香港中文大學
The Chinese University of Hong Kong

CENG2400 Embedded System Design

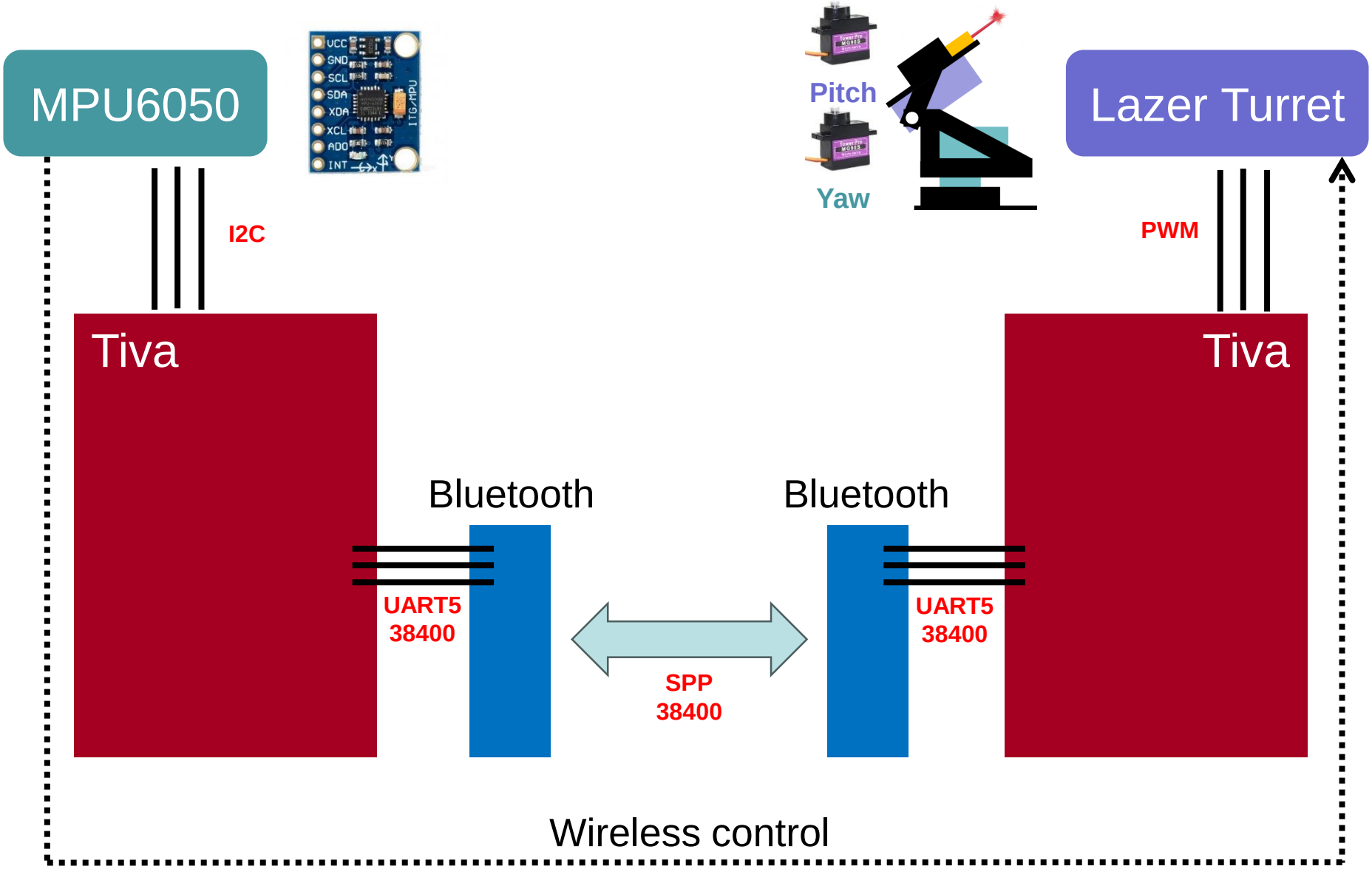
Lecture 08: Motion Control

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk

Thanks to Prof. Q. Xu and Drs. K. H. Wong, Philip Leong, Y.S. Moon, O. Mencer, N. Dulay, P. Cheung for some of the slides used in this course!

FP: Motion-Controlled Lazer Turret

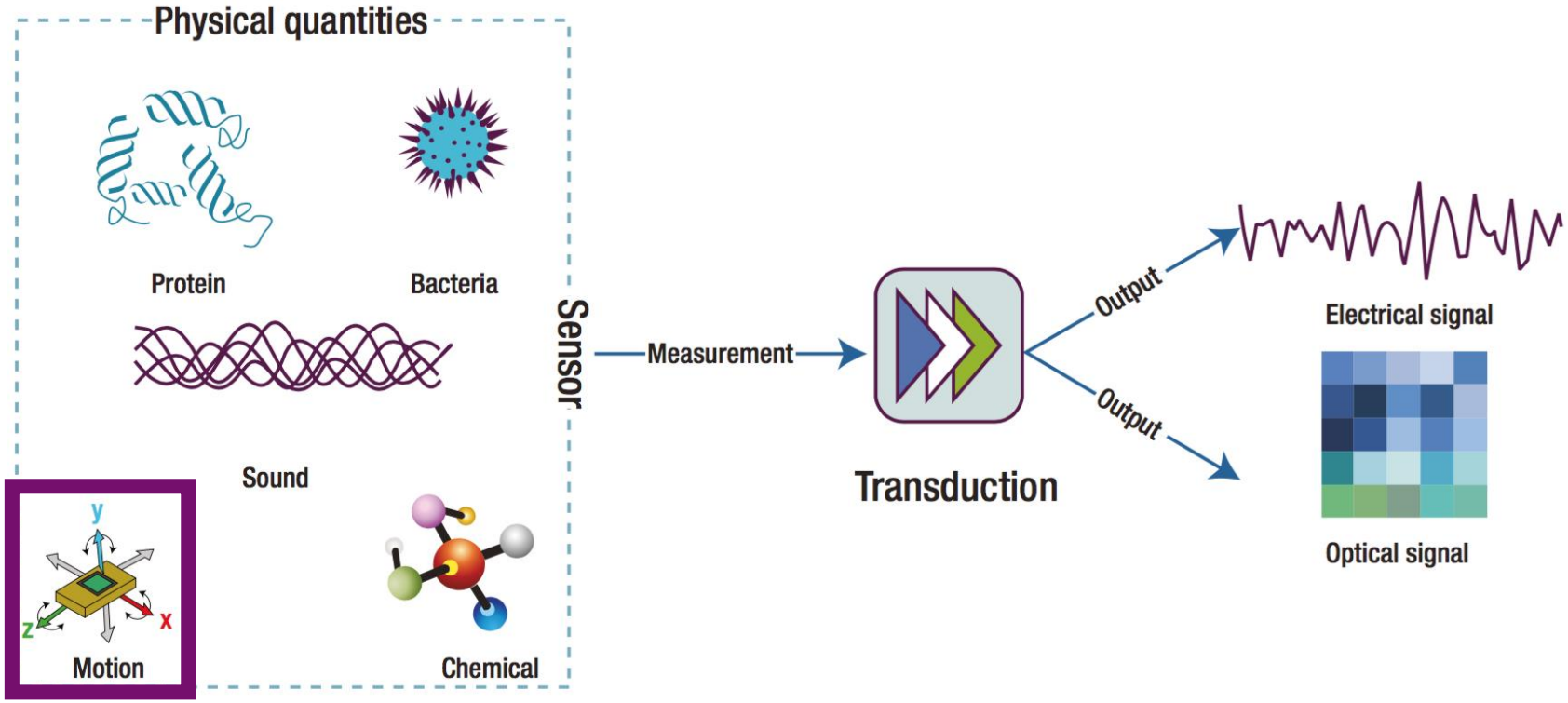


Outline

- **Motion Sensor**
  - Gyroscope & Accelerometer
  - Motion Processing Unit (MPU)
  - TivaWare™ Driver vs. Sensor Library
  - Demo Code: mpu_demo.c
- **Electric Motor**
  - Servo Motor
  - Pulse-Width Modulation (PWM)
  - PWM Module on Tiva™
  - PWM in TivaWare™ Library
  - Demo Code: servo_demo.c

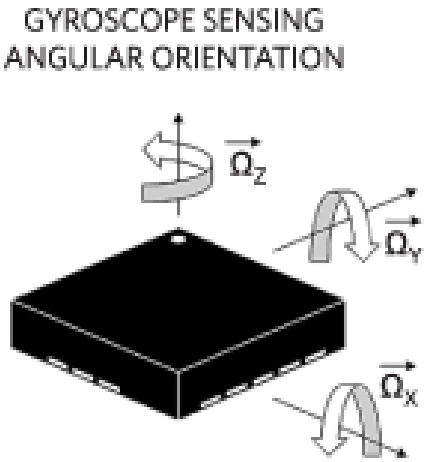
Sensor

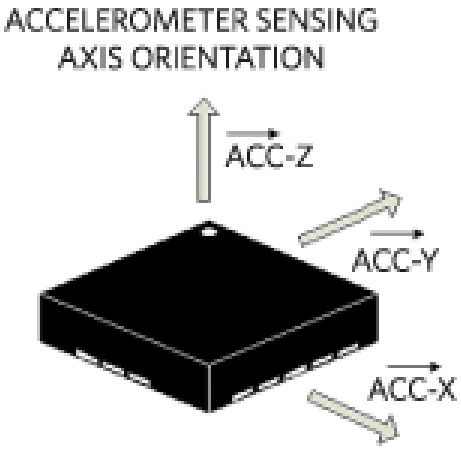
- A **sensor** receives a stimulus from the environment and responds with an electrical signal.
  - It may use a series of energy conversion steps to produce that electrical signal.



