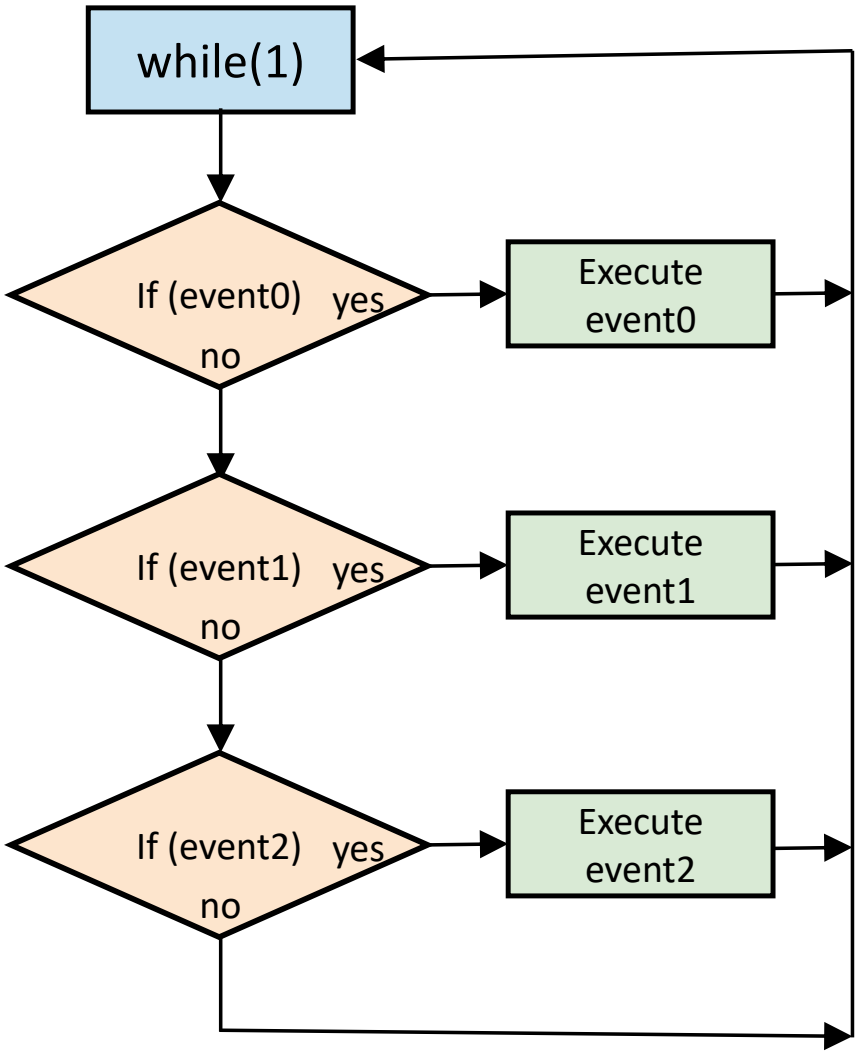
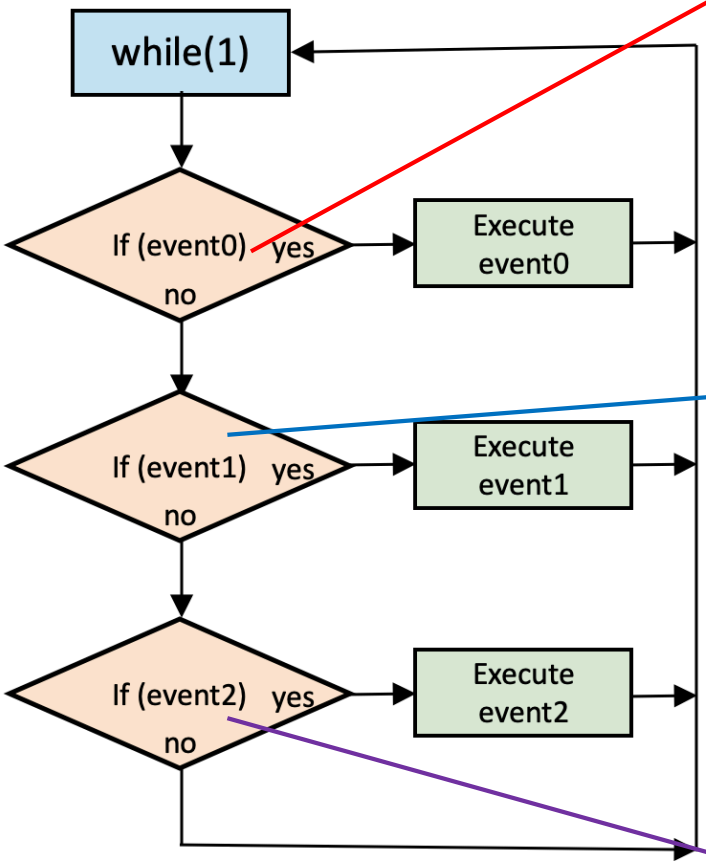
Priority Polling: Shortest deadline first

- event0 will always be serviced.
- Only if event0 is not waiting will event 1 be serviced.
- Only if event0 and event1 are not waiting will event 2 be serviced.



