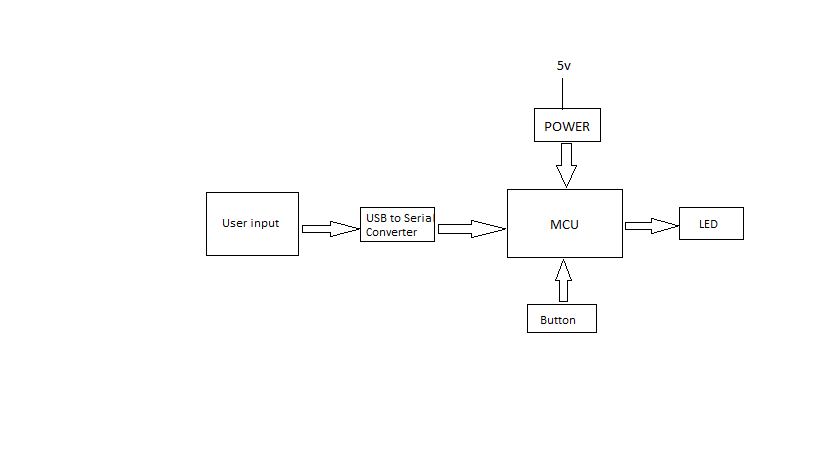
**Firmware architecture document**

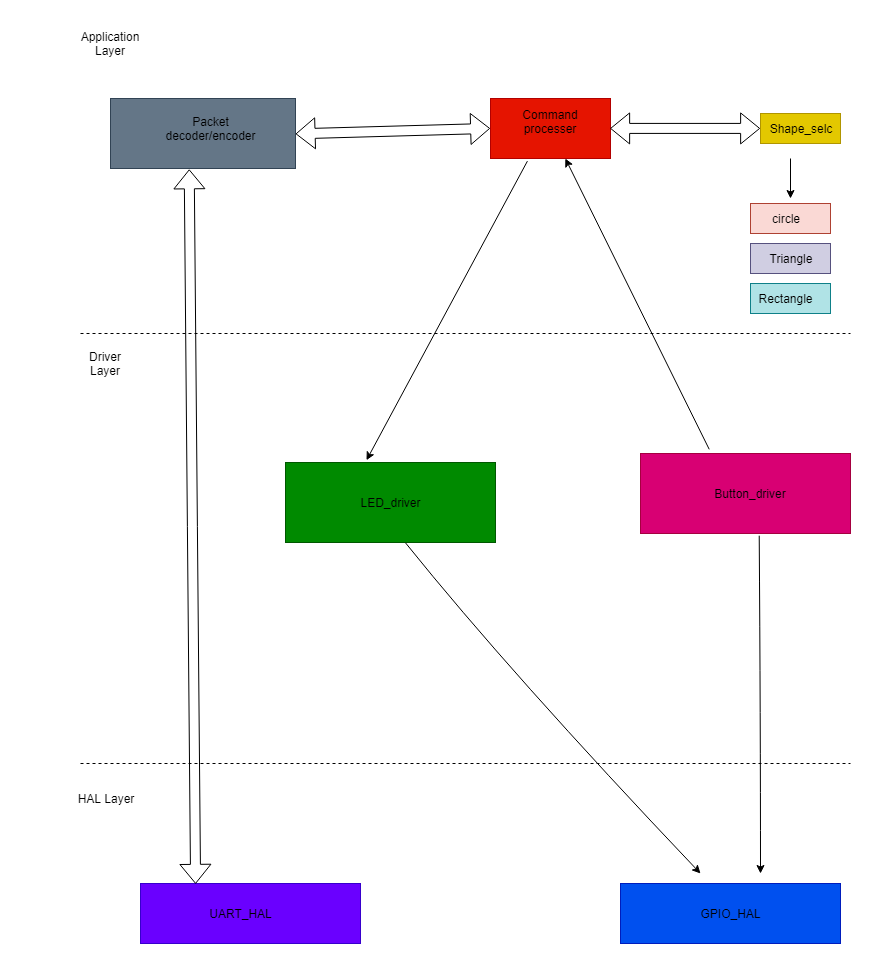
1. **Requirements**

* Turn On and turn off LEDs in the discovery board using serial commands.
* Blink LEDs three times whenever a button is pressed
* **For a given circle**, it is required to find if a given point exists within the circle or outside of the circle.