Gyroscope & Accelerometer

- A **gyroscope** is a device used for measuring or maintaining orientation and angular velocity (ω).
- An **accelerometer** is a device that measures the linear acceleration (a) of an object.

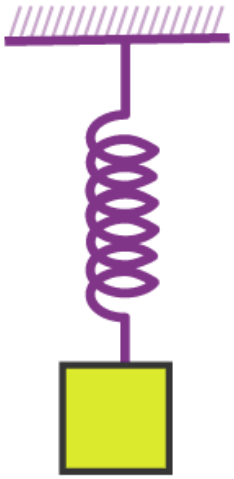




Angular	Linear
$\omega = \omega_0 + \alpha t$	$v = v_0 + at$
$\theta = \omega_0 t + \frac{1}{2}\alpha t^2$	$x = v_0 t + \frac{1}{2} a t^2$

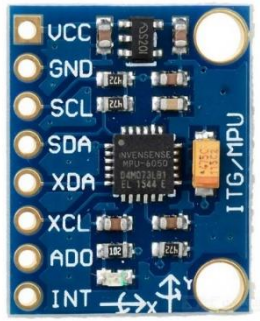
Class Exercise 8.1

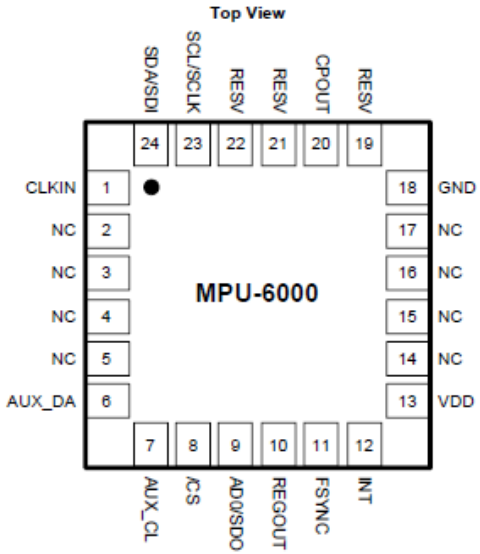
- A 1-kilogram mass is suspended motionless from a spring with a force constant of 50 (Newtons per meter).
- The mass is pulled 0.25 meters downward and then released. What is the acceleration of the mass at the moment of release?



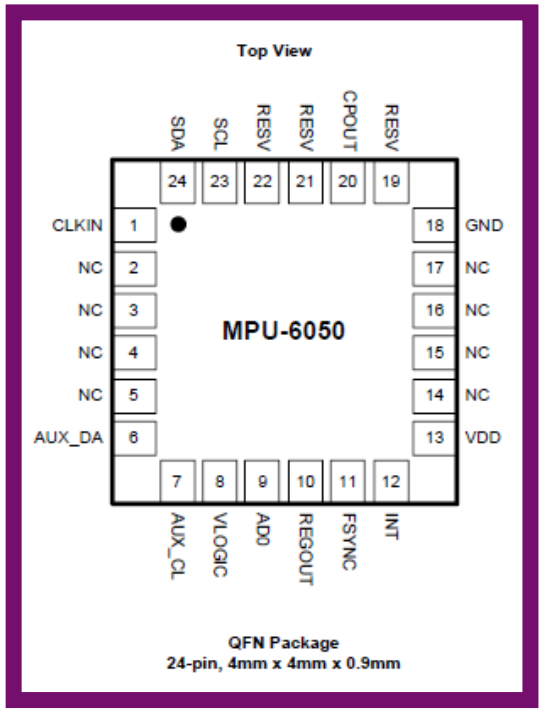
MPU-60X0 (1/2)

- **MPU-60X0** is the world's first integrated 6-axis MotionTracking device that combines a **3-axis gyroscope**, **3-axis accelerometer**, and a Digital Motion Processor™ (DMP) all in a small 4x4x0.9mm package.

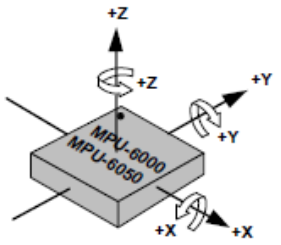




QFN Package
24-pin, 4mm x 4mm x 0.9mm



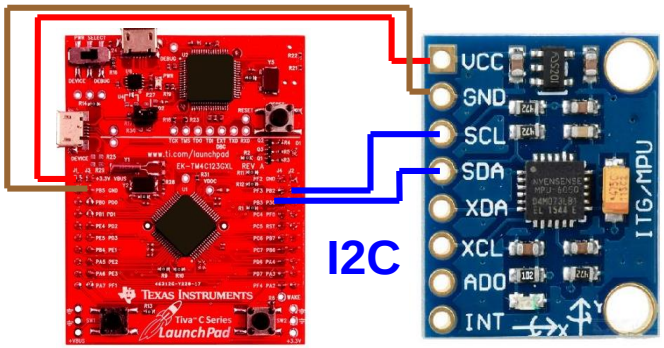
QFN Package
24-pin, 4mm x 4mm x 0.9mm

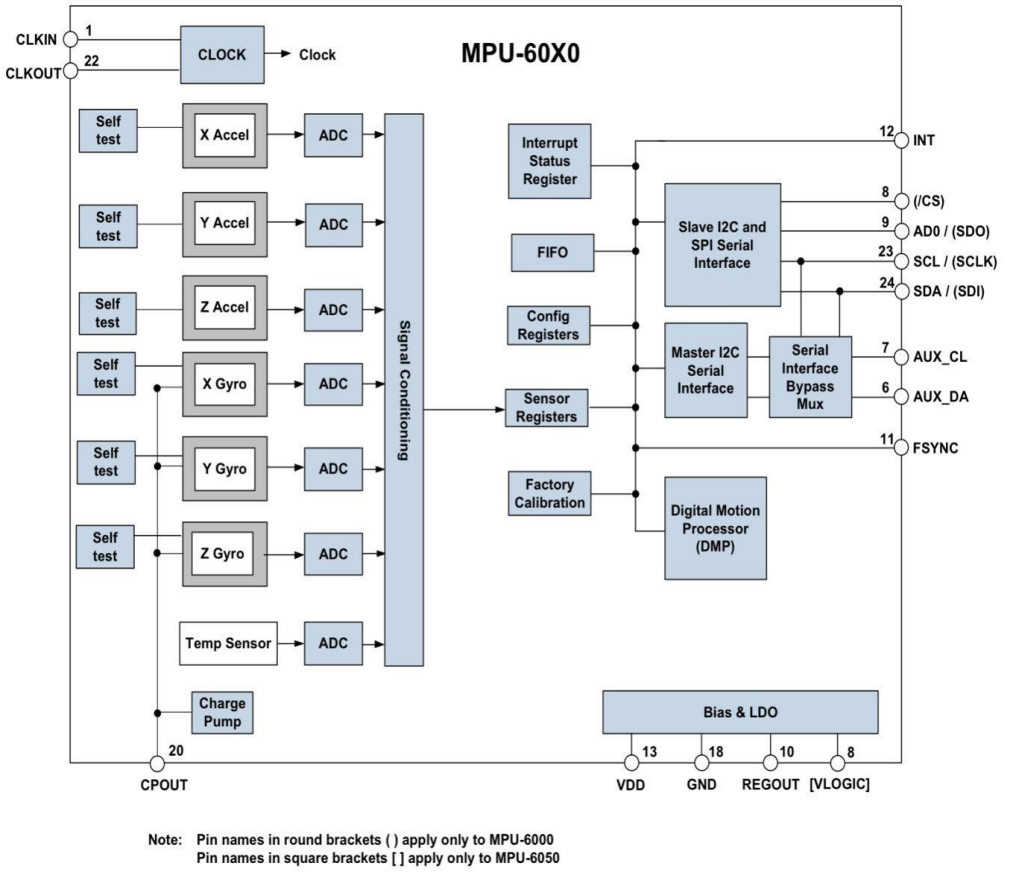


Orientation of Axes of Sensitivity and Polarity of Rotation

MPU-60X0 (2/2)

- **MPU-60X0** features six 16-bit analog-to-digital converters (ADCs) for digitizing the **gyroscope** and **accelerometer** outputs.
- Communication with registers is performed using **Inter-Integrated Circuit Bus (I2C)** (for **MPU-6050**).

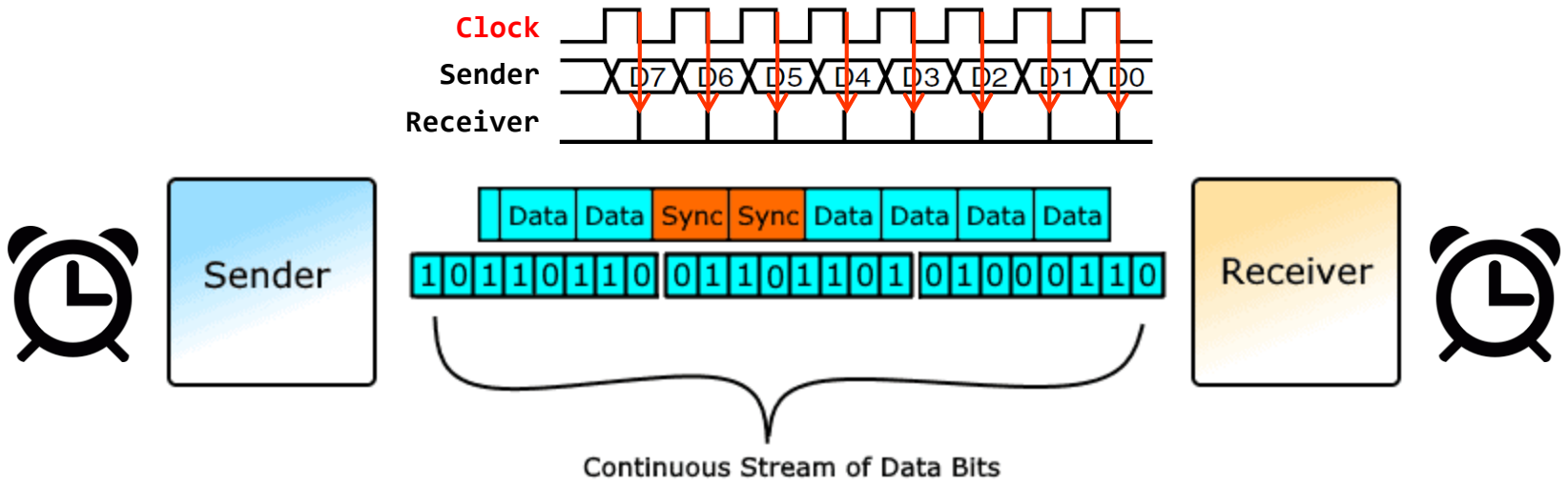




Note: Pin names in round brackets ( ) apply only to MPU-6000
Pin names in square brackets [ ] apply only to MPU-6050

Recall: Serial Communication

- **Serial comm.** can be **synchronous** or asynchronous:
  - **Synchronous Transmission**: The sender and receiver are synchronized using an external synchronization clock.



