CprE 288 – Introduction to Embedded Systems (Output Compare and PWM)

Instructors:

Dr. Phillip Jones

Announcements

- Exam 1: Making grading adjustments
- Exam 2: Week 12 (need to finalize if it will be on Tue or Thur)
  - i.e. Week of 11/5
- Quiz 7: Thursday 10/18 (Same FULL readings as Quiz 6)
- HW 6: Sunday 10/21
- Project Proposal
  - Revision 1: Monday 10/22
    - Problem Statement: HW3 User research flow for developing Project's problem statement (see rubric for grading criteria
    - Prototype: Sketch of usage of cybot, and worksheet mapping Application functionality to Cybot
  - Revision 2: Monday 10/29
- Servo motor lab

Overview of Today's Lecture

- Output Compare and Pulse Wave Modulation (PWM)
  - Textbook reading: Section 9.1, 9.2

Output Compare and PWM

Output Compare is used to Generate certain digital waveforms for control purposes

Recall Input Capture: Recognizes digital waveforms

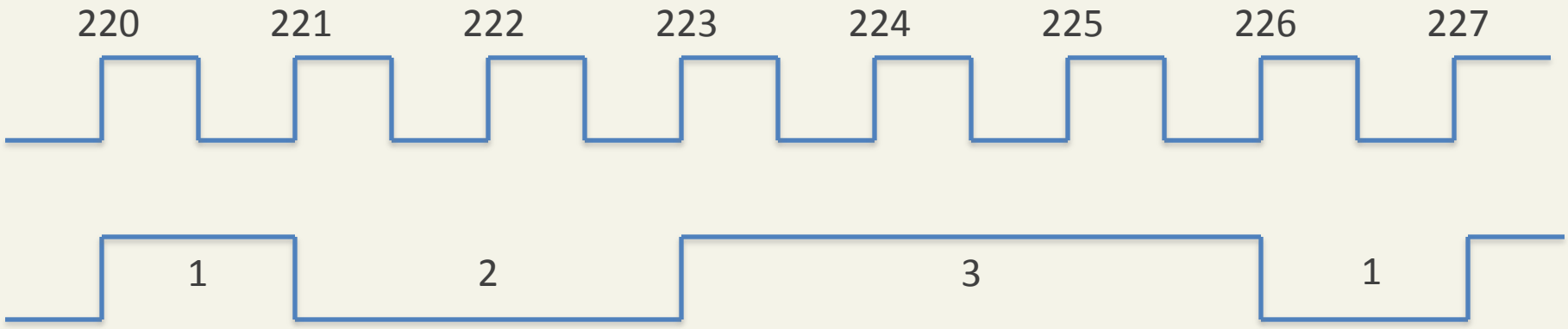
Many applications in microcontroller applications:

- Start analog devices
- Control speed of motors
- Control power output rate
- Communications
- Control servo (lab 8)

Output Compare

Example: Generate a waveform that is 1-cycle high, 2-cycle low, 3-cyle high, 1-cycle low, and repeating

The MCU may generate output events (transitions) at 220 (current time), 221, 223, 226, 227 and so on with initial state as low



Output Compare: Design Principle

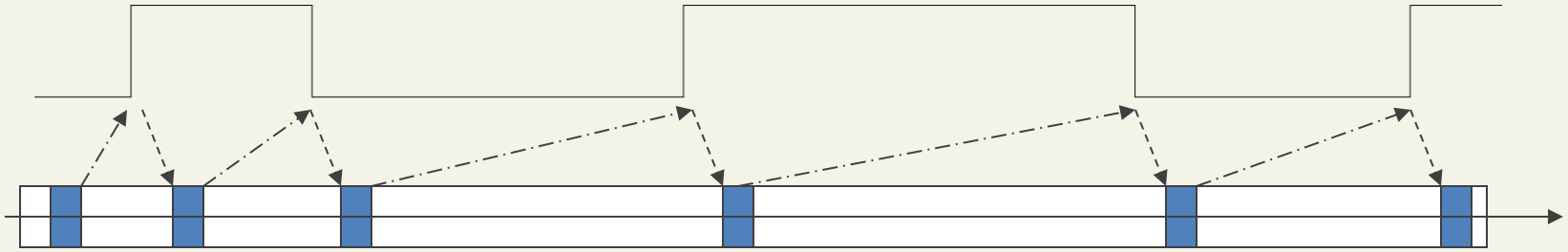
Time is important

How could a microcontroller generate events at precise time intervals?

- Use time delay functions?
  - CPU cannot do anything else
  - Not accurate
- Use interrupts?
  - Not necessarily accurate, because of interrupt overhead and possibly delays by other interrupts

Output Compare: Example

Solution: Preset the time of each event!