- What is likely to happen if event0 takes most of the CPU time?
  - Possibly 1 or 2 is starved?

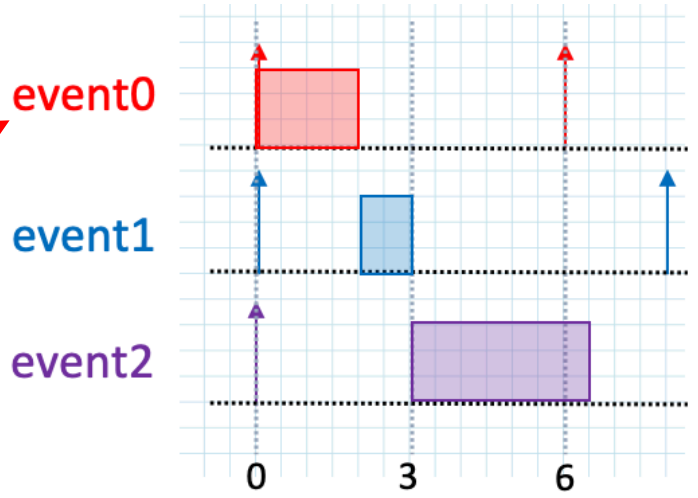
Priority Polling: event0 latency

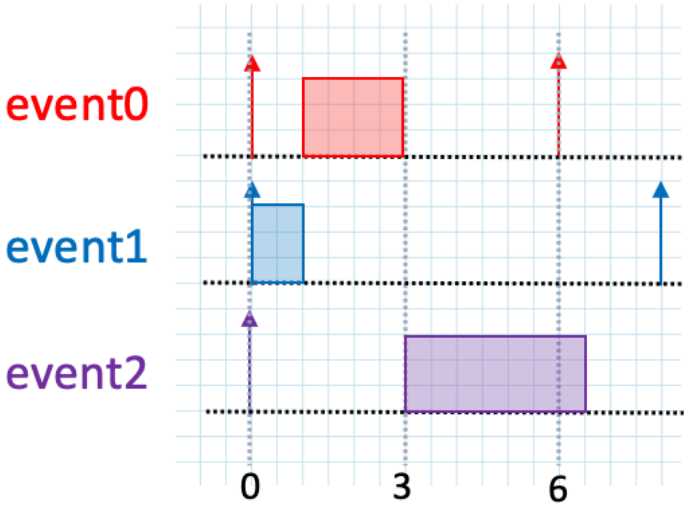


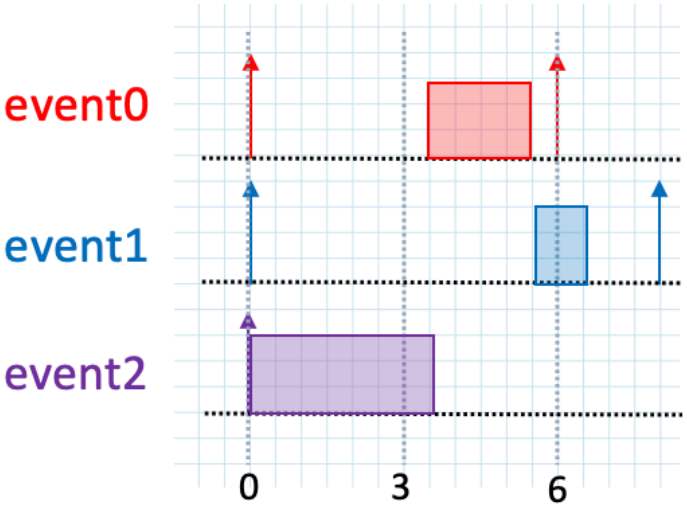
event0 latency = 0

event0 Latency = event1 execution time

event0 latency = Event2 execution time







- Max event0 latency = max(event1 execution time , event2 execution time)

Priority Polling

```
void main (void) {
  <init>;

  while (1) {
    if (<event0 occurred>) {
      <handle event0>;
    }
    else if (<event1 occurred>) {
      <handle event1>;
    }
    else if (<event2 occurred>) {
      <handle event2>;
    }
  }
}
```

t_{exec0} = 2 ms

t_{exec1} = 1 ms

t_{exec2} = 3.5 ms

Event	Period	Execution Time	Latency	Response Time (Latency + execution time)	Relative Deadline (Period)	Schedulable ? YES = response < deadline
event0	6 ms	2 ms	3.5 m (max of 1 OR 3.5)	5.5	5.5 < 6	YES
event1	8 ms	1 ms	5.5m	6.5m	6.5 <8	YES
event2	12 ms	3.5 ms	3 m	6.5m	6.5 <12	YES

- What will happen if event0 period gets shorter?
  - ?

ece3849_int_latency
Example 2: event2 = 3.5, Priority Polling

- Run settings

```
33 //#define ROUND_ROBIN_POLLING  1
34 #define PRIORITY_POLLING     1
35
36 // event and handler definitions
37 #define EVENT0_PERIOD           6007    // [us] event0 period
38 #define EVENT0_EXECUTION_TIME   2000    // [us] event0 handler execution time
39
40 #define EVENT1_PERIOD           8101    // [us] event1 period
41 #define EVENT1_EXECUTION_TIME   1000    // [us] event1 handler execution time
42
43 #define EVENT2_PERIOD           12301   // [us] event2 period
44 #define EVENT2_EXECUTION_TIME   3500    // [us] event2 handler execution time
45
```

