Sec 4. General Purpose Timer

Timing control and PWM

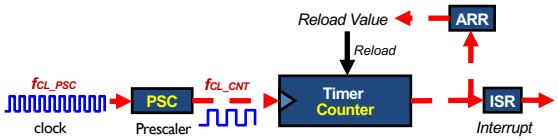
1

Timer

- Free-run counter (independent of processor)
- Functions
  - Input capture
  - Output compare
  - Pulse-width modulation (PWM) generation
  - One-pulse mode output

2

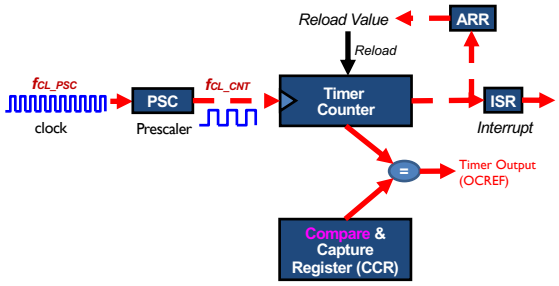
Timer: Clock



$$f_{CK\_CNT} = \frac{f_{CL\_PSC}}{PSC + 1}$$

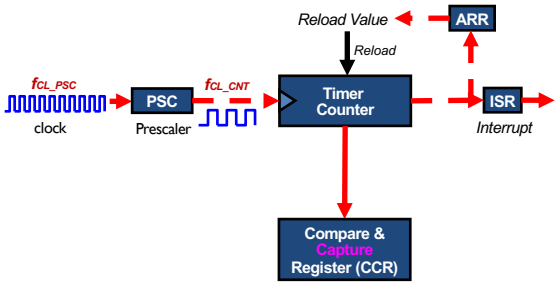
3

Timer: Output



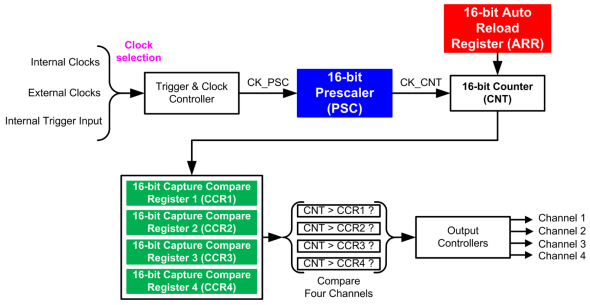
4

Timer: Input Capture



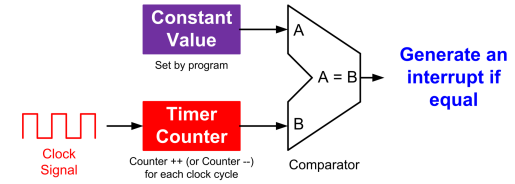
5

Multi-Channel Outputs



6

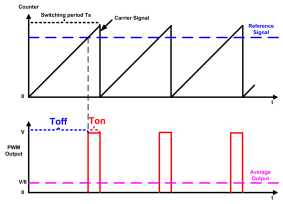
Output Compare



Output Compare Mode (OCM)	Timer Output (OCREF)
000	Frozen
001	High if CNT == CCR
010	Low if CNT == CCR
011	Toggle if CNT == CCR
100	Forced low (always low)
101	Forced high (always high)

7

PWM Mode

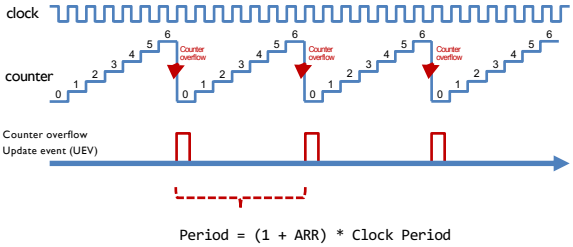


Mode	Counter < Reference	Counter ≥ Reference
PWM mode 1 (Low True)	Active	Inactive
PWM mode 2 (High True)	**Inactive**	**Active**

8

Edge-aligned Mode (Up-counting)

ARR = 6, RCR = 0

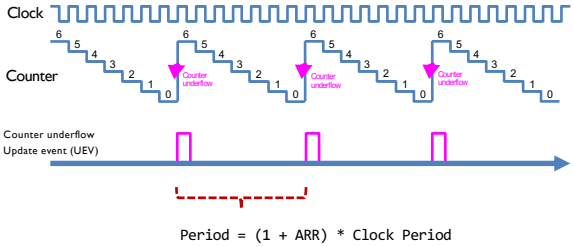



```
Period = (1 + ARR) * Clock Period
       = 7 * Clock Period
```

