Lab 2: Hardware Manipulation and Timers with the ESP32

Intro to Registers, Timers, PWM, and ADC

ECE/CSE 474
Autumn 2025

Lab Objectives

- Understand and manipulate memory using pointers.
- Control hardware directly through registers.
- Implement timing exercises to manage tasks.
- Interface and control simple peripherals.
- Read and react to analog inputs from sensors.

Agenda for today

- Review:
  - Bitwise Operations
  - Pointers
  - Registers
- Part I: GPIO Registers
- Part II: Clocks and Timers on the ESP32
  - Example Exercise
- Part III: Analog I/O and PWM
- Demo of Part III
- Part IV: Building an Alarm
- Submission Requirements

Review: Bitwise Operations

```
volatile uint8_t reg = 0b00000000;


// set a bit with the OR operator

reg |= (1 << 2); // reg is now 0b00000100


// clear a bit with the AND operator and a negated bitmask

reg &= ~(1 << 2); // reg is back to 0b00000000


// toggle a bit with the XOR operator

reg ^= (1 << 3); // reg is now 0b00001000

reg ^= (1 << 3); // reg is back to 0b00000000
```

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Bitwise left shift
>>	Bitwise right shift

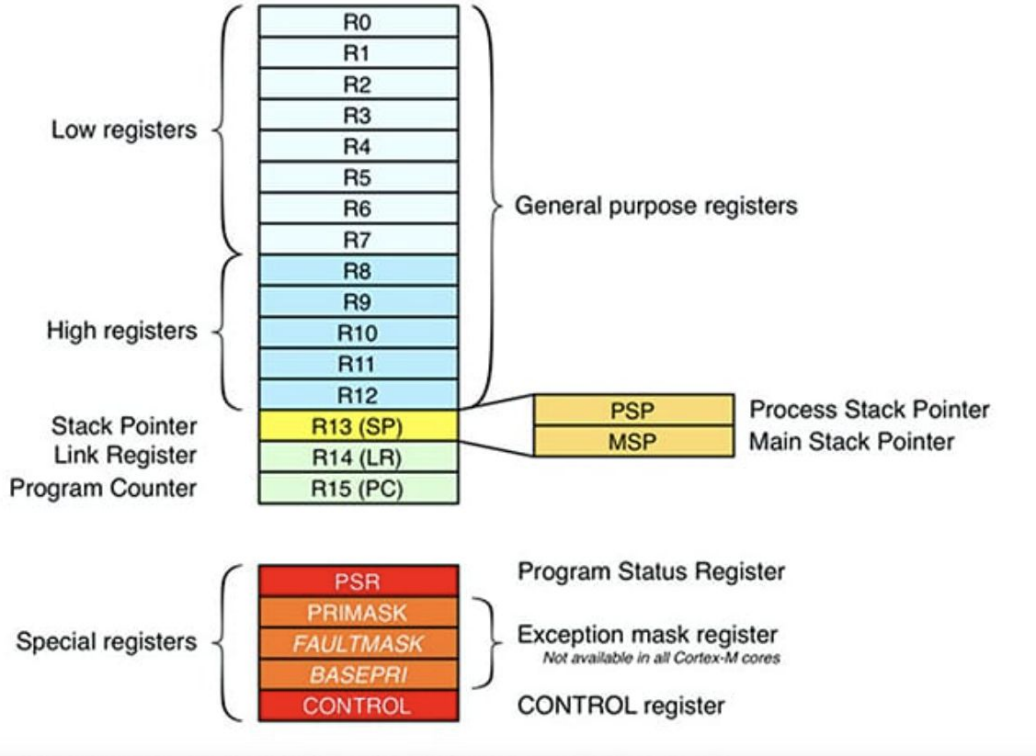
Review: Pointers

Pointers store the address of another variable:

```
int var1 = 10;

double var2 = 10.2;

int* iPtr = &var1; // integer pointer that holds the address in memory of var1

double* dPtr = &var2; // holds the address of var2


// dereference operator * to access the contents stored at the address iPtr points to

int newVar = *iPtr;  // so newVar is equal to 10
```

Review: Registers

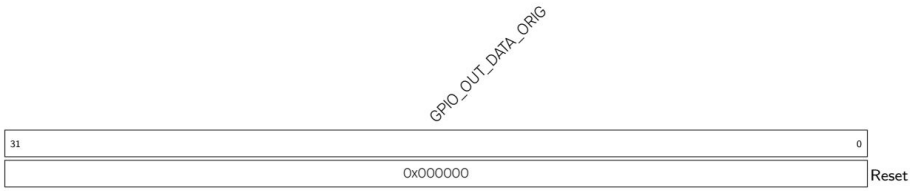
- Small storage locations within a CPU or microcontroller
- Used for temporary data storage and manipulation
- Three main types:
  - General-purpose
  - Special-purpose
  - Status registers



Part I: GPIO Registers

- GPIO_OUT_REG

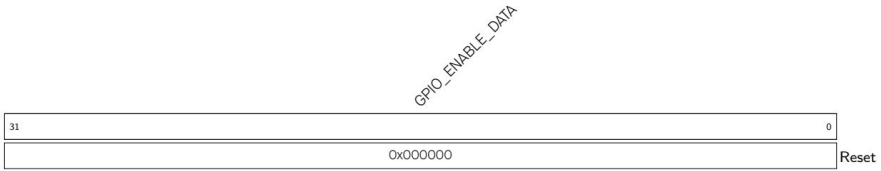
Register 6.2. GPIO_OUT_REG (0x0004)



GPIO_OUT_DATA_ORIG GPIO0 ~ 21 and GPIO26 ~ 31 output values in simple GPIO output mode. The values of bit0 ~ bit21 correspond to the output values of GPIO0 ~ 21, and bit26 ~ bit31 to GPIO26 ~ 31. Bit22 ~ bit25 are invalid. (R/W)