Ex. Circle is a 4cm radius and x,y coordinates of the center point is 5,1. If we define point in the plain as 4,1, algorithm is required to indicate if the point is inside the circle area or outside from the circle area.

**It should be able to provide an interface (UART command) when the circle parameters and point parameters provided, response should indicate if the point is inside the circle or outside.**

1. **Hardware Block Diagram**
2. **Firmware Block Diagram**

****

***Using event-driven firmware architecture***

The main while loop is a idling loop with a delay function. Where the delay function is constantly interrupted by the events like button press event, UART receive event and timer interrupt event in the led driver.

1. **Functions exposed by Application layer to the user**

/\*

\* @file : app.c

\* @brief :These functions are used to Initiate the necessary hardware

\*/

void Application\_Init();

/\*

\* @file : app.c

\* @brief This function is used to process user commands and then return the results

\*/

void handle\_user\_requests(SERIAL\_RX\_PACKET\_t\* request);

/\*

\*@brief This function calculate if a given point is inside or outside the circle

\*@param - x,y position of a point

\*@return - 1 if point is inside the circle 0 otherwise

\*/

uint8\_t calculate\_position(int x, int y);

**4.1 Packet handler Functions**

Packet handler use the UART interrupt to get notified on each byte reception. That byte is fed to a packet decoder function. This packet decoder function is a state machine which changes the state according to the received byte

/\*

\* @file :peacket\_handler.c

\* @brief Transmit complete callback function pointer

\*/

typedef void (\*transmit\_cmplt\_cb) (uint32\_t status);

/\*@brief Receive complete callback function pointer\*/

typedef void (\*receive\_cmplt\_cb) (SERIAL\_RX\_PACKET\_t\* tl\_packet, uint32\_t error);

/\*@brief initiate packet hander \*/

void packethandler\_init(PH\_CALLBACKS\_t \* cb);

/\*@brief transmit packet\*/

void packethandler\_transmit\_packet(void);

/\*@brief Get the status of the packet handler\*/

void PH\_STATUS\_t ph\_get\_status(void)

/\*\* Packet structure\*/

typedef struct {

char sb1;

char sb2;

char sb3;

char sb4;

char sig;

char type;

char length;

char data[7]; // 7 data bytes

char low\_crc;

char high\_crc;

}SERIAL\_RX\_PACKET\_t;

/\* @brief packet handler status are stored in the structure defined bellow \*/

typedef enum{

PH\_STATUS\_READY\_TO\_SEND = 0,

   PH\_STATUS\_TRANSMITTING

}PH\_STATUS\_t;

/\* @brief packet handler application callbacks are stored in the structure defined bellow \*/

typedef struct {

transmit\_cmplt\_cb\_t tx\_complete\_cb;

receive\_cmplt\_cb\_t  rx\_complete\_cb;

}PH\_CALLBACKS\_t;

/\* @brief packet collector application status are stored in the structure defined bellow\*/

typedef enum{

   /\*\* Waiting for start byte 1 \*/

   PACKET\_COLLECTOR\_STATE\_STATE\_STX\_1 = 0,

   /\*\* Waiting for start byte 2 \*/

   PACKET\_COLLECTOR\_STATE\_STATE\_STX\_2,

   /\*\* Waiting for start byte 3 \*/

   PACKET\_COLLECTOR\_STATE\_STATE\_STX\_3,

   /\*\* Waiting for start byte 4 \*/

   PACKET\_COLLECTOR\_STATE\_STATE\_STX\_4,

   /\*\* Waiting for packet signature 1 \*/

   PACKET\_COLLECTOR\_STATE\_PKT\_SIG,

   /\*\* Waiting for packet type \*/

   PACKET\_COLLECTOR\_STATE\_TYPE,

   /\*\* Waiting for low byte of packet length \*/

   PACKET\_COLLECTOR\_STATE\_STATE\_LENGTH,

   /\*\* Collecting data bytes \*/

   PACKET\_COLLECTOR\_STATE\_STATE\_DATA,

   /\*\* Waiting for low byte of 16bit CRC \*/

   PACKET\_COLLECTOR\_STATE\_STATE\_CRC\_L,

   /\*\* Waiting for high byte of 16bit CRC \*/

   PACKET\_COLLECTOR\_STATE\_STATE\_CRC\_H,

}PACKET\_COLLECTOR\_STATE\_t;