- Run results

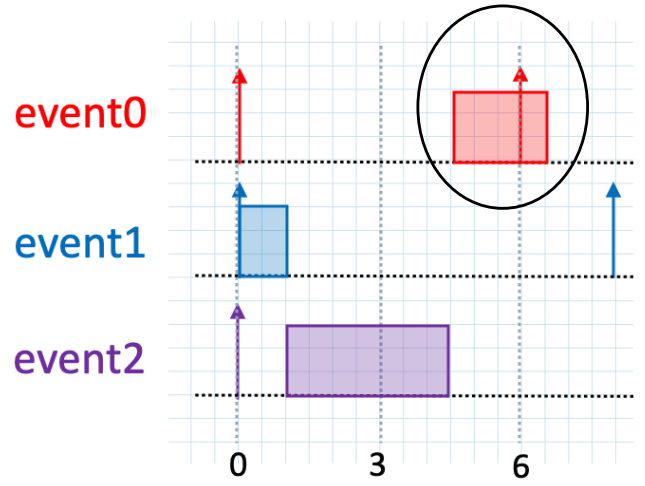
(x)= Variables | Expressions | Registers

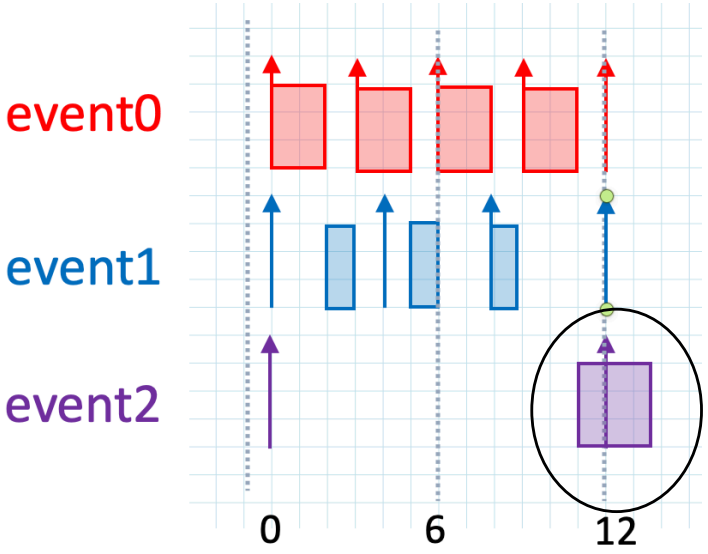
Expression	Type	Value	Expected Results
(x)= event0_latency/120.0f	float	3500.92505	3.5 ms
(x)= event1_latency/120.0f	float	5500.2334	5.5 ms
(x)= event2_latency/120.0f	float	3002.07495	3.0 ms
(x)= event0_response_time/120.0f	float	5501.56689	5.5 ms
(x)= event1_response_time/120.0f	float	6500.94189	6.5 ms
(x)= event2_response_time/120.0f	float	6502.75	6.5 ms
(x)= event0_missed_deadlines	unsigned int	0	
(x)= event1_missed_deadlines	unsigned int	0	
(x)= event2_missed_deadlines	unsigned int	0	
Add new expression			

Response time lower
No missed deadlines!

Round Robin vs. Priority Polling

- Round Robin Scheduling
  - Pro: Will guarantee each event gets processed.
  - Con: Response time is limited by the sum of the execution times of all events.
    - May cause events with short deadlines to miss their deadline.
- Prioritize Short Deadline First
  - Pro: Short dead lines will be met.
  - Con: Starvation
    - Events with long deadlines may never get serviced if higher priority tasks occur at a fast enough rate.
  - Figuring out maximum latency for low priority tasks complicated.





Example 3: Event 2 = 6 msec

- Lets try Priority Polling again

t_{exec0} = 2 ms

t_{exec1} = 1 ms

t_{exec2} = 6.0 ms

```
void main (void) {
  <init>;

  while (1) {
    if (<event0 occurred>) {
      <handle event0>;
    }
    else if (<event1 occurred>) {
      <handle event1>;
    }
    else if (<event2 occurred>) {
      <handle event2>;
    }
  }
}
```

Event	Period	Execution Time	Latency	Response Time (Latency + execution time)	Relative Deadline (Period)	Schedulable ? YES = response < deadline
event0	6 ms	2 ms	6	8	8 > 6	NO
event1	8 ms	1 ms	6+2 = 8	9	9 > 8	No
event2	12 ms	6 ms	2+1 = 3	9	9 < 12	YES

ece3849_int_latency
Example 3: event2 = 6.0, Priority Polling

- Run settings

```
33 //#define ROUND_ROBIN_POLLING  1
34 #define PRIORITY_POLLING     1
35
36 // event and handler definitions
37 #define EVENT0_PERIOD           6007    // [us] event0 period
38 #define EVENT0_EXECUTION_TIME   2000    // [us] event0 handler execution time
39
40 #define EVENT1_PERIOD           8101    // [us] event1 period
41 #define EVENT1_EXECUTION_TIME   1000    // [us] event1 handler execution time
42
43 #define EVENT2_PERIOD           12301   // [us] event2 period
44 #define EVENT2_EXECUTION_TIME   6000    // [us] event2 handler execution time
```

