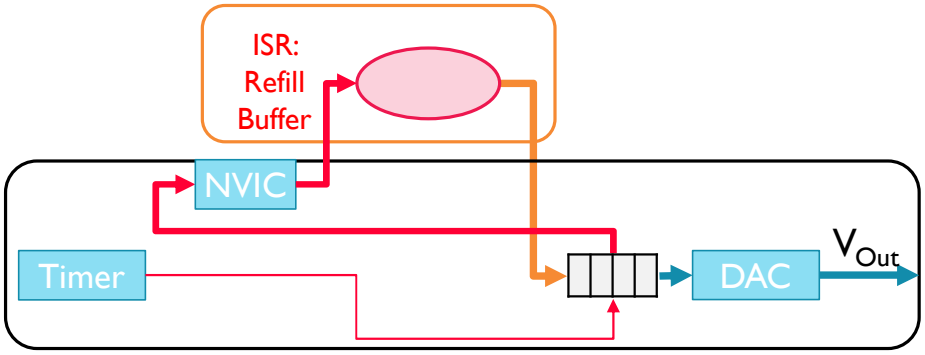
- Buffer is queue: first-in, first-out (FIFO)
- Buffering delays data, so will not work for all applications

Sequence Diagram



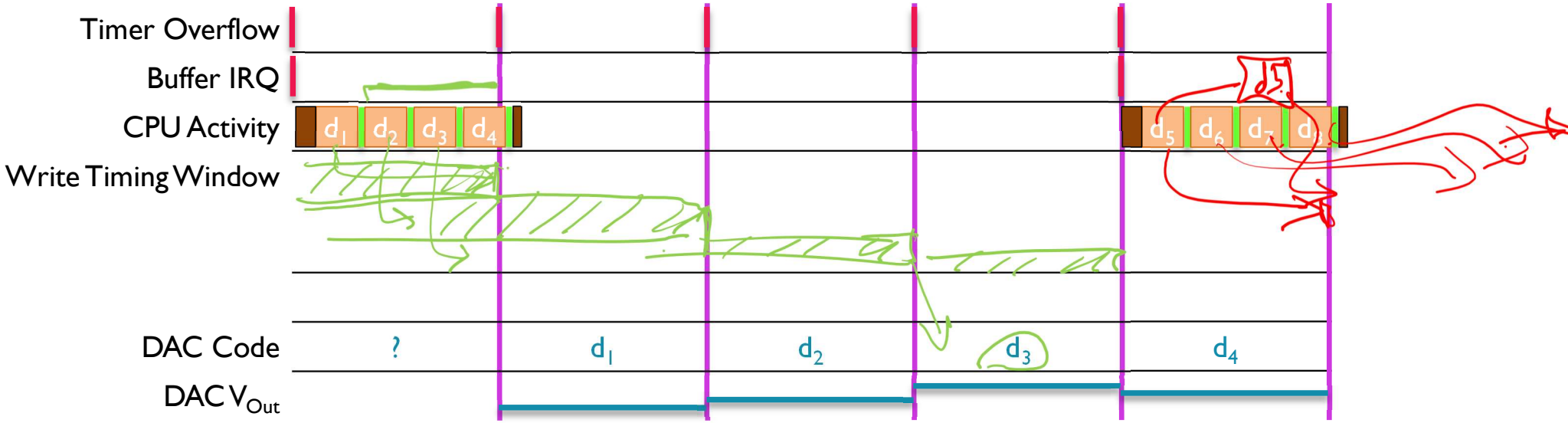


E2. DAC Buffer Generates Low/Empty Interrupt

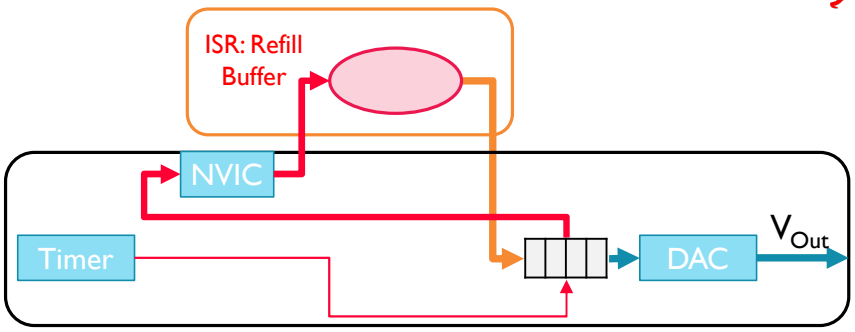


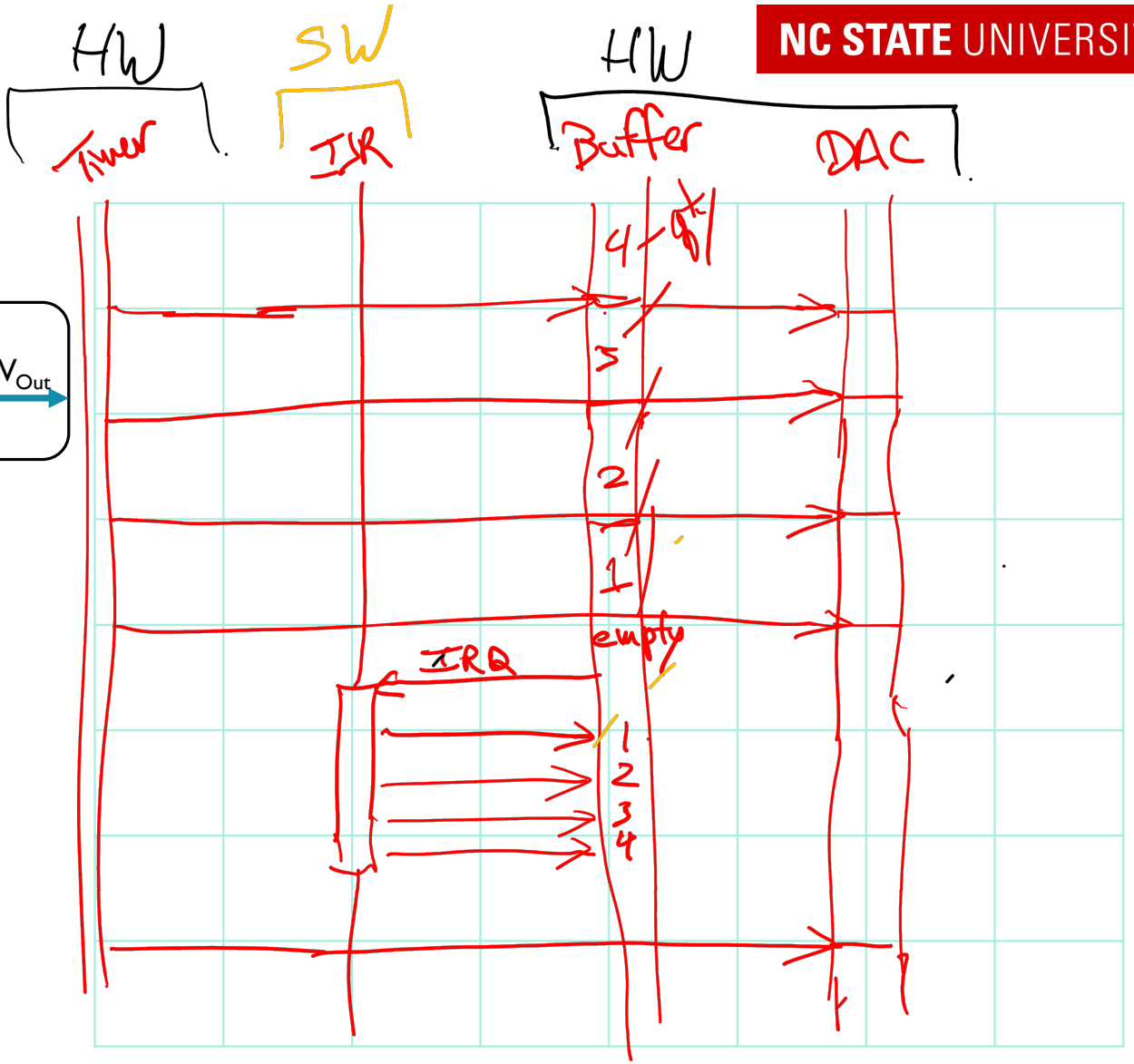

```
(code) {
}
```

- Burst/batch processing, so fewer interrupts needed
  - Divides number of interrupts by free buffer size

