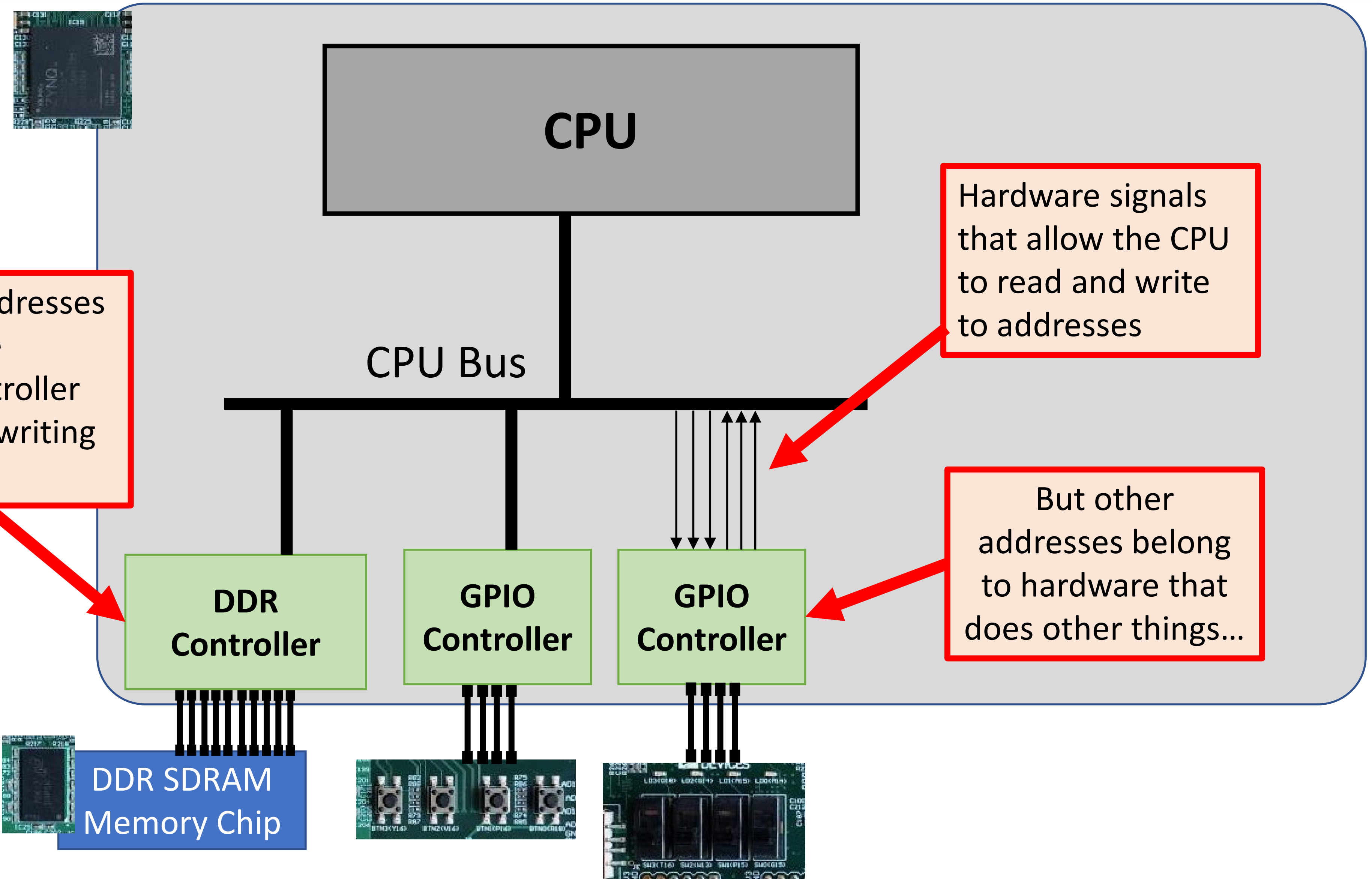
Lab 3: Interval Timer

ECEN 330

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

CPU Bus

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

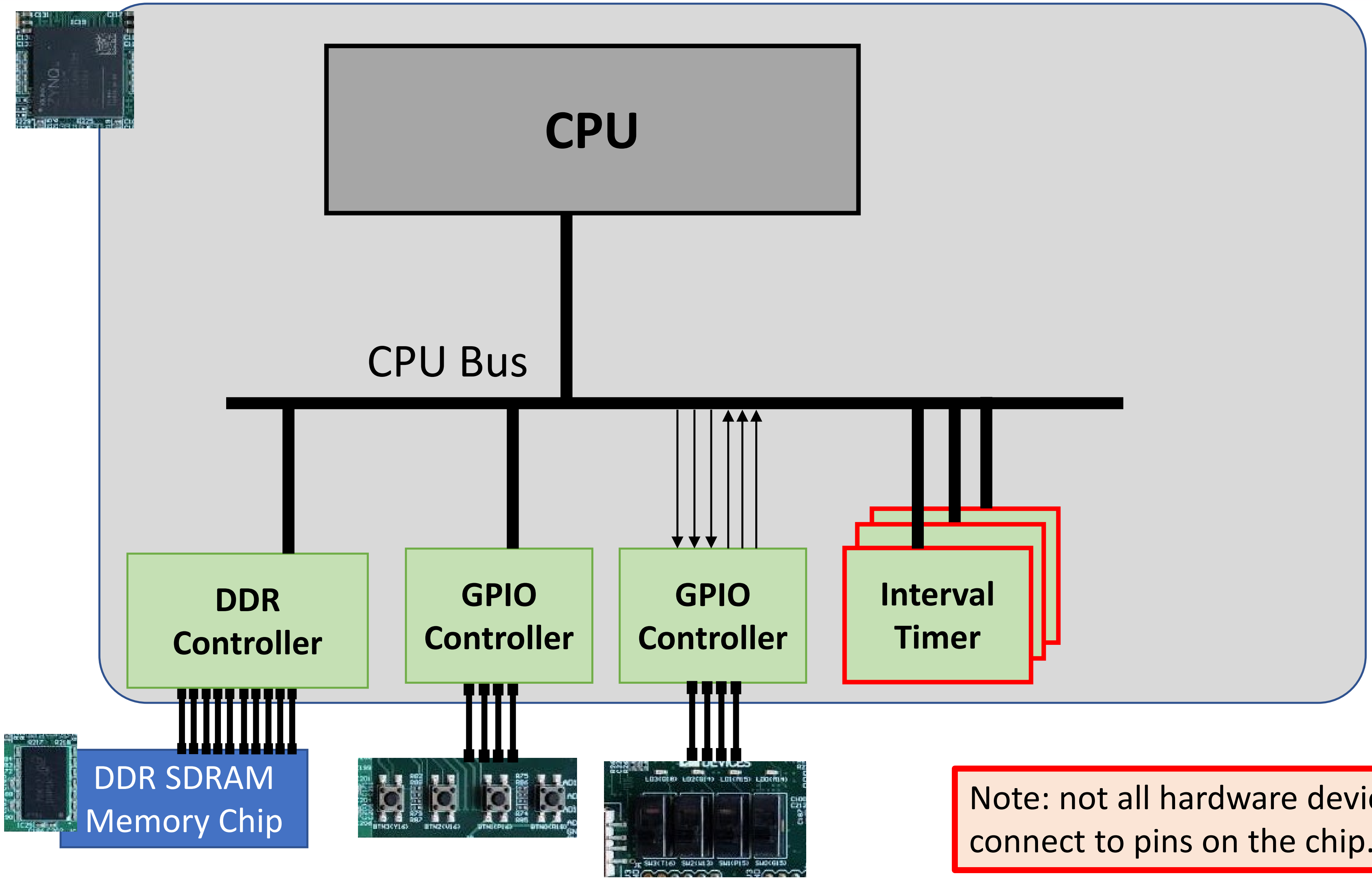


