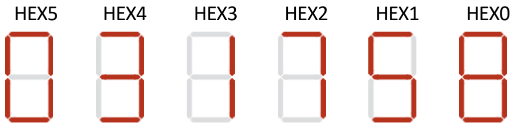
ECE3375B Lab 2: Stopwatch

Objective

In this lab, we will turn the DE-10 board into a stopwatch. We have six seven-segment displays, so we will organize them as: MM:SS:HH. That is to say, the leftmost two digits display the number of minutes counted, the middle two digits display the number of seconds counted and the rightmost two digits display the count in hundredths of a second.



Timer displaying a time of 3 minutes and 17.58 seconds.

The seven segment displays have decimal points, but unfortunately the decimal points have not been wired to the Cyclone V, so we'll have to imagine it. We will use the push buttons and slide switches to control the stopwatch, and one of the DE10-Standard's many timers to accurately control the time measured by the stopwatch.

Version History: This lab manual was originally written by Prof. Ken McIsaac and used in ECE3375 from 2019 to 2022. It was rewritten by Prof. John McLeod in 2023.

Remember that all the features of hardware and the debugging interface are available in the online simulator: https://cpulator.01xz.net/?sys=arm-de1soc.

Deliverables

For this lab, all you need to do is demonstrate the working timer and answer some questions from the TAs. The TAs may wish to play around with your stopwatch and press all the buttons.

You do not need to submit any report or show your code, however if you are **unable** to get the timer working in hardware then you may show the TAs your code to possibly obtain partial credit for this lab.

Stopwatch Operation

The DE10-Standard has four push-button switch. These are debounced in hardware, so you don't need to worry about debouncing them. These four switches are labelled as KEY0 to KEY3 on the board, and are memory mapped to address 0xFF200050 (called KEY_BASE in address map arm.h). We also need to use the first slide switch, SW0, which you may recall is memory mapped to address 0xFF200040 (called SW_BASE in address map arm.h). We will use these to operate the timer as follows:

Address	Switch	Bit Position	Function	Description
0xFF200050	KEY0	Bit 0	Start	Start the stopwatch
	KEY1	Bit 1	Stop	Stop the stopwatch
	KEY2	Bit 2	Lap	Store current time
	KEY3	Bit 3	Clear	Resets the stopwatch back to zero
0xFF200040	SW0	Bit 0	Display	Toggle between displaying current time and lap time

Basically, the DE10-Standard stopwatch should operate the same way as normal hand-held stopwatch.

- **Start** starts the stopwatch counting up in increments of 0.01 s. This count time should be stored somewhere in a register or memory, and if the slide switch SW0 is toggled appropriately, the count time should also be displayed in real-time on the seven-segment displays.
- **Stop** stops the stopwatch from counting up. The stopwatch should simply hold at the current time. If **Start** is subsequently pressed, the stopwatch should continue counting from the old time.
- **Lap** captures the time currently stored. This should be stored somewhere in a register or memory, and if the slide switch SW0 is toggled appropriately, the lap time should also be displayed on the seven-segment displays. Note that if **Lap** is pressed after **Start**, and the slide switch SW0 is set to display the lap time, the stopwatch *keeps counting up* even though it isn't displayed. Only one lap time can be stored. Pressing **Lap** again overwrites the previously-stored lap time with the new value.
- **Clear** returns the stopwatch to the initial state. The clock time and lap time are reset to zero and the seven-segment display should read 00:00:00.

The description above states that the slide switch is "toggled appropriately". The switch is binary, 0 or 1. You can decide which value displays the count time or lap time.

The above instructions are ambiguous whether or not pressing **Clear** also stops the count. If someone presses **Start** and then presses **Clear**, there are two possible things that can happen:

- The count clears to 00:00:00 and the stopwatch stops counting.
- The count clears to 00:00:00 but the stopwatch immediately resumes counting.

In other words, it is ambiguous whether the *only* way to stop the stopwatch is to press **Stop**, or if **Clear** will also stop the stopwatch. You can choose whichever operation makes the most sense to you, both are valid.

Finally, the stopwatch count timer or lap time should be displayed across all six of the seven-segment display panels. The first four panels (corresponding to the seconds and hundredths of seconds), HEX3 to HEX0, are memory mapped to address 0xFF200020 (called HEX3_HEX0_BASE in address map arm.h). As you may recall from lab 1, each display panel uses a byte, so these four displays span the entire word at 0xFF200020. The next two panels are memory mapped to address 0xFF200030 (called HEX5_HEX4_BASE in address map arm.h). These two occupy only the two least-significant bytes of the word at 0xFF200030.