GPTMTnR=TOP

CPU sets next event time

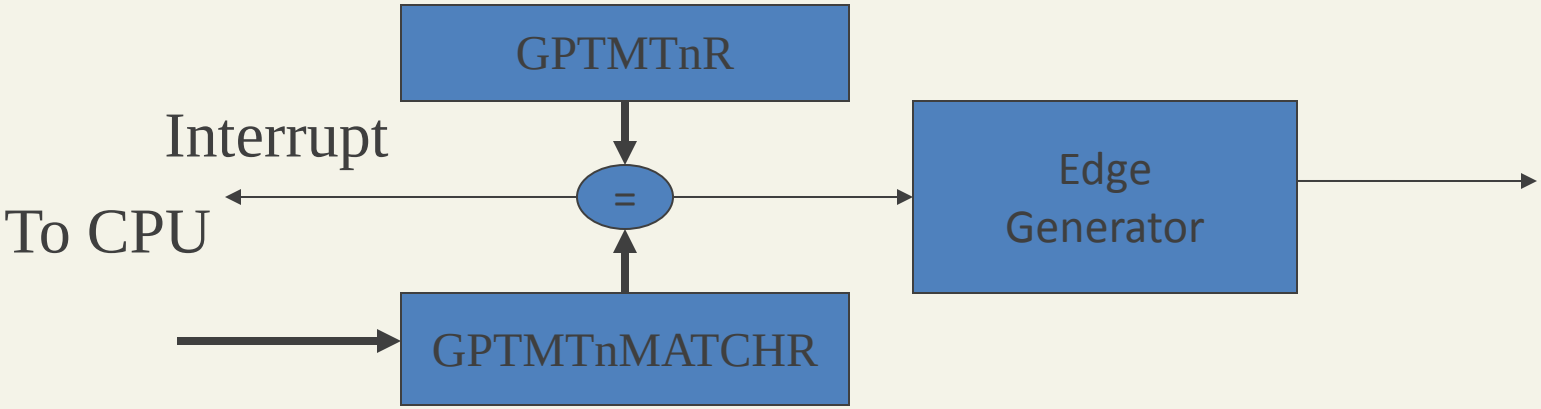
Interrupt to CPU

CPU Interrupt processing

CPU Foreground computation

Output Compare: Design Principle

Time value (clock count) is set first by the CPU and then used by the output compare unit



GPTMTnRn: Timer/Counter

GPTMTnMATCHR : Output Compare Register

Under what condition does the hardware generate the precise waveform?

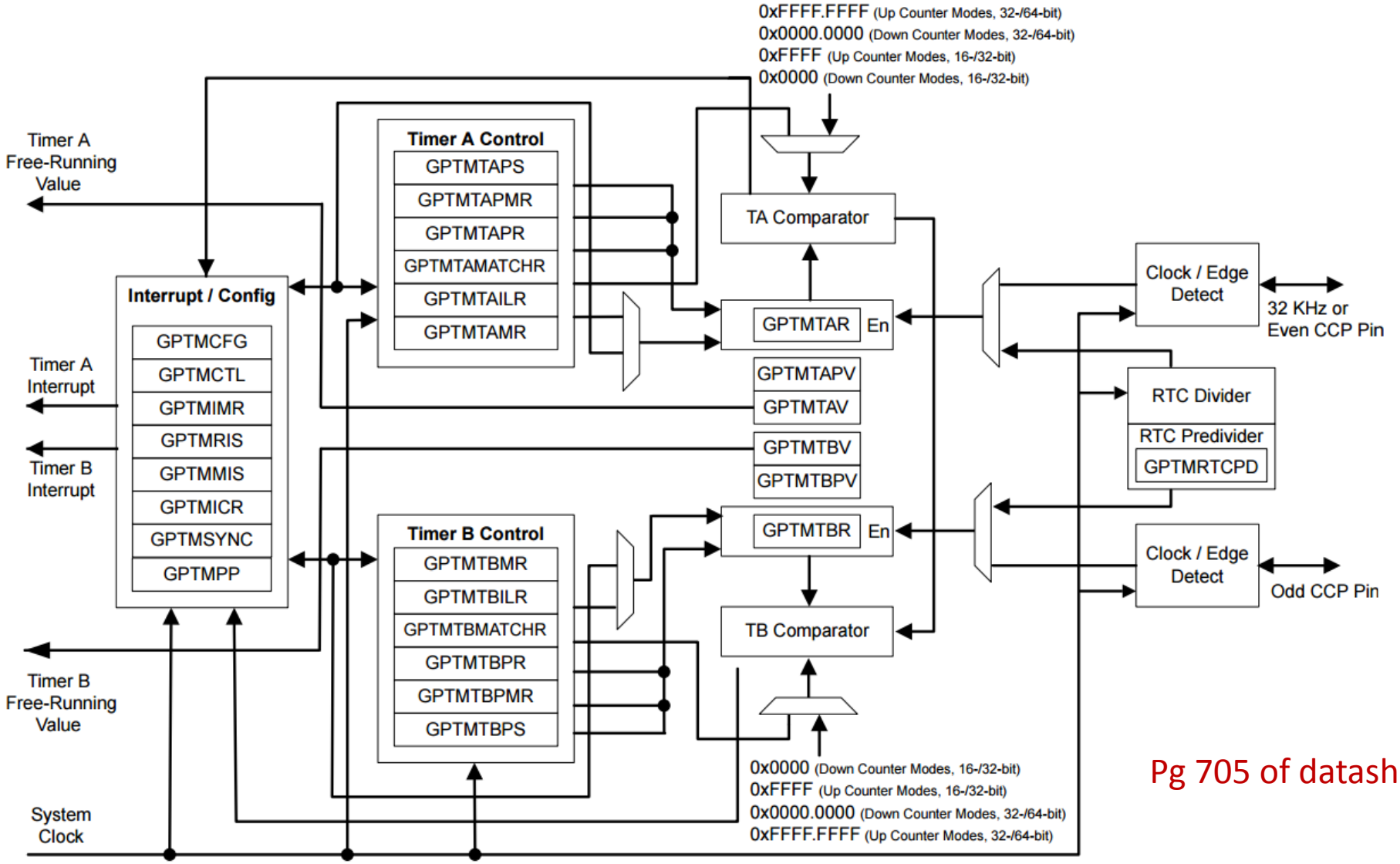
A: The CPU is not overloaded by interrupt processing