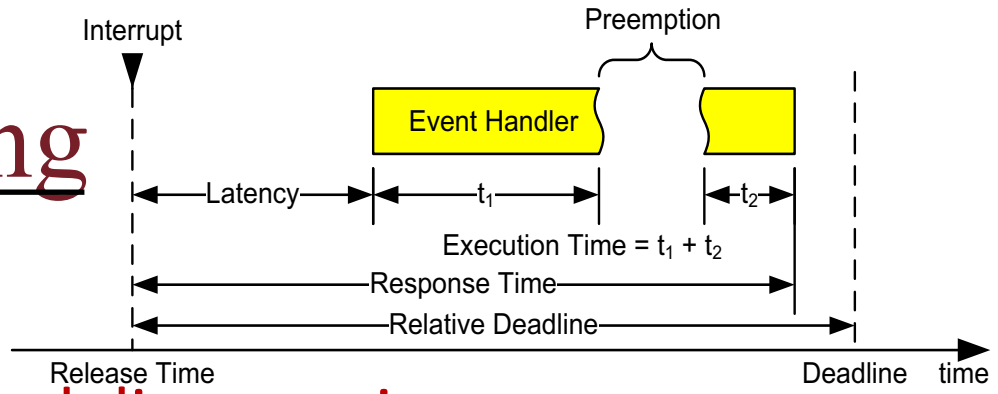
- Run results

(x)= Variables | Expressions | Registers

Expression	Type	Value	Expected Results
(x)= event0_latency/120.0f	float	6000.875	6.0 ms
(x)= event1_latency/120.0f	float	8098.4585	8.0 ms
(x)= event2_latency/120.0f	float	3002.3833	3.0 ms
(x)= event0_response_time/120.0f	float	8001.5415	8.0 ms
(x)= event1_response_time/120.0f	float	9099.18359	9.0 ms
(x)= event2_response_time/120.0f	float	9003.05859	9.0 ms
(x)= event0_missed_deadlines	unsigned int	940	
(x)= event1_missed_deadlines	unsigned int	332	Tons of missed deadlines!
(x)= event2_missed_deadlines	unsigned int	0	
Add new expression			

- What strategies can we try next to meet all the deadlines?
  - Try adding preemption?

Preemptive scheduling



- The simplest preemptive scheduling requires an interrupt controller.
  - The Cortex-M4 is designed for real time systems and supports an interrupt-driven strategy.
- The infinite while loop runs non-critical tasks in the foreground.
- Interrupt controller automatically calls an Interrupt Service routine (ISR) as soon as an event occurs.
  - The foreground tasks are preempted when ISRs are called.
  - ISR's are said to run in the background.
  - Use of foreground and background is arbitrary and sometimes swapped.
- Higher priority ISRs can interrupt / preempt lower priority ones.

Interrupt-Driven Design

- Beginning of main
  - Initialize Interrupts and hardware.
  - While loop (in foreground)
    - Performs low priority non-interrupt driven tasks.
    - Or it just loops if nothing to do.
  - ISR_event0 – Highest priority ISR
  - ISR_event1 – Mid priority ISR
  - ISR_event2 – Lowest priority ISR

Outline of an interrupt-driven real-time system:

```
void main (void) {
  <init>;

  while (1) {
    <perform non real-time work>;
  }
}

void ISR_event0(void) {
  <handle event0>;
}

void ISR_event1(void) {
  <handle event1>;
}

void ISR_event2(void) {
  <handle event2>;
}
```

- Interrupt controller automatically calls ISR routines to operate in the background

Prioritization of Interrupts

- Higher priority ISR can preempt / interrupt lower priority ISRs.
- How should we prioritize the interrupts?
  - Shorter periods should have higher priority.
  - Smaller deadlines
- Once setup it is possible to turn on and off preemption.
  - When preemption is off, the system behaves like a priority polling design.
  - But we need to respond more quickly than this.

ece3849_int_latency: Enable Interrupts

- Comment out polling option and enable ISR.

```
33 //#define ROUND_ROBIN_POLLING  1
34 //#define PRIORITY_POLLING     1
35
36 // event and handler definitions
37 #define EVENT0_PERIOD           6007    // [us] event0 period
38 #define EVENT0_EXECUTION_TIME   2000    // [us] event0 handler execution time
39
40 #define EVENT1_PERIOD           8101    // [us] event1 period
41 #define EVENT1_EXECUTION_TIME   1000    // [us] event1 handler execution time
42
43 #define EVENT2_PERIOD           12301   // [us] event2 period
44 #define EVENT2_EXECUTION_TIME   6000    // [us] event2 handler execution time
45
46 // build options
47 #define ENABLE_ISR 1
48 //#define DISABLE_INTERRUPTS_IN_ISR // if defined, interrupts are disabled in t
49
```

ece3849_int_latency: Enable Interrupts

- ENABLE_ISR allows the IntMasterEnable function to be called.

```
117 #ifdef ENABLE_ISR
118     IntMasterEnable();
119 #endif
```