  - An extra channel is typically employed to transmit the **clock signal**.
    - It provides **synchronous clock pulses** that define a **constant time interval** for data transmission.
  - Common **synchronous** serial communication protocols: **Serial Peripheral Interface (SPI)** and **Inter-Integrated Circuit Bus (I^2C)**.

TivaWare™ Driver vs. Sensor Library

- **Driver Library (*driverlib*)**
  - Functionality: a low-level hardware abstraction library that provides functions to interact directly with the microcontroller's peripherals (such as I2C, UART, GPIO, timers, etc.).
  - Example: Using driverlib, you would manually initialize the I2C peripheral with specific settings that update the value of corresponding register.
- **Sensor Library (*sensorlib*)**
  - Functionality: a higher-level library specifically designed for interacting with sensors, like the MPU6050.
  - It abstracts away much of the complexity of directly interfacing with the sensor to configure and read data.
  - Example: Using sensorlib, you simply call a functions which internally handles the I2C setup and sensor initialization, abstracting the configuration details.

TivaWare™ Peripheral Driver Library

Table of Contents

TivaWare™ Sensor Library (1/2)

- **TivaWare™ Sensor Library** is a collection of functions & drivers for interacting with sensors.
  - Such as accelerometers, gyroscopes, magnetometers, and so on.

- It consists of the following components:
  - A set of drivers for I2C connected sensors.
  - An interrupt-driven I2C master driver that handles the sequence of operations required to perform an I2C transfer, as well as provides a request queue to ease sharing of the I2C bus between multiple drivers.
  - A set of routines for performing common, sensor-independent operations on sensor data.

TivaWare™ Sensor Library (2/2)

Table of Contents

Sensor Library - MPU6050 Driver

15 MPU6050 Accelerometer and Gyroscope Driver

15.1 Introduction

The MPU6050 is an integrated three-axis accelerometer and gyroscope produced by Invensense Inc. In addition to an internal accelerometer and gyroscope, this device has an auxiliary I2C bus that can be used to automatically sample external sensors. This driver allows the MPU6050 to be accessed via the I2C bus.

When initialized, a soft reset of the MPU6050 is performed, putting it into its default state. The default accelerometer range of +/- 2 g and gyroscope range of 250 degrees/second are therefore selected.

This driver is contained in `sensorlib/mpu6050.c`, with `sensorlib/mpu6050.h` containing the API declarations for use by applications.

15.2 API Functions

Functions

- void MPU6050DataAccelGetFloat (tMPU6050 *psInst, float *pfAccelX, float *pfAccelY, float *pfAccelZ)
- void MPU6050DataAccelGetRaw (tMPU6050 *psInst, uint_fast16_t *pui16AccelX, uint_fast16_t *pui16AccelY, uint_fast16_t *pui16AccelZ)
- void MPU6050DataGyroGetFloat (tMPU6050 *psInst, float *pfGyroX, float *pfGyroY, float *pfGyroZ)
- void MPU6050DataGyroGetRaw (tMPU6050 *psInst, uint_fast16_t *pui16GyroX, uint_fast16_t *pui16GyroY, uint_fast16_t *pui16GyroZ)
- uint_fast8_t MPU6050DataRead (tMPU6050 *psInst, tSensorCallback *pfnCallback, void *pvCallbackData)
- uint_fast8_t MPU6050Init (tMPU6050 *psInst, tI2CMInstance *psI2CInst, uint_fast8_t ui8I2CAddr, tSensorCallback *pfnCallback, void *pvCallbackData)
- uint_fast8_t MPU6050Read (tMPU6050 *psInst, uint_fast8_t ui8Reg, uint8_t *pui8Data, uint_fast16_t ui16Count, tSensorCallback *pfnCallback, void *pvCallbackData)
- uint_fast8_t MPU6050ReadModifyWrite (tMPU6050 *psInst, uint_fast8_t ui8Reg, uint_fast8_t ui8Mask, uint_fast8_t ui8Value, tSensorCallback *pfnCallback, void *pvCallbackData)
- uint_fast8_t MPU6050Write (tMPU6050 *psInst, uint_fast8_t ui8Reg, const uint8_t *pui8Data, uint_fast16_t ui16Count, tSensorCallback *pfnCallback, void *pvCallbackData)

15.2.1.5 MPU6050DataRead

Reads the accelerometer and gyroscope data from the MPU6050.

Prototype:

```
uint_fast8_t
MPU6050DataRead(tMPU6050 *psInst,
                tSensorCallback *pfnCallback,
                void *pvCallbackData)
```

Parameters:

psInst is a pointer to the MPU6050 instance data.

pfnCallback is the function to be called when the data has been read (can be **NULL** if a callback is not required).

pvCallbackData is a pointer that is passed to the callback function.

Description:

This function initiates a read of the MPU6050 data registers. When the read has completed (as indicated by calling the callback function), the new readings can be obtained via:

- MPU6050DataAccelGetRaw()
- MPU6050DataAccelGetFloat()
- MPU6050DataGyroGetRaw()
- MPU6050DataGyroGetFloat()

Returns:

Returns 1 if the read was successfully started and 0 if it was not.

15.2.1.1 MPU6050DataAccelGetFloat

Gets the accelerometer data from the most recent data read.

Prototype:

```
void
MPU6050DataAccelGetFloat(tMPU6050 *psInst,
                         float *pfAccelX,
                         float *pfAccelY,
                         float *pfAccelZ)
```

15.2.1.3 MPU6050DataGyroGetFloat

Gets the gyroscope data from the most recent data read.

Prototype:

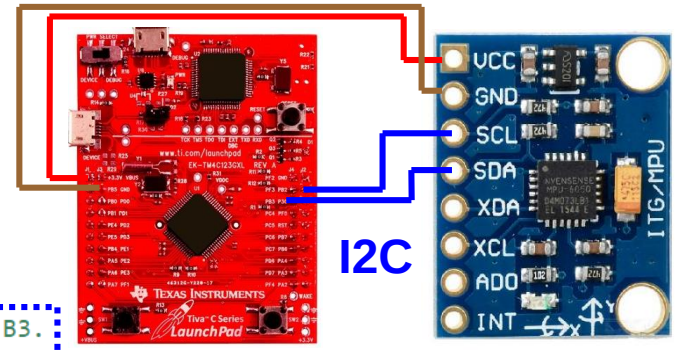
```
void
MPU6050DataGyroGetFloat(tMPU6050 *psInst,
                        float *pfGyroX,
                        float *pfGyroY,
                        float *pfGyroZ)
```

Demo Code: mpu_demo.c (1/2)

```c
int main()
{
    // Set the system clock to use the PLL with a 16 MHz crystal oscillator.
    // The clock is divided by 1 (SYSCTL_SYSDIV_1) and uses an internal oscillator (SYSCTL_OSC_INT).
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_PLL | SYSCTL_OSC_INT | SYSCTL_XTAL_16MHZ);

    // Initialize the system (e.g., peripherals, hardware components)
    Initialization();

    // Declare arrays to store accelerometer and gyroscope data
    float fAccel[3], fGyro[3];

    // Reset the MPU6050 sensor by writing to the power management register (PWR_MGMT_1)
    g_bMPU6050Done = false;
    MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_1, 0x00, 0b00000010 & MPU6050_PWR_MGMT_1_DEVICE_RESET, MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

    // Configure the MPU6050 to not be low power mode by writing to the power management register (PWR_MGMT_2)
    g_bMPU6050Done = false;
    MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_2, 0x00, 0x00, MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

    // Main infinite loop to repeatedly read data from the MPU6050
    while (1)
    {
        //
        // Request another reading from the MPU6050 sensor
        //
        g_bMPU6050Done = false;
        MPU6050DataRead(&sMPU6050, MPU6050Callback, &sMPU6050);
        while (!g_bMPU6050Done)
        {
        }

        // Extract the accelerometer data (in floating-point format)
        MPU6050DataAccelGetFloat(&sMPU6050, &fAccel[0], &fAccel[1], &fAccel[2]);

        // Extract the gyroscope data (in floating-point format)
        MPU6050DataGyroGetFloat(&sMPU6050, &fGyro[0], &fGyro[1], &fGyro[2]);
    }
}
```

Demo Code: mpu_demo.c (2/2)