TM4C123G Timer module

- 6 general purpose 16/32-bit timer blocks
- 6 wide general purpose 32/64-bit timer blocks
- 11 timer modes
- **2 independent matching units per block (A / B)**
- Pulse width modulation output
- Many other features

***Note that there is also a separate PWM module

Timer Block Diagram



Pg 705 of datasheet

TM4C123G 16/32-bit Timer/Counter

- Timer 16/32 bit
  - two 16bit timers (A & B) or single 32bit (A)
  - 6 Channels (0 – 5)
    - 11 modes
      - One shot
      - Periodic
      - Periodic Snapshot
      - Wait-for-Trigger
      - Real-Time Clock
      - Input Edge Count
      - Input Edge Time
      - PWM (one shot or periodic)
      - DMA
      - Synchronizing GP-Timer Blocks
      - Concatenated Modes

Lab 8 Important Timer Modes

- **Periodic mode**
  - **Count down** to zero from a preset value
    - Single shot stops timer once zero is reached
    - Periodic will reset timer with initial value and start again
  - Can set interrupt to fire when timer reaches end value or /and match value
  - Also can configure to start at zero and count to specified value (up counter)
  - Periodic also has a snapshot mode (pg 698 in book)

Though this mode can generate wave forms there is a lot of overhead (interrupts)

Lab 8 Important Timer Modes

- **PWM mode**
  - 24 bit or 48 bit count down counter
    - Prescale register used to increase size and is **not** used as prescale divisor
    - 16bits + 8bits in the prescale register
    - 32bits + 16bits in prescale register
  - Allows for creation of periodic or one shot output square wave
  - Handles switching of output (off/on) for the developer (**no need for interrupts**)

Options for Generating a Waveform

- Generic waveform generation
  - use Timer in **Periodic Mode**,
  - an ISR has to set the new values to the MATCH and GPIO register
  - can generate any arbitrary waveform
  - There is CPU overhead executing the ISR
- PWM waveform generation (**Used for Lab 8**)
  - use Timer in **PWM Mode:** the Output Compare (OC) hardware generates a PWM waveform without CPU involvement
  - can only easily generate a PWM waveform
  - no CPU overhead, since ISRs are not required