- Creates a while loop with nothing in it.

```
156 #ifdef ENABLE_ISR
157     while(true) {
158
159     }
160 #endif
```

- In the vector table in tm4c1294ncpdt_startup_css.c the event handler functions are assigned to the Timer A interrupts

```
109     event0_handler,
110     IntDefaultHandler,
111     event1_handler,
112     IntDefaultHandler,
113     event2_handler,
```

```
// Timer 0 subtimer A
    // Timer 0 subtimer B
// Timer 1 subtimer A
    // Timer 1 subtimer B
// Timer 2 subtimer A
```

ece3849_int_latency: Setting Priority

```
92    // initialize general purpose timers 0-2 for periodic interrupts
93    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
94    TimerDisable(TIMER0_BASE, TIMER_BOTH);
95    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
96    TimerLoadSet(TIMER0_BASE, TIMER_A, TIMER0_PERIOD - 1);
97    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
98    IntPrioritySet(INT_TIMER0A, 0); // 0 = highest priority, 32 = next lower
99    IntEnable(INT_TIMER0A);
100
101   SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
102   TimerDisable(TIMER1_BASE, TIMER_BOTH);
103   TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
104   TimerLoadSet(TIMER1_BASE, TIMER_A, TIMER1_PERIOD - 1);
105   TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
106   IntPrioritySet(INT_TIMER1A, 32); // 0 = highest priority, 32 = next lower
107   IntEnable(INT_TIMER1A);
108
109   SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
110   TimerDisable(TIMER2_BASE, TIMER_BOTH);
111   TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC);
112   TimerLoadSet(TIMER2_BASE, TIMER_A, TIMER2_PERIOD - 1);
113   TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
114   IntPrioritySet(INT_TIMER2A, 64); // 0 = highest priority, 32 = next lower
115   IntEnable(INT_TIMER2A);
```

highest priority = lowest number = 0

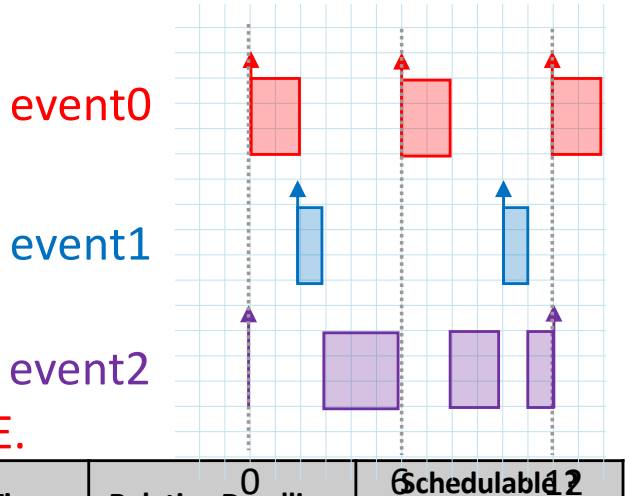
Mid priority =32

Lowest priority = highest number = 64

- The priority value register set by IntPrioritySet function is an 8-bit value.
- However, only the 3 most significant bits of the value are used for interrupt priority.
  - Valid values are 0, 32, 64, 96, 128, 160, 192, 224 (multiples of 32)

Example 3: Event 2 = 6 msec

- Preemptive Interrupt Driven Design
  - Interrupt controller calls ISR immediately on event0.
    - There will be a small delay to call ISR.
  - Higher priority events interrupt/preempt lower priority events.
    - Higher priority events may interrupt multiple times.
    - Events do not have fixed phase and can happen anytime relative to each other.

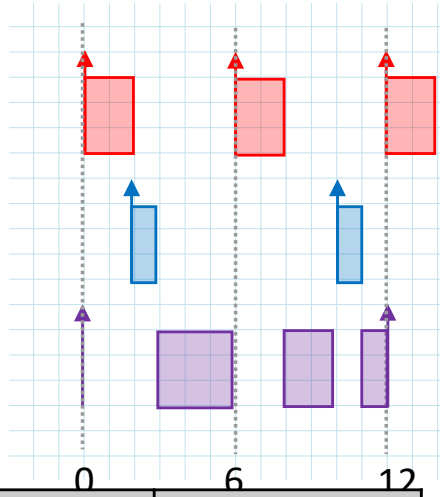


SEE CLARIFICATION ON NEXT PAGE.

Event	Period	Execution Time	Latency (See clarification on next page)	Response Time (Latency + ?)	Relative Deadline (Period)	Schedulable? YES = response < deadline
event0	6 ms	2 ms	0 because it is top dog	0+2 = 2	6	YES
event1	8 ms	1 ms	2 ms, only waits for 0. Maybe 0 happens twice	2 + 1 = 3 ? Maybe 5? Simulation shows 3	(3 or 5) < 8	YES
event2	12 ms	6 ms	2*2ms +2*1 = 6ms Wait to start + interrupt	12 m	12 == 12	YES, But marginal.