Preparation

This lab might be tricky to complete. It is definitely possible to complete the lab in the 3-hour session, but it would be a lot easier if you came prepared. Ideally with an entire working program...

If you are having difficulty figuring out how to implement a stopwatch, the TAs can help during you lab session. You can make your life (and everyone else's) a lot easier by **implementing as much functionality as possible** before coming to the lab.

- Even if you aren't sure how to implement a stopwatch, you know it will use a timer. You can write subroutines to initialize the timer to a particular interval, start the timer, stop the timer, check if the timer is finished counting, wait until the timer is done counting, etc.
- You can test all of these subroutines in the simulator just by observing whether or not the timer hardware is working as expected when you run the code.

That way once you figure out how the timer should be used in your stopwatch, you can start coding the main program without worrying about the technical details of the timer.

Similarly, another difficult part of this lab is displaying values across the 6 seven-segment display panels. But again, you can implement a subroutine that takes any arbitrary integer (or 6 digits, depending on how you want to implement it) and writes them across all 6 panels.

Tips on Software Design

Implementing a stopwatch is not a trivial problem, and you will need to think about how you want to do it. The stopwatch needs to count up to 1 hour in increments of 1/100th of a second before rolling over.

One major design question you should thin about is: *how is the timer hardware controlled by software*?

- One approach is to have the timer start counting continuously when the **Start** button is pressed, and have it stop counting when the **Stop** button is pressed. Software keeps track of the timer's progress to update stopwatch time.
- Another approach is to have the timer count down once, and each time it does so increment a counter in software. The timer is manually restarted by software each time as long as the **Start** button was previously pressed and the **Stop** button has not yet been pressed. Software keeps track of the manual timer resets to update stopwatch time.
- An interesting idea is to have the timer count continuously when the program is powered on and *never stop*. The **Start** and **Stop** buttons are just used to record whether or not the stopwatch time is updated based on the timer's progress.

The *timing accuracy* of the three ideas above is different, but is it different enough for anyone to notice?

A second major design question you should think about is: *how will the current time (and lap time) be stored*?

- One approach would be to implement the stopwatch time in software, and use the timer hardware to count for the basic time step. Once the timeout flag on the timer is set, you increment the stopwatch time in software and display it.
- Another approach is to have the timer hardware count for a longer duration (seconds, minutes - but can any of the 32-bit timers count for an hour?) and frequently check the count on the timer to see when to update the display.
- Obviously the lap time must be stored as a variable in software, since the timer hardware doesn't have any extra memory, but the way it is stored should mimic the way the current stopwatch time is stored, so it can be displayed the same way.

Remember that the timers are an external piece of hardware. Once started they will continue counting even if the CPU does other things (this is obvious in the simulator, if a counter is set to "count-down-and-repeat" it will keep going even if you stop execution of your code).

Another major design question to think about is: *how will the current time (and lap time) be displayed*?

- You wrote some code to display a single digit on the hex display in Lab 1. But now you need to display 6 digits.
- It is easy to store the current time (or lap time) as a single integer (presumably in increments of 1/100th of a second), but that make it more difficult to show on a 6-digit display.
- Splitting the time into 6 variables (for tens of minutes, minutes, tens of seconds, seconds, tenths of seconds, and hundredths of seconds) makes it easier to show each digit, but then it is more complicated to increment the count time.

It might be useful to remember the modulo operator: `(n%m)` divides `n` by `m` and keeps the remainder. The modulo operator is natively implemented in C. It is a lot trickier to implement in Assembly. Remember also that unless you are explicitly doing floating-point arithmetic, all division results in an integer. How is this useful? Consider `(27%10) = 7` and `(27/10) = 2`. This shows you how to isolate individual digits in a larger decimal number.

There are a lot of ways you can implement this design, depending on how you answer these questions.

Timer Peripherals

This lab obviously requires using a timer peripheral. It simply isn't acceptable to fool around with dead loops to try and figure out the appropriate number of loop iterations for the system to pause for the correct amount of time (the way we did in lab 1), we need accuracy in this device.

