PSoC 6 101 Training

1. Introduction

* Introduction to PSoC 6 and PSoC Creator
* Dual Core features of PSoC 6 using cortex-m0+ and cortex-m4 cpu.
* Hello world program in PSoC creator for blinking LED in 3 modes

1. Without CPU
2. From Firmware in M0 core
3. from M4 core .

* Introduction to PDL libraries.
* The M0 and M4 cores in PSoC 6 use the same registers and buses allowing for resource sharing between the 2 core. This enable application developers to partition their code to the core that best suits the needs.

1. Free RTOS

* Free RTOS is used in the projects developed in this tutorial for handling multiple tasks with ease and without delay in real time.
* RTOS feature was added by selecting the checkbox for Free RTOS in the Peripheral driver library under the Build settings for the project.  
  The Macros in the FreeRTOSConfig.h file is to be modified to enable semaphores and the heap size was specified to be 48 x 1024 bytes.  
  NOTE: Comment or remove the #warning in this file.  
  Free RTSO Task was created with semaphores to run UART task on key press. And print a message.
* Semaphores are enabled and used for ease of resource sharing between tasks. This allows those tasks which has no events occurring for them be put to sleep so as to execute other time critical tasks. For example, here the UART Task is often put to sleep using the xSemaphoreTake() API routine which waits for the uartSemaphore semaphore to be given from the uart ISR that is triggered only when data is received in the UART receive buffer.
* Event groups are used to indicate if an even has occurred or not. Here, this allows to notify observation tasks of change in the data that they are responsible for monitoring which allows them to get the latest data and update the data base with the same. For example, to inform EZI2Cand BLE tasks of the change in the motor PWM duty cycle which can then be communicated to user via the BCP and notify the CySmart application respectively.
* Message queues are used to exchange data between multiple tasks .For example, here this is used to communicate the new PWM duty cycle percentage generated by the capsense or IMU tasks to the motor task which can then set the PWM on the motor pins to control the motor position

1. UART and Retarget IO

* A basic test firmware was developed to test and verify the operation of printf using the UART with a terminal application like tera term.
* The UART component was used with FreeRTOS where a UartTask() Task handles the UART receive operations using interrupts while the Retarget IO feature handles the transmission and reception of data over UART using the printf and getchar APIs.
* The peripheral driver library “Retarget io” was enabled in the projects build settings to re route the messages from printf() to UART on SCB5 for displaying the messages in terminal application.
* The retarget io feature is enabled in all projects in this tutorial as well for printing out debug messages and receiving user commands from UART.
* The receive functionality over UART was also verified where an interrupt is triggered each time data is available in the UART buffer and the same is retrieved from the buffer using the getchar() routine.
* The receive interrupt performs the operation of giving the semaphore which allows the UART Task loop to retrieve the data from the UART buffer using the getchar() API.

1. EZI2C

* Basic EZI2Cprogram was developed and same was used to send and retrieve data from the PSoC 6 using the bridge control panel.
* Chart feature of Bridge control panel was used to observe the changes in the values retrieved from PSoC using the EZI2C.
* The EZI2Cfeature was integrated into the main controller project to observe the motor speed as a graph in the charts tab of the bridge control panel.

1. Logical Kill Switch

* Implemented a kill switch circuit using logic components in thetop schematic of PSoC creator which makes use of digital multiplexer, not gates and a t-flipflop to stop PWM’s controlling motor operation by enabling and connecting the t-flipflop output to the kill pin of each of the PWM’s.
* The kill switch makes use of PWM to blink a red led when enabled and solid green led when disabled by clicking on the user switch sw2 of the pioneer BLE kit.
* The project source can be found under [BasicKillSwitch\_UDB.cydsn](https://github.com/Denzerek/BLEControlledArm/tree/main/BLE_ARM_Workspace/BasicKillSwitch_UDB.cydsn) in the GitHub repository.