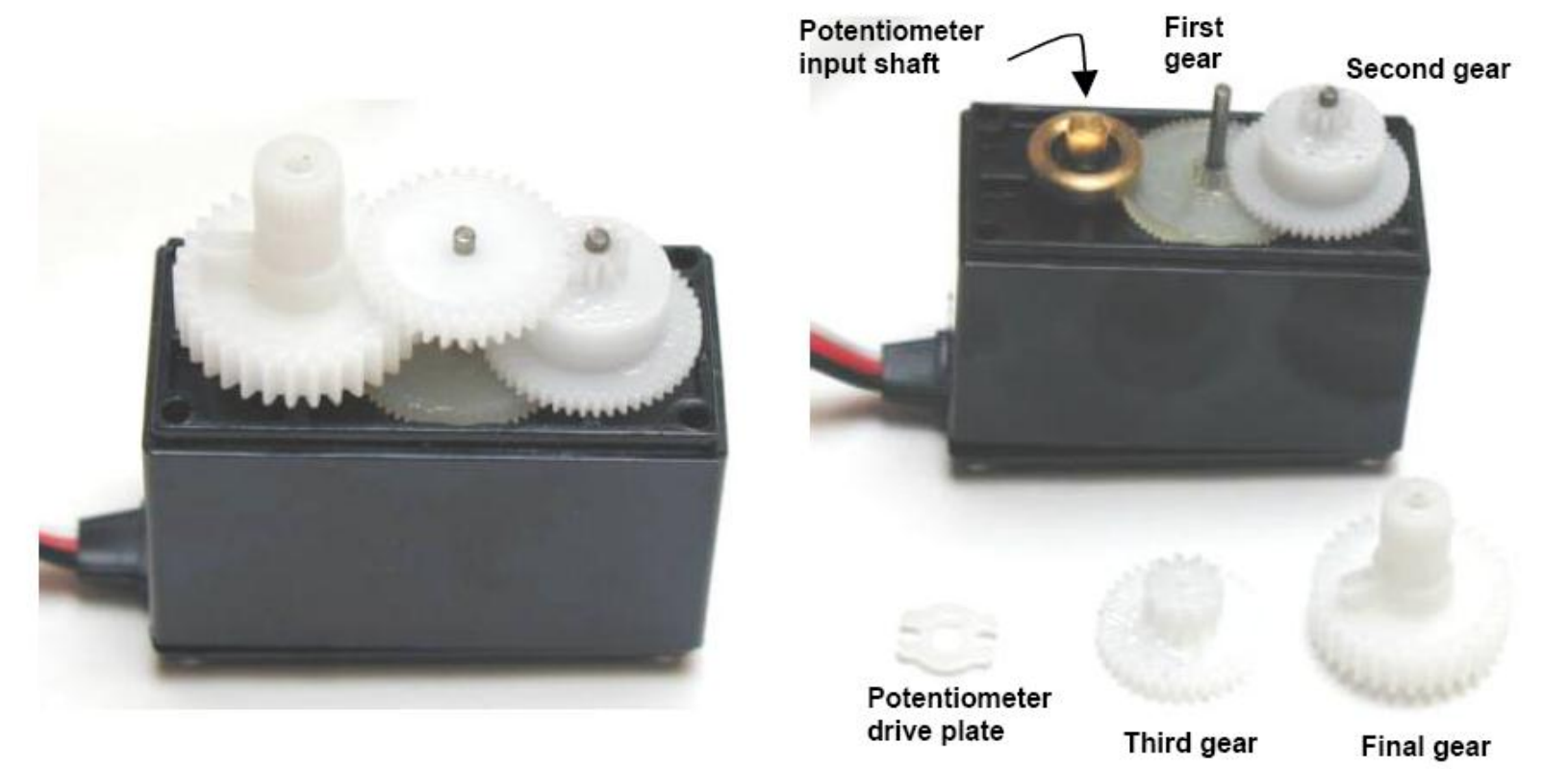
Servo Control

A **servo** is a special motor with built-in position feedback

- Can stop the shaft at a given position
- Relatively precise
- **Needs calibration**

Servo Control

The potentiometer plays as position sensor (see the next two slides)



Source: Parallax Robotics Student guide, V1.4

Servo Control

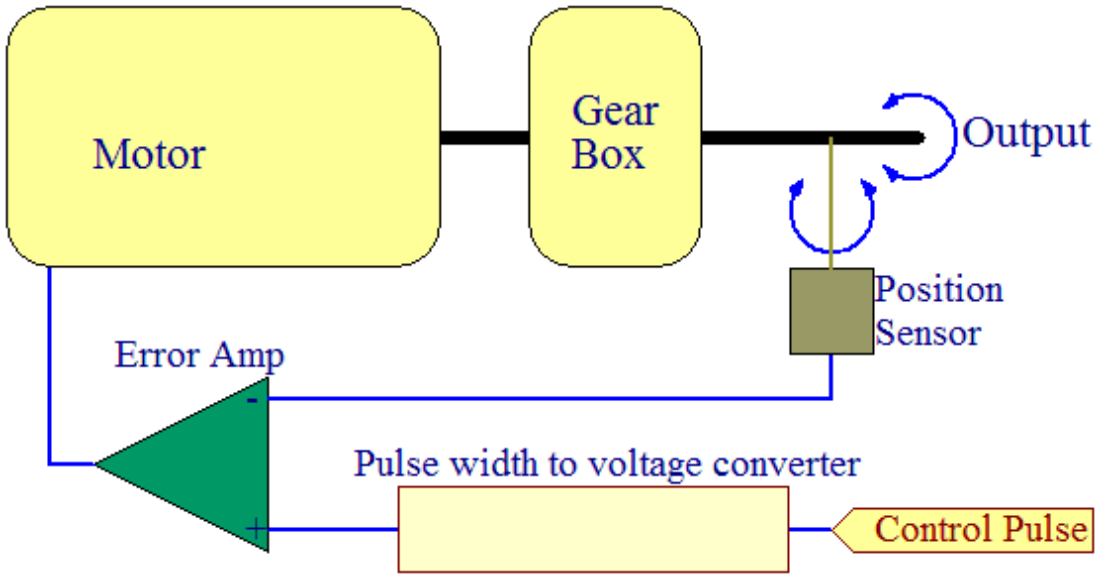
http://en.wikipedia.org/wiki/Potentiometer