- GPIO_ENABLE_REG

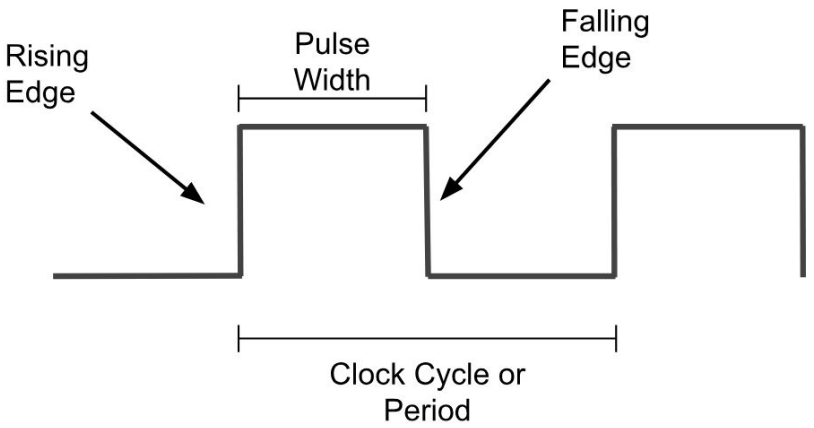
Register 6.9. GPIO_ENABLE_REG (0x0020)



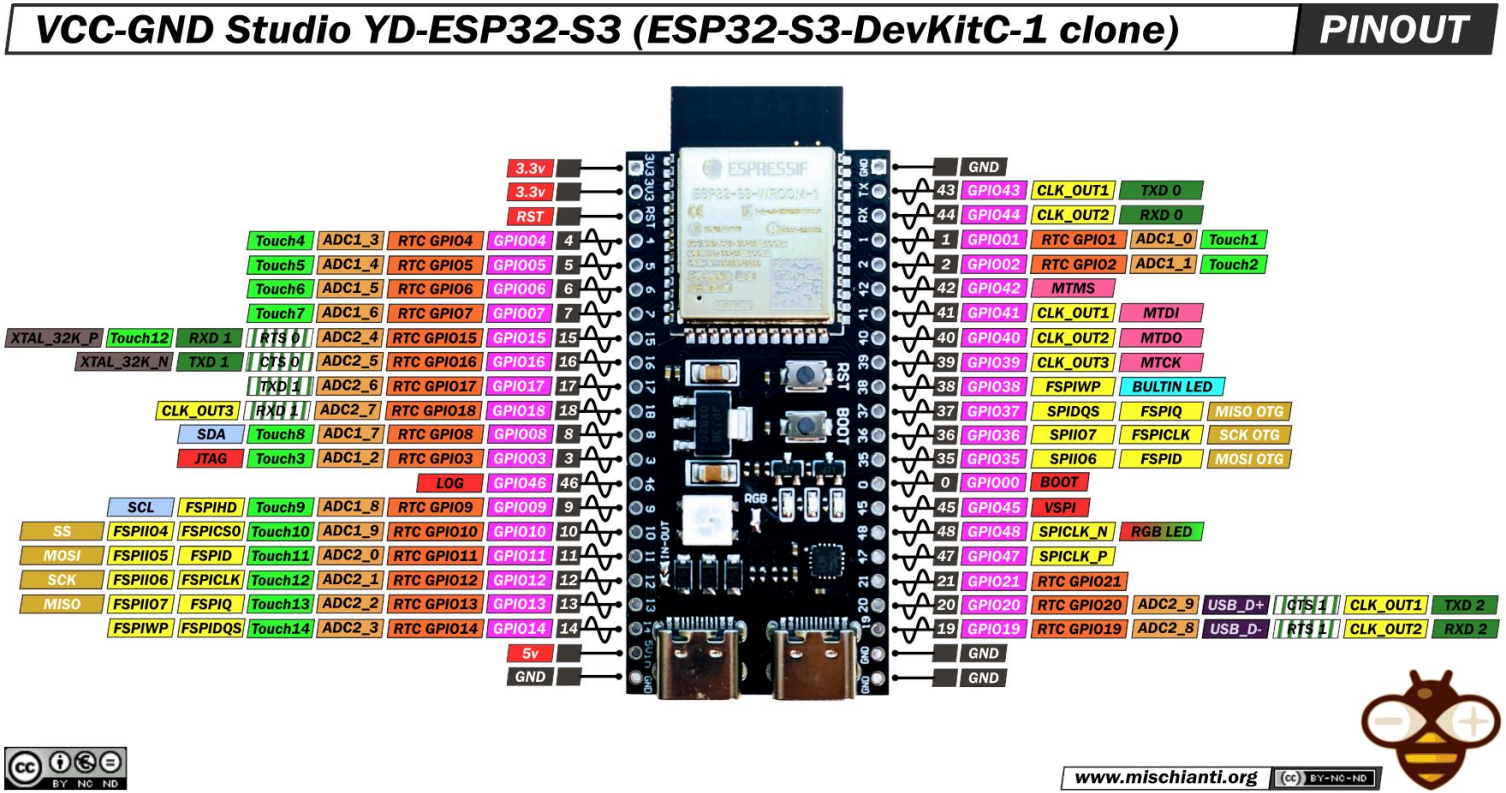
GPIO_ENABLE_DATA GPIO0~31 output enable register. (R/W)

Part II: Clocks on an MCU

- A clock is the “heartbeat” of a microcontroller:
  - Syncs operations
  - Coordinates tasks
- The clock produces a periodic signal (typically a square wave)