When the addresses belong to the memory controller we are read/writing to memory

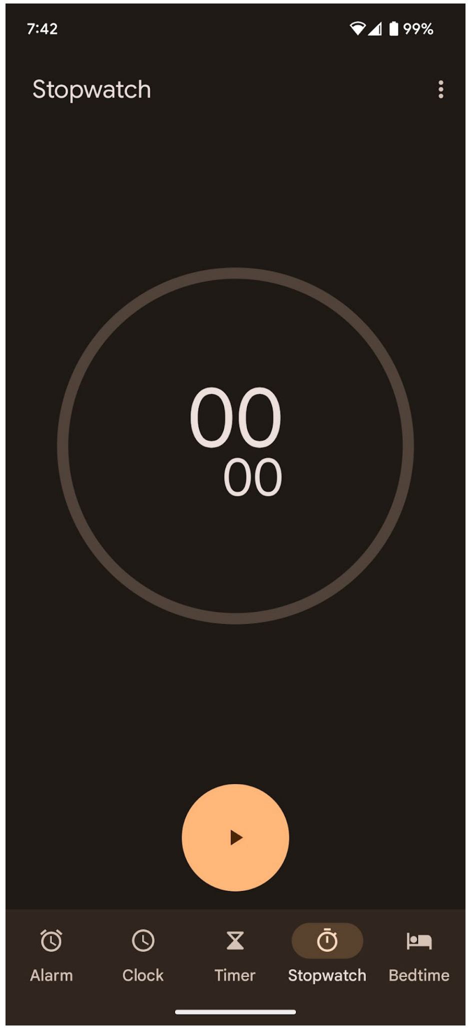
Hardware signals that allow the CPU to read and write to addresses

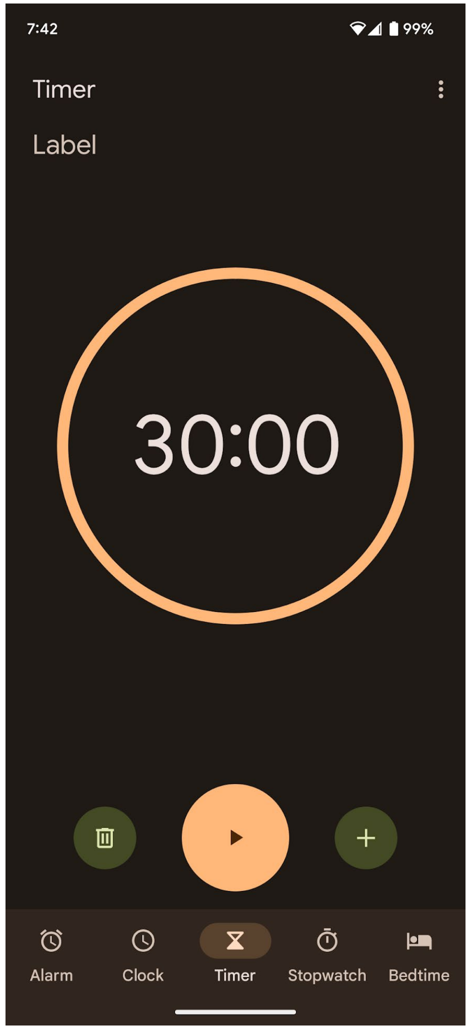
But other addresses belong to hardware that does other things…