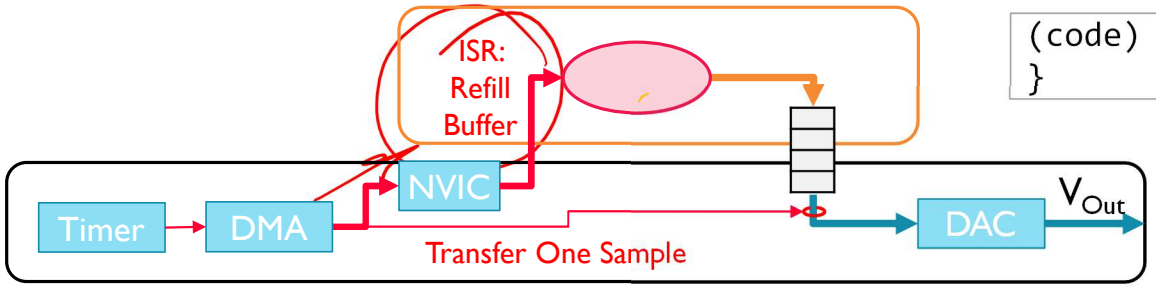


Sequence Diagram



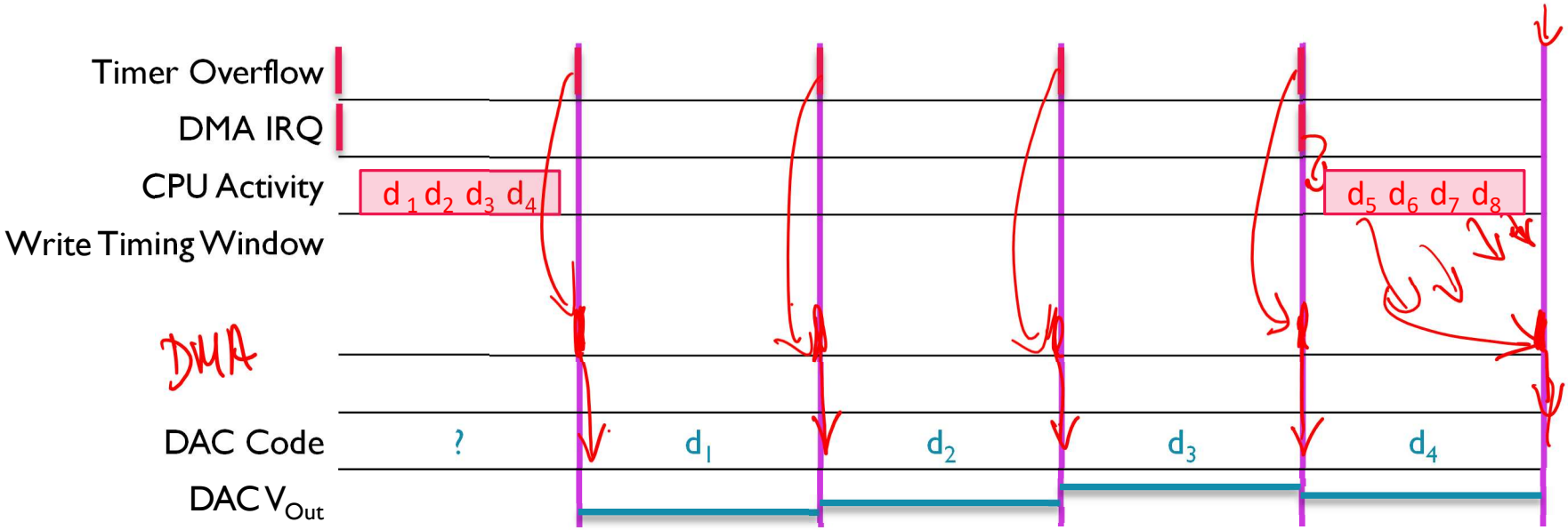


F. Timer Triggers DMA Transfer, DMA Request Refill with IRQ



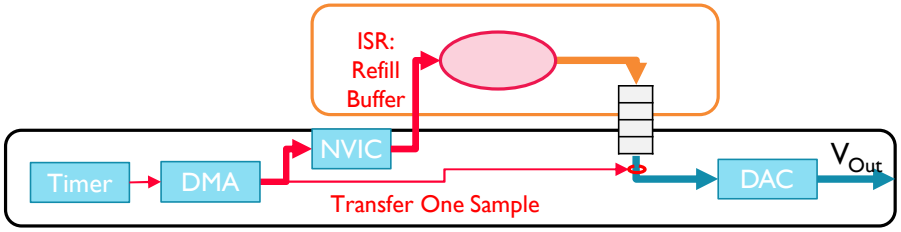

```
(code) {
}
```

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

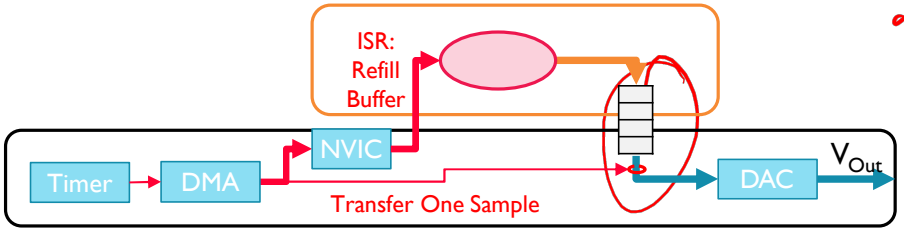


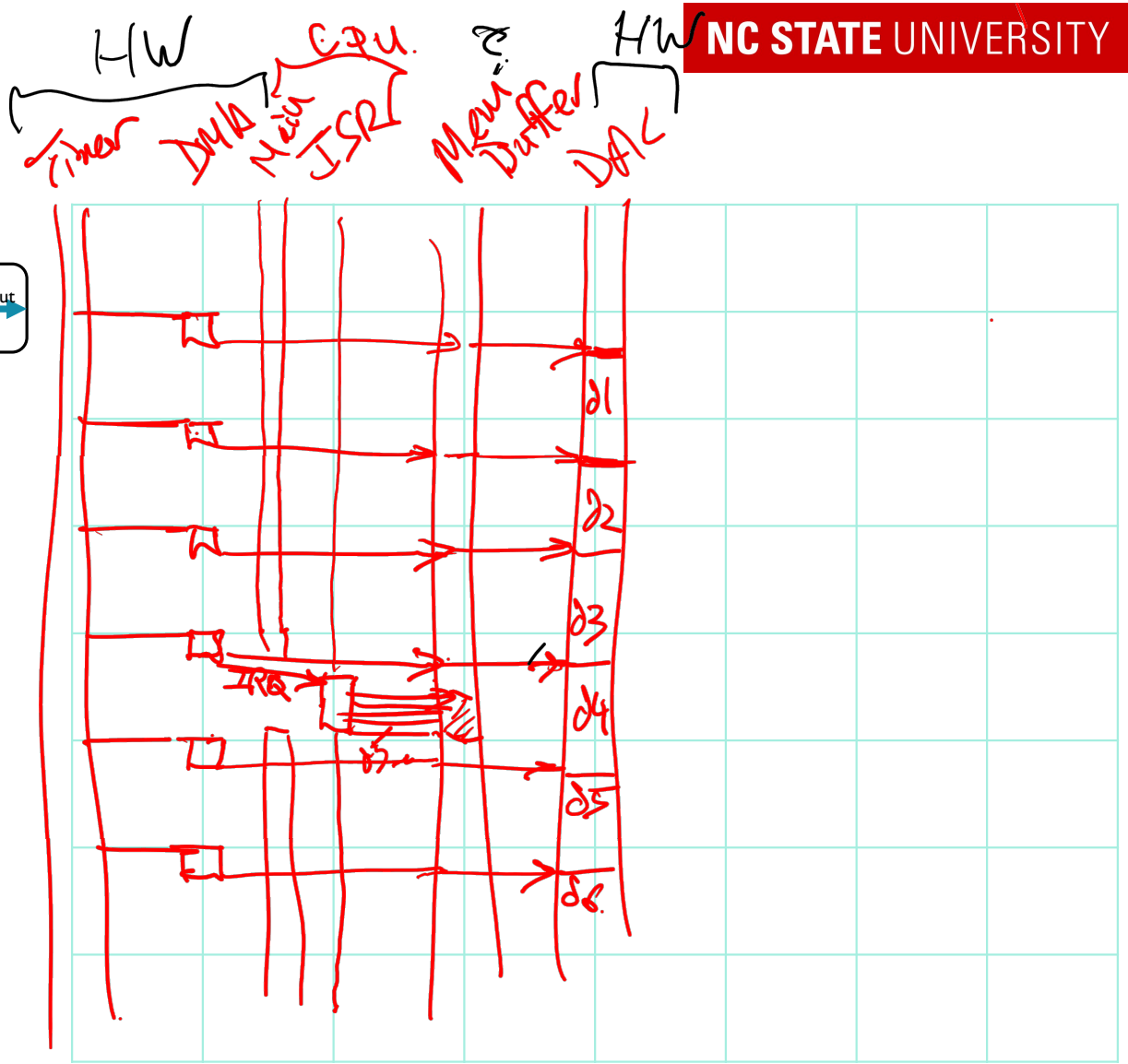