9

Edge-aligned Mode (down-counting)

ARR = 6, RCR = 0

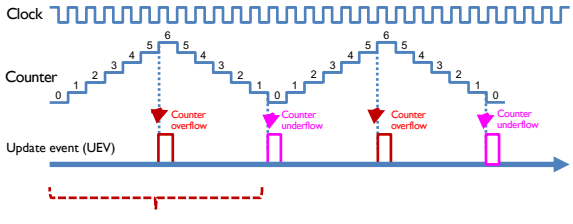



```
Period = (1 + ARR) * Clock Period
       = 7 * Clock Period
```

10

Center-aligned Mode

ARR = 6, RCR = 0



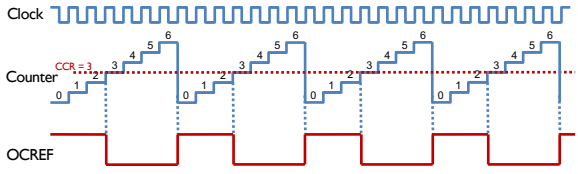

```
Period = 2 * ARR * Clock Period
       = 12 * Clock Period
```

11

PWM Mode 1 (Low-True)

Mode 1
Timer Output = { High if counter < CCR
Low if counter ≥ CCR

Upcounting mode, ARR = 6, CCR = 3, RCR = 0



$$\text{Duty Cycle} = \frac{CCR}{ARR + 1} = \frac{3}{7}$$

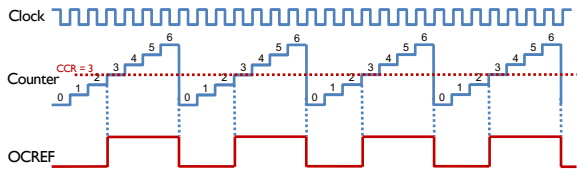
```
Period = (1 + ARR) * Clock Period
       = 7 * Clock Period
```

12

PWM Mode 2 (High-True)

Mode 2 Timer Output = { Low if counter < CCR; High if counter ≥ CCR }

Upcounting mode, ARR = 6, CCR = 3, RCR = 0



Duty Cycle = $1 - \frac{CCR}{ARR + 1} = \frac{4}{7}$

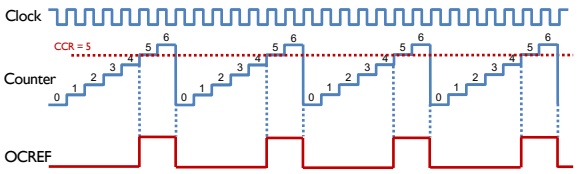
Period = (1 + ARR) * Clock Period = 7 * Clock Period

13

PWM Mode 2 (High-True)

Mode 2 Timer Output = { Low if counter < CCR; High if counter ≥ CCR }

Upcounting mode, ARR = 6, CCR = 3, RCR = 0



Duty Cycle = $1 - \frac{CCR}{ARR + 1} = \frac{2}{7}$

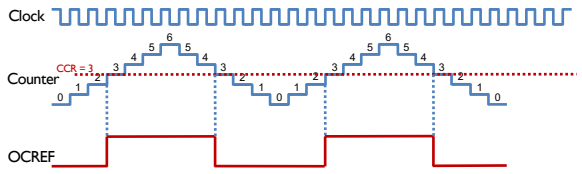
Period = (1 + ARR) * Clock Period = 7 * Clock Period

14

PWM Mode 2 (High-True)

Mode 2 Timer Output = { Low if counter < CCR; High if counter ≥ CCR }

Center-aligned mode, ARR = 6, CCR = 3, RCR = 0



Duty Cycle = $1 - \frac{CCR}{ARR} = \frac{1}{2}$

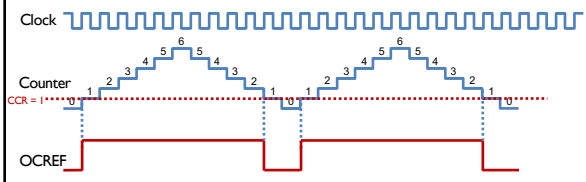
Period = 2 * ARR * Clock Period = 12 * Clock Period

15

PWM Mode 2 (High-True)

