Profiling

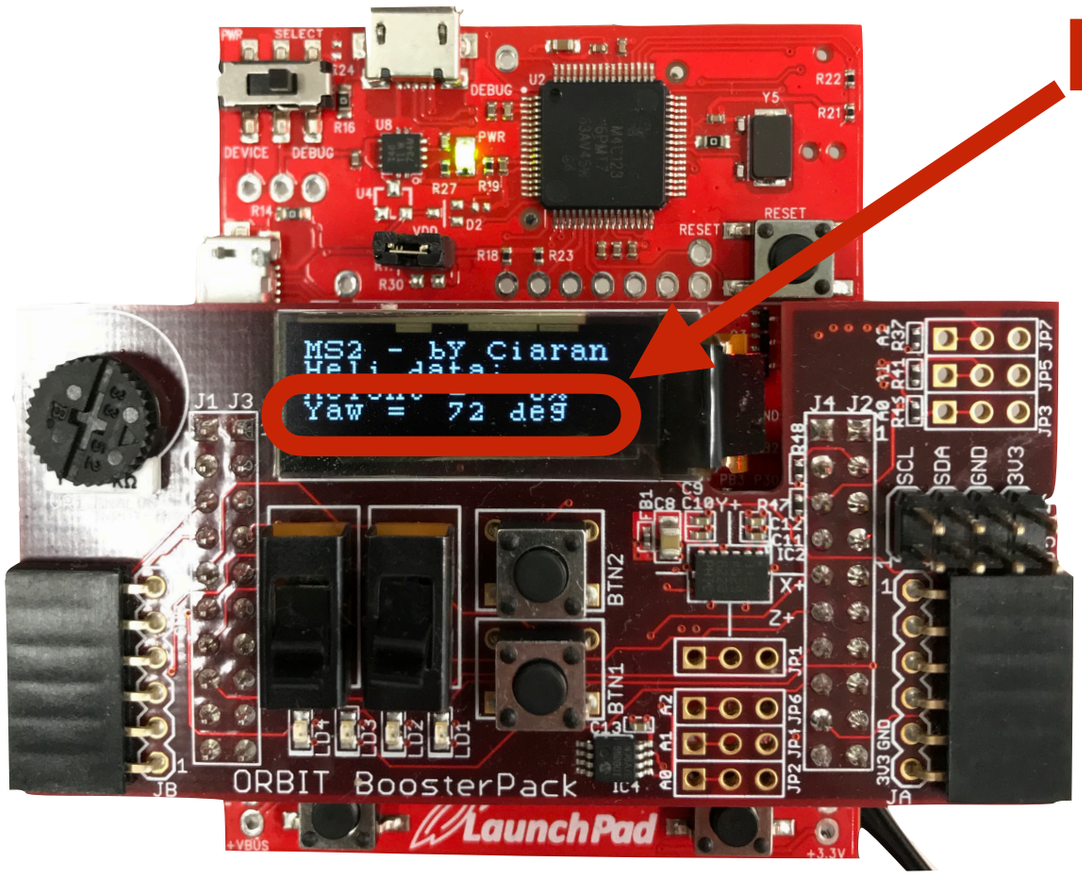
ENCE361: Design & Architecture: Lecture Block 1

Roadmap

		ENCE361 Lecture, Tutorial & Lab Schedule – 2020				v. 20.3		Updated 09/05/2020
		Lec 1		Lec 2		Lec 3		Lab
Wk	**Starting**	**Mon 1p**		**Wed 2p**		**Fri 2p**		**Mon 11a, Tue 9a, Tue 11a, Wed 11a**
10	**11 May**	23	Profiling	26	Load Analysis	27	MCU Interfacing	
11	**18 May**	28	Memory structures	29	MCU memory types	30	Arm Arithm./ Logic ccts	**Project demos in usual lab slot**
12	**25 May**	31	The ARM ISA	32	ARM Assembly language	33	Revision & Exam Prep	**Project report & code due Fri 29 May**

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing



Sometimes ISRs took ‘too long’ to complete, causing pin changes to be missed, giving an inaccurate yaw angle. How long is too long? Can we measure this?

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing code in CCS

- CCS has a Profile Clock that can automatically time code
- Unfortunately, the profile clock hardware is not available in the Cortex M core used by our TiVA board
- Instead, we will time code by reading special registers

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing code in CCS

- Use a Count Event to measure the number of cycles between breakpoints.