- Potentiometer: Three-terminal resistor with a sliding **mid contact**
- In the servo, the motor rotates the shaft that slides the mid contact
- The voltage at the mid contact provides feedback to the power circuits driving the motor

Servo Control

Control feedback loop

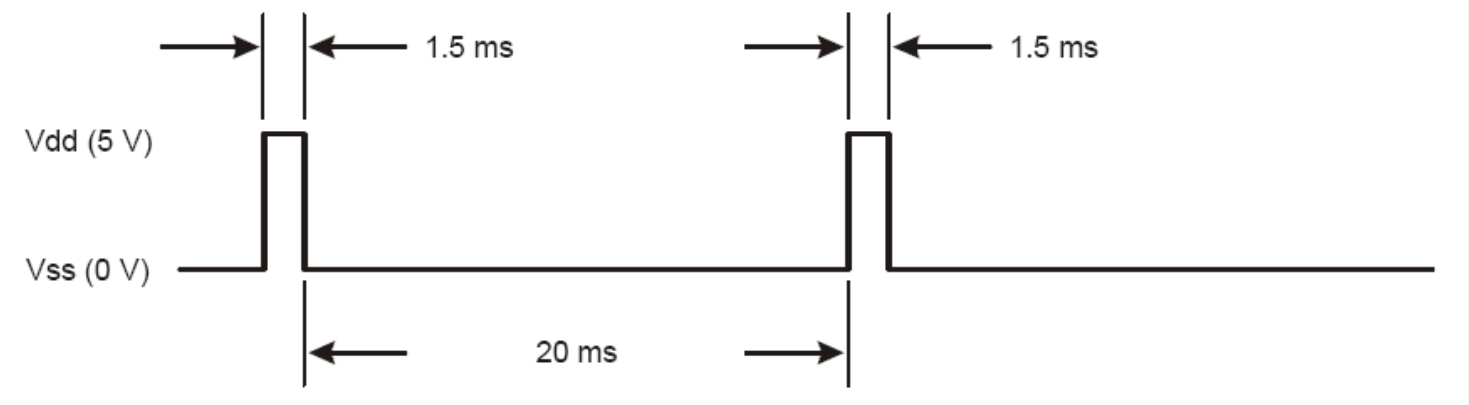
- A control pulse is converted to a target voltage
- If the servo is not at the **target angular position**, there will be a **error** between the target voltage and the mid contact voltage
- The voltage error is amplified to drive the motor in opposite direction of the error, until the error reaches zero



Servo Control

The Servo input: Periodic digital waveform of pulses

- The pulse width decides the target voltage (a property of the circuit inside the servo)
- The pulses must be separated by a time between 10ms and 40ms
- How does your program generate the waveform?



- This is a form of periodic PWM waveform: Pulse width = 1.5ms, pulse period = 21.5ms

Servo Control

In Lab 8, your program should be able to

1. Send pulses to the servo to make it move
2. Make the servo stop at the center position
3. Make the servo stop at different degrees of angle, do calibration

   0, 45, 90 (center), 135, 180

Servo Control

Use digital waveform to inform the servo the target position

If the servo is **ideal**:

- 1ms pulse – clockwise far end
- 1.5ms pulse – center position
- 2ms pulse – counterclockwise far end
- 10-40ms interval between pulses (doesn't have to be precise)

Must repeat until shaft arrives the target position

*The **actual servos require calibration***

Servo Control

Programming tasks: Generate a periodic waveform with a certain pulse width and a fixed period

1.0~2.0ms corresponds to 0~180 degree counterclockwise

Suggested by servo's document. Again, ***calibration IS necessary***

Pulse Width Modulation = PWM

Parameters: Period Length and Pulse Width

Duty Cycle = Pulse width / Period Length

Programming: How to set the two parameters?

PWM

Two parameters in PWM programming

- Pulse width: by writing to the Match Register
- Pulse period: by writing a TOP value. lower 16bits to the Interval Load Register and the higher 8 bits to the Timer Prescale register.

How does the hardware work in Timer/Counter *n*

- GPTMT*n*R decrements every cycle
- The first **match event** occurs when GPTMT*n*R = GPTMT*n*MATCHR
- The second match event occurs when GPTMT*n*R is reset (after it reaches 0 and is reset to TOP)

PWM – REMINDER!