Mode 2 Timer Output = { Low if counter < CCR; High if counter ≥ CCR }

Center-aligned mode, ARR = 6, CCR = 3, RCR = 0



Duty Cycle = $1 - \frac{CCR}{ARR} = \frac{5}{6}$

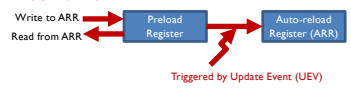
Period = 2 * ARR * Clock Period = 12 * Clock Period

16

Auto-Reload Register (ARR)

- Auto-Reload Preload Enable (ARPE) bit in TIMx_CR1

ARPE = 1 (Syn Update)



If UDIS bit in TIMx_CR1 is 1, UEV event is disabled.

ARPE = 0 (Asyn Update)

17

PWM Output Polarity

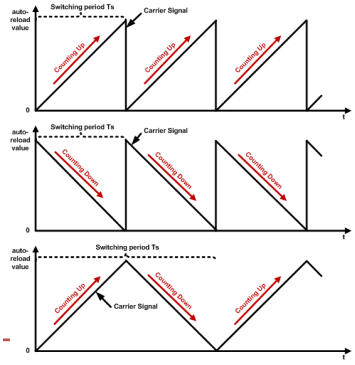
Mode	Counter < CCR	Counter ≥ CCR
PWM mode 1 (Low True)	Active	Inactive
PWM mode 2 (High True)	Inactive	Active

Output Polarity:
- Software can program the CCxP bit in the TIMx_CCER register

	Active	Inactive
Active High	High Voltage	Low Voltage
Active Low	Low Voltage	High Voltage

18

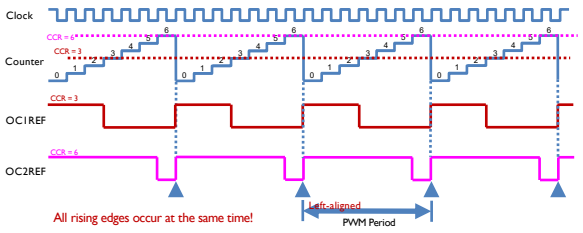
Counting up, down, center



19

Up-Counting: Left Edge-aligned

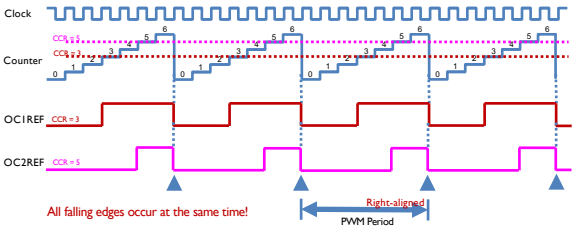
Upcounting mode, ARR = 6, CCR = 3, RCR = 0



20

PWM Mode 2: Right Edge-aligned

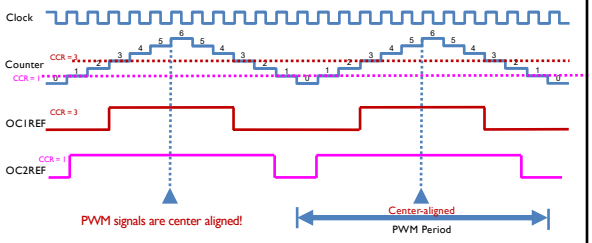
Upcounting mode, ARR = 6, CCR = 3, RCR = 0



21

PWM Mode 2: Center Aligned

Center-aligned mode, ARR = 6, CCR = 3, RCR = 0



22

The devil is in the detail

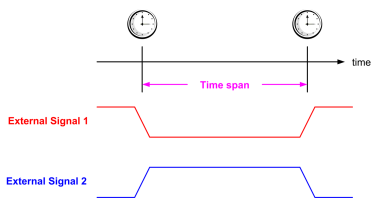
- Timer output control
- Enable Timer Output
  - MOE: Main output enable
  - OSSI: Off-state selection for Idle mode
  - OSSR: Off-state selection for Run mode
  - CCxE: Enable of capture/compare output for channel x
  - CCxNE: Enable of capture/compare complementary output for channel x