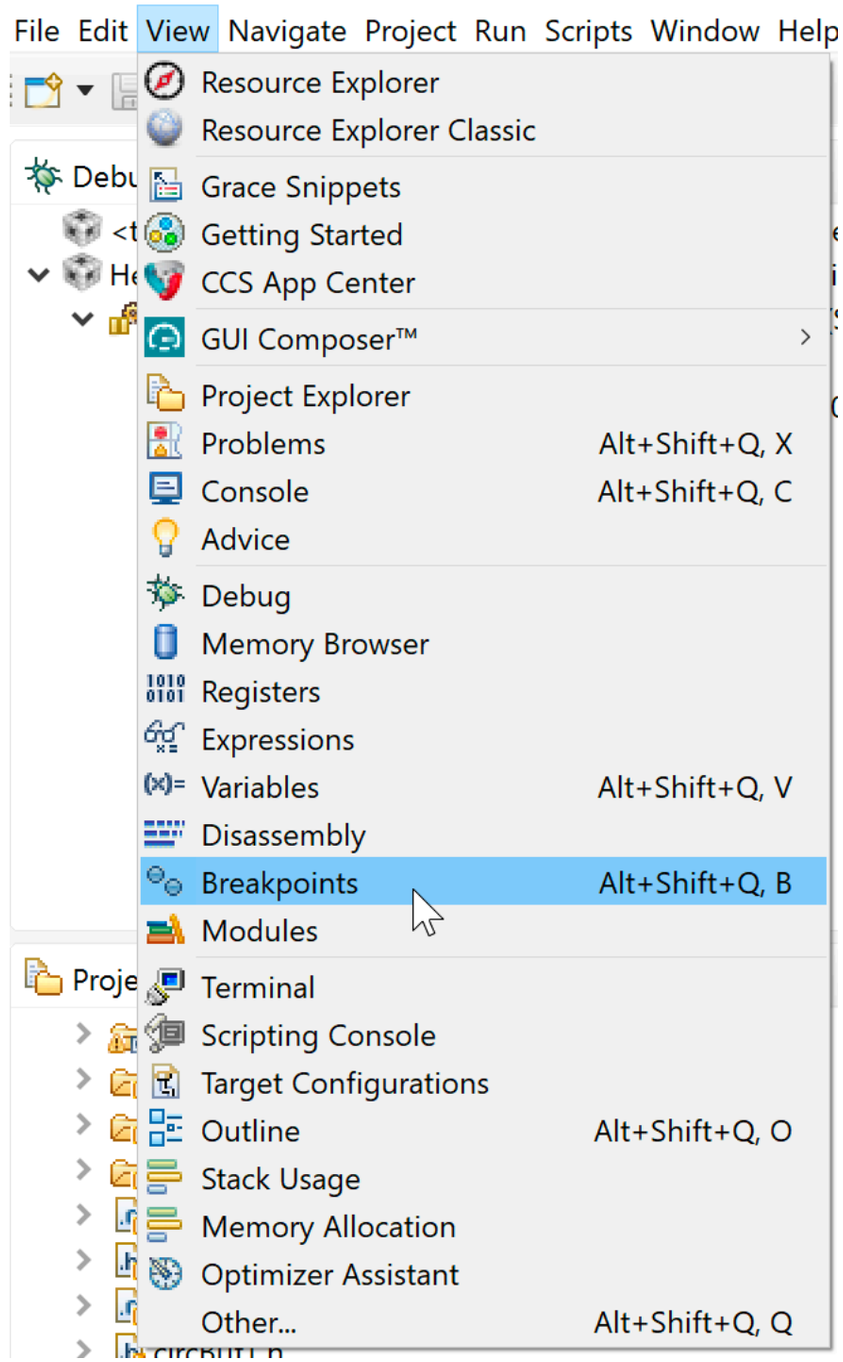
Expressions | Breakpoints

Identity	Name	Condition	Count
Clock Cycles [H/W]	Count Event		226334
MS2.c, line 373 (CL21) [H/W	Breakpoint		0 (0)
MS2.c, line 374 (CL21 + 0x10)	Breakpoint		0 (0)

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing code in CCS

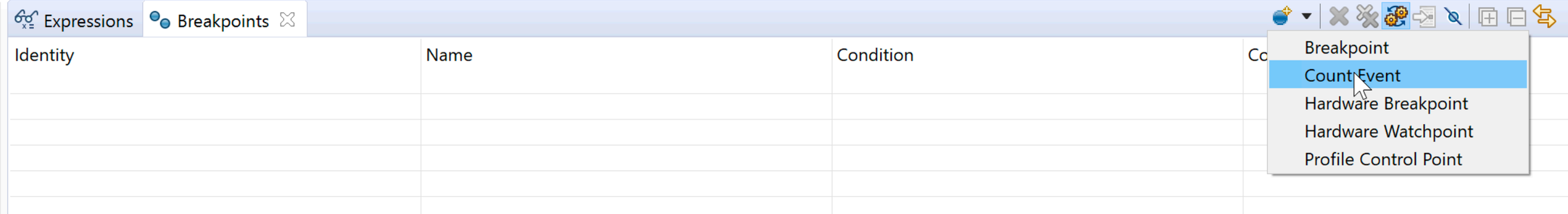
- Put CCS in Debug mode.
- Open breakpoints view:
  - View > Breakpoints (ALT+SHIFT+Q, B)



Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing code in CCS

- In the Breakpoints pane, add a Count Event:

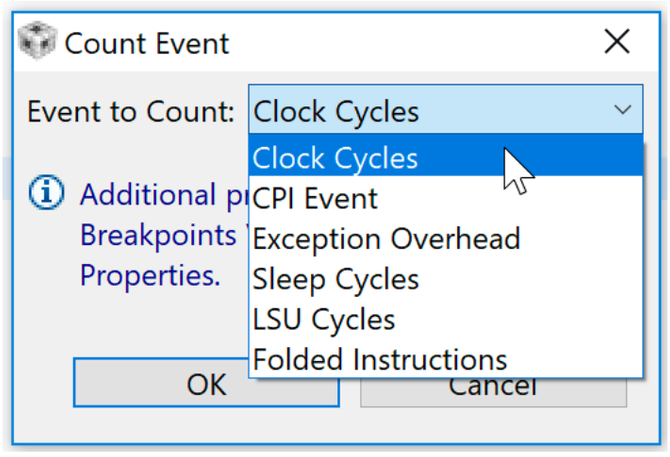


- NB: Delete all other breakpoints before adding the count event.