NC STATE UNIVERSITY

Sequence Diagram

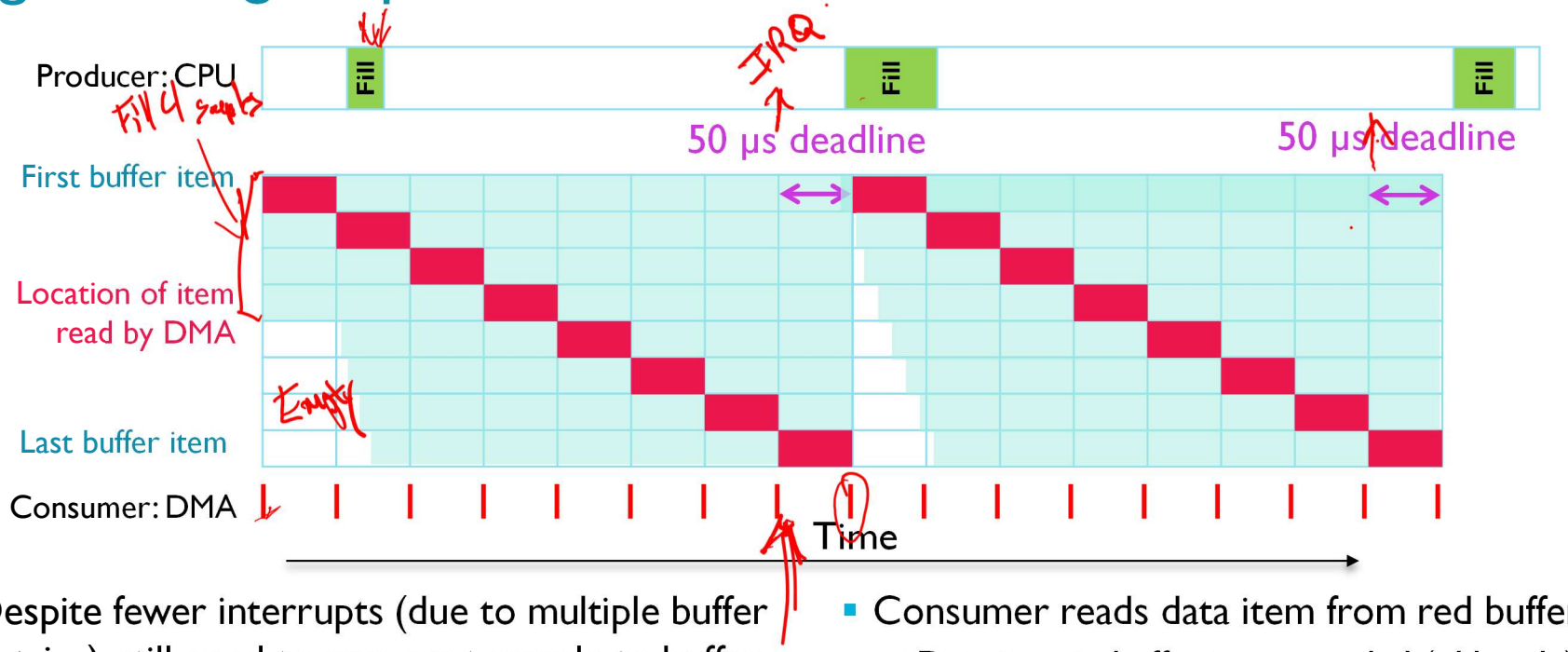


Sequence Diagram



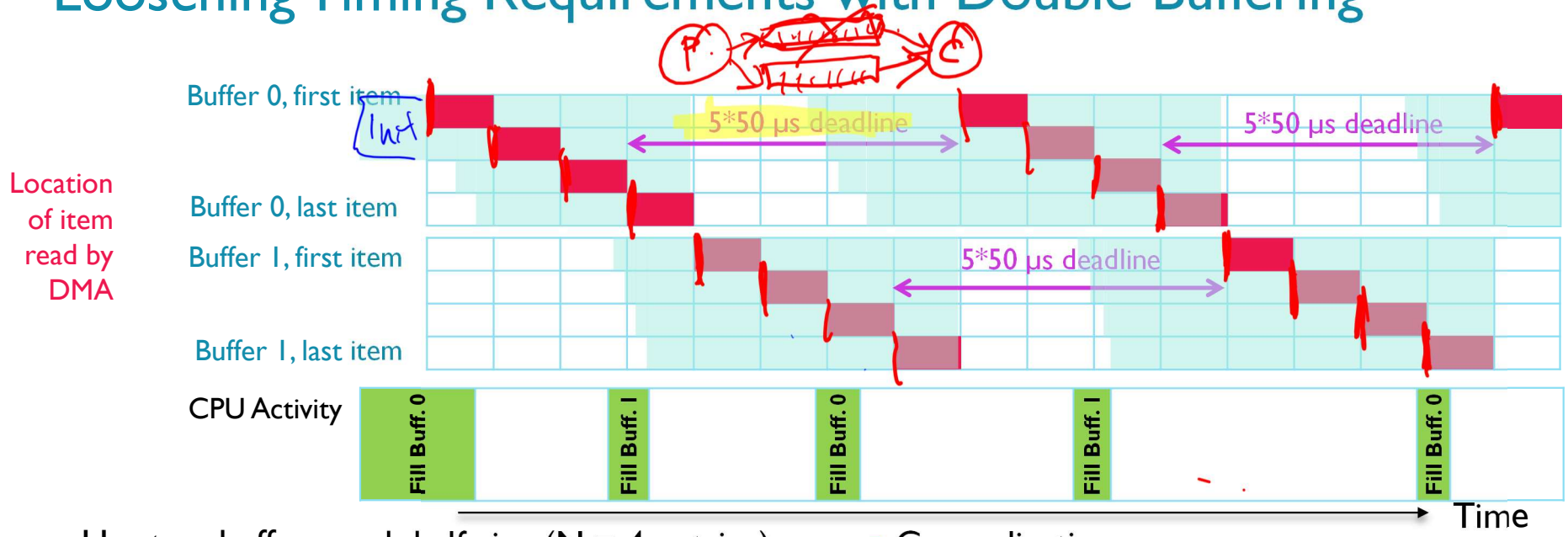


Tight Timing Requirements for Buffer



- Despite fewer interrupts (due to multiple buffer entries), still need to save next sample to buffer **before** DMA reads it