Pin mapping


VCC-GND Studio YD-ESP32-S3 (ESP32-S3-DevKitC-1 clone)
PINOUT
3.3v
3.3v
RST
Touch4 ADC1_3 RTC GPIO4 GPIO004 4
Touch5 ADC1_4 RTC GPIO5 GPIO005 5
Touch6 ADC1_5 RTC GPIO6 GPIO006 6
Touch7 ADC1_6 RTC GPIO7 GPIO007 7
XTAL_32K_P Touch12 RXD 1 RTS 0 ADC2_4 RTC GPIO15 GPIO015 15
XTAL_32K_N TXD 1 CTS 0 ADC2_5 RTC GPIO16 GPIO016 16
TXD 1 ADC2_6 RTC GPIO17 GPIO017 17
CLK_OUT3 RXD 1 ADC2_7 RTC GPIO18 GPIO018 18
SDA Touch8 ADC1_7 RTC GPIO8 GPIO008 8
JTAG Touch3 ADC1_2 RTC GPIO3 GPIO003 3
LOG GPIO046 46
SCL FSPIHD Touch9 ADC1_8 RTC GPIO9 GPIO009 9
SS FSPII04 FSPICS0 Touch10 ADC1_9 RTC GPIO10 GPIO010 10
MOSI FSPII05 FSPID Touch11 ADC2_0 RTC GPIO11 GPIO011 11
SCK FSPII06 FSPICLK Touch12 ADC2_1 RTC GPIO12 GPIO012 12
MISO FSPII07 FSPIQ Touch13 ADC2_2 RTC GPIO13 GPIO013 13
FSPIWP FSPIDQS Touch14 ADC2_3 RTC GPIO14 GPIO014 14
5v
GND
GND
43 GPIO43 CLK_OUT1 TXD 0
44 GPIO44 CLK_OUT2 RXD 0
1 GPIO001 RTC GPIO1 ADC1_0 Touch1
2 GPIO002 RTC GPIO2 ADC1_1 Touch2
42 GPIO042 MTMS
41 GPIO041 CLK_OUT1 MTDI
40 GPIO040 CLK_OUT2 MTDO
39 GPIO039 CLK_OUT3 MTCK
38 GPIO038 FSPIWP BULTIN LED
37 GPIO037 SPIDQS FSPIQ MISO OTG
36 GPIO036 SPII07 FSPICLK SCK OTG
35 GPIO035 SPII06 FSPID MOSI OTG
0 GPIO000 BOOT
45 GPIO045 VSPI
48 GPIO048 SPICLK_N RGB LED
47 GPIO047 SPICLK_P
21 GPIO021 RTC GPIO21
20 GPIO020 RTC GPIO20 ADC2_9 USB_D+ CTS 1 CLK_OUT1 TXD 2
19 GPIO019 RTC GPIO19 ADC2_8 USB_D- RTS 1 CLK_OUT2 RXD 2
GND
GND
www.mischianti.org

Part II: Clocks on the ESP32

- The ESP32-S3 has a variety of clocks:
  - Fast clocks like the XTAL_CLK (40 MHz)
  - Slow clocks like the RC_SLOW_CLK (136 kHz)
  - etc

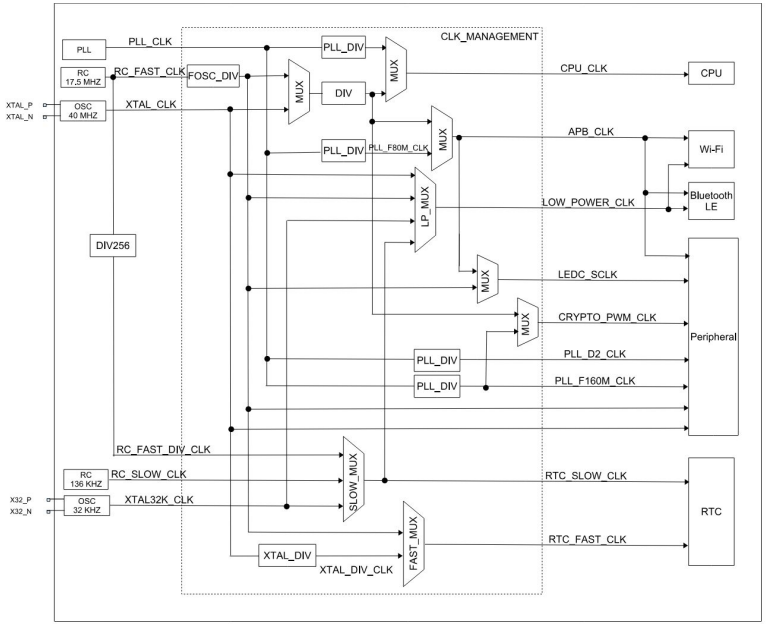


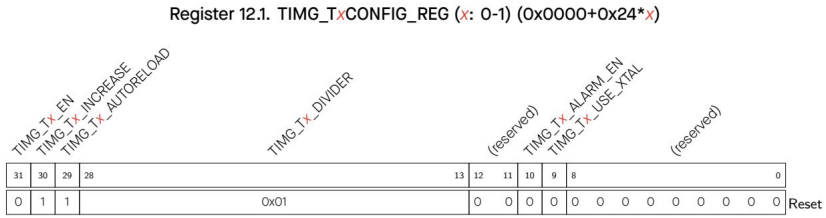
Figure 7-2. Clock Structure

Part II: Timers on the ESP32-S3

- Timers count clock cycles or pulses
  - Used for creating delays, measuring time lengths, performing tasks at regular intervals
- Three main types of timers
  - General Purpose: used for a wide variety of tasks
  - Watchdog timers: used for system reliability
  - RTC: keeps track of actual date and time
- For this lab we will use the general purpose timers
  - Four 54-bit general purpose timers divided into two timer groups (Group 0 and Group 1) with two timers each.
    - 16-bit prescaler
    - Autoreload capable
    - Up/down functionality
    - Programmable alarm

Part II: Timer Registers

- TIMG_T0CONFIG_REG is the configuration register
  - Enable timer, set prescaler, etc
  - The **`TIMG_T0CONFIG_REG(0)`** macro stores the address of this register
- TIMG_T0LO_REG is the lower 32-bits of timer 0.
  - **`TIMG_T0LO_REG(0)`**
- TIMG_T0UPDATE_REG copies the current value of Timer0 to HI/LO
  - **`TIMG_T0UPDATE_REG(0)`**



TIMG_Tx_USE_XTAL 0: Use APB_CLK as the source clock of timer group; 1: Use XTAL_CLK as the source clock of timer group. (R/W)

TIMG_Tx_ALARM_EN When set, the alarm is enabled. This bit is automatically cleared once an alarm occurs. (R/W/SC)