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing code in CCS

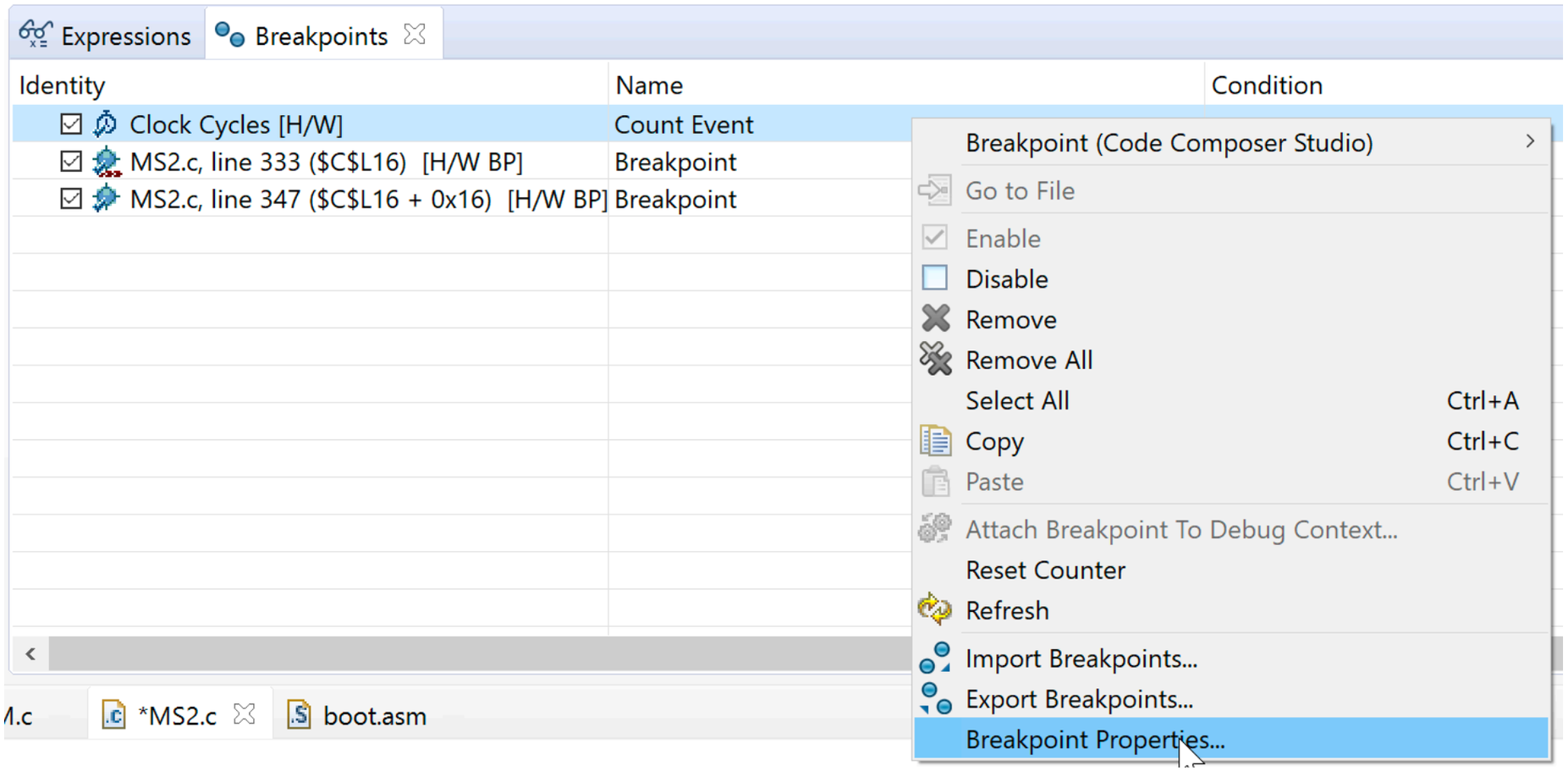
- Select ‘clock cycles’ as the event you wish to count:



Ciaran Moore
ciaran.moore@canterbury.ac.nz

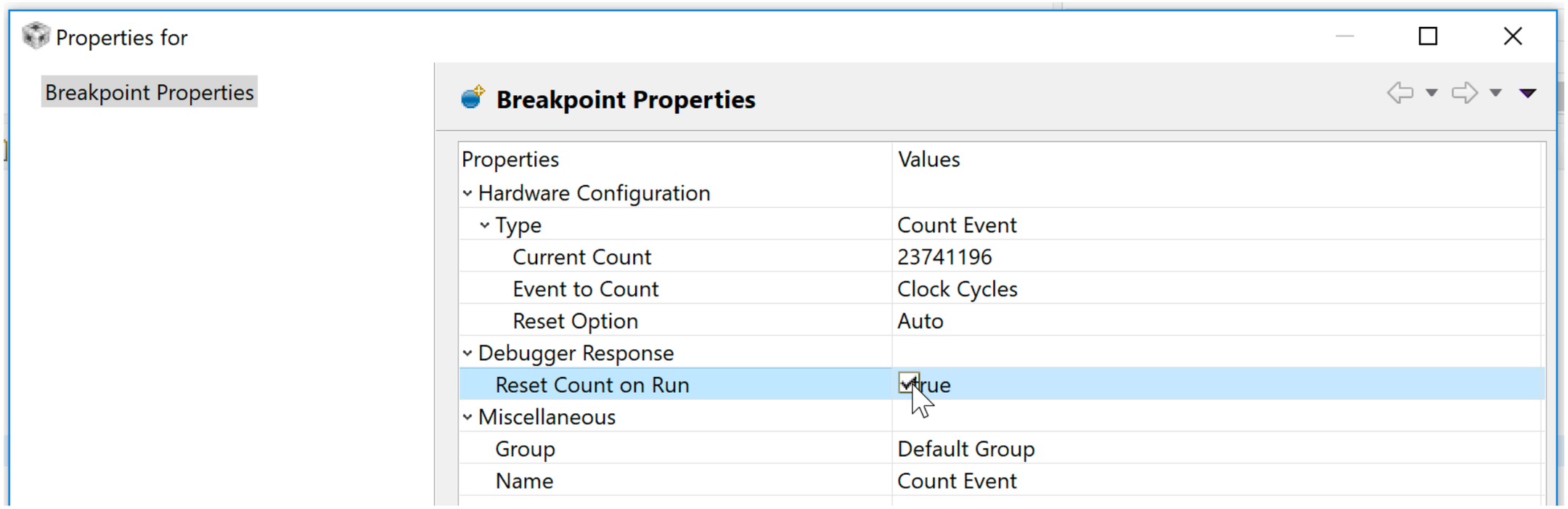
Timing code in CCS

- Right click on the Count Event breakpoint and edit its properties to Reset Count on Run = TRUE



