COS10004 Computer Systems

Lecture 8.3 LED Flash (part 3) – a better busy wait (the Timer Register)

CRICOS provider 00111D

Dr Chris McCarthy

INSERT DELAY WITH BUSY WAIT TIMER

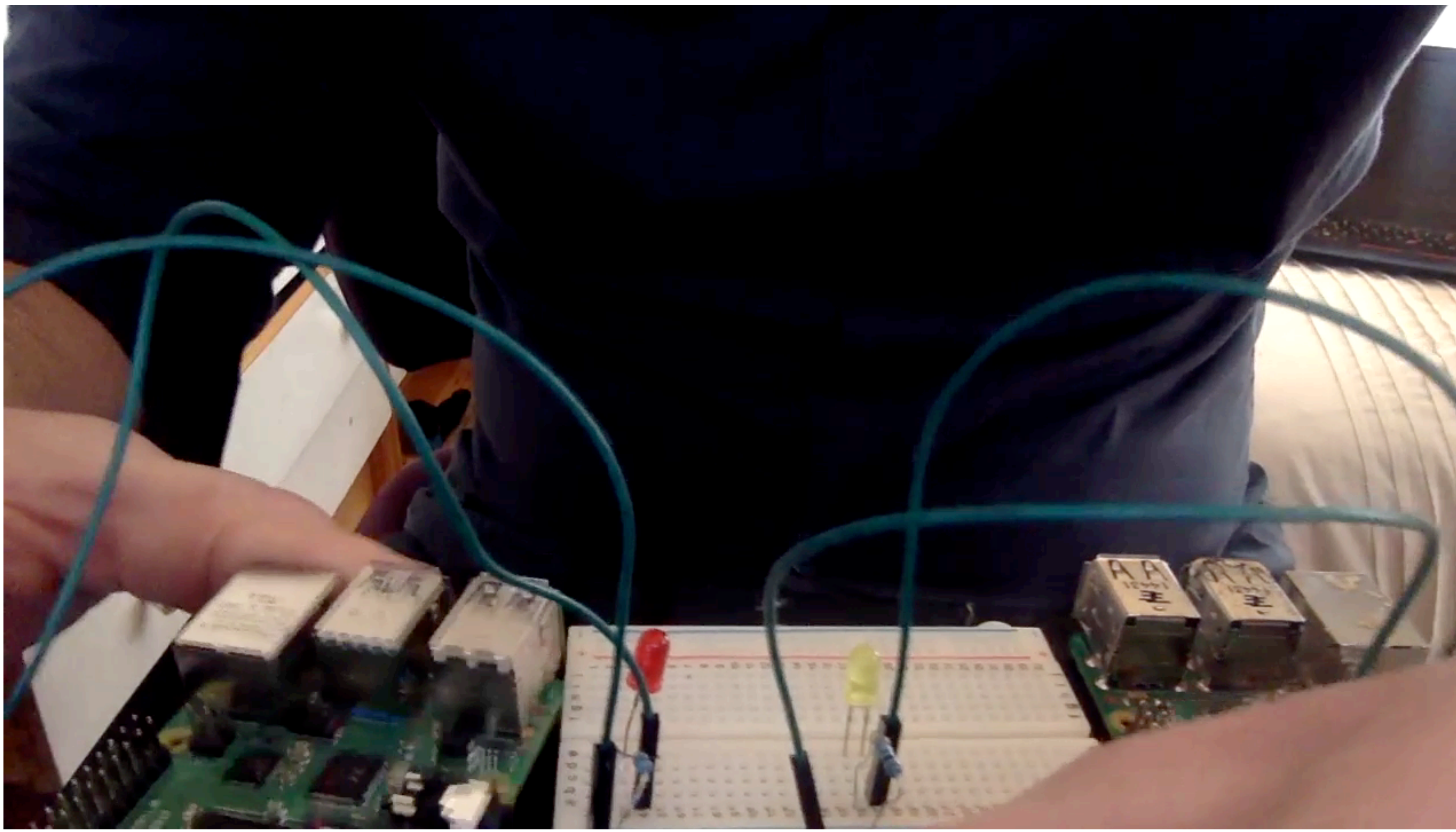
- Pseudocode:
  - Program GPIO18 LED for output
  - Loop1:
    - Turn LED on (pull GPIO18 high)
    - Busy wait
    - Turn LED off (pull GPIO18 low)
    - Busy wait
  - branch to loop1