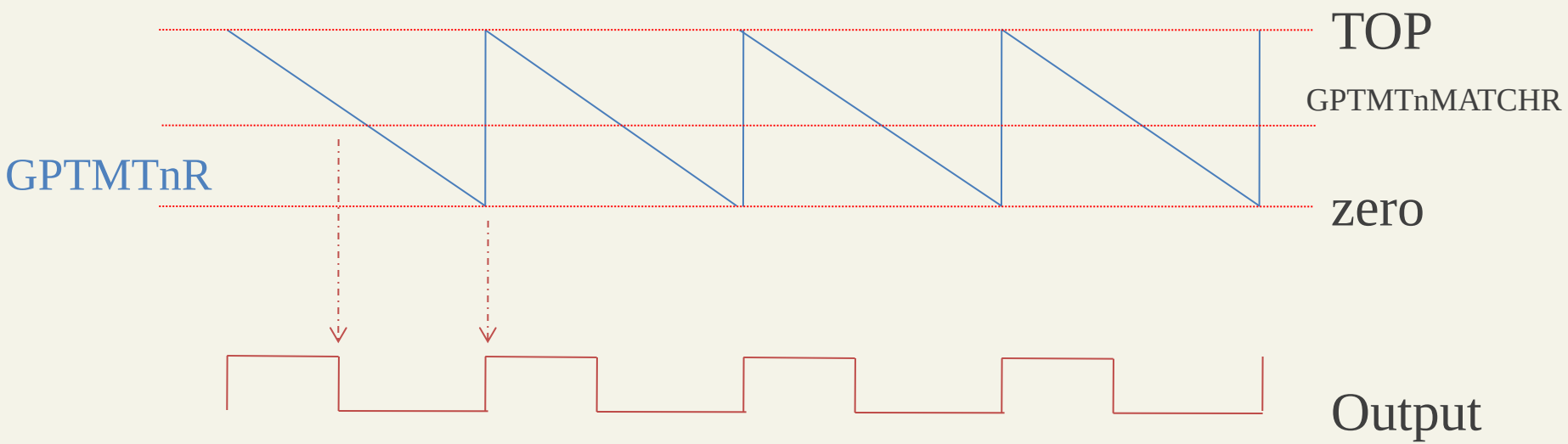
In PWM mode a 16bit timer acts as a 24bit timer.

- Lower 16bits (15:0) go in the Timers Interval Load register
- Higher 8bits (23:16) go in the Timer Prescale Register
- Match register also works this way with it's prescale holding most significant 8 bits

Also remember that Timers in PWM mode **count down** from load value

PWM

Generate single-slope PWM waveform continuously



GPTMTnR changes: 0, 1, …, GPTMTnMATCHR, …, TOP, 0, 1, …
First event at the end of the cycle GPTMTnR = GPTMTnMATCHR
Second event at the end of the cycle GPTMTnR = TOP
Pulse width = GPTMTnMATCHR , Pulse period = TOP

Servo Programming

```
unsigned pulse_period = …;          //  pulse period in cycles

void timer0_init()
{
    //***set GPIO PB5, turn on clk, alt. function, output, enable***
    TIMER0_CTL_R = _____ //disable timer to config
    SYSCTL_RCGCTIMER_R = _____ //turn on clk for timer0
    TIMER0_TAMR_R = _____ //periodic and PWM enable
    TIMER0_CFG_R = _____ //set size of timer to 16
    TIMER0_TAILR_R = pulse_period & 0xFFFF //lower 16 bits of the interval
    TIMER0_TAPR_R =  pulse_period >> 16 //set the upper 8 bits of the interval
    GPTMTAMATCHR0 = pulse_period - mid_width; // if you want to move servo
    to the middle
    TIMER0_CTL_R = _____ //enable timer
}
```

Servo Programming

```
void move_servo(unsigned degree)
{
  unsigned pulse_width;                 // pulse width in cycles

  …                                     // calculate pulse width in cycles

 TIMER0_TAMATCHR_R = period_width - pulse_width;       // set pulse width

  // you need to call timer_waitMillis( ) here to enforce a delay for the servo to
  // move to the position
}
```

Review of OC Programming Interface

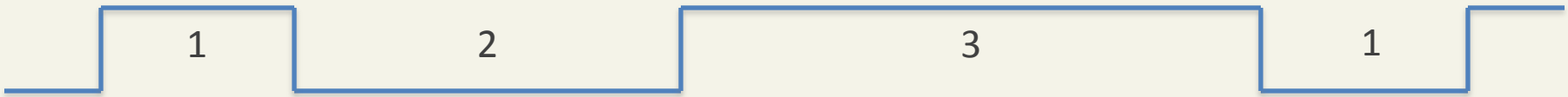
- GPTMCTL – Control
- GPTMCFG – Configuration
- GPTMTnMR – Timer n mode
- GPTMTnPR – Timer n prescale / 8 bits PWM
- GPTMTnILR – Timer n interval load
- GPTMTnPMR – Timer n prescale match
- GPTMTnMATCHR – Timer n match
- GPTMIMR – Interrupt mask
- GPTMRIS – Raw interrupt status
- GPTMICR – Interrupt clear