Timing code in CCS

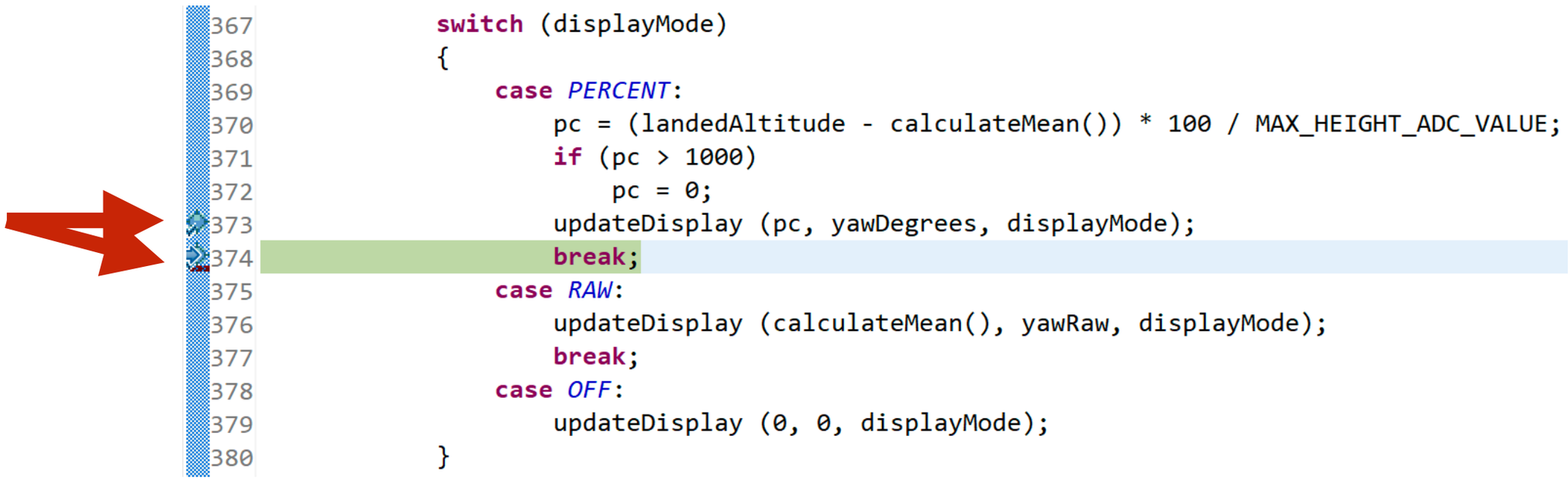
- Right click on the Count Event breakpoint and edit its properties to Reset Count on Run = TRUE



Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing code in CCS

- Add two breakpoints immediately before and immediately after the code you want to time.

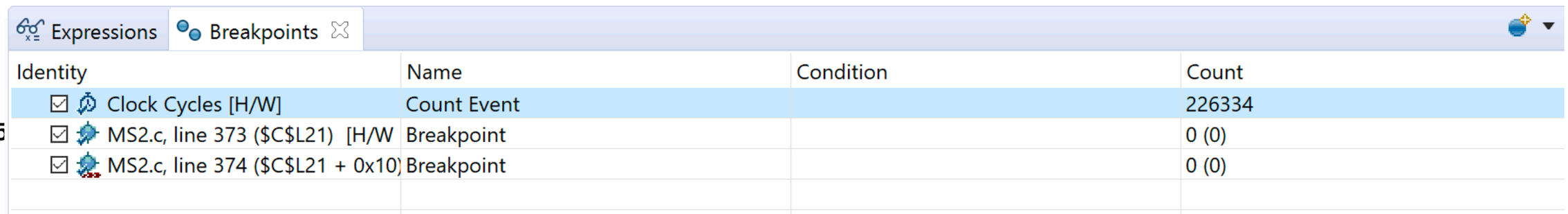


```
367             switch (displayMode)
368             {
369                 case PERCENT:
370                     pc = (landedAltitude - calculateMean()) * 100 / MAX_HEIGHT_ADC_VALUE;
371                     if (pc > 1000)
372                         pc = 0;
373                     updateDisplay (pc, yawDegrees, displayMode);
374                     break;
375                 case RAW:
376                     updateDisplay (calculateMean(), yawRaw, displayMode);
377                     break;
378                 case OFF:
379                     updateDisplay (0, 0, displayMode);
380             }
```


Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing code in CCS

- Add two breakpoints immediately before and immediately after the code you want to time.
- Run to the first breakpoint. The event counter will count the number of cycles taken up to this point.
- Run to the second breakpoint. The event counter will show the number of cycles taken since the last breakpoint.



X

Timing code in CCS

- We can ensure that we are timing only the code block we are interested in by adding

```
IntMasterDisable();
```

… and …

```
IntMasterEnable();
```

above and below our code block.

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing data for HeliRig example

- How long does code take to execute?

Function or ISR	Clock Cyles			
SysTick	49			
ADCIntHandler	83			
YawIntHandler	48			
updateButtons	221 or 232			
updateDisplay	226 k			
Main Loop (w/out updateDisplay and SysCtlDelay)	305			

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing data

- How fast *should* code run?
  - Depends on what code is doing.
- Memory access time depends on type of memory (flash, RAM, external).
- External peripherals are generally slower than integrated components.
- In general, user-based I/O is slooooow (buttons, displays, etc.)

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing data for HeliRig example

- How long do we have for code to execute?

Function or ISR	Clock Cyles Required	Desired Period	Desired Frequency = 1/Period	Cycles in Desired Period = f_{clk} x Period
SysTick	49		100 Hz	
ADCIntHandler	83		100 Hz	
YawIntHandler	48		900 Hz	
updateButtons	221 or 232		50 Hz	
updateDisplay	226 k		4 Hz	
Main Loop (w/out updateDisplay and SysCtlDelay)	305		50 Hz	

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing data

- Period = 1 / frequency
- Clock cycles available = System clock rate / execution frequency
- Yaw interrupt frequency = interrupts/rev. × max. expected rate of rev. = 112 slots/rev. × 2 interrupts / slot / sensor × 2 sensors × 2(?) rev./second

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing data for HeliRig example

- How long do we have for code to execute?

Function or ISR	Clock Cyles Required	Desired Period	Desired Frequency = 1/Period	Cycles in Desired Period = f_{clk} x Period
SysTick	49	10 ms	100 Hz	200 k
ADCIntHandler	83	10 ms	100 Hz	200 k
YawIntHandler	48	1.1 ms	900 Hz	22 k
updateButtons	221 or 232	20 ms	50 Hz	400 k
updateDisplay	226 k	250 ms	4 Hz	5 M
Main Loop (w/out updateDisplay and SysCtlDelay)	305	20 ms	50 Hz	400 k

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing data for HeliRig example

- What happens if we move code around?
  - Can we put **updateButtons** in **SysTickIntHandler**?

Function or ISR	Clock Cyles Required	Desired Period	Desired Frequency = 1/Period	Cycles in Desired Period = f_{clk} x Period
SysTick	49	10 ms	100 Hz	200 k
ADCIntHandler	83	10 ms	100 Hz	200 k
YawIntHandler	48	1.1 ms	900 Hz	22 k
updateButtons	221 or 232	20 ms	50 Hz	400 k
updateDisplay	226 k	250 ms	4 Hz	5 M
Main Loop (w/out updateDisplay and SysCtlDelay)	305	20 ms	50 Hz	400 k

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing data for HeliRig example

- What happens if we move code around?
  - Can we **updateDisplay** at 120 fps?