CPU Bus

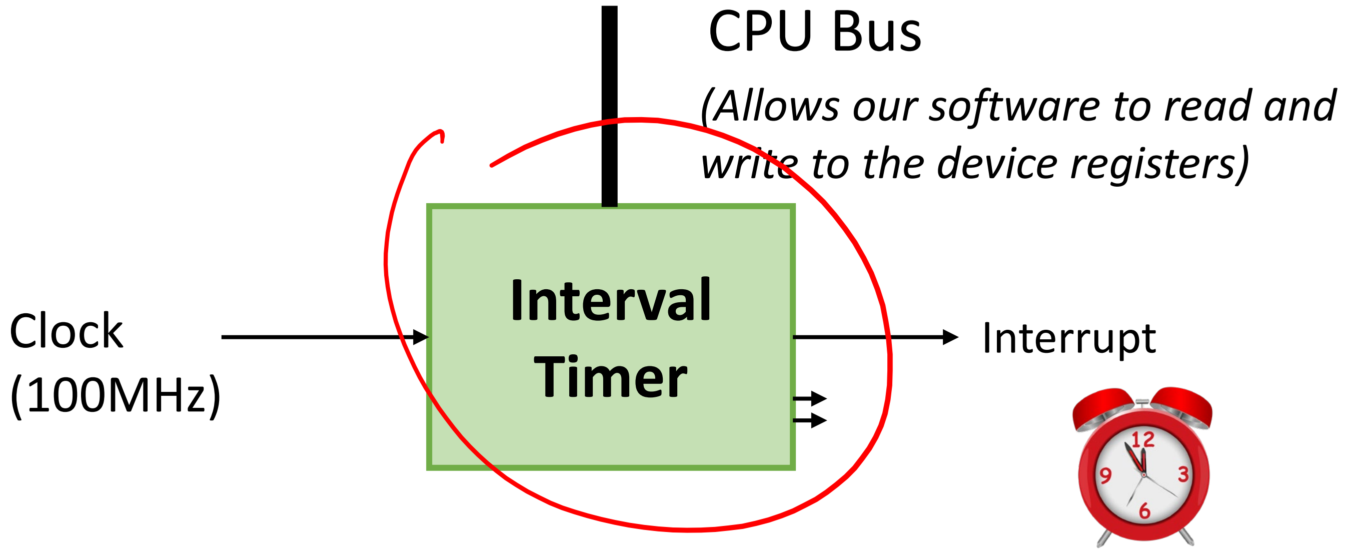


Note: not all hardware devices need to connect to pins on the chip.


7:42
99%
Stopwatch
00
00
Alarm
Clock
Timer
Stopwatch
Bedtime


7:42
99%
Timer
Label
30:00
Alarm
Clock
Timer
Stopwatch
Bedtime

Interval Timer: Connections

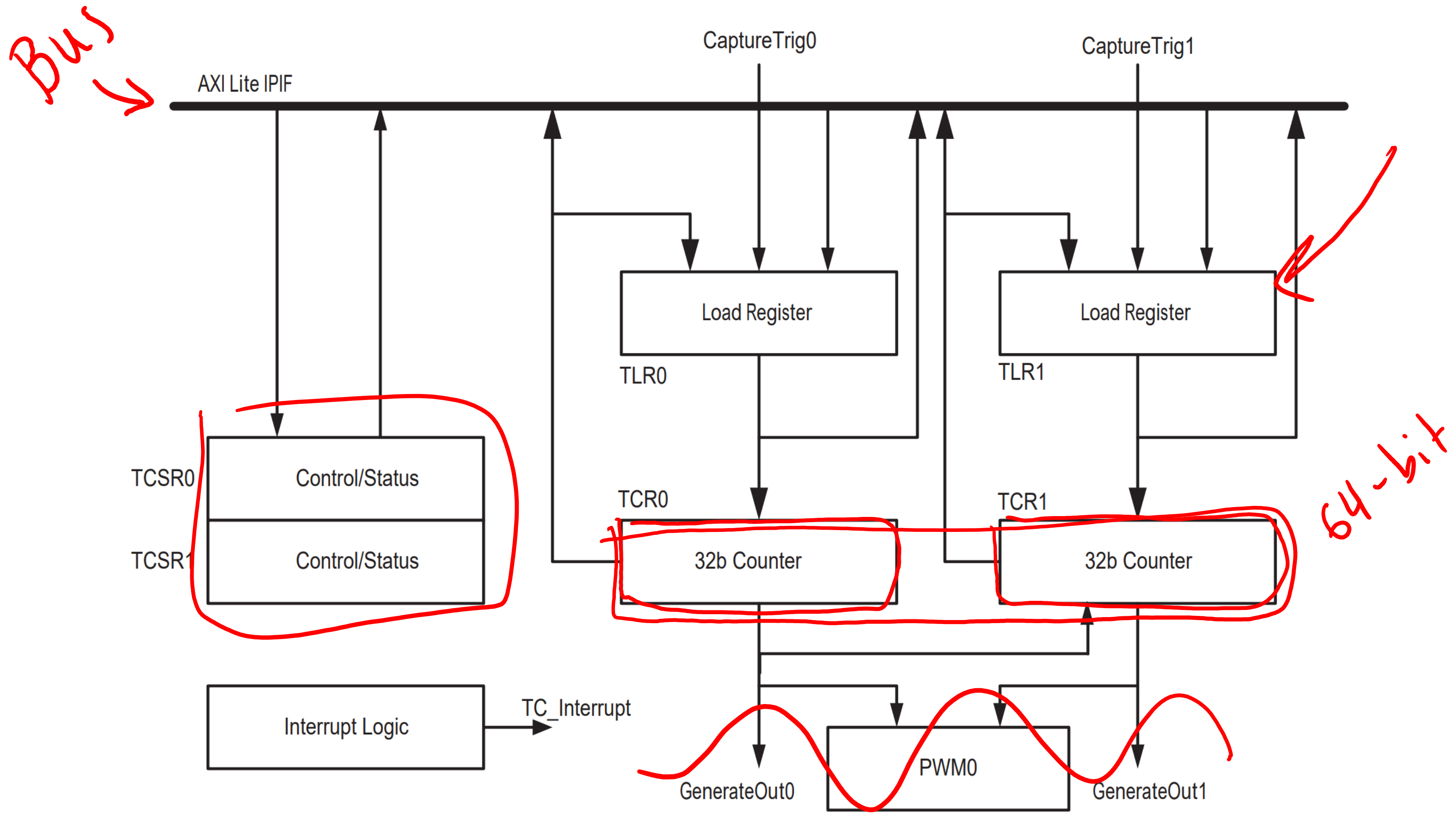

CPU Bus
(Allows our software to read and write to the device registers)
Clock (100MHz)
Interval Timer
Interrupt