See page 726 of data sheet for more info

Output Compare: General Purpose Waveform

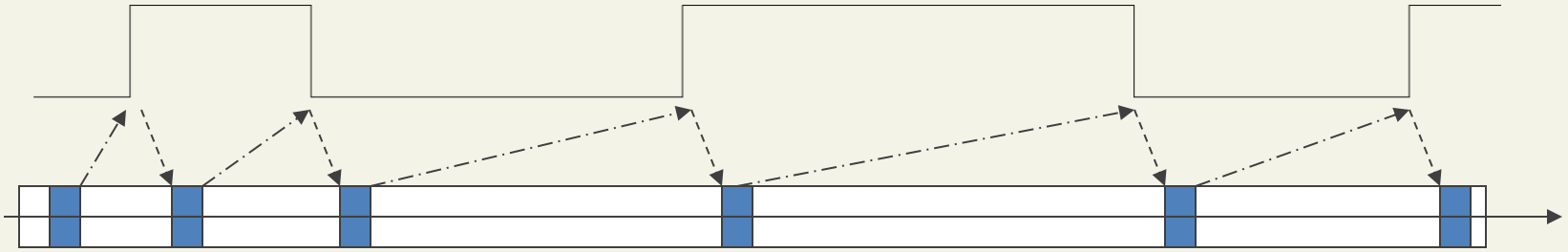
How to generate an output **waveform of arbitrary shape**?

Recall the example: Generate a waveform that is 1-cycle high, 2-cycle low, 3-cyle high, 1-cycle low, and repeating



Output Compare: General Purpose Waveform

Recall the solution: Preset the time of each event

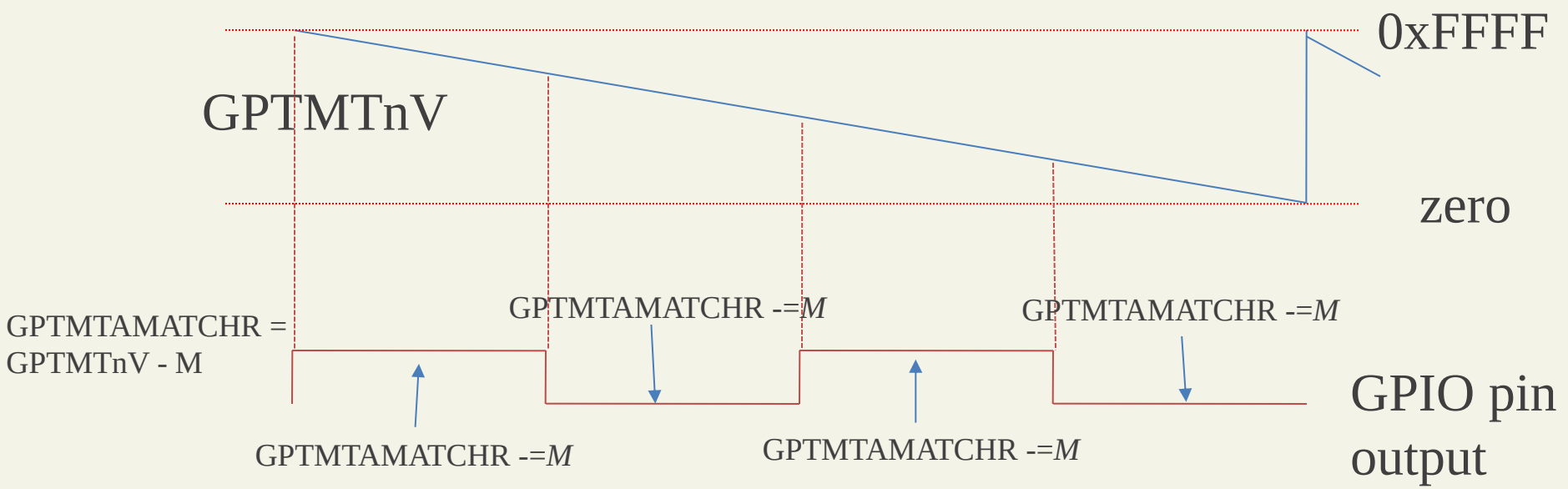


GPTMTnR=0

- CPU sets next event time
- Interrupt to CPU
- CPU Interrupt processing
- CPU Foreground computation

General Purpose Waveform

Example: Use general purpose waveform generation to make a square waveform (timer A used)



Cycle time: $2 \times M$

Programming Example: General Purpose Waveform

```
// Use Periodic Timer Mode to generate a square
   waveform of 2*M timer cycles, using Timer0A. Assume
   Timer, GPIO (PF0), and NVIC initialized already

TIMER0A_Handler(void)
{
    //check that a Match interrupt occurred
   if(TIMER0_MIS_R & TIMER_MIS_TAMMIS)
   {
     // clear interrupt flag
     TIMER0_ICR = TIMER0_ICR | TIMER_MIS_TAMMIS;

     //set next match time
      TIMER0_TAMATCHR_R = TIMER0_TAMATCHR_R - M;

     if(GPIO_PORTF_DATA_R & 0x01)
     {
        GPIO_PORTF_DATA_R &= ~0x01;//set low
      }
     else
     {
        GPIO_PORTF_DATA_R |= 0x01;//set high
      }
   }
}
```

Programming Example: General Purpose Waveform

Generate a periodic waveform repeating the following:

100-cycle low, 100 high, 200 low, 200 high, 300 low, 300 high.