Function or ISR	Clock Cyles Required	Desired Period	Desired Frequency = 1/Period	Cycles in Desired Period = f_{clk} x Period
SysTick	49	10 ms	100 Hz	200 k
ADCIntHandler	83	10 ms	100 Hz	200 k
YawIntHandler	48	1.1 ms	900 Hz	22 k
updateButtons	221 or 232	20 ms	50 Hz	400 k
updateDisplay	226 k	250 ms	4 Hz	5 M
Main Loop (w/out updateDisplay and SysCtlDelay)	305	20 ms	50 Hz	400 k

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Profiling

- The process of measuring the clock cycles (or time) taken by each function can be automated by a Profiler.
- Profilers also typically report how often a function is called.
- Profiling is typically implemented by the IDE, but needs hardware support for embedded systems

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Profiling

- An example from Matlab:

Profile Summary

Generated 06-May-2019 11:07:30 using performance time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
TMM	1	0.214 s	0.033 s	
addInterfaces	1	0.140 s	0.140 s	
plotTF	1	0.013 s	0.001 s	
publishPlot	1	0.010 s	0.002 s	
findall	2	0.008 s	0.007 s	
axis	1	0.007 s	0.001 s	
xlabel	1	0.005 s	0.004 s	
axis>LocCheckCompatibleLimits	1	0.005 s	0.005 s	
hold	1	0.005 s	0.003 s	

Ciaran Moore
ciaran.moore@canterbury.ac.nz

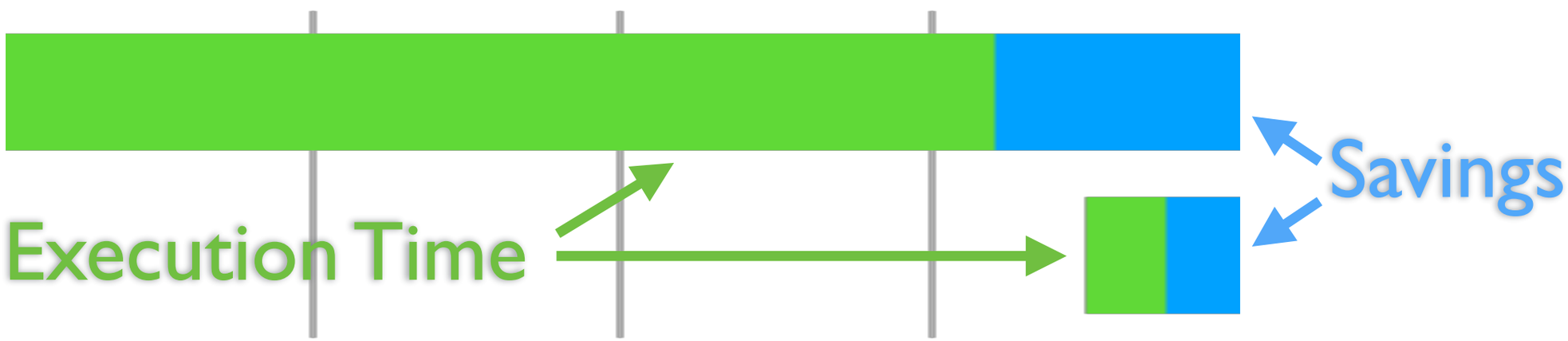
Making Code Faster

- Common attributes of Good Code:
  - Clear Structure
  - Low Coupling
  - High Cohesion
- Making code faster often means moving away from these ideal attributes

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Making Code Faster

- Start with code that is used most often.
  - A 20% improvement to code that takes 80% of compute time gives larger gains than a 50% improvement to code that takes 10% of compute time.



Ciaran Moore
ciaran.moore@canterbury.ac.nz

Making Code Faster

- Example: convert.c

```
//*********************************************************************
// convert.c - Simple interrupt driven program which samples with ADC0
// *** Version 5 - calls API functions within ISRs and allows timing
// measurements with short cycle background and no display. ***
//
// Author: P.J. Bones UCECE
// Last modified:18.4.2012
//*********************************************************************

//*********************************************************************
// convert.c - Simple interrupt driven program which samples with ADC0
// *** Version 6 - without function calls within ISR and
// with short cycle background and no display. ***
//
// Author: P.J. Bones UCECE
// Last modified:18.4.2012
//*********************************************************************
```


Ciaran Moore
ciaran.moore@canterbury.ac.nz

Making Code Faster

- Version 5:

```
//*****************************************************************************
// The interrupt handler for the for SysTick interrupt. ** Version 5 **
//*****************************************************************************
void SysTickIntHandler(void)
{
    // Initiate a conversion
    ADCProcessorTrigger(ADC0_BASE, 3);
    g_ulSampCnt++;
}
```


Ciaran Moore
ciaran.moore@canterbury.ac.nz

Making Code Faster

- Version 6:

```
//*****************************************************************************
// The interrupt handler for the SysTick interrupt. ** Version 6 **
//*****************************************************************************
void SysTickIntHandler(void)
{
    // Initiate a conversion
    // Alternative to call ADCProcessorTrigger(ADC0_BASE, 3);
    // Write to ADC0 PSSI register for Sequencer 3 (write-only register)
    ADC0_PSSI_R = ADC_PSSI_SS3;
    g_ulSampCnt++;
}
```


Ciaran Moore
ciaran.moore@canterbury.ac.nz

Making Code Faster

Extract from **convertV5.c** (with function calls):

```
*****************************************************
   930    ;* FUNCTION NAME: ADCIntHandler
   931    ;*
   932    ;*   Regs Modified     : A1,A2,A3,A4,V1,V2,V9,SP,LR,SR
   933    ;*   Regs Used         : A1,A2,A3,A4,V1,V2,V9,SP,LR,SR
   934    ;*   Local Frame Size  : 0 Args + 4 Auto + 12 Save = 16 byte
   935    ;*****************************************************
                              .
 946 00000000 B538    PUSH     {A4, V1, V2, LR} ; [ORIG 16-BIT INS]
                              .
 966 00000004 2180    MOVS     A2, #128            ; |66|
 967 00000006 460A    MOV      A3, A2 ; [ORIG 16-BIT INS]
 968 00000008 4628    MOV      A1, V2 ; [ORIG 16-BIT INS]
 973 0000000a FFFEF7FF!   BL       GPIOPinWrite          ; |66|
                              .
1022 0000003e 2200    MOVS     A3, #0              ; |81|
1023 00000040 2180    MOVS     A2, #128            ; |81|
1024 00000042 4628    MOV      A1, V2 ; [ORIG 16-BIT INS]
1029 00000044 FFFEF7FF!   BL       GPIOPinWrite          ; |81|
                              .
1037 00000048 BD38    POP      {A4, V1, V2, PC} ;
```