1. Capsense

* The capsense block was used to develop a basic program to vary the PWM duty cycle of the signal fed to an RGB led.
* The capsense linear slider and 2 buttons were used.
* Additional gesture controls were enabled, specifically the flick gesture on the slider which allows user to switch the color on the RGB led between blue ,green or red by flicking on the slider to the left or right.  
  After selecting the required led color by flicking, the slider can be used to adjust the led brightness by slowly sliding the users finger on the slider.
* Clicking on the capsense buttons can start or stop the PWM that controls the RGB pins.
* When PWM is disabled, the Blue RGB is on.
* Here additionally, UART is enabled with retarget IO to print out the different gestures that are detected by the capsense module for learning purposes.
* The project source can be found under [CapSensing\_basic.cydsn](https://github.com/Denzerek/BLEControlledArm/tree/main/BLE_ARM_Workspace/CapSensing_basic.cydsn) in the GitHub repository.

1. Bluetooth 5.0 Peripheral

* A basic program to control the brightness of an RGB LED over BLE was developed.
* The BLE 5.0 component was added in the top schematic and configured as a peripheral device by check marking the ‘peripheral’ checkbox in the general tab.
* An LED service that implements a characteristic called GREEN which is given read and write permissions was created in the ‘Gatt Settings’ tab.
* The device was given the name PSoC6 in the general section of the ‘Gap Settings’ tab to identify the advertisement packet in the CySmart application.
* The discovery mode was set to general with no timeout so its easy to find and connect to the device in the ‘Advertisement Settings’ of ‘Gap Settings’ tab without the advertisement being disabled after the timeout period.
* This project can be used in combination with the BLECentralBasic project.
* The advertisement packet was configured in the ‘Gap Settings’ tab to contain the device name and the LED service UUID.
* API Description:

1. genericEventHandler():  
   To handle generic Bluetooth events like device connect, disconnect and write events.  
   When the device is disconnected, the ‘Blink’ PWM component is started which will blink the RED RGB led and the BLE advertisement is started  
   When the deivce is connected to, the ‘Blink’ PWM component is stopped and the ‘PWM\_DIM’ component is started which causes the green RGB led to glow.  
   A write operation to the ‘GREEN’ characteristic from the connected device, in this case CySmart BLE application, with an appropriate hex value changes the brightness of the RGB LED. (allowed write value range from 0x00-0x64)
2. bleInterruptNotify():  
   This routine is registered as the BLE application host callback for releasing the RTOS semaphore for processing BLE events when any BLE events occur.
3. bleTask();  
   This task is executed by the xTaskCreate() and vTaskStartScheduler() in the main routine to start the BLE stack and subsequent connect, disconnect and write event processes.

* The project source can be found under [CustomBLEBasic.cydsn](https://github.com/Denzerek/BLEControlledArm/tree/main/BLE_ARM_Workspace/CustomBLEBasic.cydsn) in the GitHub repository.

1. Bluetooth 5.0 Central

* A basic program that uses the BLE component as central device to connect with a BLE peripheral device was developed.
* It uses the same LED service that was used in the [CustomBLEBasic.cydsn](https://github.com/Denzerek/BLEControlledArm/tree/main/BLE_ARM_Workspace/CustomBLEBasic.cydsn) project. (By saving this service and then loading the same in this BLE central project so that the service with the same UUID is available in both projects.). This makes it possible for the firmware to access the UUID of the LED service via ‘cy\_ble\_customCServ[CY\_BLE\_CUSTOMC\_LED\_SERVICE\_INDEX].uuid’ and then compare this UUID with the one that was obtained after scanning the surrounding BLE devices advertisement packets. If the comparison returns true, then we have found the device that we have to connect to and can successfully accomplish the same.
* API Description:

1. genericEventHandler():  
   The stack on and device disconnect events will start the BLE GAP scan operation and at the same time turn on the red led ,to indicate device disconnection, and turns off the green led.  
   The scan progress result event is triggered when a device is detected in the scan operation and a compare operation on the UUID of the LED service is performed with the UUID of the service that was detected. If the UUID’s match, then a connect operation is performed and the scan operation is stopped.  
   Additional BLE events for unsuccessful/successful write, discovery status and connect/disconnect events are also handled.
2. bleTask()  
   It registers the genericEventHandler() and processes the BLE events in an infinite loop.  
   It also monitors the UART SCB5 for incoming characters ‘+’ and ‘-‘ to increase and decrease the LED brightness.
3. writeLED():  
   This routine takes the brightness value and sends the same to the GREEN characteristic in LED service .