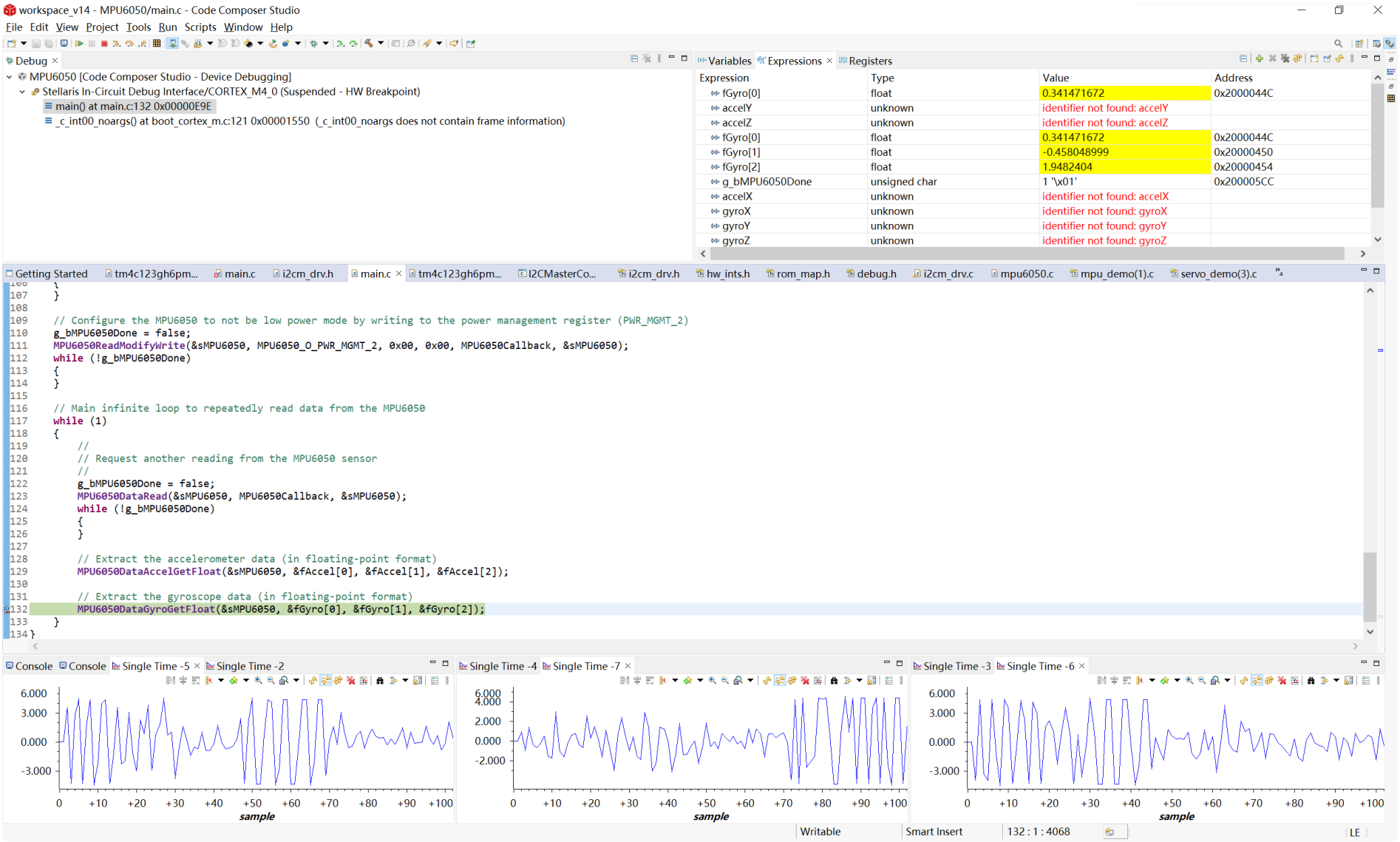
```
void Initialization(void)
{
    //enable I2C module 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);

    //reset module
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);

    //enable GPIO peripheral that contains I2C 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

    // Configure the pin muxing for I2C0 functions on port B2 and B3.
    GPIOPinConfigure(GPIO_PB2_I2C0SCL);
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);

    // Select the I2C function for these pins.
    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);

    // Enable and initialize the I2C0 master module.
    // Use the system clock for the I2C0 module.
    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), true);

    //clear I2C FIFOs
    HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;

    // Initialize the I2C master driver.
    I2CMInit(&g_sI2CMSimpleInst, I2C0_BASE, INT_I2C0, 0xff, 0xff, SysCtlClockGet());
    // Register the interrupt handler for I2C interrupts
    I2CIntRegister(I2C0_BASE, I2CMSimpleIntHandler);

    // Configure the MPU6050
    g_bMPU6050Done = false;
    MPU6050Init(&sMPU6050, &g_sI2CMSimpleInst, 0x68, MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

}
```



- 3.3V-VCC
- GND-GND
- PB2-SCL
- PB3-SDA

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type[a]	Description
I2C0SCL	47	PB2 (3)	I/O	OD	I²C module 0 clock. Note that this signal has an active pull-up. The corresponding port pin should not be configured as open drain.
I2C0SDA	48	PB3 (3)	I/O	OD	I²C module 0 data.
I2C1SCL	23	PA6 (3)	I/O	OD	I²C module 1 clock. Note that this signal has an active pull-up. The corresponding port pin should not be configured as open drain.
I2C1SDA	24	PA7 (3)	I/O	OD	I²C module 1 data.
I2C2SCL	59	PE4 (3)	I/O	OD	I²C module 2 clock. Note that this signal has an active pull-up. The corresponding port pin should not be configured as open drain.
I2C2SDA	60	PE5 (3)	I/O	OD	I²C module 2 data.
I2C3SCL	61	PD0 (3)	I/O	OD	I²C module 3 clock. Note that this signal has an active pull-up. The corresponding port pin should not be configured as open drain.
I2C3SDA	62	PD1 (3)	I/O	OD	I²C module 3 data.

Expected Results



Outline

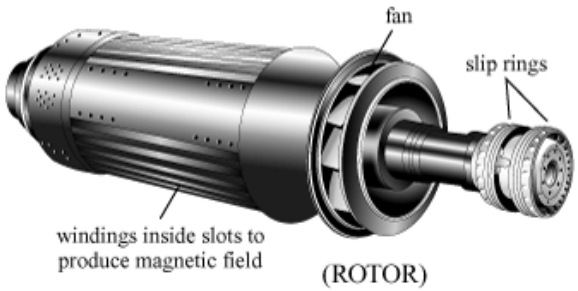
- **Motion Sensor**
  - Gyroscope & Accelerometer
  - Motion Processing Unit (MPU)
  - TivaWare™ Driver vs. Sensor Library
  - Demo Code: mpu_demo.c
- **Electric Motor**
  - Servo Motor
  - Pulse-Width Modulation (PWM)
  - PWM Module on Tiva™
  - PWM in TivaWare™ Library
  - Demo Code: servo_demo.c

Actuator

- **Actuator:** a device used to produce motion or action.
  - For instance, acceleration and displacement resulting from a force or torque.
  - Actuation can be from few microns to few meters.

Electric Motor (1/2)

- The most ubiquitous type of actuator.
  - It converts electrical energy into mechanical energy with the help of electromagnetics.
- All electric motor consists of two parts:
  - **Stator**: The fixed part in the motor and consists of magnet and coils wrapped around/inside it.
  - **Rotor**: A moving axle that goes through the stator and magnetic forces are exerted on it causing it to spin.



Electric Motor (2/2)

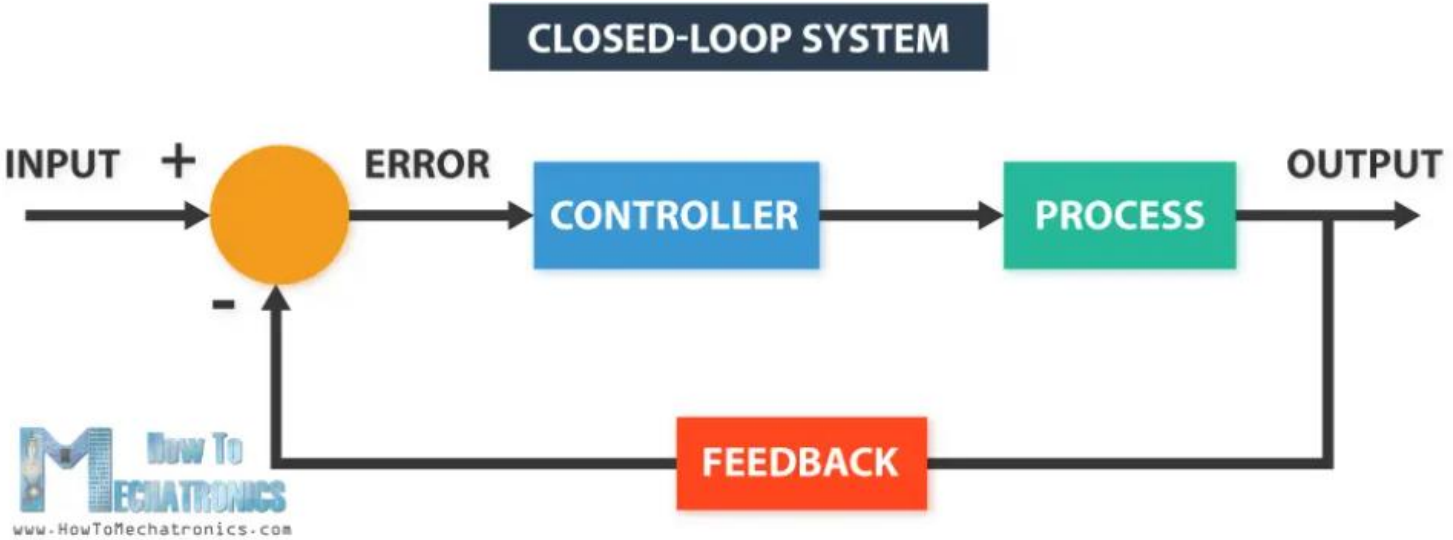
- There are numerous types of motors, and they differ in required power, power efficiency, speed range, torque, etc.