Ciaran Moore
ciaran.moore@canterbury.ac.nz

Making Code Faster

Extract from **convertV6.c** (no function calls):

```
*****************************************************************
   873    ;* FUNCTION NAME: ADCIntHandler
   874    ;*
   875    ;*   Regs Modified     : A1,A2,A3,A4,V1,V2,SP,SR
   876    ;*   Regs Used         : A1,A2,A3,A4,V1,V2,SP,LR,SR
   877    ;*   Local Frame Size  : 0 Args + 0 Auto + 12 Save = 12 byte
   878    ;*****************************************************************
```

no PUSH operation

```
912 00000002 4B0C    LDR     A4, $C$CON13       ; |77| ;
913 00000004 2080    MOVS    A1, #128           ; |77| ;
914 00000006 6018    STR     A1, [A4, #0]       ; |77| ;
```

```
943 0000002c 2108    MOVS    A2, #8             ; |95| ;
944 0000002e 6011    STR     A2, [A3, #0]       ; |95| ;
946 00000030 6018    STR     A1, [A4, #0]       ; |98| ;
```

no POP operation

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Making Code Faster

- Writing directly to registers removes the need to PUSH & POP the state of the program onto and off the stack
- There are also extra instructions associated with the function calls in V5 not present in V6

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Making Code Faster

Version	Description	ADC core	ST complete	ADC complete	ST + ADC complete	epi + pro for ADC
V5	With API calls	3.8	2.5	5.8	8.4	2.0
V6	Without API calls	1.5	1.7	3.3	5.0	1.8

All times are in µsec. The processor clock rate is 20 MHz.

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Making Code Faster

- Write good code first
  - THEN make it fast (if you need to)

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Homework – Profiling

- Considering the timing data for the HeliRig example, what's the lowest system clock frequency that would allow our program to execute without difficulty?
- The difference between the two programs convertV5.c and convertV6.c is that the former uses function calls within the ISRs, while the latter uses direct register accesses. Why might this difference matter?
- When optimising code for speed, should you start with functions that take the largest number of clock cycles to complete, or should you first tackle functions that are called the most frequently, or is there some other approach that you should use to set optimisation priorities?

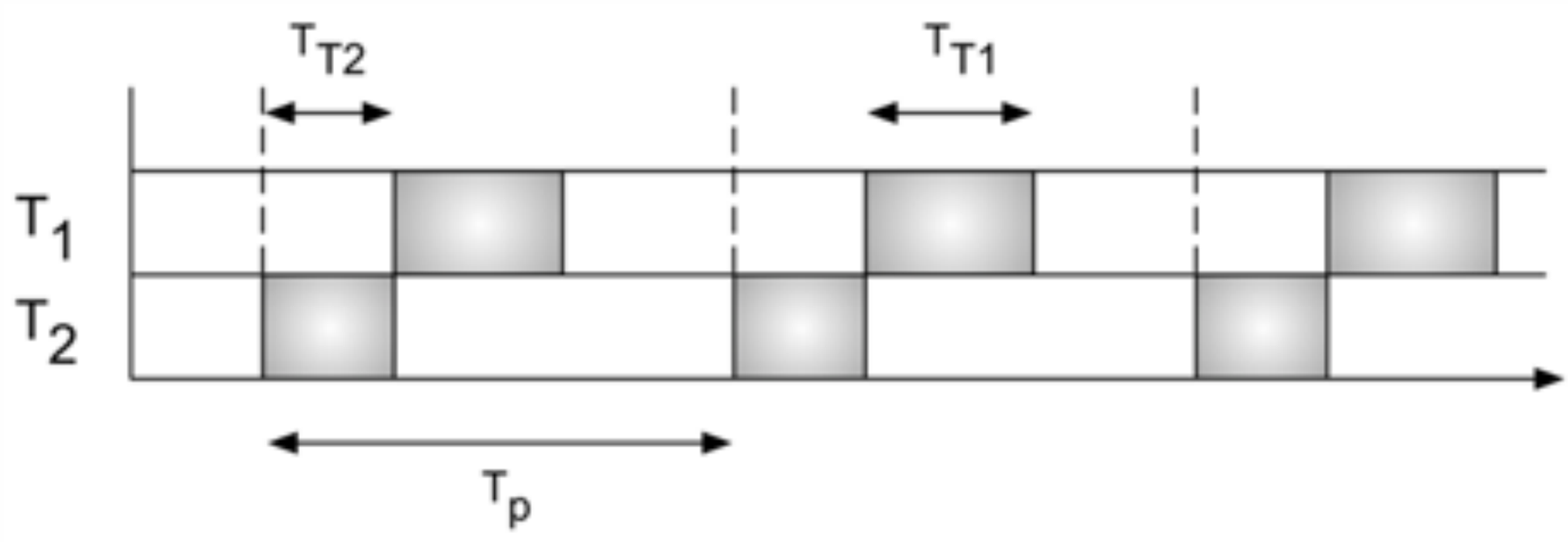
Ciaran Moore
ciaran.moore@canterbury.ac.nz

CPU Load Utilisation

ENCE361: Design & Architecture: Lecture Block 1

CPU Load

- We can extend the idea of quantifying the time taken to complete a function and the time *available* to complete a function to give a new metric: CPU Load, *L*.



CPU Load

- The load for the k^{th} task is

$$L_k = \frac{T_k}{P_k} = T_k \times F_k$$

- where T_k = task's (max) execution time
  P_k = task's (min) execution period
  F_k = task's (max) execution frequency

Ciaran Moore
ciaran.moore@canterbury.ac.nz

CPU Load

- The total load on the CPU is given by

$$L_{TOT} = \sum_k \frac{T_k}{P_k}$$

- If $L_{TOT} > 1$ then the design is not feasible!