This timer is very similar to the counters you made in ECEN 220, except:

- Very little I/O aside from the CPU bus = this hardware was made to be controlled from software
- Has more features

Looking Inside

…from the commercial documentation…

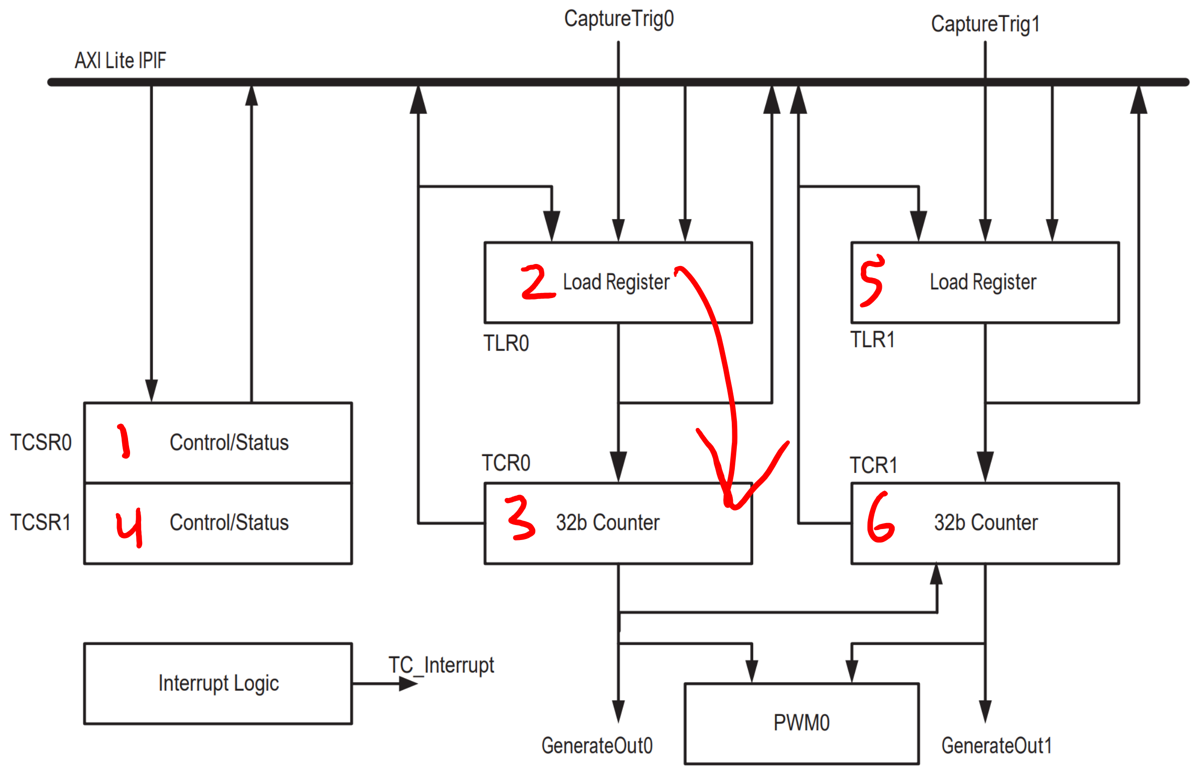


Looking Inside

…from the commercial documentation…

Table 4: **Register Overview**

Register Name	Address (hex)	Access Type	Default Value (hex)	Description
TCSR0 1	0x00	Read/Write	0x0	Control/Status Register 0
TLR0 2	0x04	Read/Write	0x0	Load Register 0
TCR0 3	0x08	Read	0x0	Timer/Counter Register 0
TCSR1	0x10	Read/Write	0x0	Control/Status Register 1
TLR1	0x14	Read/Write	0x0	Load Register 1
TCR1	0x18	Read	0x0	Timer/Counter Register 1



Remember we have 3 of these….