- Examine and understand timing requirements for buffer
- Producer adds data (light blue-green) to fill in buffer
  - In example, first four items have already been added
- Consumer reads data item from red buffer entry
  - 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)

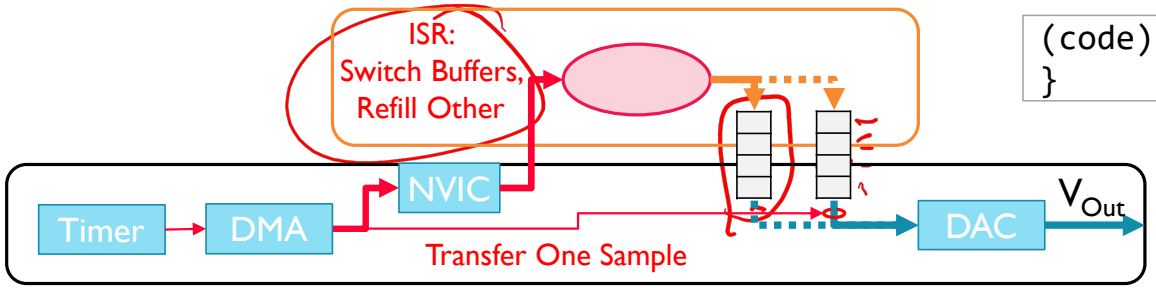
Loosening Timing Requirements with Double-Buffering