Example clarification

- Latency: Is ALWAYS the time it takes to start the event.
- Execution time is the time it takes to complete the event
- For cases with preemption
  - Response time = Latency + Execution time + Preemption time.



Event	Period	Execution Time	Latency (Correction)	Response Time (Latency + Execution + premption)	Relative Deadline (Period)	Schedulable ? YES = response < deadline
event0	6 ms	2 ms	0	0+2 = 2	6	YES
event1	8 ms	1 ms	2 ms, only waits for event0.	2 + 1 = 3 ? Maybe 5 if event 1 preempts 2x	(3 or 5) < 8	YES
event2	12 ms	6 ms	2 ms + 1ms = 3ms, Waits for event 1 and event2	12 ms = 3 ms latency +6 ms+ 3ms if preemption from event 0 and 1	12 == 12	YES, But marginal.

ece3849_int_latency
Example 3: event2 = 6.0, Interrupt Driven

- Run settings

```
33 //#define ROUND_ROBIN_POLLING  1
34 //#define PRIORITY_POLLING     1
35
36 // event and handler definitions
37 #define EVENT0_PERIOD           6007    // [us] event0 period
38 #define EVENT0_EXECUTION_TIME   2000    // [us] event0 handler execution time
39
40 #define EVENT1_PERIOD           8101    // [us] event1 period
41 #define EVENT1_EXECUTION_TIME   1000    // [us] event1 handler execution time
42
43 #define EVENT2_PERIOD           12301   // [us] event2 period
44 #define EVENT2_EXECUTION_TIME   6000    // [us] event2 handler execution time
45
46 // build options
47 #define ENABLE_ISR 1
48 //#define DISABLE_INTERRUPTS_IN_ISR // if defined, interrupts are disabled in t
```

- Run results

(x)= Variables | Expressions | Registers

Expression	Type	Value
(x)= event0_latency/120.0f	float	0.266666681
(x)= event1_latency/120.0f	float	2000.14172
(x)= event2_latency/120.0f	float	3001.25
(x)= event0_response_time/120.0f	float	2000.84998
(x)= event1_response_time/120.0f	float	3001.95825
(x)= event2_response_time/120.0f	float	12005.1504
(x)= event0_missed_deadlines	unsigned int	0
(x)= event1_missed_deadlines	unsigned int	0
(x)= event2_missed_deadlines	unsigned int	0

Expected Results
0 ms
2.0 ms
3.0 ms
2.0 ms
3.0 ms
12.0 ms (marginal)

- Why is the event0 latency not 0?

Latency of Highest Interrupt

- If ISR are enabled, the latency of the highest priority interrupt will be small but not zero.
  - The current operation needs to complete.
  - The context of the CPU state is saved.
  - The first instruction of the ISR is fetched.

- Also in this the ece3849_int_latency program several operations were executed before the latency was calculated.
  - The time was read to calculate the latency.
  - This would normally not be considered part of latency.

ece3849_int_latency
Example 3: event2 = 6.0,
Interrupt Driven, all the same priority

```
92      // initialize general purpose timers 0-2 for periodic interrupts
93      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
94      TimerDisable(TIMER0_BASE, TIMER_BOTH);
95      TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
96      TimerLoadSet(TIMER0_BASE, TIMER_A, TIMER0_PERIOD - 1);
97      TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
98      IntPrioritySet(INT_TIMER0A, 0); // 0 = highest priority, 32 = next lower
99      IntEnable(INT_TIMER0A);
100
101     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
102     TimerDisable(TIMER1_BASE, TIMER_BOTH);
103     TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
104     TimerLoadSet(TIMER1_BASE, TIMER_A, TIMER1_PERIOD - 1);
105     TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
106     IntPrioritySet(INT_TIMER1A, 0); // 0 = highest priority, 32 = next lower
107     IntEnable(INT_TIMER1A);
108
109     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
110     TimerDisable(TIMER2_BASE, TIMER_BOTH);
111     TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC);
112     TimerLoadSet(TIMER2_BASE, TIMER_A, TIMER2_PERIOD - 1);
113     TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
114     IntPrioritySet(INT_TIMER2A, 0); // 0 = highest priority, 32 = next lower
115     IntEnable(INT_TIMER2A);
```

- What happens if we make them all the same priority?
  - Reverts to Priority Polling?

ece3849_int_latency
Example 3: event2 = 6.0,
Interrupt Driven, all the same priority

- Interrupts with the same software priority are still given priority in the hardware through the vector table.
- If all events have same software priority, system behaves the same as in priority polling design.