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Timing data for HeliRig example

Function or ISR	Clock Cyles Required, T_k	Desired Period	Desired Frequency	Cycles Available, P_k
SysTick	49	10 ms	100 Hz	200 k
ADCIntHandler	83	10 ms	100 Hz	200 k
YawIntHandler	48	1.1 ms	900 Hz	22 k
updateButtons	221 or 232	20 ms	50 Hz	400 k
updateDisplay	226 k	250 ms	4 Hz	5 M
Main Loop (w/out updateDisplay and SysCtlDelay)	305	20 ms	50 Hz	400 k

$$L_{TOT} = \sum_k \frac{T_k}{P_k}$$

Ciaran Moore
ciaran.moore@canterbury.ac.nz

ABS example

Function	Req.	Exec
ISR - SysTick	10^5 / s	2.5 µs
ISR - Wheels	5280 / s	20 µs
speedFL	264 / s	30 µs
speedFR	264 / s	30 µs
speedRL	264 / s	30 µs
speedRR	264 / s	30 µs
monitorWheels	200 / s	250 µs
monitorDriver	50 / s	1.5 ms
activateABS	Infreq.	5 ms
displayABS	20 / s	7.5 ms

$$L_{TOT} = \Sigma_k \left(T_k \times F_k \right)$$

Ciaran Moore
ciaran.moore@canterbury.ac.nz

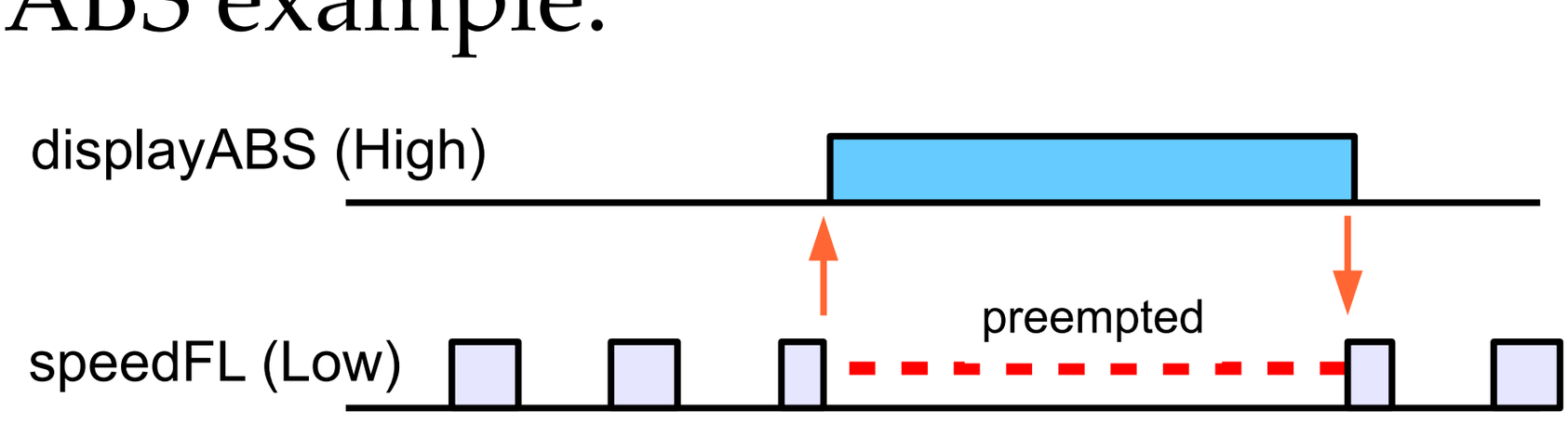
Preemption and Task Priority

- Recap:
  Assigning different priorities to tasks allows some to be interrupted when processing time is limited.

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Preemption and Task Priority

- Consider a UI task (displayABS) and a core IO task (speedFL) from the earlier ABS example:

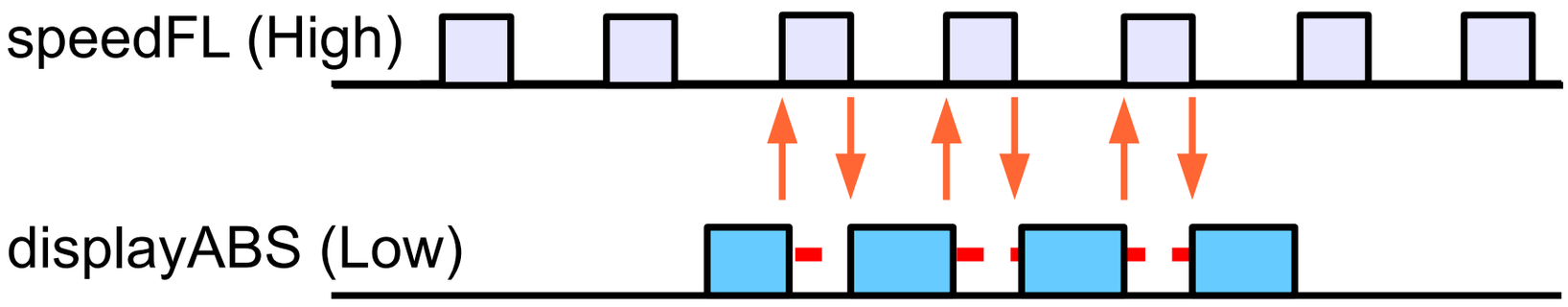


  - displayABS: 50ms deadline, 7.5ms exec. time
  - speedFL: 3.8ms deadline, 30 μs exec. time

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Preemption and Task Priority

- By assigning a higher priority to the speedFL task, we can preemptively interrupt the displayABS task:



  - displayABS: 50ms deadline, 7.5ms exec. time
  - speedFL: 3.8ms deadline, 30 μs exec. time