- 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 $(N+1)*T_{Sample}$ ☺

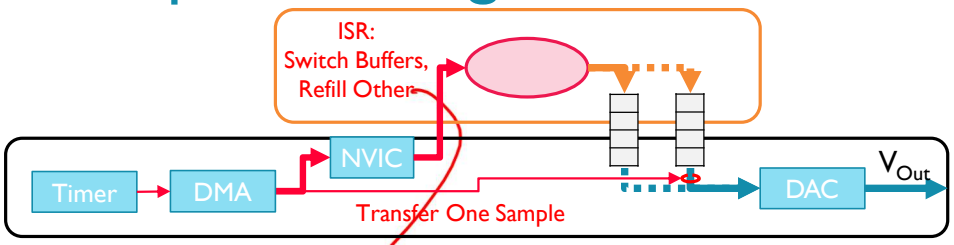
V2

G. Double Buffering

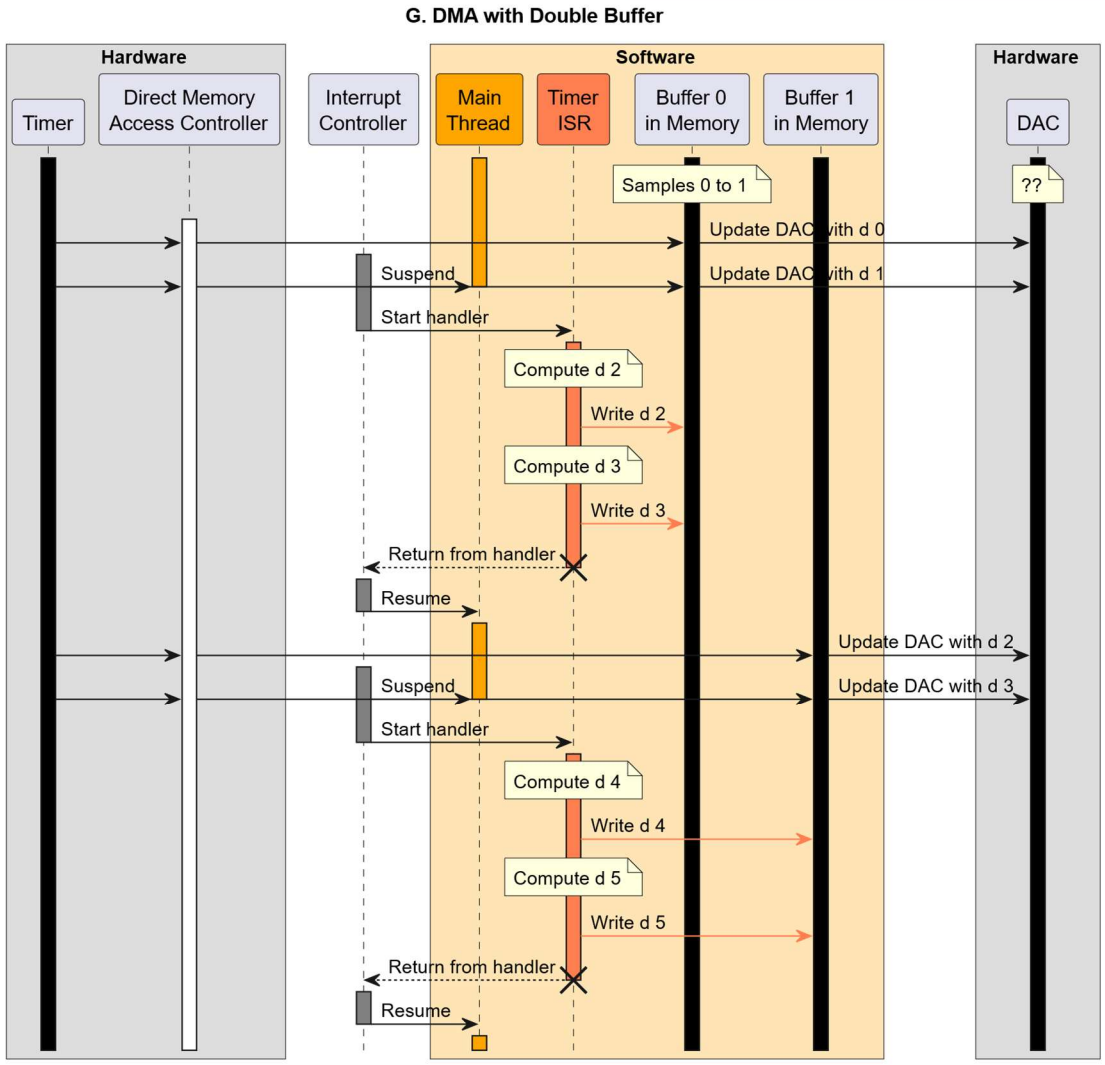