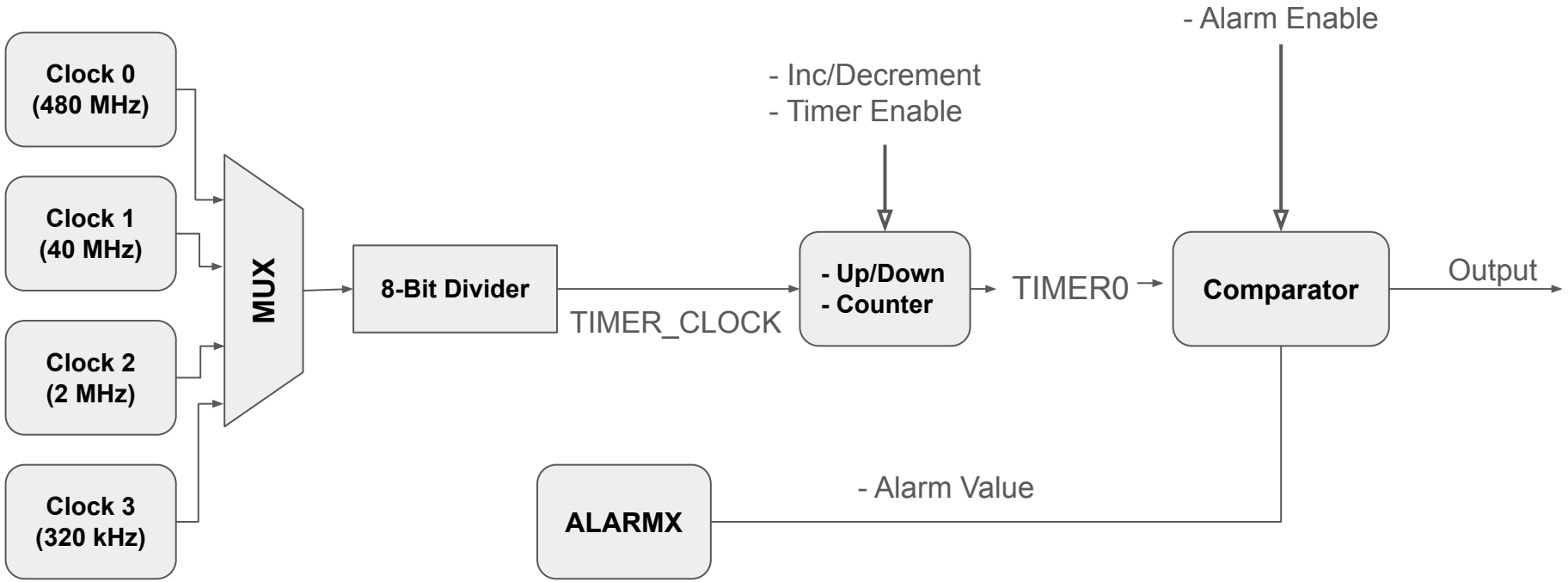
TIMG_Tx_DIVIDER Timer *x* clock (Tx_clk) prescaler value. (R/W)

TIMG_Tx_AUTORELOAD When set, timer *x* auto-reload at alarm is enabled. (R/W)

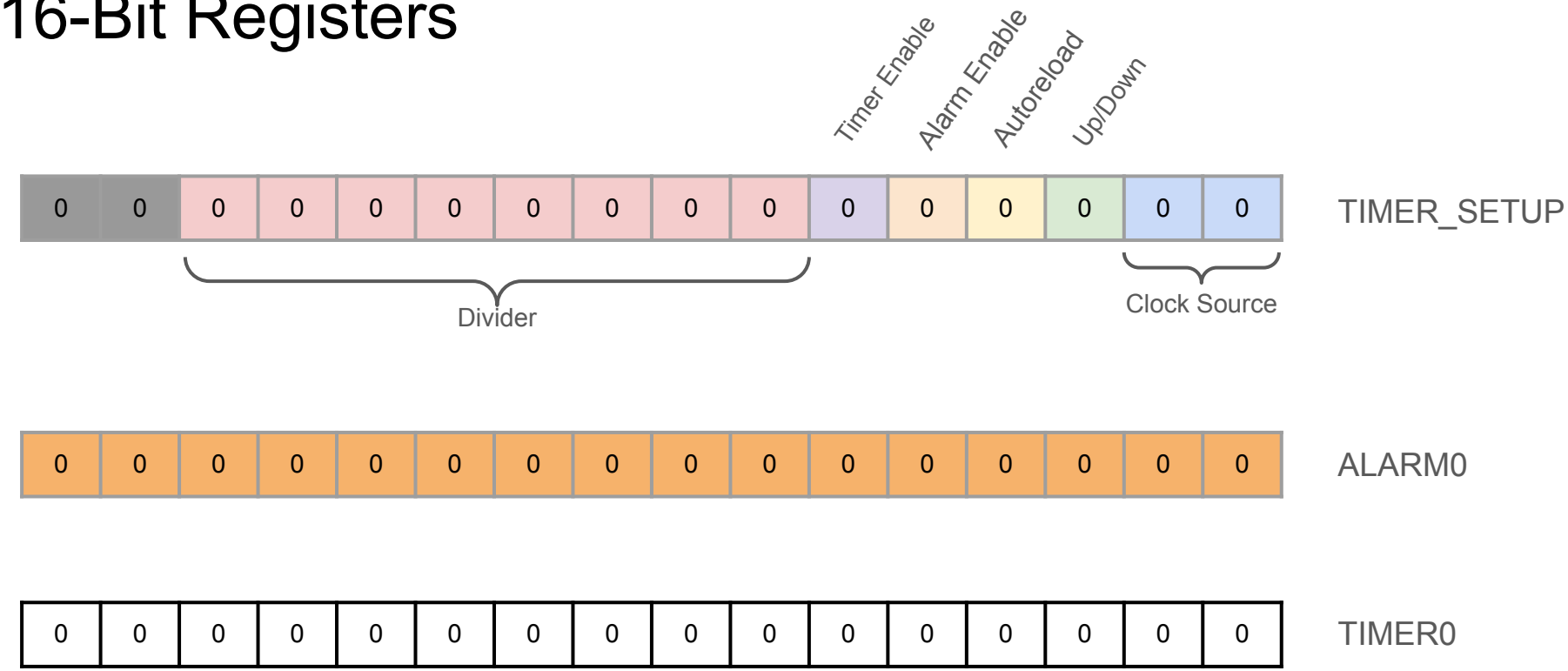
TIMG_Tx_INCREASE When set, the timer *x* time-base counter will increment every clock tick. When cleared, the timer *x* time-base counter will decrement. (R/W)