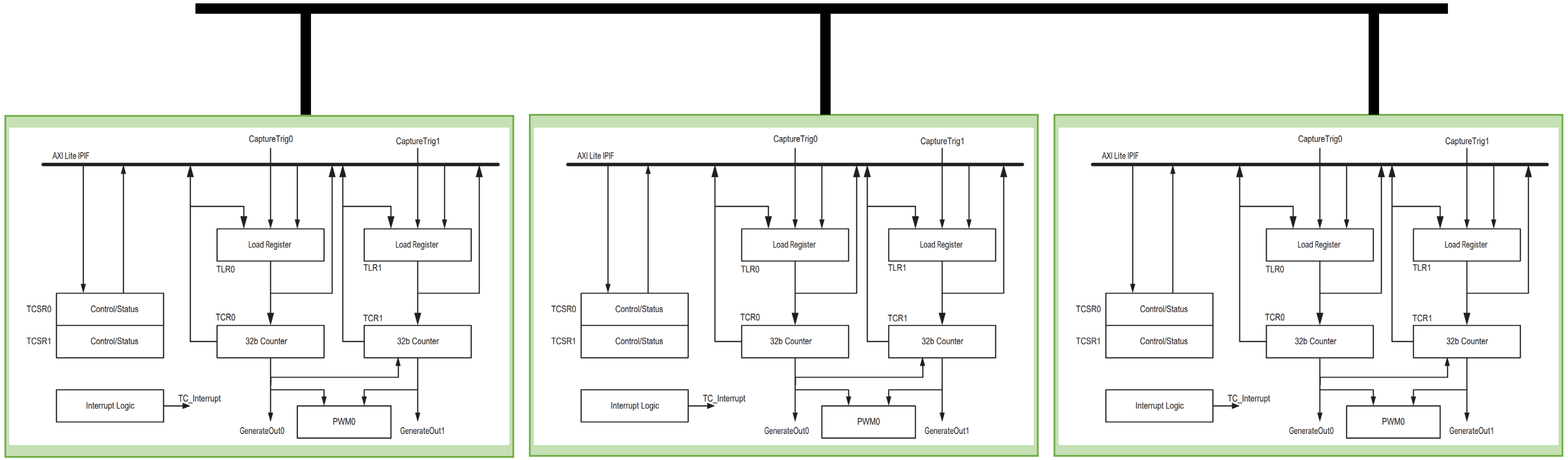


Table 4: **Register Overview**

Register Name	Address (hex)	Access Type	Default Value (hex)	Description
TCSR0	0x00	Read/Write	0x0	Control/Status Register 0
TLR0	0x04	Read/Write	0x0	Load Register 0
TCR0	0x08	Read	0x0	Timer/Counter Register 0
TCSR1	0x10	Read/Write	0x0	Control/Status Register 1
TLR1	0x14	Read/Write	0x0	Load Register 1
TCR1	0x18	Read	0x0	Timer/Counter Register 1

Table 4: **Register Overview**

Register Name	Address (hex)	Access Type	Default Value (hex)	Description
TCSR0	0x00	Read/Write	0x0	Control/Status Register 0
TLR0	0x04	Read/Write	0x0	Load Register 0
TCR0	0x08	Read	0x0	Timer/Counter Register 0
TCSR1	0x10	Read/Write	0x0	Control/Status Register 1
TLR1	0x14	Read/Write	0x0	Load Register 1
TCR1	0x18	Read	0x0	Timer/Counter Register 1

Table 4: **Register Overview**

Register Name	Address (hex)	Access Type	Default Value (hex)	Description
TCSR0	0x00	Read/Write	0x0	Control/Status Register 0
TLR0	0x04	Read/Write	0x0	Load Register 0
TCR0	0x08	Read	0x0	Timer/Counter Register 0
TCSR1	0x10	Read/Write	0x0	Control/Status Register 1
TLR1	0x14	Read/Write	0x0	Load Register 1
TCR1	0x18	Read	0x0	Timer/Counter Register 1

Control/Status

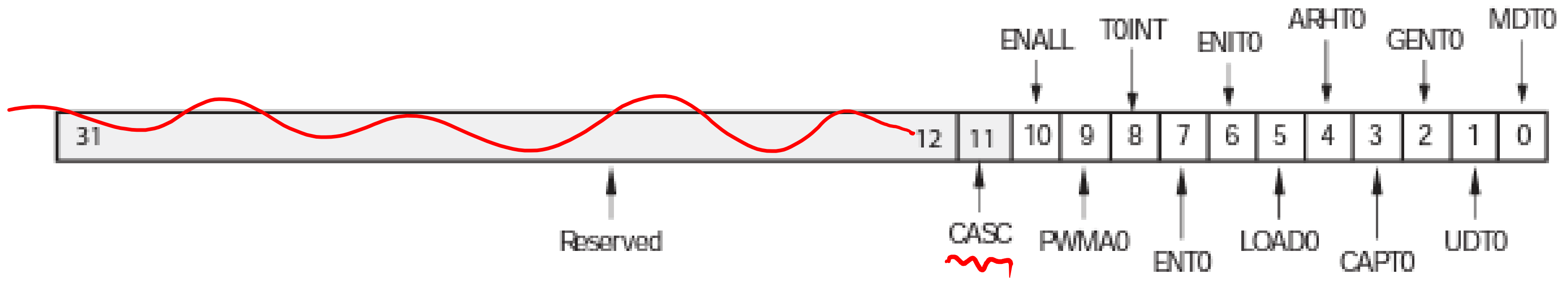


Figure 6: **Control/Status Register 0**

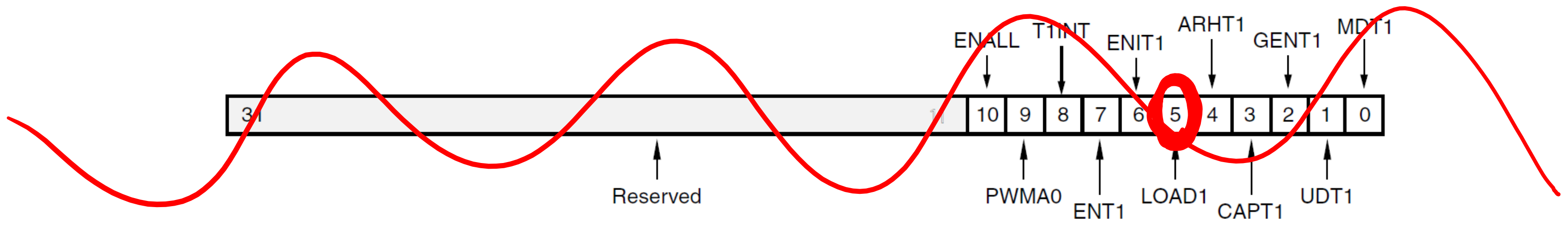


Figure 7: **Control/Status Register 1**

Helpful snippet from the documentation:

- "Timer Control Status Register for timer/counter 0 (TCSR0) acts as the control and status register for the cascaded counter. TCSR1 is ignored in this mode.*"
- "TCSR1 register is used only for loading the TLR1 register in cascade mode."