 AC Motor	 DC Motor	 Shunt Motor	 Separately Motor	 Compound Motor
 PDMC Motor	 Induction Motor	 Synchronous Motor	 Stepper Motor	 Brushless Motor
 Universal Motor	 Hysteresis Motor	 Reluctance Motor	 Servo Motor	 Linear Motor

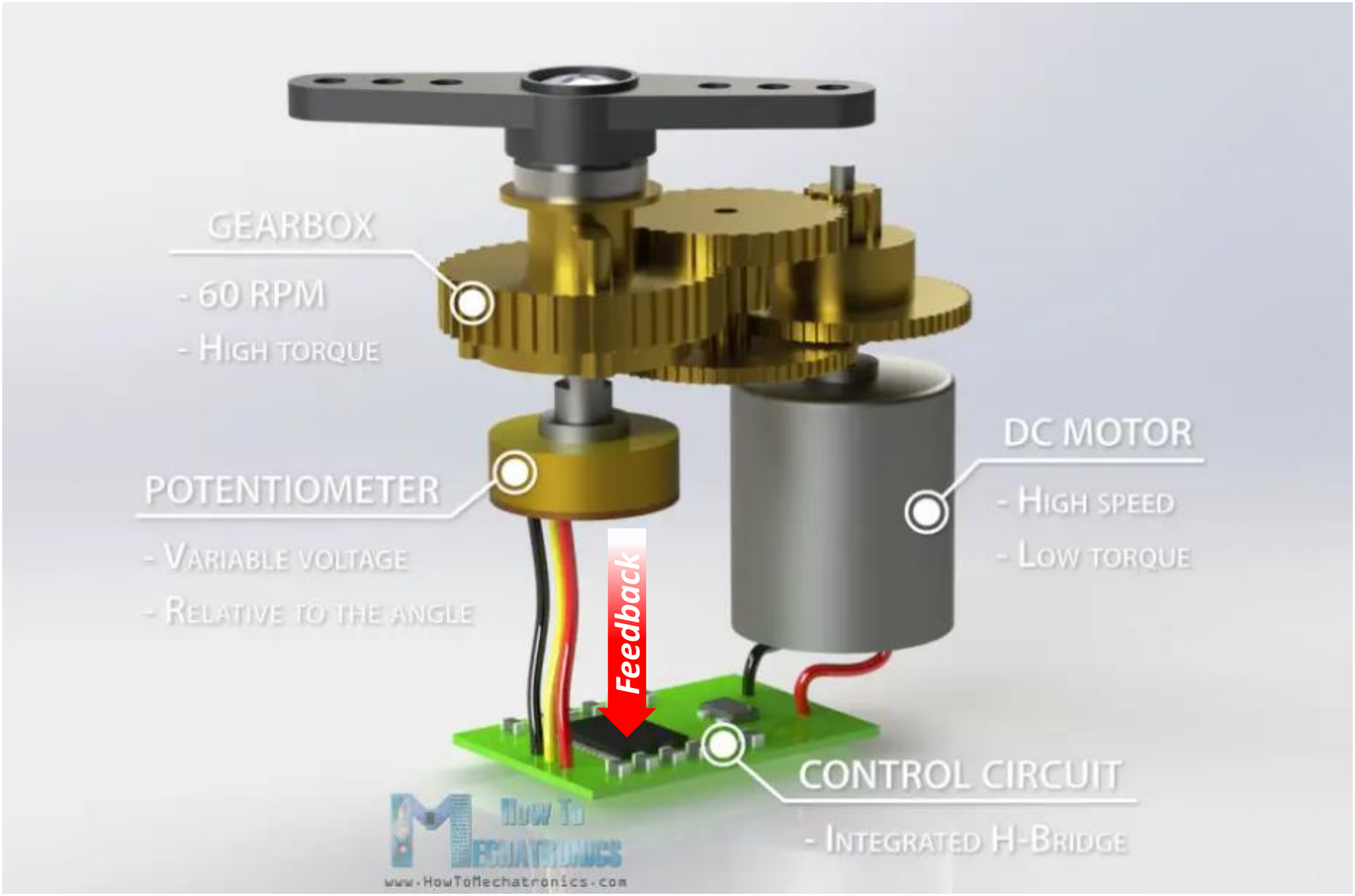
https://www.youtube.com/watch?app=desktop&v=ZGGQBPA-v6o

Servo Motor

- A servo motor is a closed-loop system that uses "position feedback" to control its motion and final position.
  - It features the ability to precisely control the position of its shaft.

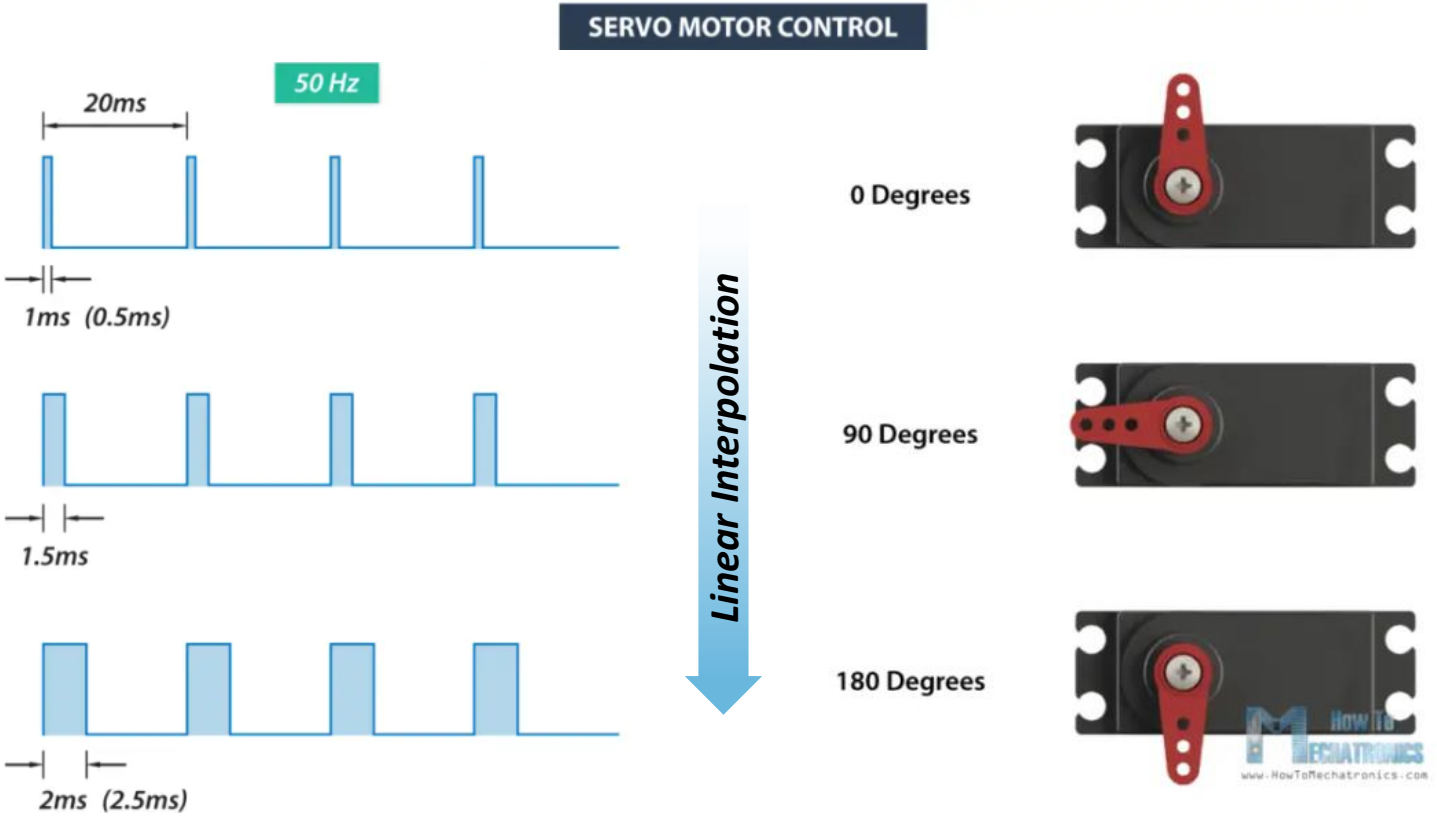


How servo motor works?



How to control servo motor?

- A servo motor is controlled by sending a series of pulses (with different widths) through the signal line.