- Assume: 1) Timer already configured in count-down periodic mode, 2) Assumer Port F wire 0 will be the output and is already properly configured, 3) Assume MATCH interupts have already been enabled, and the NVIC has been configured.

- **Give code to place in the Timer ISR**

Programming Example: General Purpose Waveform

```
volatile unsigned int count[6]={100, 100, 200,200, 300,
   300};
int pos = 0;

//Assume output is initially high
TIMER0A_Handler(void)
{
   if(TIMER0_MIS_R & TIMER_MIS_TAMMIS) //check IRQ type
   {
     // clear interrupt flag
     TIMER0_ICR = TIMER0_ICR | TIMER_MIS_TAMMIS;

     //set next match time
     TIMER0_TAMATCHR_R = TIMER0_TAMATCHR_R - count[pos];
     pos =(pos+1) % 6;

     if(GPIO_PORTF_DATA_R & 0x01) // Toggle output
     {
        GPIO_PORTF_DATA_R &= ~0x01;//set low
     }
     else
     {
        GPIO_PORTF_DATA_R |= 0x01;//set high
      }
   }
}
```

Programing Example: General Purpose Waveform

Initialize Timer/Counter 0A's OC unit as periodic for general purpose waveform gen

```
timer_init(){
//init GPIO, and enable Timer clock, and count-down
...
TIMER0_CTL_R &= ~TIMER_CTL_TAEN; //disable timer0A
TIMER0_CFG_R |= TIMER_CFG_16_BIT; //set to 16bit
TIMER0_TAMR_R = TIMER_TAMR_PERIOD; //set to periodic
TIMER0_TAPR_R = 0; //set timer prescaler
TIMER0_TAILR_R = 0xFFFF; //set period
TIMER0_TAPMR_R = 0; // set match prescaler
TIMER0_TAMATCHR_R = first_match; // set value for initial intpt
TIMER0_ICR_R |= TIMER_IMR_TAMIM; //clear interrupts
TIMER0_IMR_R |= TIMER_IMR_TAMIM; //enable match interrupts
IntRegister(INT_TIMER0A, TIMER0A_Handler); //Bind intrupt handle

// NVIC setup
...
IntMasterEnable(); //enable global interrupts
TIMER0_CTL_R |= TIMER_CTL_TAEN; //enable timer0A
}
```

Generating Periodic Interrupts (needs updating)

Example: Use Periodic Timer Mode to generate periodic interrupt (lab 4)

```
void periodic_interrupt_init(void)
{
  // Code for enabling Timer Clock
  ...
  //clear timeout interrupts
   TIMER0_ICR_R |= TIMER_IMR_TATOIM;
   TIMER0_IMR_R |= TIMER_IMR_TATOIM; //enable timeout ints

  TIMER0_TAPR_R = 0; //set timer prescaler
  TIMER0_TAILR_R = 0xFF; //set period

  //Bind interupt handler
   IntRegister(INT_TIMER0A, TIMER0A_Handler);

   IntMasterEnable(); //enable global interrupts
   TIMER0_CTL_R |= TIMER_CTL_TAEN; //enable timer0A
}
```

Review of OC Programming Interface

- GPTMCTL – Control
- GPTMCFG – Configuration
- GPTMTnMR – Timer n mode
- GPTMTnPR – Timer n prescale / 8 bits PWM
- GPTMTnILR – Timer n interval load
- GPTMTnPMR – Timer n prescale match
- GPTMTnMATCHR – Timer n match
- GPTMIMR – Interrupt mask
- GPTMRIS – Raw interrupt status
- GPTMICR – Interrupt clear

See page 726 of data sheet for more info

Summary of OC General Purpose Waveform

Good for generating **waveforms of any shape**

Programming: Use Interrupt to pre-set the timing of the next event

Cons:

- Interrupt overhead can be high
- Cannot generate high-frequency waveforms
- CPU cannot sleep into deep power-saving modes

Summary of PWM

Good for generating **Pulse Width Modulation waveforms** and **Clock waveforms**

Two parameters: **Pulse Width** and **Period Length** (they decide the timing of two events)

Programming

- Lower 16bits (15:0) go in the Timers Interval Load register
- Higher 8bits (23:16) go in the Timer Prescale Register
- Match stores (Top - Pulse_Width) for down counter mode

PWM: Summary example

Objectives:

- Generate a PWM wave that has a pulse width of 39 ticks and a period of 200 ticks
- The PWM wave should be generated on PB5 in lab