(x)= Variables | Expressions | Registers

Expression	Type	Value	Priority Polling
(x)= event0_latency/120.0f	float	6000.4834	6.0 ms
(x)= event1_latency/120.0f	float	8097.3418	8.0 ms
(x)= event2_latency/120.0f	float	3001.1416	3.0 ms
(x)= event0_response_time/120.0f	float	8001.1001	8.0 ms
(x)= event1_response_time/120.0f	float	9097.94141	9.0 ms
(x)= event2_response_time/120.0f	float	9001.75879	9.0 ms
(x)= event0_missed_deadlines	unsigned int	834	Not schedulable
(x)= event1_missed_deadlines	unsigned int	291	
(x)= event2_missed_deadlines	unsigned int	0	
Add new expression			

Disabled Interrupts in the ISR

- In ece3849_int_latency, the interrupts can be disabled in the ISR routines.

```
46 // build options
47 #define ENABLE_ISR 1
48 #define DISABLE_INTERRUPTS_IN_ISR // if defined, interrupts are disabled in the body of each ISR
49
```

```
203 void event0_handler(void)
204 {
205 #ifdef DISABLE_INTERRUPTS_IN_ISR
206     IntMasterDisable();
207 #endif
208     uint32_t t = TIMER0_PERIOD - TIMER0_TAR_R; // read Timer A count using direct regist
209     if (t > event0_latency) event0_latency = t; // measure latency
210     TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); // clear interrupt flag
211
212     delay_us(EVENT0_EXECUTION_TIME); // handle event0
213
214     if (TimerIntStatus(TIMER0_BASE, 1) & TIMER_TIMA_TIMEOUT) { // next event occurred
215         event0_missed_deadlines++;
216         t = 2 * TIMER0_PERIOD; // timer overflowed since last event
217     }
218     else t = TIMER0_PERIOD;
219     t -= TimerValueGet(TIMER0_BASE, TIMER_A); // read Timer A count using driver
220     if (t > event0_response_time) event0_response_time = t; // measure response time
221 #ifdef DISABLE_INTERRUPTS_IN_ISR
222     IntMasterEnable();
223 #endif
224 }
225
```

Interrupts Globally disabled at start of ISR

Interrupts Globally re-enabled at end of ISR

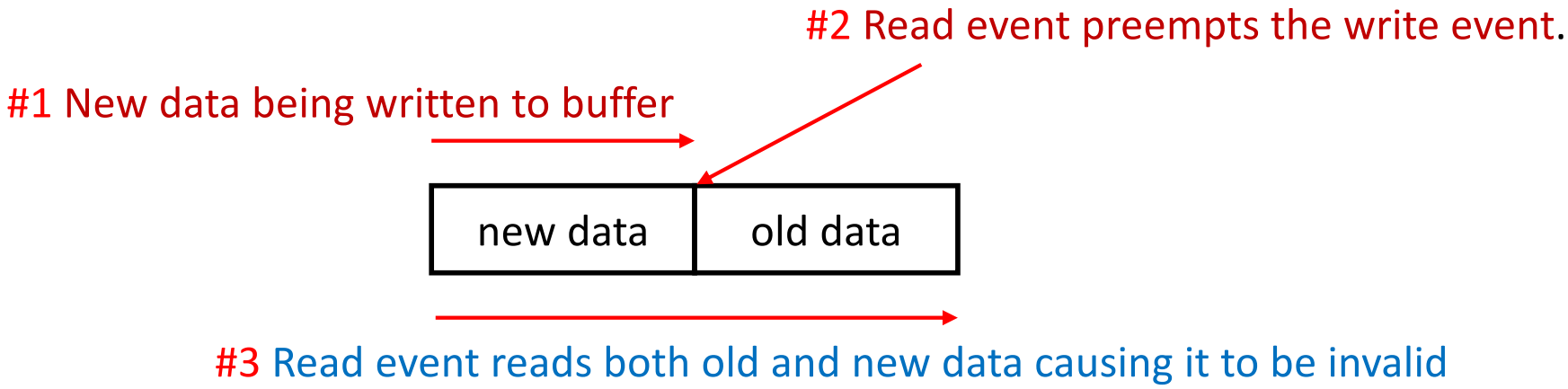
Effect of Disabled Interrupts in the ISR

- Under what conditions is it important to disable interrupts?
  - Low priority event with some special need for continuous operation.
  - Data hazard – shared variable between low and high priority events.

- What happens to the response time if we do this?
  - Back to Priority Polling performance - failing deadlines?