A DUMB TIMER

- Variables:
  - r0 = GPIO base address
  - r1 = working memory (for setting bits, registers)
  - use r2 for timing
- mov r2,$3F0000
  loop1:
   sub r2,#1
   cmp r2,#0
  bne loop1

Does a 'busy wait' - uses 100% of CPU

OK2 RUNNING ON PI 2B AND PI 4B



A BETTER TIMER

- RPi has a dedicated timer register:
  - independent of clock speed.
  - housed inside the same chip as the ARM CPU, the GPIO, GPU, RAM and most other things.
  - This chip is called the SoC (System on a Chip).
- The Timer registers start at BASE address + 0x3000
  - Timer counts 1 microsecond intervals (10^6 per second)

A BETTER TIMER

- The RPi timer registers:

Byte offset (from BASE)	Size / Bytes	Name	Description	Read or Write
0x3000	4	Control / Status	Register used to control and clear timer channel comparator matches.	RW
0x3004	8	Counter	A counter that increments at 1MHz.	R
0x300C	4	Compare 0	0th Comparison register.	RW
0x3010	4	Compare 1	1st Comparison register.	RW
0x3014	4	Compare 2	2nd Comparison register.	RW
0x3018	4	Compare 3	3rd Comparison register.	RW

PSEUDOCODE

- pseudocode:

  store base address of timer

  store delay

  mov start_time (=*now*)

  loop:

  read *now*

  remaining_time = (*now* – starttime)

  compare remaining_time, delay

  loop if remaining_time <= delay

Still does a 'busy wait' – but measures real time

Reading the time (timestamp)

- Variables:
- r3 - base address of timer
- r4 - desired delay
- r6, r7 - where the current time will be stored
- r5 - start time
- r8 - elapsed time (r6-r5)

Timer returns 64-bit time, but we only need the lower register (r6) - must be an even register

WITH REGISTERS

store base address of timer (r3)

store delay (r4)

mov start_time (r5)(=current_time (r6))

timerloop:

    read current_time (r6)

    remaining_time (r8)= current_time (r6) – start_time (r5)

    compare remaining_time (r8), delay (r4)

    loop if LE (remaining_time <= delay)

RECALL LOAD REGISTER ?

delimiter

```
ldr r0,[r1]
```

register 0

r1 must contain a pointer to a value in memory

- load register 0 with the value pointed to by r1

A BETTER TIMER

- BASE = $FE000000
  TIMER_OFFSET = $3000
  mov r3,BASE
  orr r3,TIMER_OFFSET ;base address of ti

- ldrd r6,r7,[r3,#4] ;copy timestamp [4 bytes afte
  byte), r7(high byte).
  - d suffix means double-word (64 bit)
  - http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0338g/Chddeedh.html

- mov r4,$80000 ;0.524sec

d suffix means double-word (64 bits)

WITH CODE (4B)

```
BASE = $FE000000
TIMER_OFFSET = $3000
mov r3,BASE
orr r3,TIMER_OFFSET ;sto  as
mov r4,$80000  ;store delay (r4)
ldrd r6,r7,[r3,#4]
mov r5,r6 ;mov starttime (r5)(=cu
timerloop:
    ldrd r6,r7,[r3,#4] ;read currenttime (r6)
    sub r8,r6,r5  ;remainingtime (8)= currenttime (r6) - starttime (r5)
    cmp  r8,r4 ;compare remainingtime (r8), delay (r4)
    bls timerloop ;loop if LE (reaminingtime <= delay)
```