```
(code) {
}
```

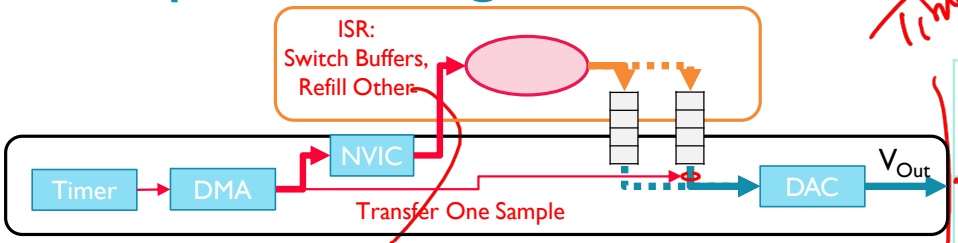
Sequence Diagram



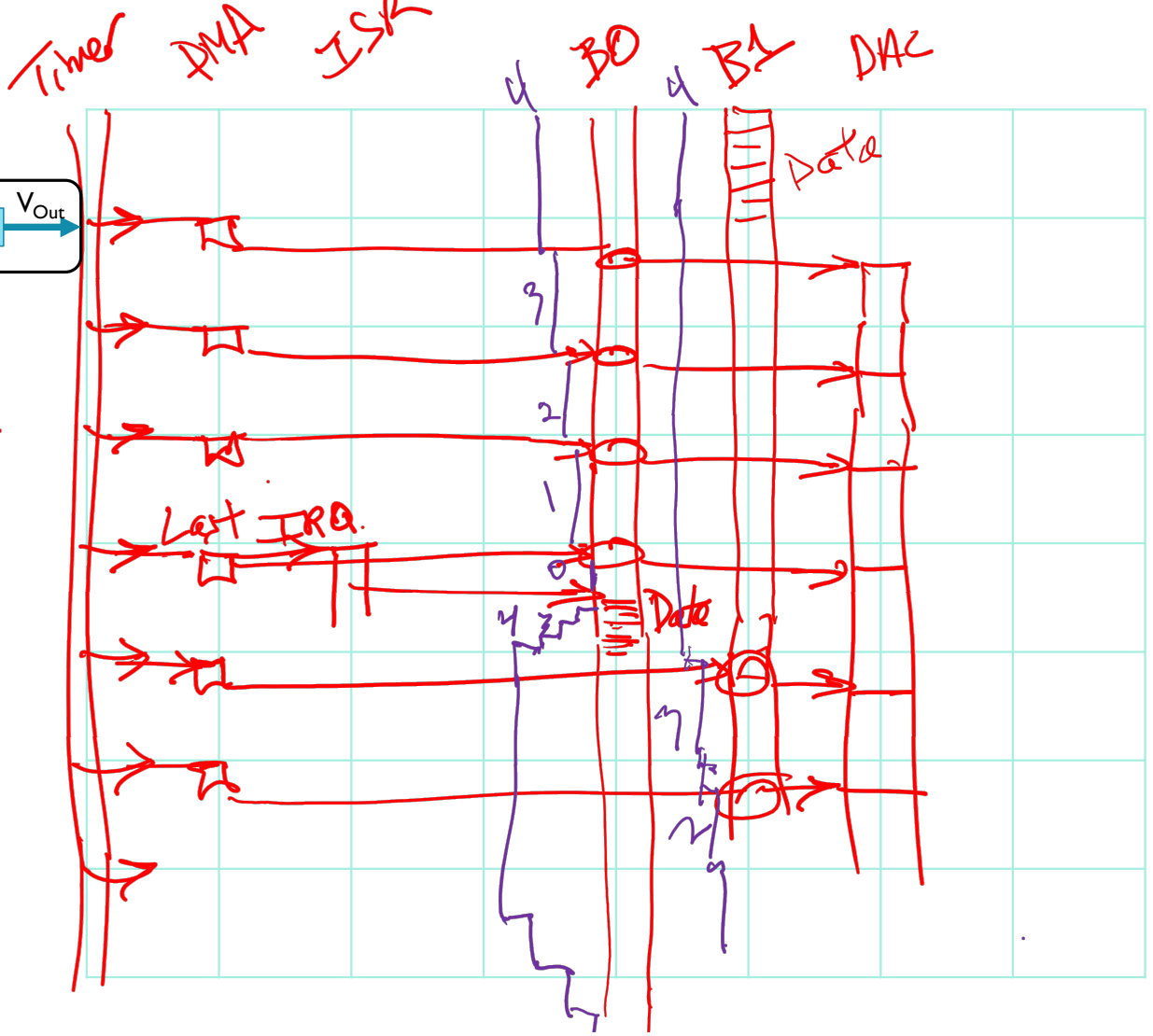
- Switch me to other buffer
- Refill this one