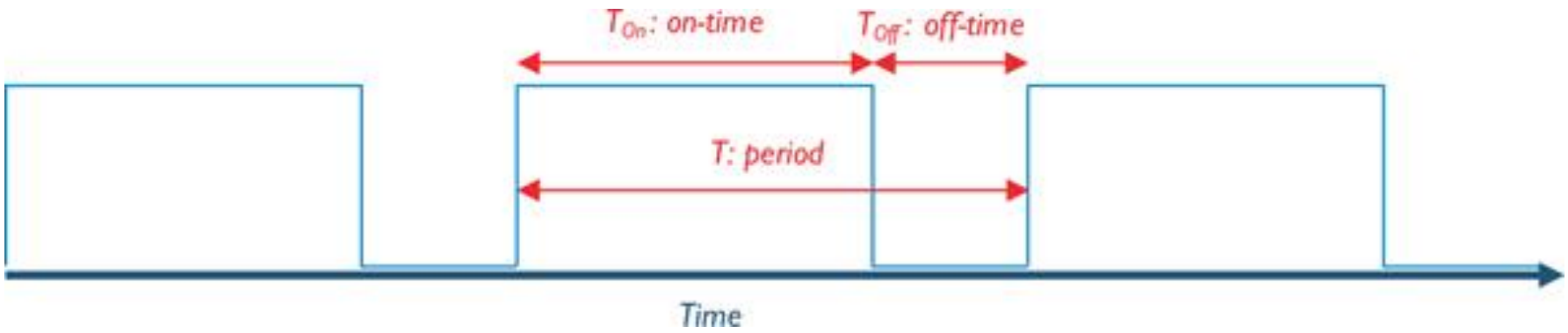
  - *Note: The pulses can vary with different brands (e.g., 0.5 ms for 0 degrees and 2.5 ms for 180 degrees position).*

Recall: Timer Peripherals on Tiva™

- **SysTick** (§3.3 in MCU Data Sheet; §28 in TivaWare Library)
  - Provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism.
- **General-Purpose Timers** (§11 in MCU Data Sheet; §29 in TivaWare Library)
  - Provides programmable timers to count or time events.
  - There are six 16/32-bit GPTM & six 32/64-bit Wide GPTM, each module (M) provides two timers/counters (Timer A/B).
    - **Modes**: one-shot, periodic, prescaler, real-time, count-up/down, etc.
- **Watchdog Timers** (§12 in MCU Data Sheet; §33 in TivaWare Library)
  - Used to regain control when a system has failed.
  - There are two Watchdog Timer Modules (Watchdog Timer 0/1).
- **Pulse Width Modulator** (§20 in MCU Data Sheet; §21 in TivaWare Library)
  - Used for digitally encoding analog signal levels.
  - There are two PWM modules, each with four PWM generator blocks and a control block.

Pulse-Width Modulation (PWM)

- **PWM** is a method to send more than one bit of information on a single digital signal line.
  - The information is encoded as the fraction of time that the signal is a logic one (called the duty cycle).
  - It is much less vulnerable to electrical noise, as the signal is sent in a digital format.



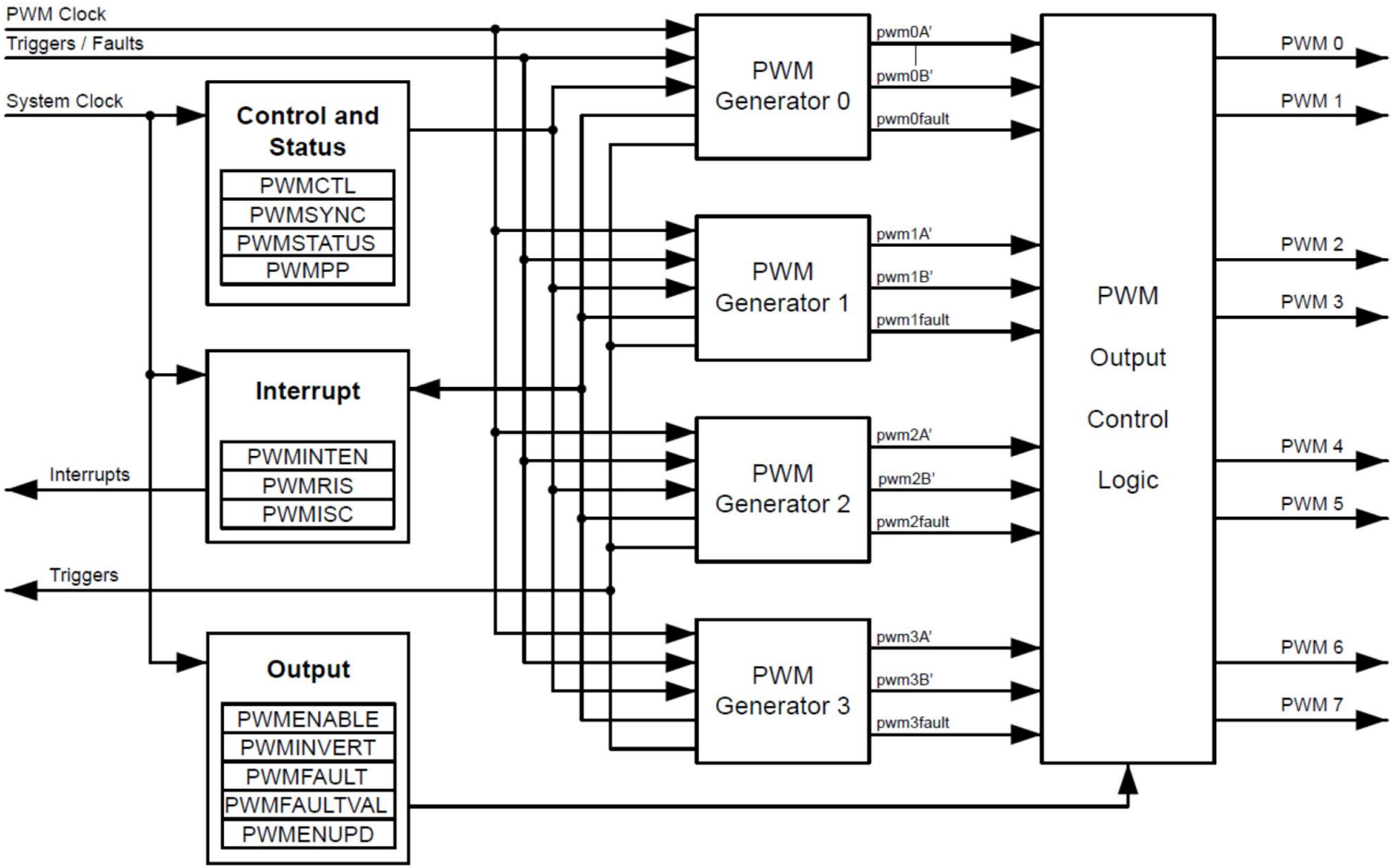
The duty cycle is the on-time divided by the period: $D = T_{On}/T$

Class Exercise 8.2

- Assume a servo motor has a control signal range of 0.5 ms to 2.5 ms (corresponding to a rotation range of 0 degrees to 180 degrees) and the control frequency for the servo motor is 50 Hz.
- If the PWM signal has a high time of 2 ms, determine the corresponding angle of the servo motor.

PWM Module on Tiva™ (1/2)

- There are **two** PWM modules, each with **four** PWM signal generators (each can produce **two** outputs).



PWM Module on Tiva™ (2/2)

Table 20-1. PWM Signals (64LQFP)

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type[a]	Description
MOFAULT0	30 53 63	PF2 (4) PD6 (4) PD2 (4)	I	TTL	Motion Control Module 0 PWM Fault 0.
MOPWM0	1	PB6 (4)	O	TTL	Motion Control Module 0 PWM 0. This signal is controlled by Module 0 PWM Generator 0.
MOPWM1	4	PB7 (4)	O	TTL	Motion Control Module 0 PWM 1. This signal is controlled by Module 0 PWM Generator 0.
MOPWM2	58	PB4 (4)	O	TTL	Motion Control Module 0 PWM 2. This signal is controlled by Module 0 PWM Generator 1.
MOPWM3	57	PB5 (4)	O	TTL	Motion Control Module 0 PWM 3. This signal is controlled by Module 0 PWM Generator 1.
MOPWM4	59	PE4 (4)	O	TTL	Motion Control Module 0 PWM 4. This signal is controlled by Module 0 PWM Generator 2.
MOPWM5	60	PE5 (4)	O	TTL	Motion Control Module 0 PWM 5. This signal is controlled by Module 0 PWM Generator 2.
MOPWM6	16 61	PC4 (4) PD0 (4)	O	TTL	Motion Control Module 0 PWM 6. This signal is controlled by Module 0 PWM Generator 3.
MOPWM7	15 62	PC5 (4) PD1 (4)	O	TTL	Motion Control Module 0 PWM 7. This signal is controlled by Module 0 PWM Generator 3.
M1FAULT0	5	PF4 (5)	I	TTL	Motion Control Module 1 PWM Fault 0.
M1PWM0	61	PD0 (5)	O	TTL	Motion Control Module 1 PWM 0. This signal is controlled by Module 1 PWM Generator 0.
M1PWM1	62	PD1 (5)	O	TTL	Motion Control Module 1 PWM 1. This signal is controlled by Module 1 PWM Generator 0.
M1PWM2	23 59	PA6 (5) PE4 (5)	O	TTL	Motion Control Module 1 PWM 2. This signal is controlled by Module 1 PWM Generator 1.
M1PWM3	24 60	PA7 (5) PE5 (5)	O	TTL	Motion Control Module 1 PWM 3. This signal is controlled by Module 1 PWM Generator 1.
M1PWM4	28	PF0 (5)	O	TTL	Motion Control Module 1 PWM 4. This signal is controlled by Module 1 PWM Generator 2.
M1PWM5	29	PF1 (5)	O	TTL	Motion Control Module 1 PWM 5. This signal is controlled by Module 1 PWM Generator 2.
M1PWM6	30	PF2 (5)	O	TTL	Motion Control Module 1 PWM 6. This signal is controlled by Module 1 PWM Generator 3.
M1PWM7	31	PF3 (5)	O	TTL	Motion Control Module 1 PWM 7. This signal is controlled by Module 1 PWM Generator 3.

PWM in TivaWare™ Library (1/2)

21 Pulse Width Modulator (PWM)

21.1 Introduction

Each instance of a Tiva PWM module provides up to four instances of a PWM generator block, and an output control block. Each generator block has two PWM output signals, which can be operated independently or as a pair of signals with dead band delays inserted. Each generator block also has an interrupt output and a trigger output. The control block determines the polarity of the PWM signals and which signals are passed through to the pins.