- r3=BASE + TIMER_OFFSET + 4 (0x3F003004)
- [r3,#4] means value at ((address in r3) + 4)

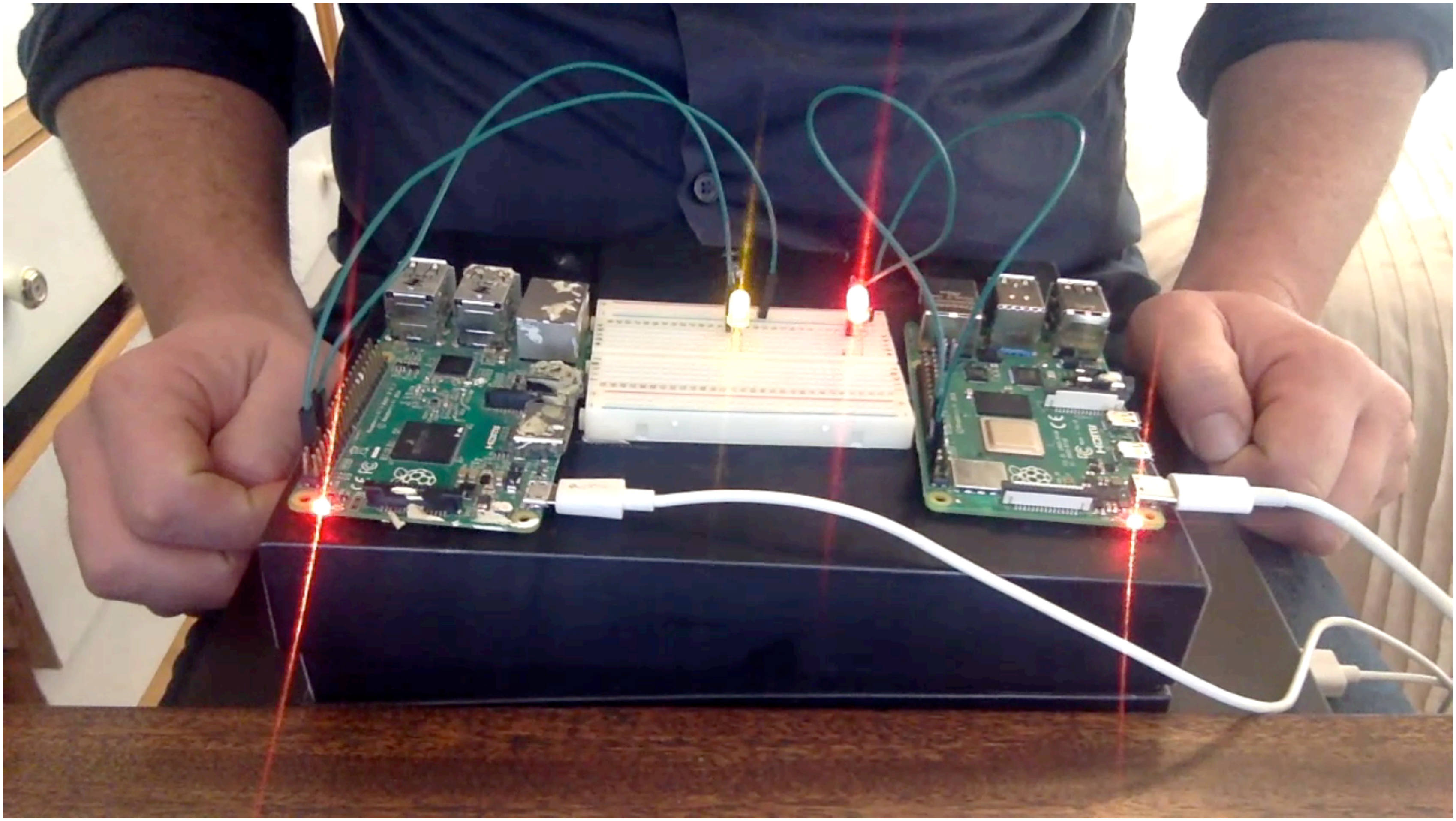
can't re-use loop label - each must be unique: loop1, loop2, loop3...