/\*  
\* @brief This function initialize the packet handler.  
\*@param cb - pointer to a structure where holds the callback functions to be called at  
\* different occasions

\*/  
void ​ph\_init(​PH\_CALLBACKS\_t​ \* cb);

/\* @brief This function reads the receved bytes from a rx\_buffer and put them in to the packet structure \*/

SERIAL\_RX\_PACKET\_t packet\_collector(char​\* buff,​ SERIAL\_RX\_PACKET\_t​\* tl\_packet);

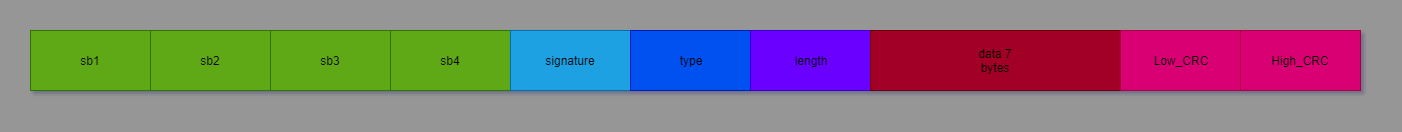
/\* @brief This function transmit uart packet by adding headers and all other thigs \*/

void ​ph\_transmit\_packet(​char\* result​);

/\* @brief This function return the status of the packet handler \*/

PH\_STATUS\_t ​ph\_get\_status(​void​)

**packet structure**

****

1. **Functions exposed by Driver layer to application layer**
   1. **LED driver**

To invoke different led functions, like blink 3 times and turn it on and off, timer interrupt is used. At each timer interrupt, we check the what function is given by the application

/\* Enumeration to define led, in future we can add more led \*/

typedef enum {

  LED\_1 ,

LED\_2 ,

}LED\_t;

/\* Enumeration to define different led functions \*/

typedef enum {

  LED\_FN\_ON = 0,

  LED\_FN\_OFF,

LED\_FN\_BLINK\_3T

} LED\_FN\_t;

/\*

\* @file : led.c

\*@brief LED init function

\*/

Int32\_t led\_init(void);

/\*

\* @file : led.c

\* @brief set the different function of led

\* @param

\* led -> led which using

\* function -> function of the led

\*/

int32\_t led\_set(LED\_t led, LED\_FN\_t function);

* 1. **Button driver**

Button event is implemented as an external interrupt. Application can register a callback function to get notified when a button is pressed or released.

/\* Enumuration to define buttons, in future we can add more buttons \*/

typedef enum{

   BUTTON\_1 ,

BUTTON\_2,

}BUTTON\_ID\_t;

/\* Enumuration to define buttons events \*/

typedef enum{

   PRESSED = 0,

   RELEASED,

}BUTTON\_EVENT\_t;

/\* @brief Function callback prototype where the application can register different callbacks for different functions \*/

typedef void (\*BTN\_CALLBACK\_f) (BUTTON\_ID\_t button\_id, BUTTON\_EVENT\_t button\_event );

/\* @brief initialize the button \*/

int button\_init(void);

/\* @brief Application use this function to register different callbacks for different button events \*/

int32\_t button\_register\_callback(BUTTON\_ID\_t button\_id, BTN\_CALLBACK\_f cb);

1. **Functions exposed by HAL layer to driver layer**
   1. **GPIO\_HAL**

HAL\_GPIO\_Init();

HAL\_GPIO\_Deinit();

HAL\_GPIO\_Readpin();

HAL\_GPIO\_Writepin();

HAL\_GPIO\_Togglepin();

HAL\_GPIO\_EXTI\_Irqhandle();

* 1. **UART\_HAL**

HAL\_UART\_Init();

HAL\_UART\_Transmit();

HAL\_UART\_Receive();

HAL\_UART\_Irqhandle();