Some of the features of the Tiva PWM module are:

- Up to four generator blocks, each containing
  - One 16-bit down or up/down counter
  - Two comparators
  - PWM generator
  - Dead band generator
- Control block
  - PWM output enable
  - Output polarity control
  - Synchronization
  - Fault handling
  - Interrupt status

This driver is contained in `driverlib/pwm.c`, with `driverlib/pwm.h` containing the API declarations for use by applications.

21.2 API Functions

Functions

- uint32_t PWMClockGet (uint32_t ui32Base)
- void PWMClockSet (uint32_t ui32Base, uint32_t ui32Config)
- void PWMDeadBandDisable (uint32_t ui32Base, uint32_t ui32Gen)
- void PWMDeadBandEnable (uint32_t ui32Base, uint32_t ui32Gen, uint16_t ui16Rise, uint16_t ui16Fall)
- void PWMFaultIntClear (uint32_t ui32Base)
- void PWMFaultIntClearExt (uint32_t ui32Base, uint32_t ui32FaultInts)
- void PWMFaultIntRegister (uint32_t ui32Base, void (*pfnIntHandler)(void))
- void PWMFaultIntUnregister (uint32_t ui32Base)

- void PWMGenConfigure (uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Config)
- void PWMGenDisable (uint32_t ui32Base, uint32_t ui32Gen)
- void PWMGenEnable (uint32_t ui32Base, uint32_t ui32Gen)
- void PWMGenFaultClear (uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Group, uint32_t ui32FaultTriggers)
- void PWMGenFaultConfigure (uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32MinFaultPeriod, uint32_t ui32FaultSenses)
- uint32_t PWMGenFaultStatus (uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Group)
- uint32_t PWMGenFaultTriggerGet (uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Group)
- void PWMGenFaultTriggerSet (uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Group, uint32_t ui32FaultTriggers)
- void PWMGenIntClear (uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Ints)
- void PWMGenIntRegister (uint32_t ui32Base, uint32_t ui32Gen, void (*pfnIntHandler)(void))
- uint32_t PWMGenIntStatus (uint32_t ui32Base, uint32_t ui32Gen, bool bMasked)
- void PWMGenIntTrigDisable (uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32IntTrig)
- void PWMGenIntTrigEnable (uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32IntTrig)
- void PWMGenIntUnregister (uint32_t ui32Base, uint32_t ui32Gen)
- uint32_t PWMGenPeriodGet (uint32_t ui32Base, uint32_t ui32Gen)
- void PWMGenPeriodSet (uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Period)
- void PWMIntDisable (uint32_t ui32Base, uint32_t ui32GenFault)
- void PWMIntEnable (uint32_t ui32Base, uint32_t ui32GenFault)
- uint32_t PWMIntStatus (uint32_t ui32Base, bool bMasked)
- void PWMOutputFault (uint32_t ui32Base, uint32_t ui32PWMOutBits, bool bFaultSuppress)
- void PWMOutputFaultLevel (uint32_t ui32Base, uint32_t ui32PWMOutBits, bool bDriveHigh)
- void PWMOutputInvert (uint32_t ui32Base, uint32_t ui32PWMOutBits, bool bInvert)
- void PWMOutputState (uint32_t ui32Base, uint32_t ui32PWMOutBits, bool bEnable)
- void PWMOutputUpdateMode (uint32_t ui32Base, uint32_t ui32PWMOutBits, uint32_t ui32Mode)
- uint32_t PWMPulseWidthGet (uint32_t ui32Base, uint32_t ui32PWMOut)
- void PWMPulseWidthSet (uint32_t ui32Base, uint32_t ui32PWMOut, uint32_t ui32Width)
- void PWMSyncTimeBase (uint32_t ui32Base, uint32_t ui32GenBits)
- void PWMSyncUpdate (uint32_t ui32Base, uint32_t ui32GenBits)

21.2.1 Detailed Description

These functions perform high-level operations on PWM modules.

The following functions provide the user with a way to configure the PWM for the most common operations, such as setting the period, generating left- and center-aligned pulses, modifying the pulse width, and controlling interrupts, triggers, and output characteristics. However, the PWM module is very versatile and can be configured in a number of different ways, many of which are beyond the scope of this API. In order to fully exploit the many features of the PWM module, users are advised to use register access macros.

When discussing the various components of a PWM module, this API uses the following labeling convention:

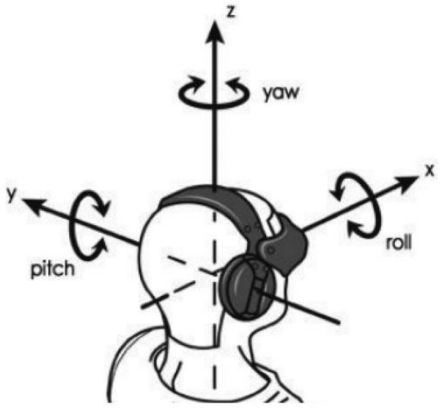
- The generator blocks are called **Gen0**, **Gen1**, **Gen2** and **Gen3**.

PWM in TivaWare™ Library (2/2)

- **SysCtlPeripheralEnable**(SYSCTL_PERIPH_PWM*x*): enables the PWM module *x* (i.e., PWM0 or PWM1).
- **SysCtlPWMClockSet**(SYSCTL_PWMDIV_*x*): sets the PWM clock as a ratio 1/*x* of the system clock (available *x*: 1/2/4/8/16/32/64) .
- **GPIOPinTypePWM**: sets pin(s) to output PWM signals.
- **GPIOPinConfigure**(GPIO_*pin*_M*x*PWM*y*): sets *pin* to output the signal *y* from the PWM module *x*.
- **PWMGenConfigure**: configures a PWM generator.
- **PWMGenPeriodSet**: sets the period of a PWM generator.
  - E.g., SysCtlClockGet()/64/50 (50 periods per second).
- **PWMGenPeriodGet**: gets the period of a PWM generator.
- **PWMGenEnable**: enables the timer/counter for a PWM generator.
- **PWMOutputState**: enables or disables PWM outputs.
- **PWMPulseWidthSet**: sets the pulse width for a PWM output.
  - E.g., PWMGenPeriodGet()*(2/20) (for 2 ms pulse / PWM period 20 ms).

Final Project: Laser Turret

- The laser turret consists of two **MG90S** servo motors: one on the **pitch**, the other on the **yaw** dimensions.



MG90S - Pitch (≈30˚ ~ ≈120˚)

MG90S - Yaw (0˚ ~ 180˚)

MG90S Servo Motor

- **MG90S** has a power wire, a ground wire, and a PWM signal wire.
- The PMW frequency should be 50 Hz (i.e., 20 ms period).

- The relationship between the angle θ (within 180°) and duty cycle ($\mathrm{D} = T_{On}/T$) is determined below:

$$\mathrm{T}_{On} = {}^{\theta}/_{90^{\circ}} + 0.5\ ms$$

0.5ms 1.0ms 1.5ms 2.0ms 2.5ms

500usec ---------- 0°

1000usec---------- 45°

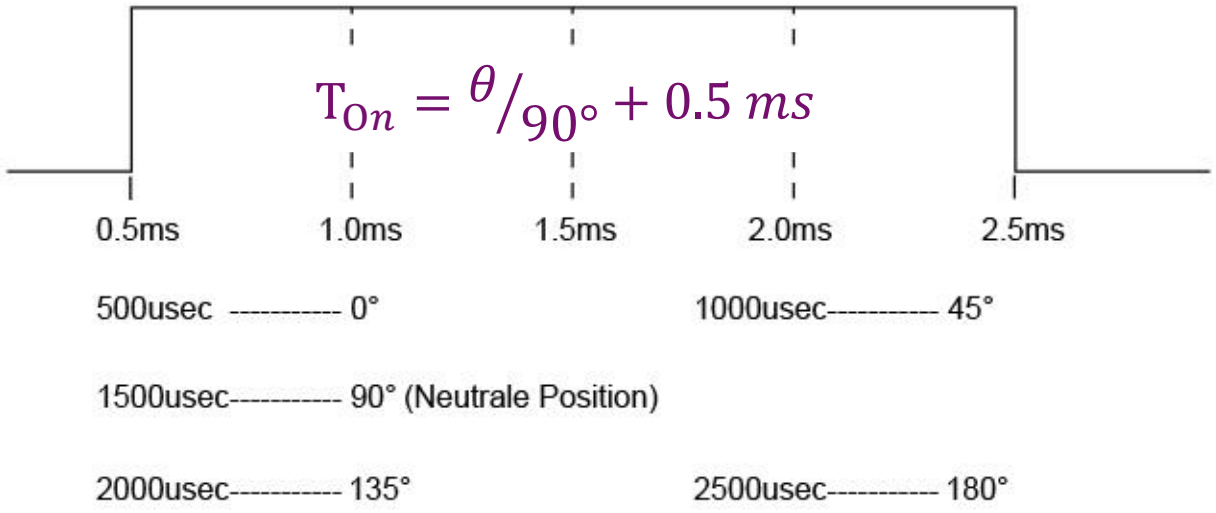
1500usec---------- 90° (Neutrale Position)

2000usec---------- 135°

2500usec---------- 180°

Demo Code: servo_demo.c (1/2)

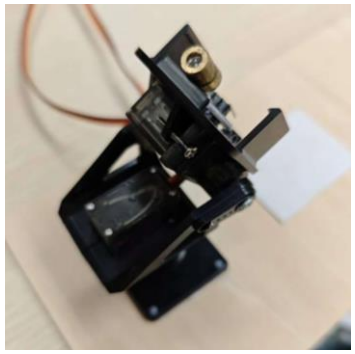
```c
float servo_pwm_freq = 50;

// determine the duty cycle according to the desired angle
float angleToPWMDutyCycle(float angle)
{
    // angle (duty cycle): 0 (0.5ms/20ms), 90 (1.5ms/20ms), 180 (2.5ms/20ms)
    // angle to pulse width: pulse_width = angle / 90 + 0.5
    // pulse width to duty cycle: duty_cycle = pulse_width / period
    // valid angle range: 0-180
    return (angle / 90 + 0.5) / (1000 / servo_pwm_freq);
}
```