Control bits					Output states[1]	
MOE bit	OSSI bit	OSSR bit	CCxE bit	CCxNE bit	OCx output state	OCxN output state
1	X	X	0	0	Output disabled (not driven by the timer: Hi-Z) OCx=0, OCxN=0	
		0	0	1	Output disabled (not driven by the timer: Hi-Z) OCx=0	OCxREF + Polarity OCxN = OCxREF xor CCxNP
			0	1	OCxREF + Polarity OCx=OCxREF xor CCxP	Output Disabled (not driven by the timer: Hi-Z) OCxN=0
			1	1	OCREF + Polarity + dead time	Complementary to OCREF (not OCREF) + Polarity + dead-time
		1	0	1	Off-State (output enabled with inactive state) OCx=CCxP	OCxREF + Polarity OCxN = OCxREF x or CCxNP
			1	0	OCxREF + Polarity OCx=OCxREF xor CCxP	Off-State (output enabled with inactive state) OCxN=CCxNP
0	0	X	X	X	Output Disabled (not driven by the timer. Hi-Z) OCx=CCxP; OCxN=CCxNP	
	1		0	0		
			0	1	Off-State (output enabled with inactive state) Asynchronously: OCx=CCxP, OCxN=CCxNP (if BRK or BRK2 is triggered). Then (this is valid only if BRK is triggered), if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state (may cause a short circuit when driving switches in half-bridge configuration). **Note**: BRK2 can only be used if OSSI = OSSR = 1.	
			1	0		
			1	1		

23

Input Capture

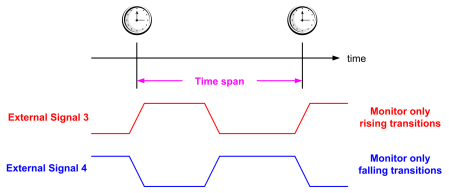
- Monitor both rising and falling edge



24

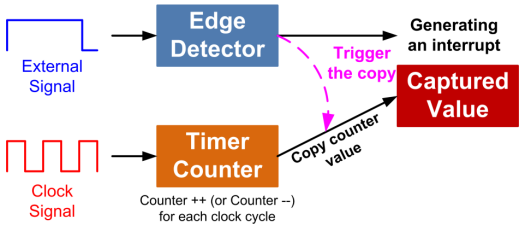
Input Capture

- Monitor only rising edges or only falling edge



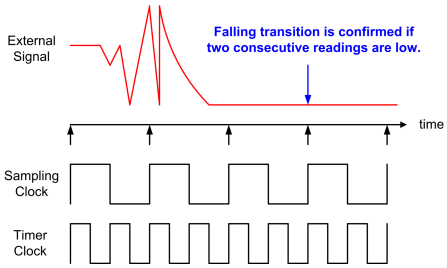
25

Input Capture



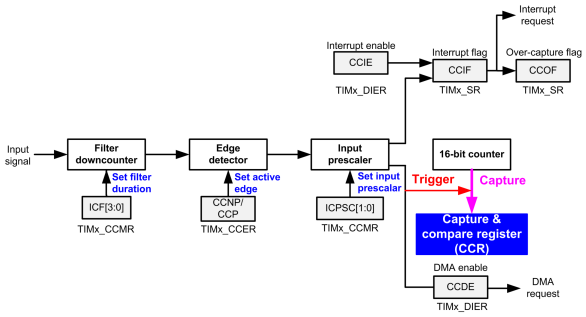
26

Input Filtering



27

Input Capture Diagram



28

Ultrasonic Distance Sensor

$$Distance = \frac{Round\ Trip\ Time \times Speed\ of\ Sound}{2}$$

$$= \frac{Round\ Trip\ Time(\mu s)}{58}$$

29

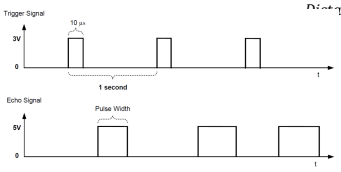
Ultrasonic Distance Sensor

The echo pulse width corresponds to round-trip time.

$$Distance\ (cm) = \frac{Pulse\ Width\ (\mu s)}{58}$$

or

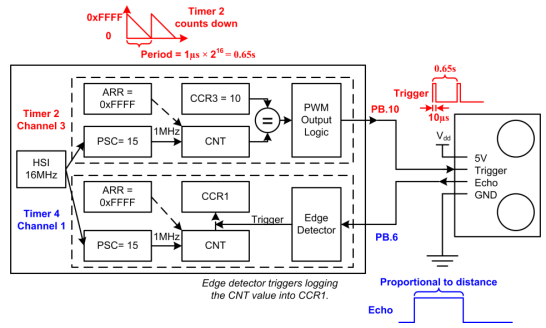
$$Distance\ (inch) = \frac{Pulse\ Width\ (\mu s)}{148}$$



If pulse width is 38ms, no obstacle is detected.