TIMG_Tx_EN When set, the timer *x* time-base counter is enabled. (R/W)

Example Exercise



16-Bit Registers

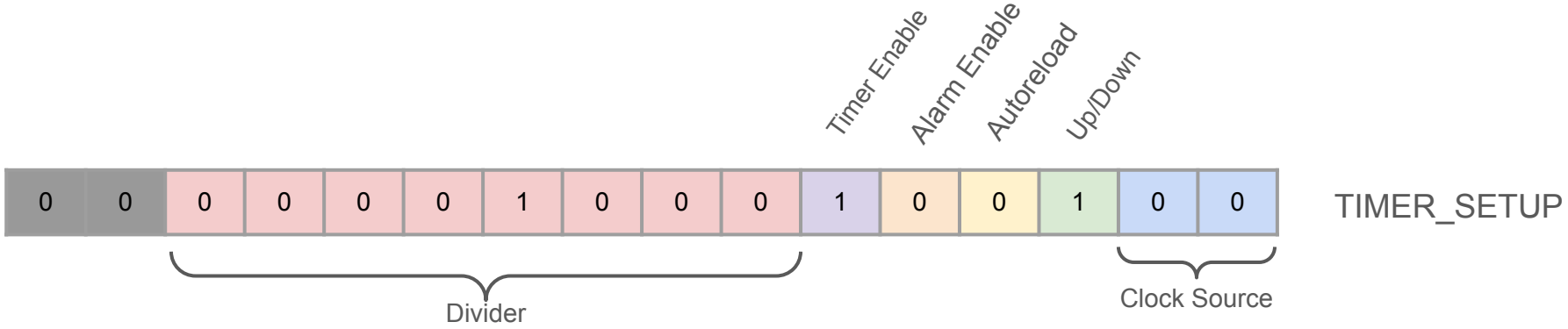


Output Signal

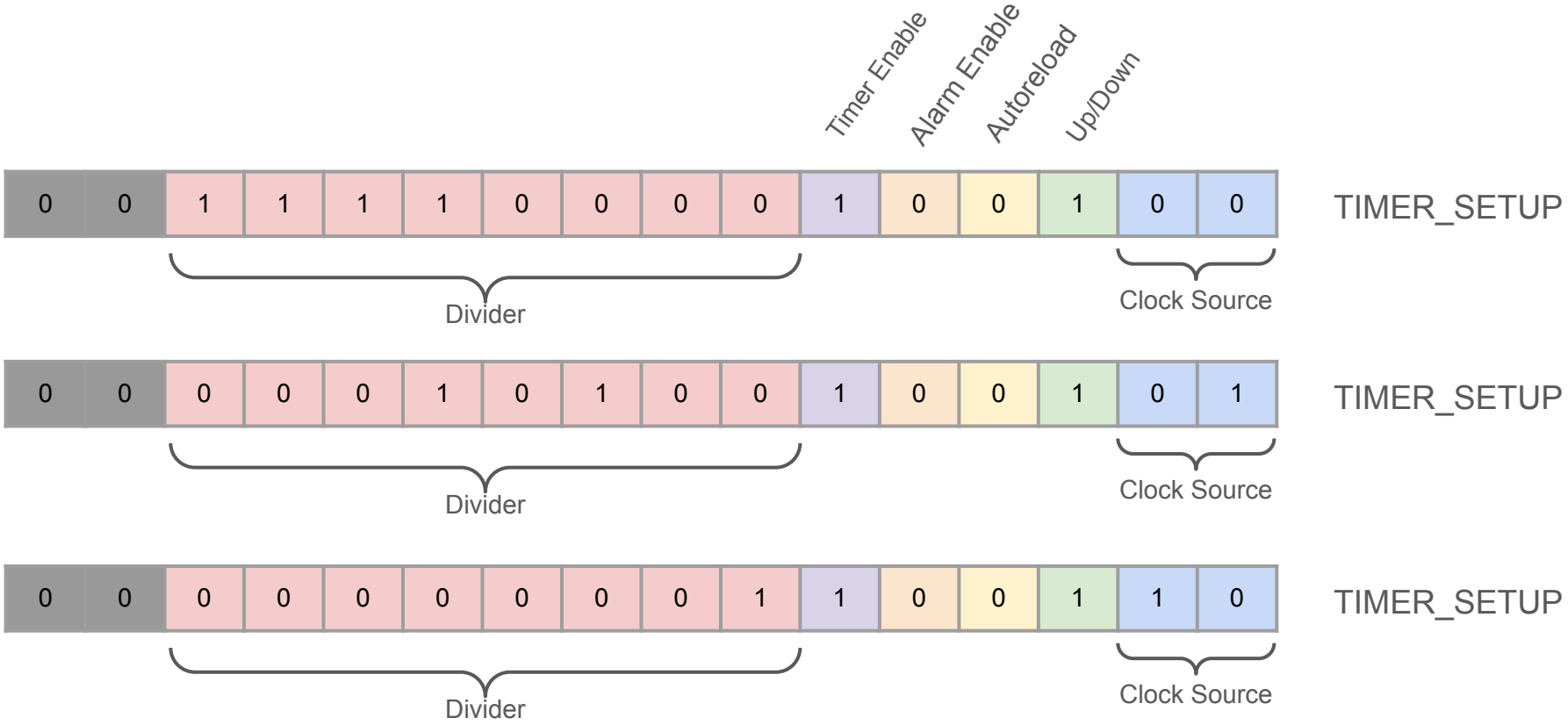
Given our constraints, let’s go through the clock sources, dividers, and flags necessary to achieve a TIMER_CLOCK with:

1. A frequency of 60 MHz
2. A frequency of 2 MHz
3. A frequency of 20 kHz
4. One-shot alarm at 20 seconds

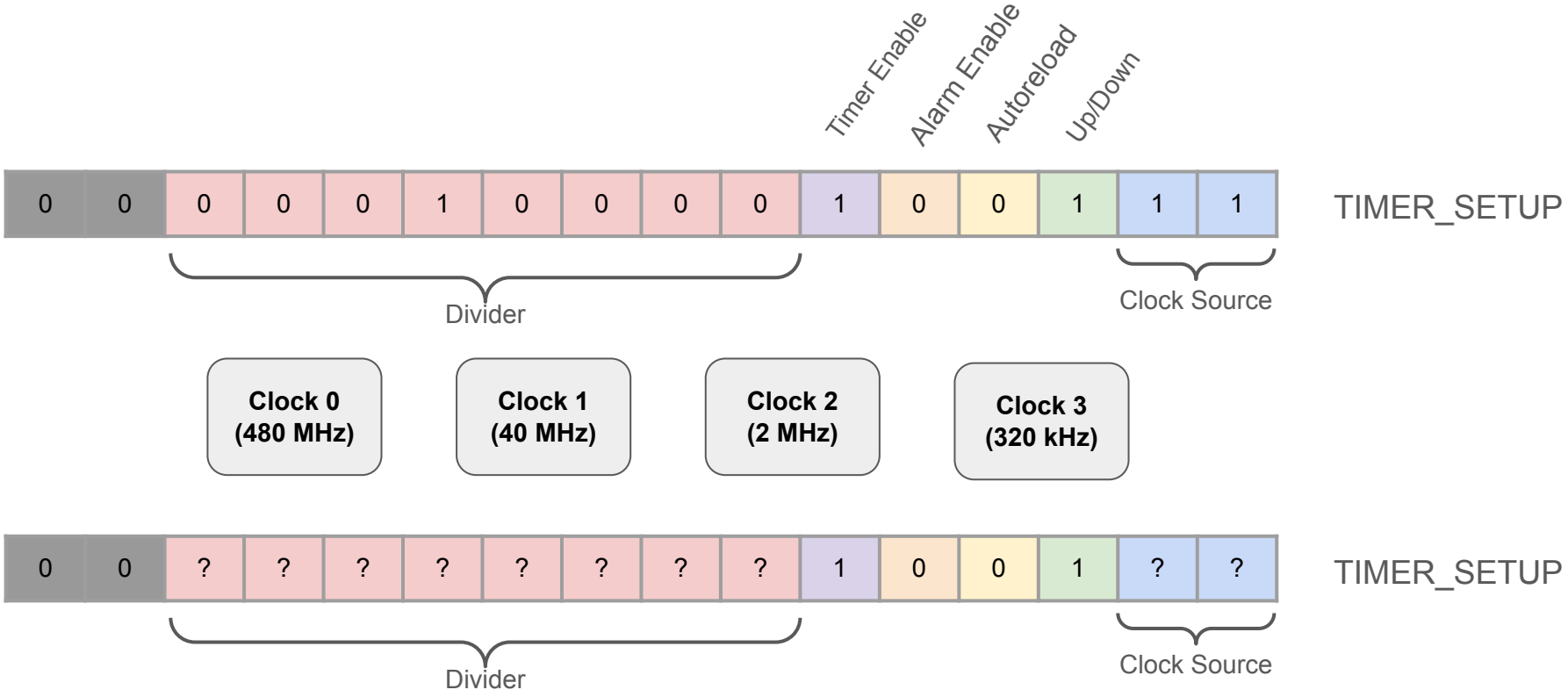
Frequency of 60 MHz



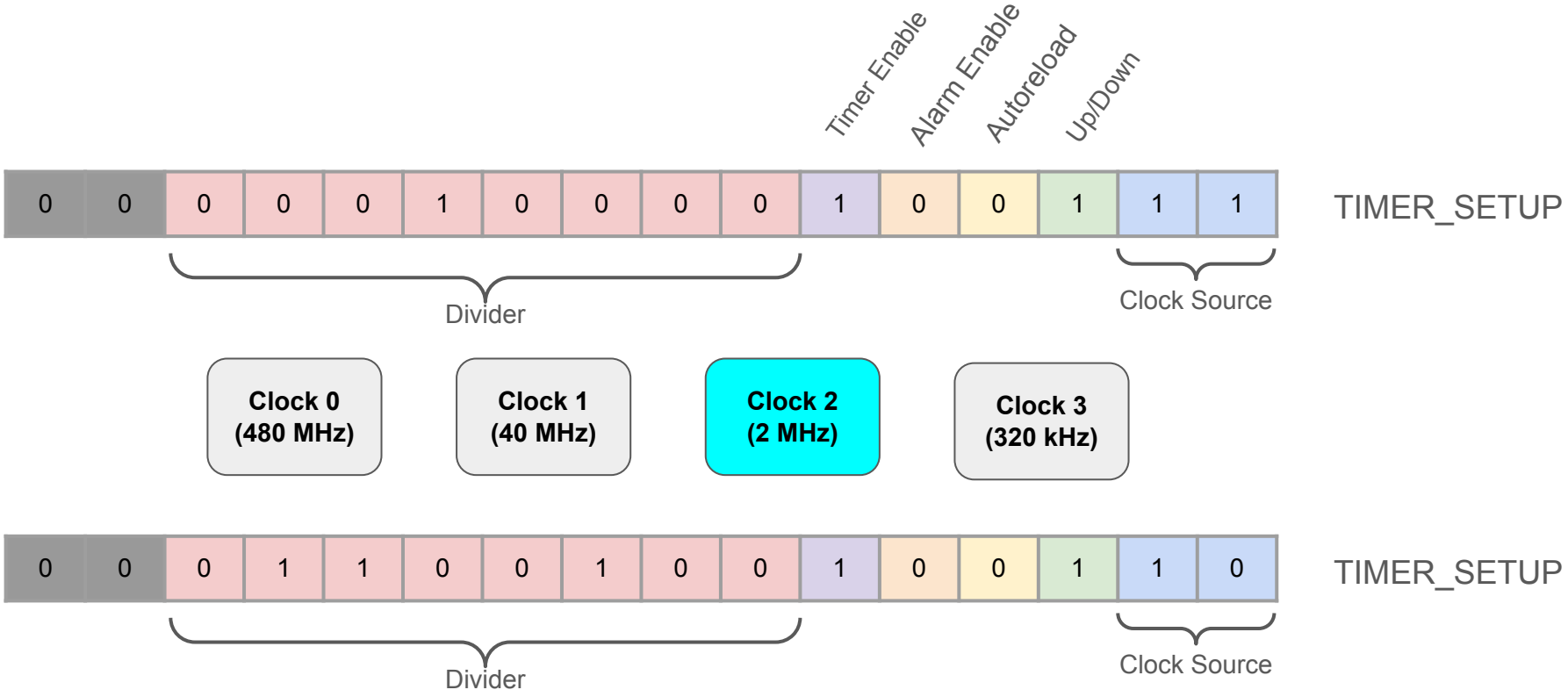
Frequency of 2 MHz



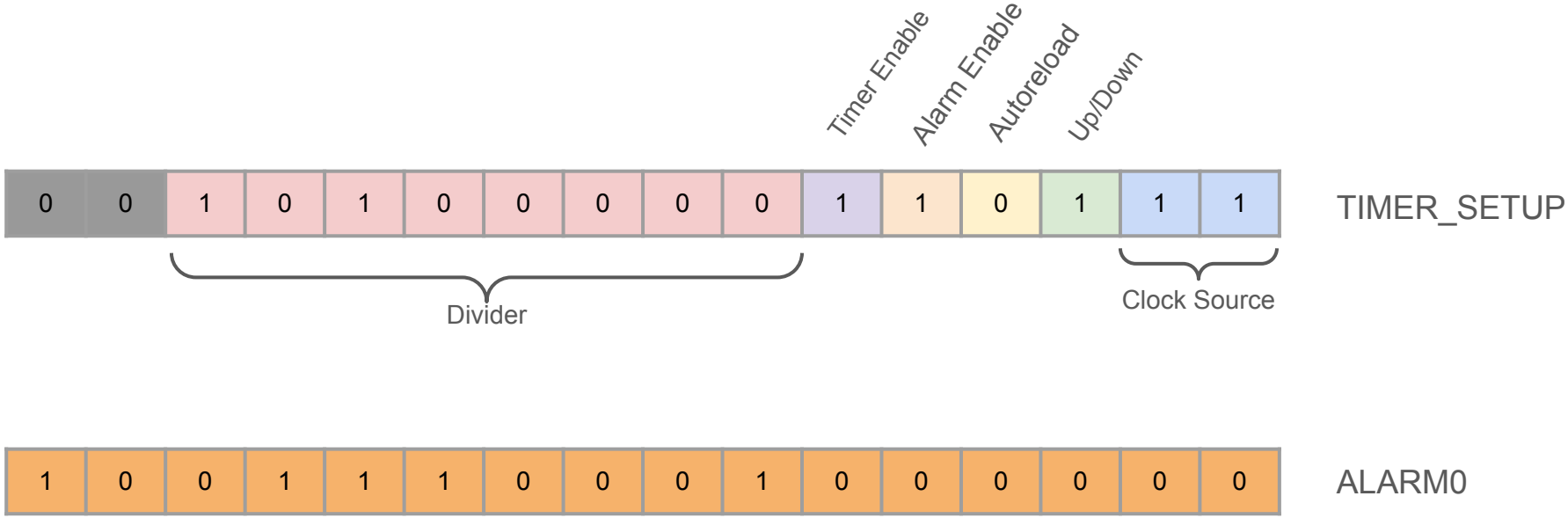
Frequency of 20 kHz