* The project source can be found under [BLECentralBasic.cydsn](https://github.com/Denzerek/BLEControlledArm/tree/main/BLE_ARM_Workspace/BLECentralBasic.cydsn)in the GitHub repository.

1. BMI160 Motion Sensor

* A basic program to read the accelerometer data (ax,ay and az) from the BMI 160 IMU from Bosch was developed.
* The small, low power BMI160 is a low noise 16-bit IMU designed for mobile applications such as AR or indoor navigation, providing highly accurate sensor data and real-time sensor data. The low current consumption of BMI160 enables always-on applications in battery-driven devices.
* The IMU is present on the E-INK display shield that comes along with the 062-BLE Pioneer kit.
* The primary communication with the IMU is via i2c and the same was achieved using the I2C component in the top schematic.
* The I2C component was configured for operation at 100Kbps and as a master.
* The [BMI 160 Sensor API files](https://github.com/BoschSensortec/BMI160_driver) were downloaded from the Bosch GitHub page and the same was integrated into the PSoC Creator project by adding the path of the API files in the Build Settings > CM4 ARM GCC > Compiler > Additional Include Directories.
* .c and .h files downloaded for the IMU from the GitHub page was added into the project using the ‘Add existing Files’ option.
* I2C Burst read and burst write functions were developed using the I2C PDL API functions of master read and write and the same was registered into the bmi160Dev.read and bmi160Dev.write variables for use by the BMI160 driver files to send and receive data from the IMU.
* An additional I2C status decoder, i2cStatusDecode(), was also developed to decode and display the i2c status codes that may be returned by the i2c bus in case of errors.
* The project source can be found under [MotionControlBasic.cydsn](https://github.com/Denzerek/BLEControlledArm/tree/main/BLE_ARM_Workspace/MotionControlBasic.cydsn) in the GitHub repository.

1. Arduino Communication

* A basic i2c Communication firmware to send i2c commands to an Arduino uno using an ATMEGA328P was developed in PSoC firmware using the I2C component in the top design schematic.
* The firmware on Arduino side was developed to configure the same as a slave device and receive i2c data based on interrupts.
* The 062-BLE Pioneer kit was configured as a master device with 100Kbps baud and the pins P9\_0 and P9\_1 of the kit on header J1 was connected to the I2c pins of the Arduino uno board.
* This communication was tested to control the servo motors from the Arduino board which has 5V output on its pins which is suitable for the servo motor PWM control (PSoC Pioneer kit is capable of upto 3.3V on its GPIO pins which is not suitable for the SG90 servo motors that require 5 V )
* The Source code for communicating with the Arduino board can be found in [ArduinoCommI2c.cydsn](https://github.com/Denzerek/BLEControlledArm/tree/main/BLE_ARM_Workspace/ArduinoCommI2c.cydsn) project.
* The firmware for the Arduino uno that is compatible with this project can be found in the [ServoControl](https://github.com/Denzerek/BLEControlledArm/tree/main/ServoControl) project.

1. PSoC 6 BLE ARM Project

In addition to the above learnings from the PSoC 6 101 tutorial, the project [PSoC-6-BLE-ARM](https://github.com/Denzerek/PSoC-6-BLE-ARM) was developed integrating all the know how that was obtained from the basic projects that were mentioned above.  
Please refer to the github page [PSoC-6-BLE-ARM](https://github.com/Denzerek/PSoC-6-BLE-ARM) for the demo video of this project.  
The 3 main project firmware’s that were developed as part of the project is described as follows (Can be found under the folder [MainProject](https://github.com/Denzerek/PSoC-6-BLE-ARM/tree/main/MainProject) in the GitHub page):