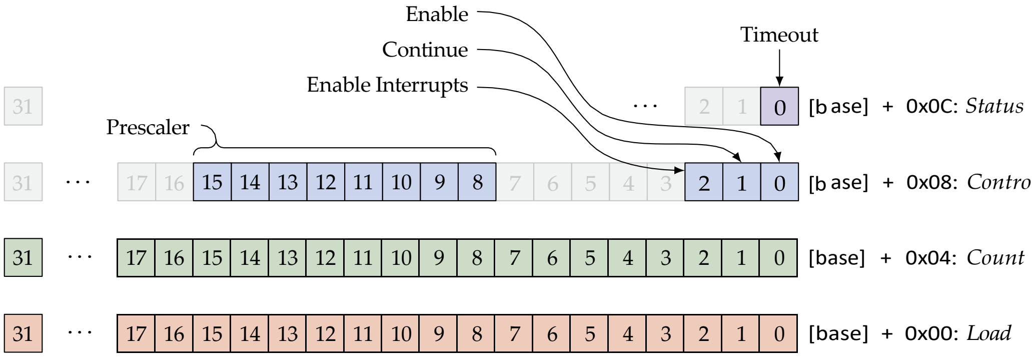
There are a lot of timers on the DE10-Standard to choose from; it doesn't really matter which one you use. Three good options are:

1. The A9 Private Timer at 0xFFFEC600. This is built into the CPU (actually there are two of these, one for each CPU code - hence the "private" in the name) and has a 200 MHz clock. This timer is also implemented in the simulator, and has a simple interface. *We recommend that you use this timer for the lab.*
2. One of the four HPS Timers at 0xFFC08000, 0xFFC09000, 0xFFD00000, or 0xFFD01000. The first two have a 100 MHz clock, the last two have a 25 MHz clock. Labs from 2020 and earlier used a HPS Timer. This also has a simple interface, but unfortunately these are *not implemented in the simulator*. As we recommend you extensively test your code on the simulator before coming to the lab, we do not recommend using this timer.
3. One of the two Interval Timers at 0xFF202000 and 0xFF202020. These both have a 100 MHz clock. The labs from 2021 used an Interval Timer. These timers unfortunately have an awkward interface: although they are 32-bit timers just like the others, the actual device hardware is just 16-bit. However, each of these 16-bit hardware registers is mapped to a 32-bit memory space! OK, it's not that hard to work with, but I think the other two timers are simpler to use. The Interval Timers are available in the simulator, so if you enjoy wrestling with bit shifts you can use this timer instead.

The detailed interfaces for these timers are given below. Please check the course notes for sample code for operating these timers.

A9 Private Timer

The A9 private timer has the structure shown below. The A9 private timer has a base address of 0xFFFEC600. This address in called MPCORE_PRIV_TIMER in address_map_arm.h.



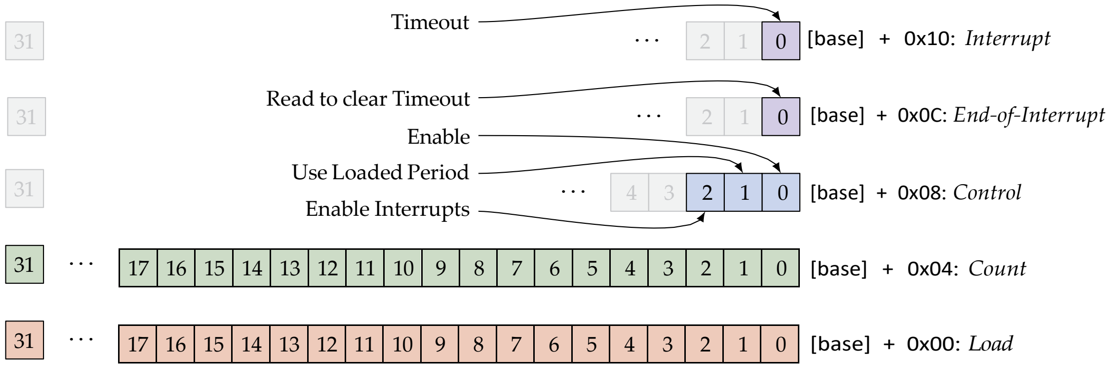
The timer functions as follows:

- To **set the count period**, write the appropriate 32-bit value to the *Load* register.
- To **start the timer counting down**, write 0b01 to the *Status* register to clear any previous timeout flag, then write 0b0001 to the *Control* register to count down once and stop, or write 0b0011 to count down and repeat.
- To **stop the timer mid-count**, write 0b0000 to the *Control* register.
- To **check if the count is done**, test if bit 0 of the *Status* register is set.
- To **check the current count**, read the *Count* register.

We can use the timer with its base frequency of 200 MHz for this lab, but you can also change the base frequency using the prescaler bits in the *Control* register if you want to and you also really love playing around with hardware. See the DE10-Standard documentation or the course notes for more information.