Frequency of 20 kHz

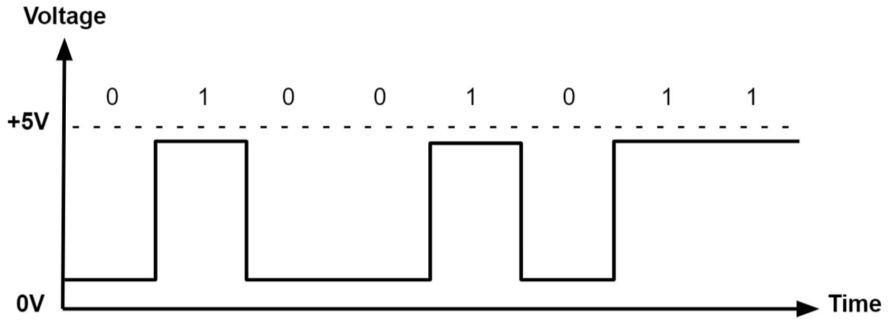


One-Shot Alarm at 20 Seconds

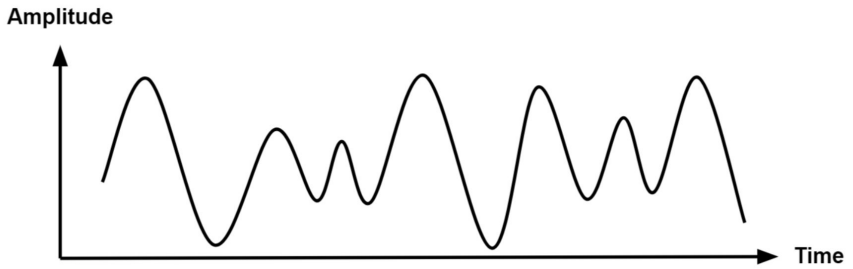


Part III: Analog I/O and PWM

- Digital signal has two states: ON and OFF
- Different peripherals need a range of values to function: motors, LEDs, buzzers, etc.