1. **BT Central Remote**
   * The [BT\_Central\_Remote.cydsn](https://github.com/Denzerek/PSoC-6-BLE-ARM/tree/main/MainProject/BT_Central_Remote.cydsn) **firmware was d**eveloped in PSoC Creator.
   * To be loaded in the PSoC 6 BLE Pioneer Kit board that is to be used as the wireless remote.
   * The BLE Pioneer Kit to which this firmware is loaded should have the E-INK display attached to the same to communicate with the BMI160 IMU to get accelerometer data.
   * Implements Bluetooth configured as central device for establishing connection to peripheral device based on compare match of Motor service UUID.
   * Cannot be used as standalone.
   * Implements firmware for data acquisition from BMI160 motion sensor from Bosch (Onboard the E-Ink display that comes packaged along with the BLE Pioneer kit) and Capsense based touch sensors from Infineon (Onboard the BLE Pioneer kit)
   * FreeRTOS for syncing BLE write operation handles from the capsense and motion sensor applications.
   * Capsense task is capable of changing the selected servo motor by clicking on the capsense buttons while the finger position on the slider will change the position of the selected servo motor.
   * If there is no movement of this board for more than 3 seconds, then motion sensor is deactivated and capsense touch sensor will be active. At this time the user can change the motor position using the slider and select the motor whose position is to be changed using the capsense buttons.
   * When the selected motor is changed via the capsense buttons, the current selected motor will be printed out in the UART.
   * User can also send UART commands from a terminal application as well to change the position of each motor as a relative to its current position. The following commands can be sent via UART
     + 'o' - Move motor 1 by -10%
     + 'p' - Move motor 1 by 10%
     + 'k' - Move motor 2 by -10%
     + 'l' - Move motor 2 by 10%
     + 'n' - Move motor 3 by -10%
     + 'm' - Move motor 3 by 10%
     + ',' - Move motor 4 by -10%
     + '.' - Move motor 4 by 10%
2. **BT Peripheral Controller**
   * The [BT\_Peripheral\_Controller.cydsn](https://github.com/Denzerek/PSoC-6-BLE-ARM/tree/main/MainProject/BT_Peripheral_Controller.cydsn) firmware was developed in PSoC Creator.
   * To be loaded in the PSoC 6 BLE Pioneer Kit board that is to be used as the central controller for communicating with the Servo slave over I2C
   * Implements Bluetooth configured as a peripheral device for starting advertisement for the central device to detect and connect to.
   * This firmware can be used as a standalone one where the user can connect to the 'BLEArm' bluetooth advertisement from an android phone using the CySmart application and write the positions of each motor from the motor characteristics for each of the 4 motors.
   * Capsense task is capable of changing the selected servo motor by clicking on the capsense buttons while the finger position on the slider will change the position of the selected servo motor.
   * When the selected motor is changed via the capsense buttons, the current selected motor will be printed out in the UART.
   * User can also send UART commands from a terminal application as well to change the position of each motor as a relative to its current position.The following commands can be sent via UART
     + 'o' - Move motor 1 by -10%
     + 'p' - Move motor 1 by 10%
     + 'k' - Move motor 2 by -10%
     + 'l' - Move motor 2 by 10%
     + 'n' - Move motor 3 by -10%
     + 'm' - Move motor 3 by 10%
     + ',' - Move motor 4 by -10%
     + '.' - Move motor 4 by 10%
   * I2C Configured for 400Khz in master mode to send I2C commands to Arduino Slave.
   * I2C Command packet is designed as follows : <command><motor\_num><position\_percent>0D
3. **Servo Control**
   * The [ServoControl.ino](https://github.com/Denzerek/PSoC-6-BLE-ARM/tree/main/MainProject/ServoControl) firmware for Arduino Uno was developed in Arduino IDE
   * Uses the wire.h library for I2C communication in slave mode and the servo.h library for servo motor pwm control.
   * Slave address is set as 0x08
   * I2C Command packet is designed as follows : <command><motor\_num><position\_percent>0D
   * Only the I2C command 0x0C is supported for now. Additional commands can be added to the switch routine later as needed.
   * Uart display of motor positions enabled.