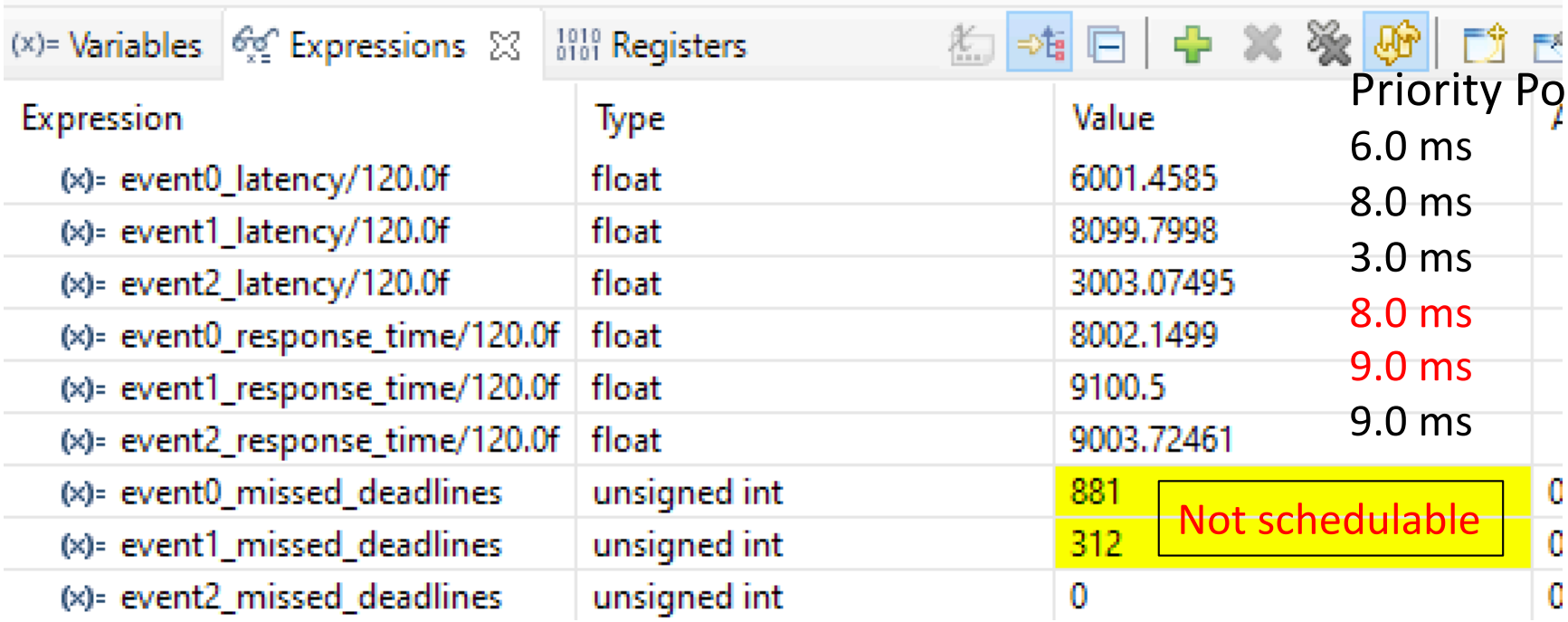
Disabling Interrupts / Shared Data

- If ISRs are disabled, the latency will be the longest time they are disabled for.
  - In the ece3849_int_latency program, this was the entire execution time of each ISR.
- We may globally disable interrupts to protect access to shared data and resources.
  - event1 is filling up a buffer.
  - event0 is reading out of the buffer.
  - If event0 preempts event1, then all the data will not get written and the data read out will be a combination of old and new.



ece3849_int_latency
Example 3: event2 = 6.0,
Interrupt Driven but disabled in ISR

- If Interrupts are disabled during the entire ISR, no preemption can happen and performance reverts back to priority polling.



Disabling Interrupts for short periods

- Maximum latency is affected by disabling interrupts.
  - It is important to minimize the time interrupts are disabled.
  - Only disable them during critical time to avoid shared data problems. Example…
    - Disable interrupts only when data being written or read to the buffer.
    - Enable at all other times.
- ece3849 int latency simulation (what do you think will happen?)

```
46 // build options
47 #define ENABLE_ISR 1
48 //#define DISABLE_INTERRUPTS_IN_ISR // if defined, interrupts are disabled in the body of each ISR
49 #define DISABLE_INTERRUPTS_SHORT 1
50
```

```
164 #ifdef DISABLE_INTERRUPTS_SHORT
165     //loop for testing the effect of disabling interrupts
166     while (true) {
167         IntMasterDisable();
168         delay_us(100);
169         //count_unloaded++;
170         //count_loaded--;
171         IntMasterEnable();
172
173         IntMasterDisable();
174         delay_us(200);
175         IntMasterEnable();
176     }
177 #endif
```

Turn interrupts off.
100 usec delay to emulate disabling during data writing.
Turn interrupts on.

Turn interrupts off.
200 usec delay to emulate disabling during data reading.
Turn interrupts on.

DISABLE_INTERRUPTS_SHORT Results

Expression	Type	Value	Interrupt Enable results
(x)= event0_latency/120.0f	float	200.333328	0 ms
(x)= event1_latency/120.0f	float	2149.29175	2.0 ms
(x)= event2_latency/120.0f	float	3201.7417	3.0 ms
(x)= event0_response_time/120.0f	float	2200.91675	2.0 ms
(x)= event1_response_time/120.0f	float	3193.28345	3.0 ms
(x)= event2_response_time/120.0f	float	14170.167	12.0 ms (marginal)
(x)= event0_missed_deadlines	unsigned int	0	
(x)= event1_missed_deadlines	unsigned int	0	
(x)= event2_missed_deadlines	unsigned int	75	

Not schedulable
14.2 > 12 msec

- Effect of disabling interrupts for short times.
  - Latency was increased by the maximum disabled time
    - Max(100 usec for write, 200 usec for read) = 200 usec.
  - All Latencies increased by 200 usec.
- Event2 now solidly not schedulable.
  - The latency caused an additional event0 event to preempt event2 adding 2 msec more to its execution time.
  - Designs without margin can be pushed from working to failing miserably by adding even tiny delays.