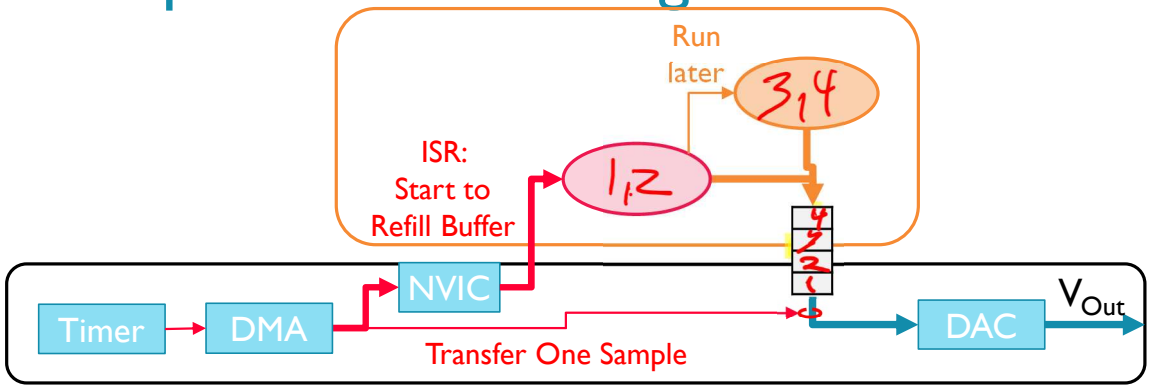
Sequence Diagram



- Switch me to other buffer
- Refill this one



H. Split Work into Urgent and Deferred

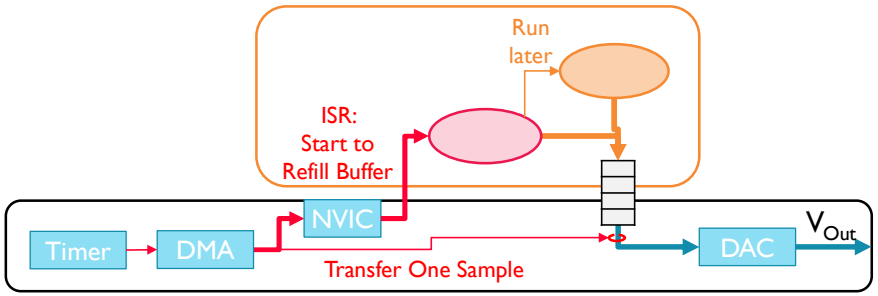


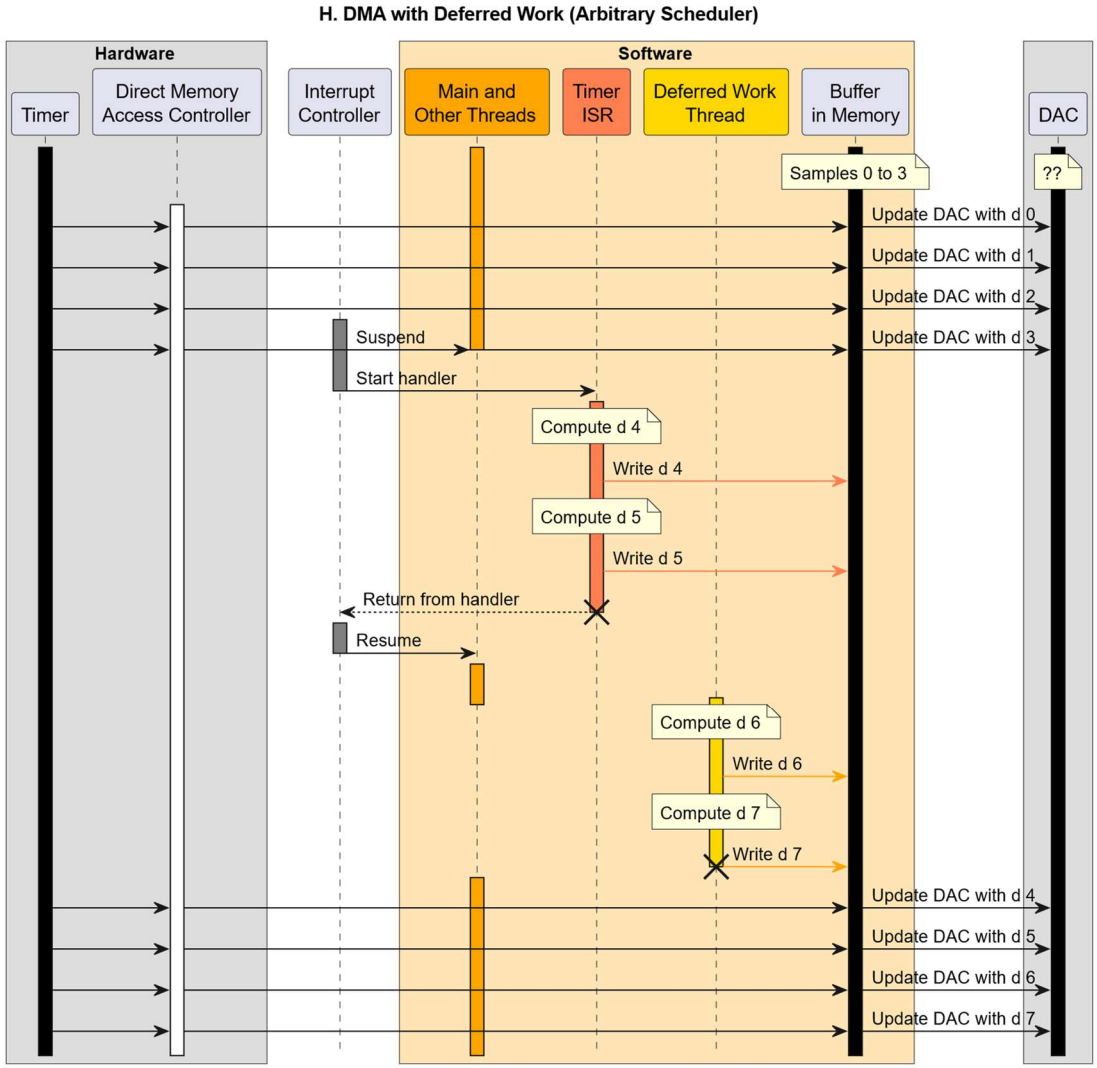

```
ISR {
}

thread_Deferred_Work {
}
```

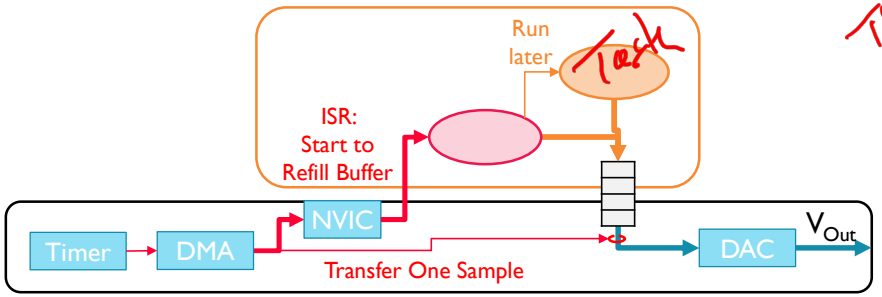
- When refill buffer ISR runs, it can delay **ISRs** (with same or lower priority) and **all** threads
  - Want to reduce time spent in ISR to improve responsiveness for other software processing
- Observation: Don't need to refill **entire buffer** in ISR
  - Refilling first sample in buffer is highest 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

