MSF_XINPUT

USER GUIDE

Zachery Littell

Version 1.0

This is the user guide for the functions included in the MSF_Xinput library for the TeensyLC. I will do my best to keep this user guide updated as the library is updated and additional functionality is created. If you have any questions regarding the implementation of this library and not just explanations of its functionality, I have included two examples (a simple one and my own fightstick) to show how you can start working with my library.

XINPUT

XINPUT(uint8_t LEDMode)

XINPUT(uint8_t LEDMode, uint8_t LEDPin)

This is the main constructor. The user must create an instance of this to use the functions available from the library in their code.
There are two ways to call this function. One takes a single argument (used to select NO LED mode) and the other takes two arguments (used to select an LED mode and a pin to use as the LED).
There are currently the following possible values for LEDMode:

NO_LED

Assigns the mode to none and sets the pin to 0

LED_ENABLED

Assigns the mode to enabled and sets the pin according to the second argument passed

Example:

Initialize controller as a class XINPUT with the LED disabled

```
XINPUT controller(NO_LED);
```

Initialize controller as a class XINPUT with the LED enabled and the LED pin set to Pin13

```
XINPUT controller(LED_ENABLED, 13);
```

buttonUpdate

buttonUpdate(uint8_t button, uint8_t buttonState)

This function updates a single button's state. The function takes the button and the button state as it's two arguments.
The possible values for the button argument are as follows:

BUTTON_A
BUTTON_B
BUTTON_X
BUTTON_Y
BUTTON_LB
BUTTON_RB
BUTTON_L3
BUTTON_R3
BUTTON_START
BUTTON_BACK
BUTTON_LOGO

buttonState can either be a 1 or a 0 depending on the state you would like the button to appear to be in. 1 being pressed and 0 being released.

Example:

Use buttonUpdate to update the A button to being pressed for controller.

```
controller.buttonUpdate(BUTTON_A, 1);
```

buttonArrayUpdate

buttonArrayUpdate(uint8_t buttonArray[11])

This function is used to update all of the buttons at once using an array of values. It can be more convenient than updating each button one at a time. You can handle all button processing ahead of time and make a single function call to update the button packets.
The function is passed a single 11 element array of unsigned 8bit integer type. The array must be in the following specific order:
[0] A Button
[1] B Button
[2] X Button
[3] Y Button
[4] LB Button
[5] RB Button
[6] L3 Button
[7] R3 Button
[8] START Button
[9] BACK Button
[10] LOGO Button
Set each element to 1 to represent being pressed and 0 to represent not being pressed.
Example:

Use buttonArrayUpdate to set RB and X as being pressed while the rest of the buttons remain unpressed.

```
uint8_t buttonStatus[11];
int x;
for (x = 0; x<11; x++){buttonStatus[x]=0;}
buttonStatus[2] = 1;
buttonStatus[5] = 1;
buttonArrayUpdate(buttonStatus);
```

dpadUpdate

dpadUpdate(uint8_t dpadUP, uint8_t dpadDOWN, uint8_t dpadLEFT, uint8_t dpadRIGHT)

This function can be used to update the status of the directional pad on the controller. The function takes 4 arguments for each direction with a type of unsigned 8bit integer. The order is Up, Down, Left, Right. The function includes an SOCD cleaner with programmed behavior of UP+DOWN = UP and LEFT + RIGHT = NEUTRAL. This allows the code to be used as a tournament legal controller.
Setting the direction to 1 indicates a pressed state while 0 indicates a released state.
Example:

Represent user pressing UP and RIGHT at the same time.

```
uint8_t up = 0;
uint8_t down = 0;
uint8_t left = 0;
uint8_t right = 0;
up = 1;
right = 1;
dpadUpdate(up, down, left, right);
```

triggerUpdate

triggerUpdate(uint8_t triggerLeftValue, uint8_t triggerRightValue)

This function is used to update both trigger values at the same time. It passes 2 arguments that are both 8 bit unsigned integers with an order of Left Trigger and then Right Trigger. With the values being 8 bit unsigned integers you have possible values of 0-255.