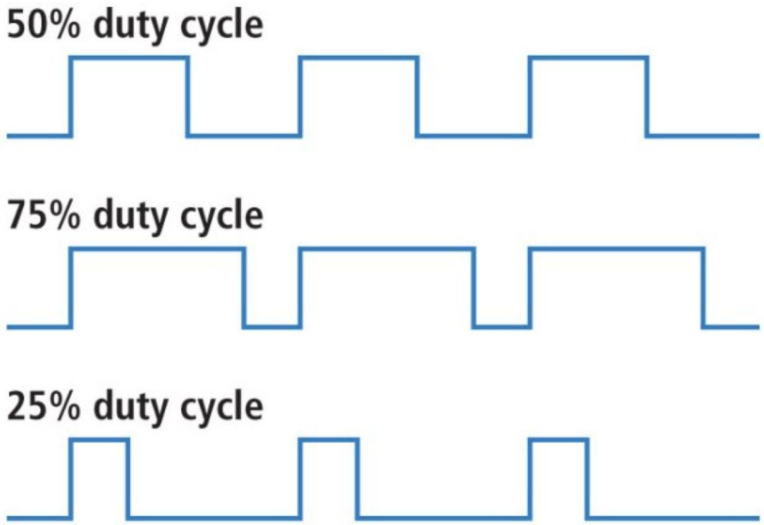
Digital Signal



Analog Signal

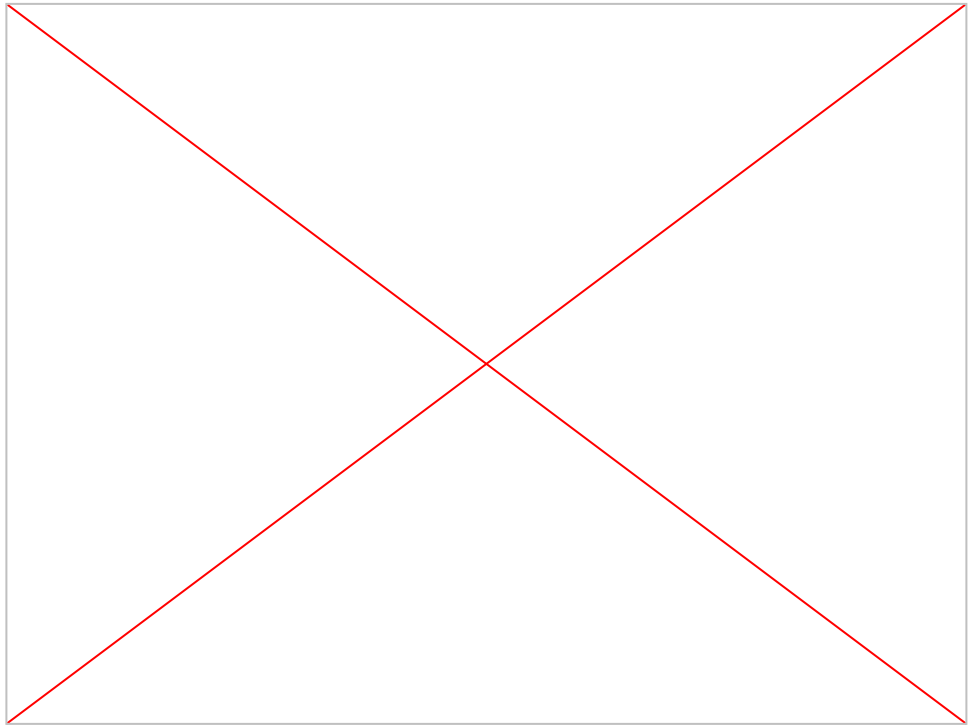
Part III: Pulse Width Modulation (PWM)

- PWM simulates this needed analog signal by by switching a digital signal on and off at a high frequency.
- Key Terms:
  - *Duty Cycle*: % of time the signal is HIGH compared to the total cycle time.
  - *Frequency*: cycles/second.
  - *Resolution*: # of bits used to define the duty cycle.

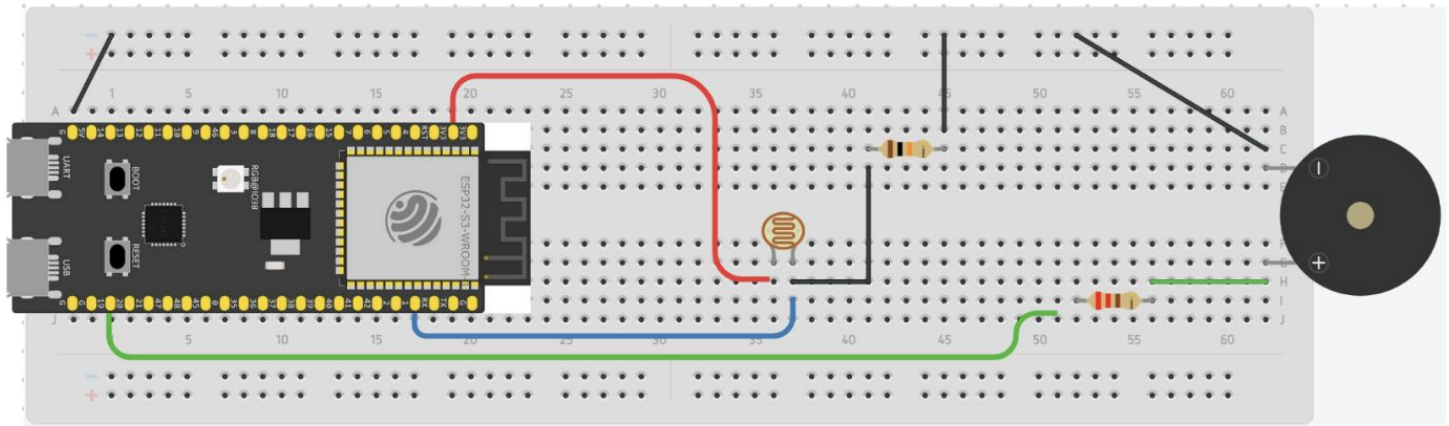


https://learn.sparkfun.com/tutorials/pulse-width-modulation/all

Demo of Part III



Part IV: Building an Alarm



What to submit for Lab 2

Final Deliverables Recap

Live Demos:

- *Part I:* The external LED blinking from Step 1.
- *Part II:* The external LED blinking from Step 3 with live oscilloscope measurements.
- *Part III:* The LED responding to changes in ambient lighting from Step 4.
- *Part IV*: The passive buzzer producing a sequence of sounds from Step 5.

Code:

NOTE: Please follow the Code Guidelines when documenting your code.

The sketch files for the following sections should be descriptively named (e.g. Lab2Part2.ino), zipped together in a single folder containing your last name(s) and the lab number (e.g. LastNameLab2.zip), and submitted through Canvas:

- *Part I:*
  - Step 1: The external LED blinking from direct register access.
  - Step 2: Comparing times between library functions and direct register access.
- *Part II:* The external LED blinking with direct register access from step 3.
- *Part III:* The LED responding to changes in ambient lighting from step 4.
- *Part IV*: The passive buzzer producing a sequence of sounds from Step 5.

Report:

- Follow the outline and instructions listed in the Lab Report Template.
- Make sure to go over the following in the discussion section:
  - Using your measured times from Step 2, compare and contrast the advantages and disadvantages of library functions vs direct register access.