ALL THE CODE (B)

```
BASE = $20000000 ;B B+. Use $3F000000 for 2
GPIO_OFFSET = $200000
mov r0,BASE
orr r0,GPIO_OFFSET ;Base address of GPIO
mov r1,#1
lsl r1,#18 ;B
str r1,[r0,#4] ;enable output
mov r1,#1
lsl r1,#16
loop$:
 str r1,[r0,#40] ;Turn on LED
 ;new timer
TIMER_OFFSET = $3000
mov r3,BASE
orr r3,TIMER_OFFSET ;store base address of timer (r3)
 mov r4,524288;store delay (r4)
 ldrd r6,r7,[r3,#4]
 mov r5,r6 ;store starttime (r5)(=currenttime (r6))
 loopt1:
  ldrd r6,r7,[r3,#4] ;read currenttime (r6)
  sub r8,r6,r5  ;remainingtime (8)= currenttime (r6) - starttime (r5)
  cmp  r8,r4  ;compare remainingtime (r8), delay (r4)
  bls loopt1 ;loop if LE (reaminingtime <= delay)
 str r1,[r0,#28]  ;turn off LED
 ;re-use timer
 ldrd r6,r7,[r3,#4]
 mov r5,r6 ;store starttime (r5)(=currenttime (r6))
 loopt2:
  ldrd r6,r7,[r3,#4] ;read currenttime (r6)
  sub r8,r6,r5  ;remainingtime (8)= currenttime (r6) - starttime (r5)
  cmp  r8,r4  ;compare remainingtime (r8), delay (r4)
  bls loopt2 ;loop if LE (reaminingtime <= delay)
 b loop$
```

ALL THE CODE (2)

```
BASE = $FE000000 ;4B
GPIO_OFFSET = $200000
mov r0,BASE
orr r0,GPIO_OFFSET ;Base address of GPIO
mov r1,#1
lsl r1,#24
str r1,[r0,#4] ;enable output
mov r1,#1
lsl r1,#18
loop$:
 str r1,[r0,#28] ;Turn on LED
 ;new timer
TIMER_OFFSET = $3000
mov r3,BASE
orr r3,TIMER_OFFSET ;store base address of timer (r3)
 mov r4, 524288  ;store delay (r4)
 ldrd r6,r7,[r3,#4]
 mov r5,r6 ;store starttime (r5)(=currenttime (r6))
 loopt1:
  ldrd r6,r7,[r3,#4] ;read currenttime (r6)
  sub r8,r6,r5  ;remainingtime (8)= currenttime (r6) - starttime (r5)
  cmp  r8,r4  ;compare remainingtime (r8), delay (r4)
  bls loopt1 ;loop if LE (reaminingtime <= delay)
 str r1,[r0,#40]  ;turn off LED
 ;re-use timer
 ldrd r6,r7,[r3,#4]
 mov r5,r6 ;store starttime (r5)(=currenttime (r6))
 loopt2:
  ldrd r6,r7,[r3,#4] ;read currenttime (r6)
  sub r8,r6,r5  ;remainingtime (8)= currenttime (r6) - starttime (r5)
  cmp  r8,r4  ;compare remainingtime (r8), delay (r4)
  bls loopt2 ;loop if LE (reaminingtime <= delay)
 b loop$
```



USING THE TIMER REGISTER

- An absolute timer (no longer dependent on processor speed!
- Our approach is still a form of busy wait though !
  - CPU still occupied for duration of delay
  - Essentially this is polling the timer
- Is there a potentially even better approach ?
  - Well yeah! We could implement an interrupt based timer
  - More complicated though (and beyond)

SUMMARY

- The timer register provides an accurate timer that can easily be accessed
- We read it using `ldrd`
  - a version of load register that reads 64 bit words into two specified 32 bit registers
- Our timer is better than before, but is still a "busy wait" timer
  - Interrupt based timing would solve this