Demo Code: servo_demo.c (2/2)

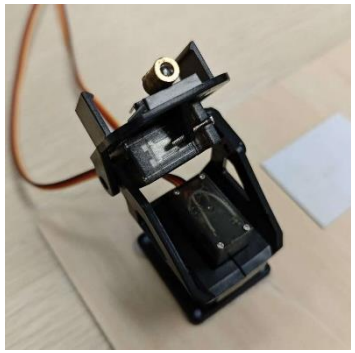
```
int main()
{
    // System clock & PMW configuration
    float pitch_angle, yaw_angle, pitch_duty_cycle, yaw_duty_cycle;

    // TODO: replace it with your own control pattern
    // ##### exemplary control pattern #####
    while (true)
    {
        yaw_angle = 0;
        yaw_duty_cycle = angleToPWMDutyCycle(yaw_angle);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, PWMGenPeriodGet(PWM1_BASE, PWM_GEN_0) * yaw_duty_cycle);
        SysCtlDelay(SysCtlClockGet() / 3);
        yaw_angle = 90;
        yaw_duty_cycle = angleToPWMDutyCycle(yaw_angle);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, PWMGenPeriodGet(PWM1_BASE, PWM_GEN_0) * yaw_duty_cycle);
        SysCtlDelay(SysCtlClockGet() / 3);
        yaw_angle = 180;
        yaw_duty_cycle = angleToPWMDutyCycle(yaw_angle);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, PWMGenPeriodGet(PWM1_BASE, PWM_GEN_0) * yaw_duty_cycle);
        SysCtlDelay(SysCtlClockGet() / 3);
        yaw_angle = 90;
        yaw_duty_cycle = angleToPWMDutyCycle(yaw_angle);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, PWMGenPeriodGet(PWM1_BASE, PWM_GEN_0) * yaw_duty_cycle);
        SysCtlDelay(SysCtlClockGet() / 3);
        pitch_angle = 50;
        pitch_duty_cycle = angleToPWMDutyCycle(pitch_angle);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, PWMGenPeriodGet(PWM1_BASE, PWM_GEN_0) * pitch_duty_cycle);
        SysCtlDelay(SysCtlClockGet() / 3);
        pitch_angle = 80;
        pitch_duty_cycle = angleToPWMDutyCycle(pitch_angle);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, PWMGenPeriodGet(PWM1_BASE, PWM_GEN_0) * pitch_duty_cycle);
        SysCtlDelay(SysCtlClockGet() / 3);
    }
    // ##### exemplary control pattern complete #####
}
```

Expected Results



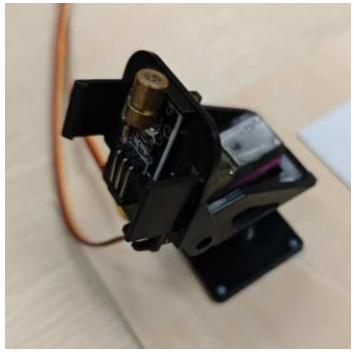
yaw: 0°

pitch: 60°



yaw: 90°

pitch: 90°



yaw: 180°

Summary

- **Motion Sensor**
  - Gyroscope & Accelerometer
  - Motion Processing Unit (MPU)
  - TivaWare™ Driver vs. Sensor Library
  - Demo Code: mpu_demo.c
- **Electric Motor**
  - Servo Motor
  - Pulse-Width Modulation (PWM)
  - PWM Module on Tiva™
  - PWM in TivaWare™ Library
  - Demo Code: servo_demo.c

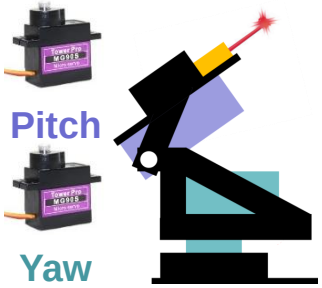
FP: Motion-Controlled Lazer Turret

MPU6050

I2C

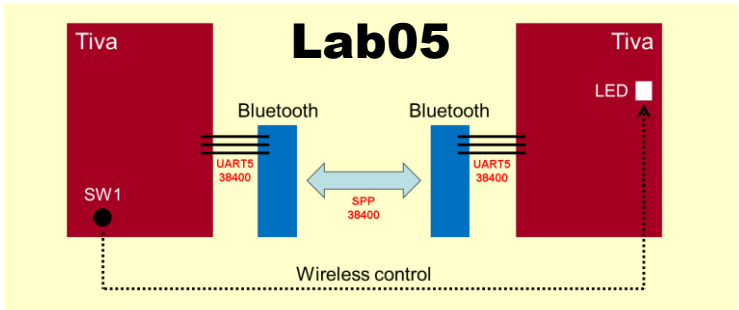
Pitch

Yaw



Lazer Turret

PWM



mpu_demo.c

Tiva

- **`fAccel[0]`**: acceleration along the X-axis (m/s²)
- **`fAccel[1]`**: acceleration along the Y-axis (m/s²)
- **`fAccel[2]`**: acceleration along the Z-axis (m/s²)
- **`fGyro[0]`**: angular velocity around the X-axis (rad/s)
- **`fGyro[1]`**: angular velocity around the Y-axis (rad/s)
- **`fGyro[2]`**: angular velocity around the Z-axis (rad/s)

Bluetooth

UART5
38400

SPP
38400

Bluetooth

UART5
38400

servo_demo.c

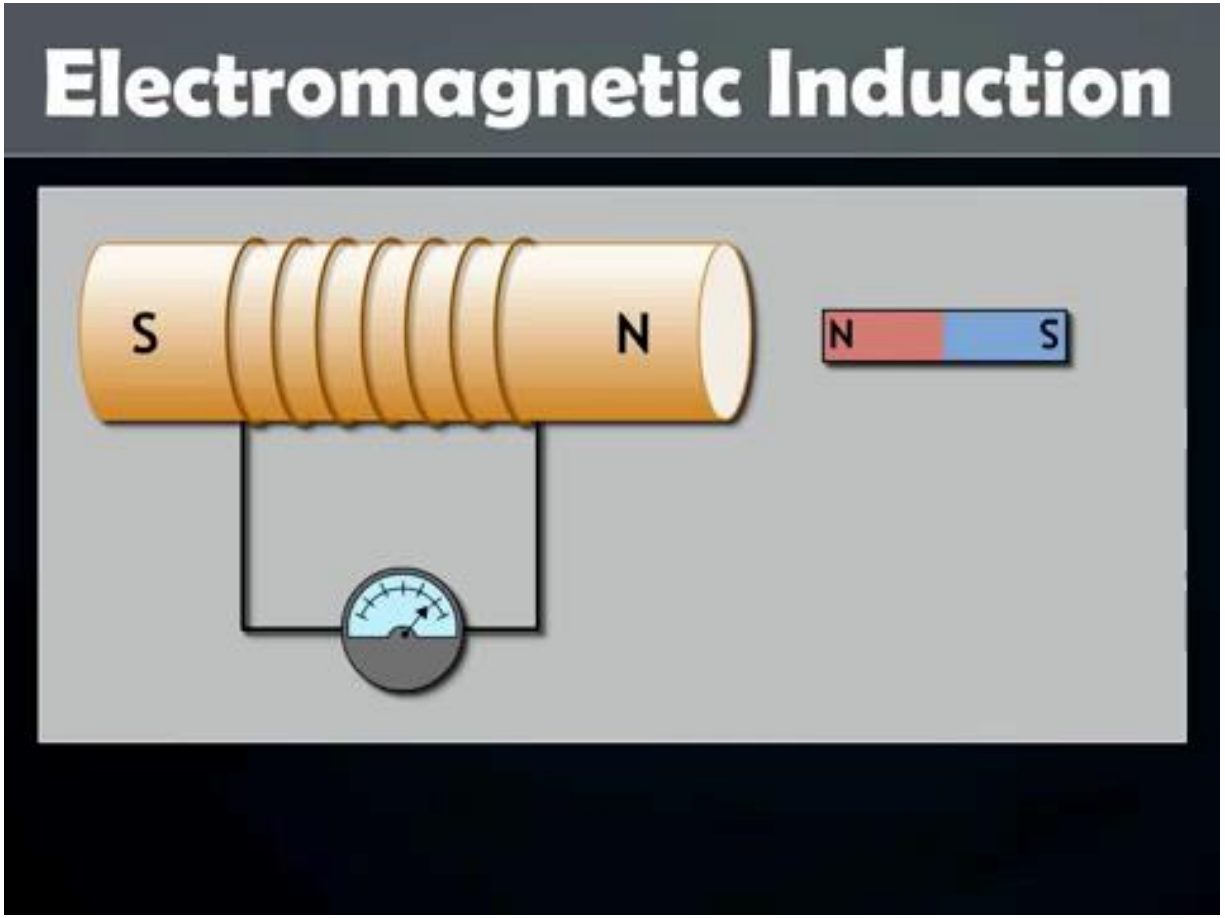
Tiva

```
// pitch
pitch_duty_cycle =
angleToPWMDutyCycle
(pitch_angle);
PWMPulseWidset
(pitch_duty_cycle);
// yaw
yaw_duty_cycle =
angleToPWMDutyCycle
(yaw_angle);
PWMPulseWidset
(yaw_duty_cycle);
```

Wireless control

Appendix: Electromagnet Induction

- The production of an electromotive force across an electrical conductor in a changing magnetic field.



https://www.youtube.com/watch?v=3HyORmBip-w