30

Ultrasonic Distance Sensor



Real-Time Clock (RTC)

- RTC is a digital clock that provides calendar time and date.
- Typical requirements:
  - Low power consumption
  - Separately powered by a battery
  - Accurate
  - Run independently from the processor core

UNIX Epoch Time

- Definition: number of seconds that have elapsed since 00:00:00 UTC, Thursday, 1 January 1970
- Example:
  - Converting 2:07:39am, April 21, 2014 (UTC) to Unix Epoch number

UNIX Epoch Number

$$= 16181 \text{ days} \times \frac{\text{seconds}}{\text{day}} + 2 \text{ hours} \times \frac{\text{seconds}}{\text{hour}} + 7 \text{ minutes} \times \frac{\text{seconds}}{\text{minutes}} + 39$$

$$= 16181 \times 86400 + 2 \times 3600 + 7 \times 60 + 39$$
$$= 1398046059$$
$$= 0x53547D6B$$

Note a day has 86400 seconds ($24 \times 60 \times 60 = 86400$)

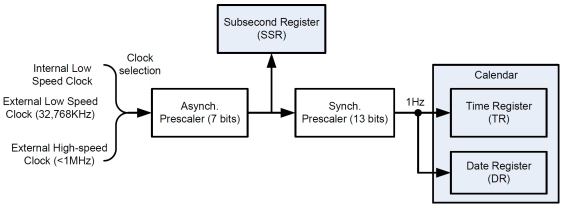
UNIX Epoch Time

- Use a signed 32-bit integer to hold the UNIX Epoch Time
  - Covers a time span of 136 years.
  - Minimum representable time is `1901-12-13`
  - Maximum representable time is `2038-01-19`
- Year 2038 Problem (also called Unix Millennium Bug)
  - The second after 03:14:07 UTC 2038-01-19 is an overflow (which became 1901-12-13).
  - Use a signed 64-bit integer to fix the problem
  - A challenge in embedded systems

Crystal Inaccuracy: PPM

- Parts Per Million (PPM) = 10^{-6}
- Crystal Inaccuracy
  - 1 PPM ⟶ ±1.1 seconds per year
  - A typical watch crystal has 20 PPM
    - Error per day: 86400 seconds × 20 × 10^{-6} = 1.728 seconds/day
    - Error per month: 30 days × 1.728 seconds/day = 51 seconds/month
- STM32 RTC
  - At 25°C, HSI and MSI have an accuracy of 100 ppm (not accurate enough for RTC)
  - Need to use external LSE crystal, typically 32.768 kHz (2^{15} Hz)

Frequency Setting



$$f_{1Hz} = \frac{f_{RTC}}{(Asynch\_Prescaler + 1) \times (Synch\_Prescaler + 1)}$$

Frequency Setting

If f_{RTC} is 32.768 kHz, *i.e.* 2^{15} Hz, then $Asynch\_Prescaler$ is 2^7-1, *i.e.* 127, and $Synch\_Prescaler$ is set as 2^8-1, *i.e.* 255, in many applications, as shown below.

$$f = \frac{f_{RTC}}{(Asynch\_Prescaler + 1) \times (Synch\_Prescaler + 1)}$$
$$= \frac{2^{15}}{(127 + 1) \times (255 + 1)}$$
$$= \frac{2^{15}}{2^7 \times 2^8}$$
$$= 1Hz$$

37

Initial Setup of Raspberry

- Connect SD card directly to Pi that is connected to a monitor through HDMI (Don't use SD card reader to format SD card, OS is already preloaded)
- Change PassWD to all up case letters
  - RASPBERRY
- WiFi Connection
  - UID = Embedded_sytem; Passwd: lab440-picar
- Enable SSH: SSH (secure shell)
  - Preference→Raspberry Pi Config→Interface
- Shut Down the Raspberry Pi
- Take SD card out

38

Download Programs to Pi

- Insert the SD card to your Pi Car
- Acquire IP address
- Use Putty to connect and operate PiCar
- Download the Pi car program
- Follow the instructions in Section 3.2 to download all the programs of the Pi car (~30 mins)

- Add the RPi Car program to auto-start. Make sure Exec path is correct.

- Auto-start Pi Car
- Client Server

39