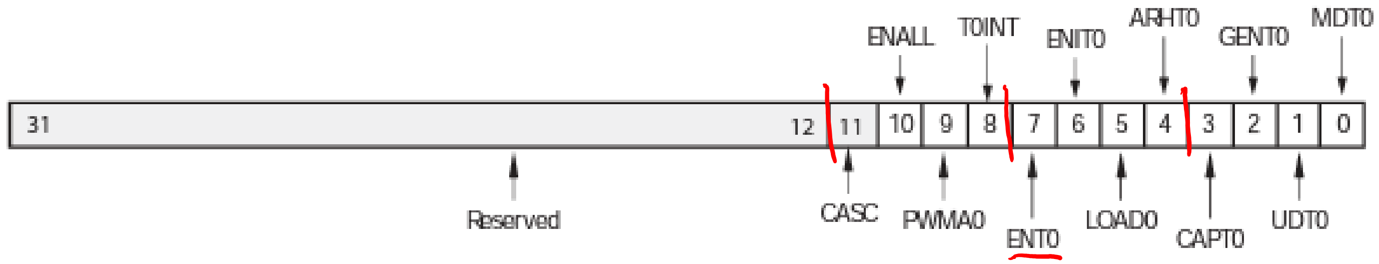


Figure 6: **Control/Status Register 0**

11	CASC	Read/Write	0	Enable cascade mode of timers 0 = Disable cascaded operation 1 = Enable cascaded operation
7	ENT0	Read/Write	0	Enable Timer0 0 = Disable timer (counter halts) 1 = Enable timer (counter runs)
5	LOAD0	Read/Write	0	Load Timer0 0 = No load 1 = Loads timer with value in TLR0
4	ARHT0	Read/Write	0	Auto Reload/Hold Timer0
1	UDT0	Read/Write	0	Up/Down Count Timer0

Init

Init

```
Init(){
 ...
 writeReg(TCSR0,0x800)
}
```

0x80 = 1000 0000
~(0x80) = 0111 1111

```
_start(){
data=readReg(TCSR0)
data=data | 0x80;
writeReg(TCSR0,data)
}
```

```
_stop(){
data = data & ~(0x80)
```

Reload

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

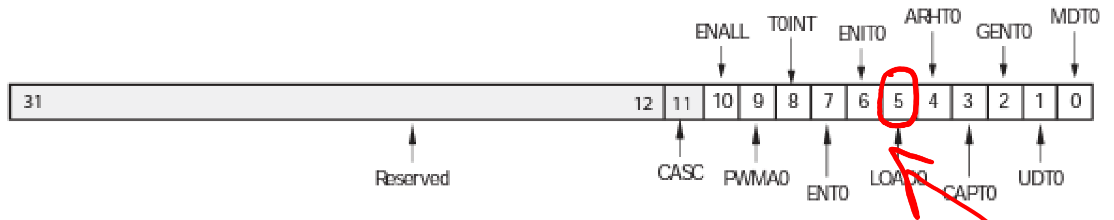


Figure 6: **Control/Status Register 0**

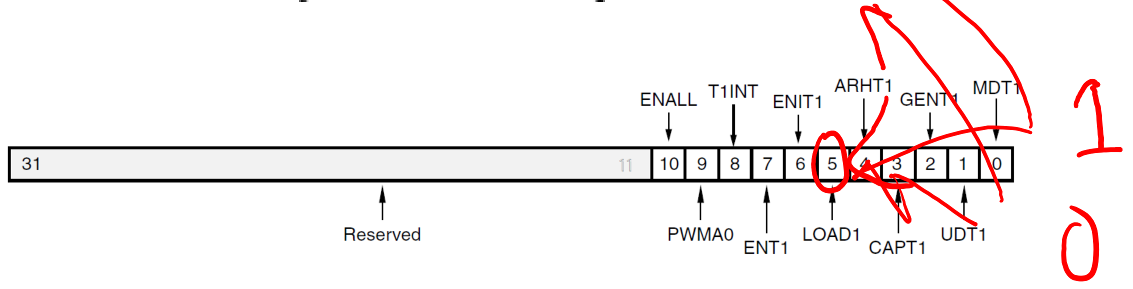
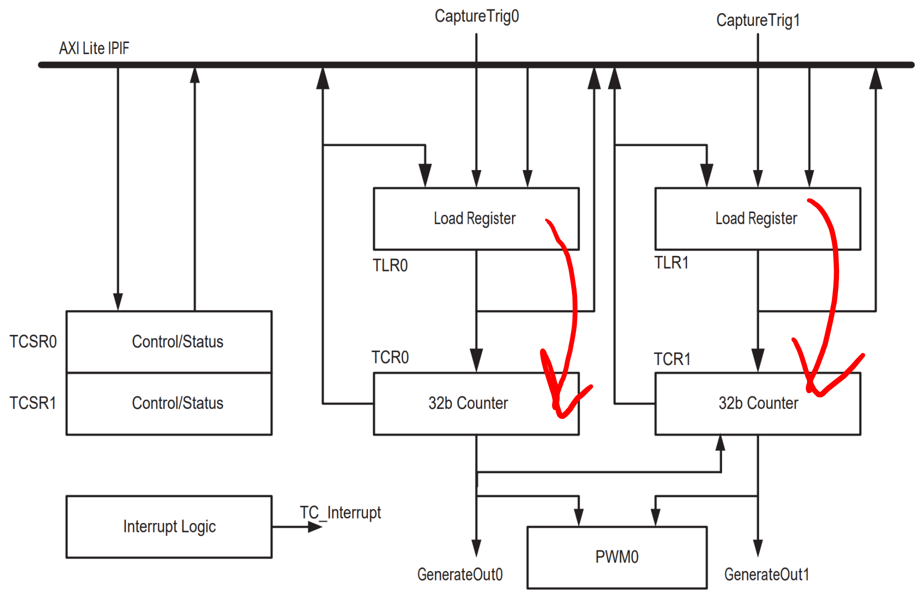