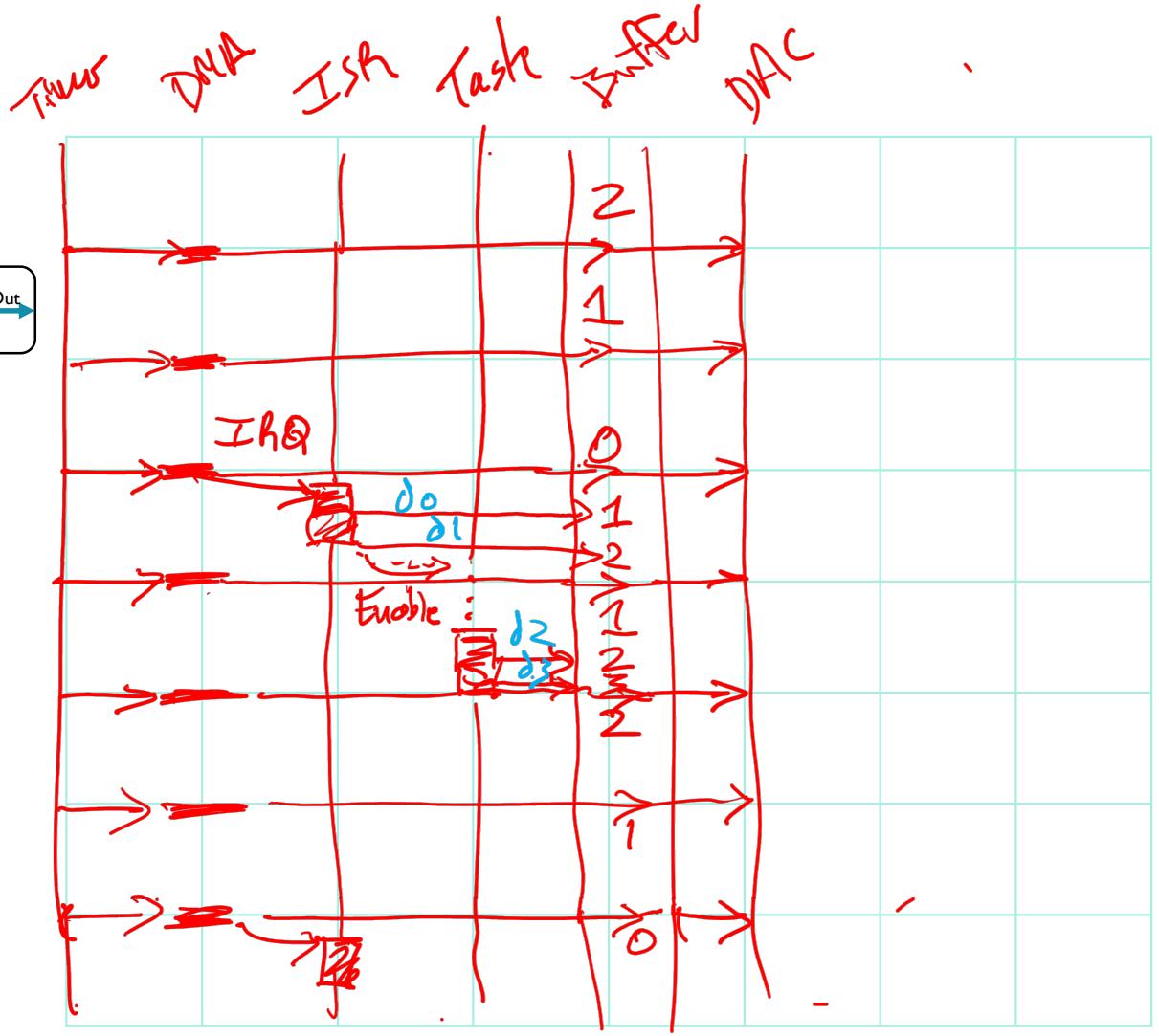
Sequence Diagram



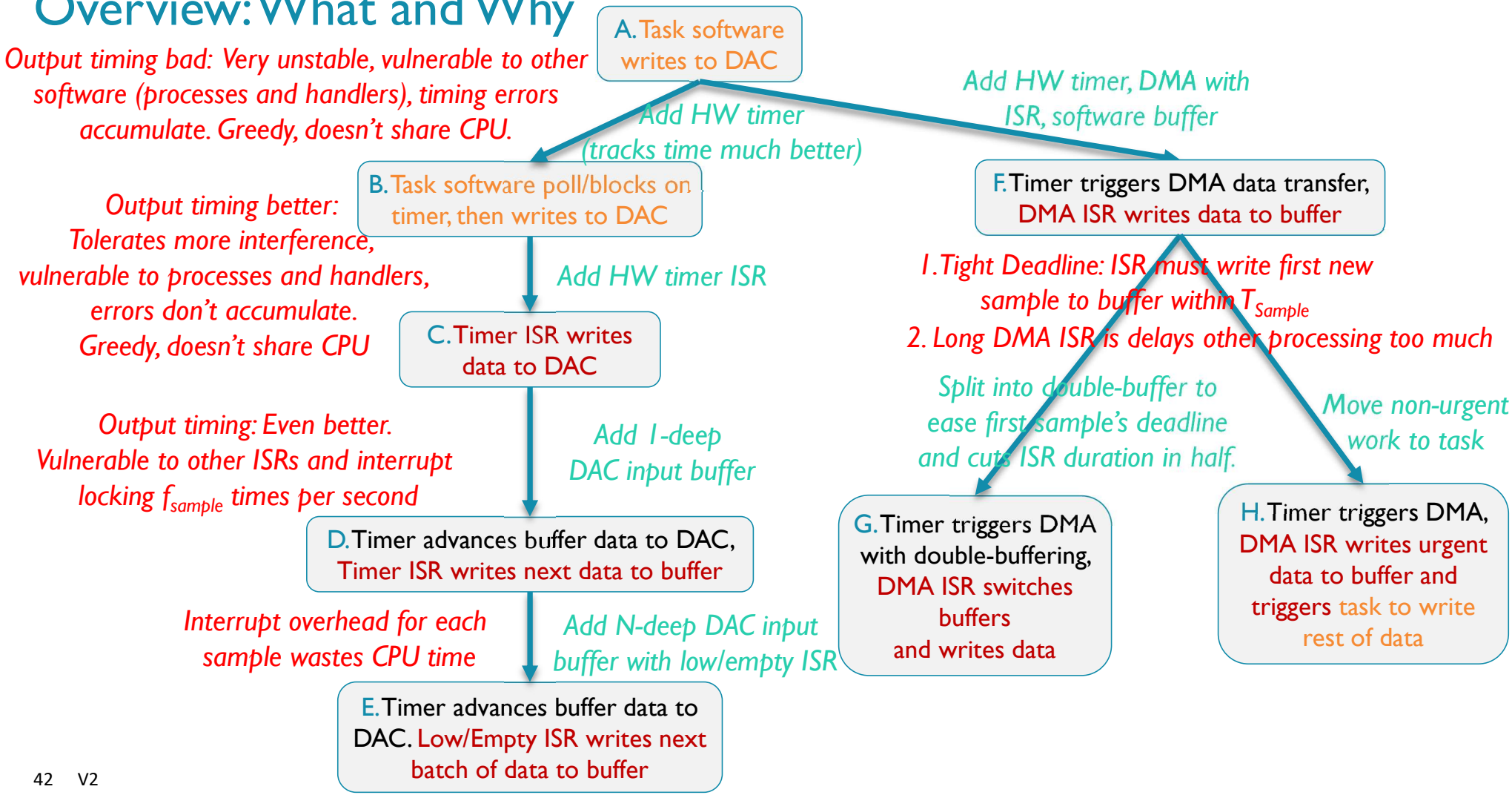


Sequence Diagram





Overview: What and Why



Vulnerabilities to Timing Interference

A. Task software 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. Low/Empty ISR writes next batch of data to buffer

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

G. Timer triggers DMA data transfer, DMA ISR switches buffers and writes data

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

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}				