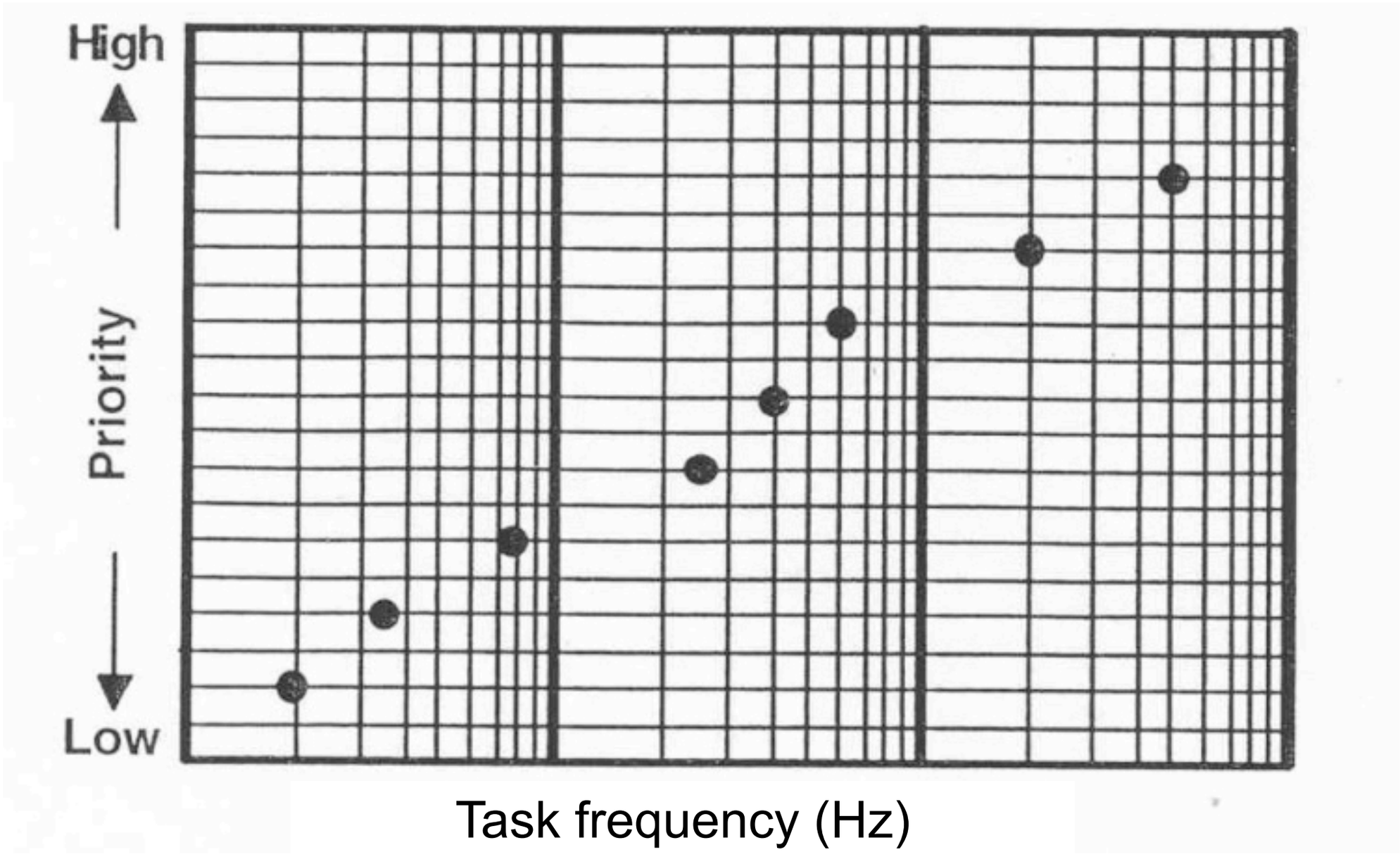
Ciaran Moore
ciaran.moore@canterbury.ac.nz

Rate Monotonic Priorities

- When priorities can be assigned, both in the foreground (interrupt priorities) and in the background (task priorities) the rate monotonic strategy sets the highest priority for the most frequently occurring interrupt or task.
- The next most frequently occurring is assigned the 2nd highest priority, and so on to the least frequently occurring, lowest priority task.

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Rate Monotonic Priorities



Ciaran Moore
ciaran.moore@canterbury.ac.nz

Rate Monotonic Priorities

- Our ABS ISRs and functions are already arranged in a rate monotonic form:

Function	Req.	Exec
ISR - SysTick	10^5 / s	2.5 µs
ISR - Wheels	5280 / s	20 µs
speedFL	264 / s	30 µs
speedFR	264 / s	30 µs
speedRL	264 / s	30 µs
speedRR	264 / s	30 µs
monitorWheels	200 / s	250 µs
monitorDriver	50 / s	1.5 ms
activateABS	Infreq.	5 ms
displayABS	20 / s	7.5 ms

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Schedulability Test

- A system using RMS is considered schedulable (realisable) if the total load, L_{TOT}, does not exceed ***ln 2 = 0.69***:

$$L_{TOT} = \frac{T_1}{P_1} + \frac{T_2}{P_2} + \ldots + \frac{T_n}{P_n} \leq 69\%$$

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Schedulability Test

- Many caveats apply to the Schedulability Test:
  - Applies only to preemptive task sets
  - Applies only to rate monotonic task sets
  - Applies only to independent, non-interacting tasks
  - Assumes all task deadlines are equal to the task periods

Ciaran Moore
ciaran.moore@canterbury.ac.nz

What About Non-Preemptive Task Sets?

- Response-time analysis becomes more useful: Ask "will task i with response time R_i meet its deadlines, D?"

$$R_i \leq D_i$$

- For round-robin scheduling, R_i = sum of all tasks
- For time-triggered scheduling, $R_i \approx 0$ (for a well-designed schedule)

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Other Analysis Options

- Energy:
  - Very roughly, CPU load is determined by current consumption: $L = I_{ave} \div I_{peak}$
    - Battery life = A · hr capacity $\div$ I_{ave}
- Bandwidth:
  - Quantity of data = samples/sec × bits/sample
  - Req'd bandwidth = Quantity × Tx task time

Ciaran Moore
ciaran.moore@canterbury.ac.nz

Homework – CPU Load

- For the ABS example on slide 8, how much could the execution time, T_k for each task k be increased (assuming only the code for that task was changed) before a CPU load of 0.69 is reached?
- For the same example, how much could the required frequency ($=1/P_k$) for each task k be increased (assuming only the code for that task was changed) before a CPU load of 0.69 is reached?
- Under what circumstances might a strictly rate monotonic priority assignment be inappropriate?

Ciaran Moore
ciaran.moore@canterbury.ac.nz