HPS Timer

The HPS timer has the structure shown below. Two HPS timers have a 100 MHz clock, these have base addresses of 0xFFC08000 and 0xFFC09000. These addresses are called HPS_TIMER0_BASE and HPS_TIMER1_BASE in address_map_arm.h. Another two HPS timers have a 25 MHz clock, these have base addresses of 0xFFD00000 and 0xFFD01000. These addresses are called HPS_TIMER2_BASE and HPS_TIMER3_BASE in address_map_arm.h. Any one of these can be used for this lab, but please note that these are *not* implemented in the online simulator.

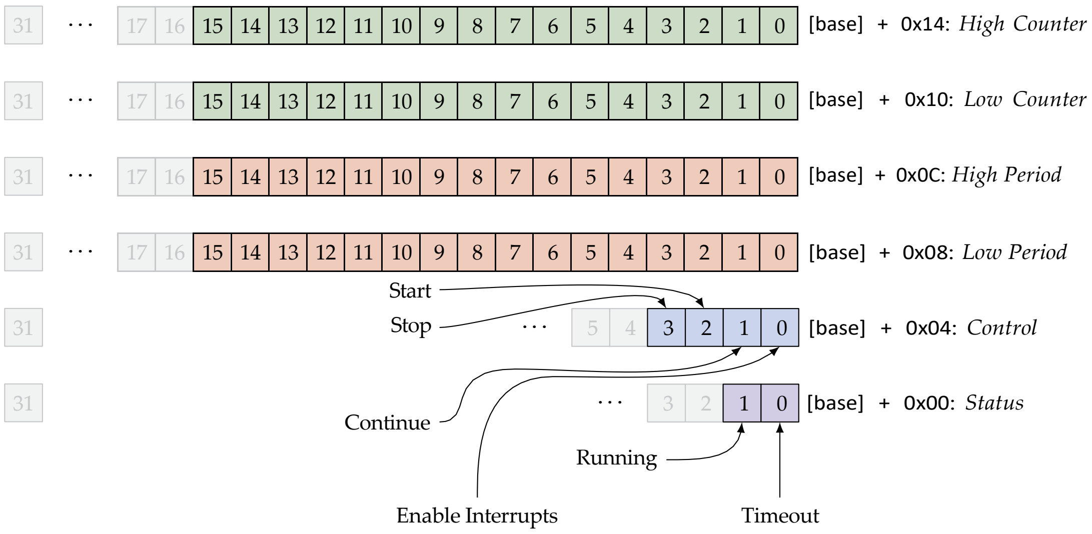


The timer functions as follows:

- To **set the count period**, first write 0b0000 to the *Control* register. Then write the appropriate 32-bit value to the *Load* register. Finally, write 0b0010 to the *Control* register to tell the hardware to use the newly loaded count time in the *Load* register.
- To **start the timer counting down**, first read from the *End-of-Interrupt* register to clear any previous timeout flag. Then write 0b0011 to the *Control* register to count down once and stop. Note that the functionality to "count down and repeat" is not available in this hardware.
- To **stop the timer mid-count**, write 0b0010 to the *Control* register.
- To **check if the count is done**, test if bit 0 of the *Interrupt* register is set.
- To **check the current count**, read the *Count* register.

Interval Timer

The interval timer has the structure shown below. There are two interval timers, each with a 100 MHz clock, at addresses 0xFF202000 and 0xFF202020. These addresses are called TIMER_BASE and TIMER_2_BASE in address_map_arm.h. Either of these can be used for this lab.



The timer functions as follows:

- To **set the count period**, write the appropriate 32-bit value to the *Low Period* register. Note that only the lowest 16-bits of this value will be written. Shift the original 32-bit value right by 16 bits and write this to the *High Period* register.
- To **start the timer counting down**, first write anything to the *Status* register to clear any previous timeout flag. Then write `0b0100` to the *Control* register to count down once and stop. Write `0b0110` to the *Control* registre to count down and repeat.
- To **stop the timer mid-count**, write `0b1000` to the *Control* register.
- To **check if the count is done**, test if bit 0 of the *Status* register is set.
- To **check the current count**, write anything to the *Low Counter* register. This instructs the hardware to refresh the current count time. Read the *Low Counter* register to obtain the lowest 16 bits of the count and read the *High Counter* register to obtain the highest 16 bits of the count.

Evaluation

You will be evaluated based on demonstrating working hardware and answering questions from the TA.

Category	Grade
Demonstrating working Stopwatch in Hardware	40
Answering Questions	10

If you are unable to demonstrate working hardware, then try to get as much of the hardware working as possible (for example, maybe the timer and pushbuttons work to control the stopwatch, but you can't finish the part to display the numbers) and show the TA your code to obtain partial credit.