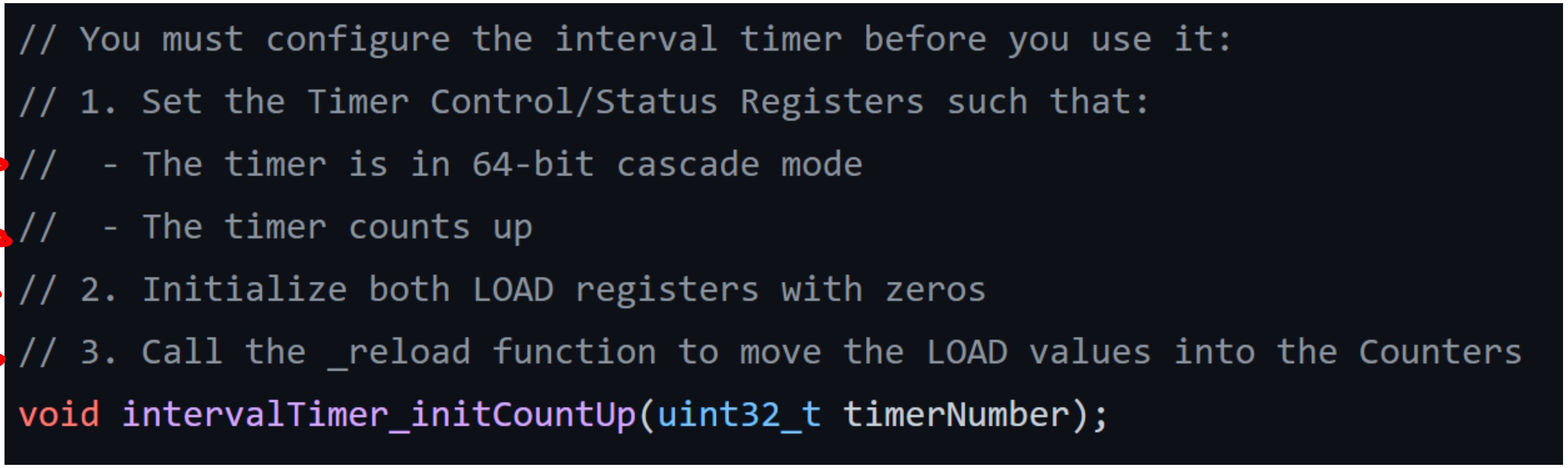
Figure 7: **Control/Status Register 1**



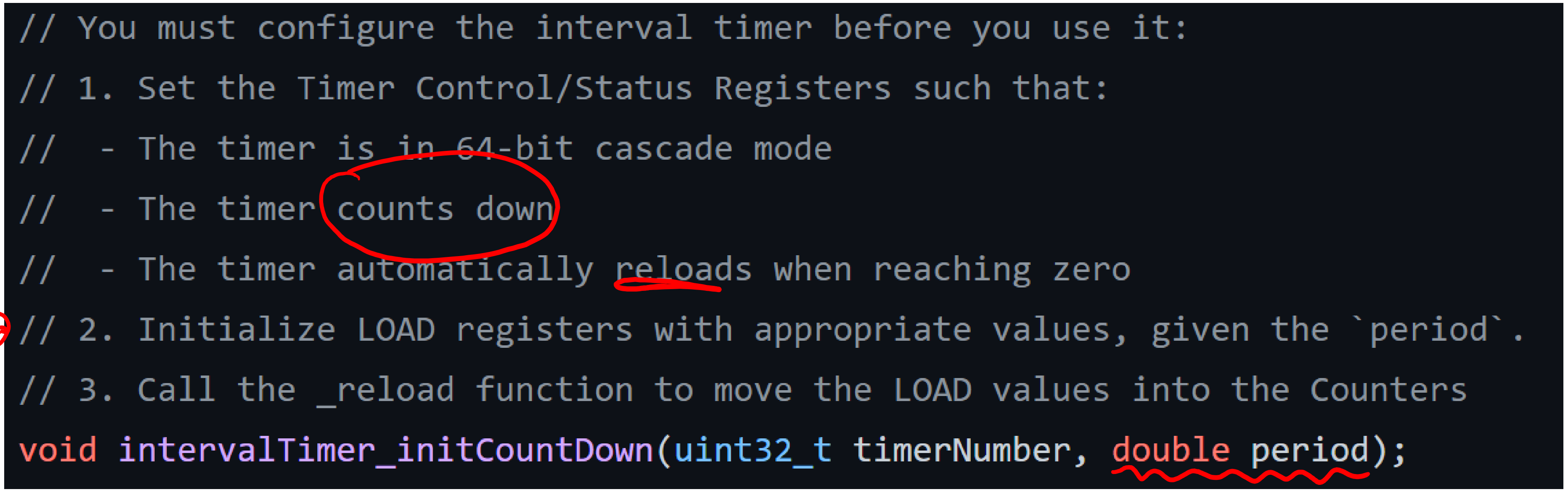
```
void intervalTimer_reload(uint32_t timerNumber);
```

Initialize

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING



```
// You must configure the interval timer before you use it:
// 1. Set the Timer Control/Status Registers such that:
//  - The timer is in 64-bit cascade mode
//  - The timer counts up
// 2. Initialize both LOAD registers with zeros
// 3. Call the _reload function to move the LOAD values into the Counters
void intervalTimer_initCountUp(uint32_t timerNumber);
```

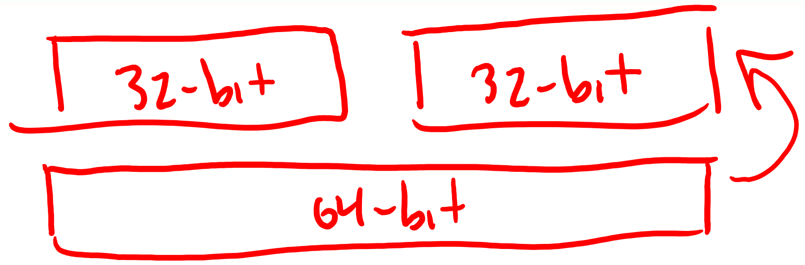


```
// You must configure the interval timer before you use it:
// 1. Set the Timer Control/Status Registers such that:
//  - The timer is in 64-bit cascade mode
//  - The timer counts down
//  - The timer automatically reloads when reaching zero
// 2. Initialize LOAD registers with appropriate values, given the `period`.
// 3. Call the _reload function to move the LOAD values into the Counters
void intervalTimer_initCountDown(uint32_t timerNumber, double period);
```

Ticks $\longleftrightarrow$ Period (s)

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

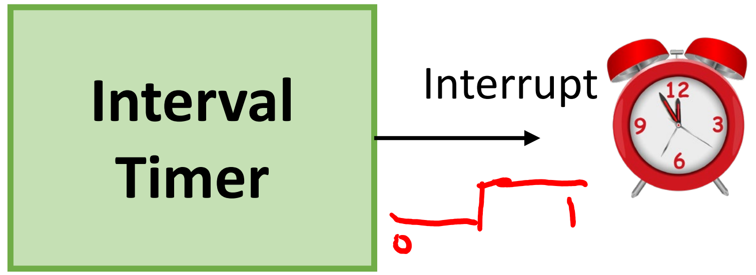
Period (s) $\longrightarrow$



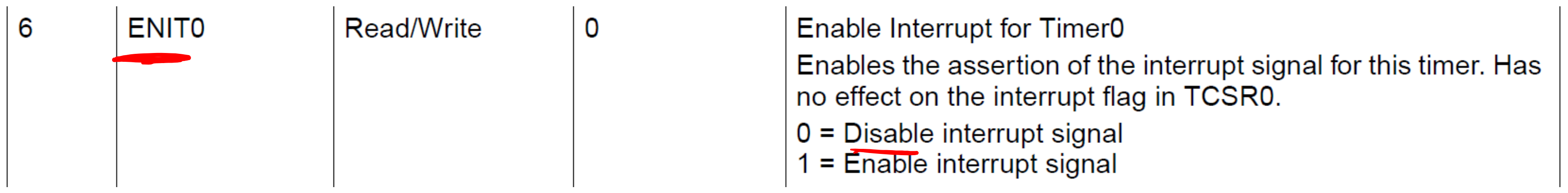
```
uint64_t ticks = 100 000 000 * period

uint32_t upper = ticks >> 32;
uint32_t lower = ticks;
```

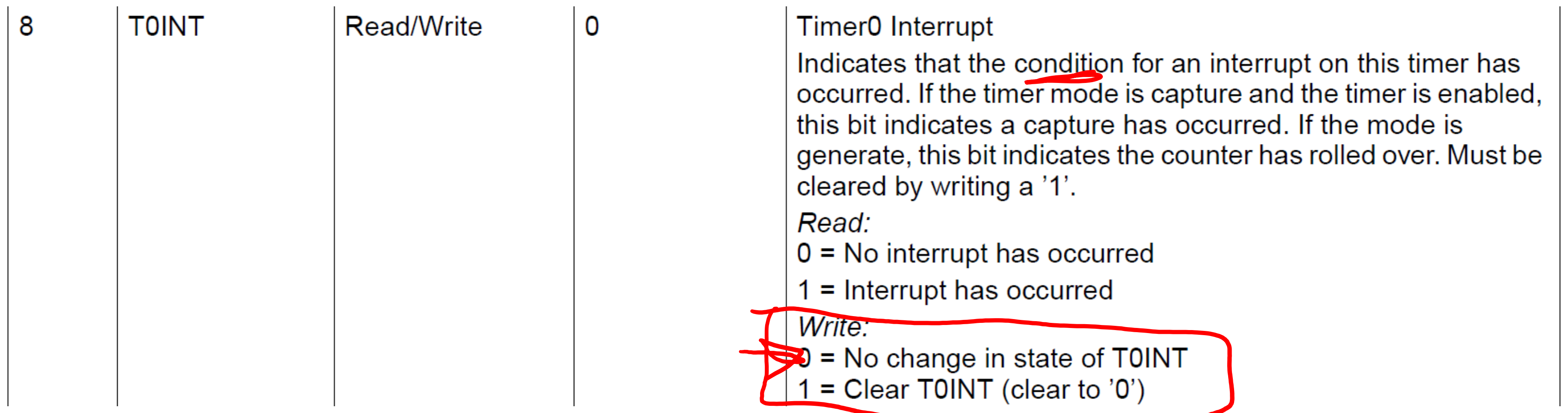
Interrupts



- “In Generate Mode, a timer interrupt is caused by the counter rolling over (the same condition used to reload the counter when ARHT is set to ’1’).”
- “The interrupt signals can be enabled or disabled with the ENIT bit in the TCSR.”
- “The interrupt status bit (TINT) in the TCSR cannot be disabled and always reflects the current state of the timer interrupt.”



6	ENIT0	Read/Write	0	Enable Interrupt for Timer0 Enables the assertion of the interrupt signal for this timer. Has no effect on the interrupt flag in TCSR0. 0 = Disable interrupt signal 1 = Enable interrupt signal



8	T0INT	Read/Write	0	Timer0 Interrupt Indicates that the condition for an interrupt on this timer has occurred. If the timer mode is capture and the timer is enabled, this bit indicates a capture has occurred. If the mode is generate, this bit indicates the counter has rolled over. Must be cleared by writing a ’1’. *Read:* 0 = No interrupt has occurred 1 = Interrupt has occurred *Write:* 0 = No change in state of T0INT 1 = Clear T0INT (clear to ’0’)

Enable/Disable

```c
// Enable the interrupt output of the given timer.
void intervalTimer_enableInterrupt(uint8_t timerNumber);
```

1 Write to ENIT

```c
// Disable the interrupt output of the given timer.
void intervalTimer_disableInterrupt(uint8_t timerNumber);
```

0

Acknowledge

```c
// Acknowledge the rollover to clear the interrupt output.
void intervalTimer_ackInterrupt(uint8_t timerNumber);
```

write 1 to T0INT

Use Register Read/Write Helper Fcns

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

```
static uint32_t readRegister(uint8_t timerNumber, uint32_t offset);
static void writeRegister(uint8_t timerNumber, uint32_t offset, uint32_t value);
```

Use helper functions for all register reads/writes (like last lab)

Except:

- Since we have 3 timers, the helper functions will take a “timerNumber” argument.
- Use this argument to choose the correct base address.