Example:

Represent the left trigger being pressed halfway while the right trigger is fully pressed.

```
uint8_t left = 127;
uint8_t right = 255;
triggerUpdate(left, right);
```

singleTriggerUpdate

singleTriggerUpdate(uint8_t trigger, uint8_t triggerValue)

This function should be used if the user wishes to update the value of a single trigger at a time. The function passes 2 arguments that are both 8 bit unsigned integers with an order of which trigger to update and then the value of that trigger. The value for trigger can either be **TRIGGER_LEFT** or it can be **TRIGGER_RIGHT**. These are provided as defines for you to use, which should increase the readability of your codebase. The value of the trigger can be anything in the range of 0-255.

Example:

Set the left trigger to a value of 26.

```
uint8_t left = 26;
singleTriggerUpdate(TRIGGER_LEFT, left);
```

stickUpdate

stickUpdate(uint8_t analogStick, int16_t stickXDirValue, int16_t stickYDirValue)

This function updates each axis of the user specified analog stick. The function passes 3 arguments with varied types. The first argument is an unsigned 8 bit integer that specifies which stick the user is updating the values for. The two possible predefined values that can be put here are **STICK_LEFT** and **STICK_RIGHT**. The next argument is a signed 16 bit integer and represents the selected sticks X Directional Value. The final argument is also a signed 16 integer and represents the selected sticks Y Directional Value. The two directional value arguments accept a value of -32,768 to +32,767.

Example:

Set the Left Stick to have a value of 0 for its X and 32,767 for its Y. This puts the left stick at being pressed all the way to the right.

```
int16_t lsX = 0;
int16_t lsY = 32767;
stickUpdate(STICK_LEFT, lsX, lsY);
```

sendXinput

sendXinput()

This function is used to send an updated packet to the PC regarding the state of the controller. It should be called regularly in the main loop after you have checked to status of any inputs and correctly updated the status of any buttons, sticks, triggers, etc that you are using with your device. There are no arguments passed with this function.

Example:

```
sendXinput();
```

receiveXinput

receiveXinput()

This functions checks if there is a received packet waiting to be processed by the microprocessor. It should be called regularly in the main loop before calling any other functions that deal with data received from the PC host itself. The function includes code to parse and store rumble and LED pattern commands from the host. Although the function passed no arguments, it does return an unsigned 8 bit integer. This value represents the type of packet that was received from the host. The possible values are as follows:

0 – Packet Not Available
1 – Packet was a Rumble Command
2 – Packet was an LED Pattern Command
3 – Packet was a command that we currently do not parse

Example:

```
uint8_t rxPacketType = 0;
rxPacketType = receiveXinput();
```

setLEDMode

setLEDMode(uint8_t LEDMode)

setLEDMode(uint8_t LEDMode, uint8_t LEDPin)

This function is used to set the style of LED you would like to use and assign the pin the LED is attached to. The function should be passed a single argument of **NO_LED** if you wish to not use any LED to represent controller state. If you pass two arguments the first should be **LED_ENABLED** and the second should be the pin that you wish to use for the LED.

Example:

```
enable the LED and use the onboard LED.
    uint8_t pinLED = 13;
    setLEDMode(LED_ENABLED, pinLED);
```

LEDUpdate

LEDUpdate()

This function should be called regularly to update the current state of the LED if one is enabled. Failure to call this in a time efficient manner can cause patterns to display a bit skewed. The function takes the current time the processor has been running in mS and then subtracts the last time the pattern was updated. If this value is greater than 150 mS then the next step in the LED pattern is executed.

Example:

```
LEDUpdate();
```

LEDPatternSelect

LEDPatternSelect(uint8_t rxPattern)

This function can be used to select the LED display pattern. Normally this function is called by receiveXinput() and not by the user. However, it can be called by the user to overwrite the current pattern. Just note that when a new LED pattern is received it will overwrite your selected pattern as normal operation intended. The possible pattern values sent from the host are as follows:
0x00 – OFF
0x01 – ALL BLINKING
0x02 – 1 Flashes, then on
0x03 – 2 Flashes, then on
0x04 – 3 Flashes, then on
0x05 – 4 Flashes, then on
0x06 – 1 on
0x07 – 2 on
0x08 – 3 on
0x09 – 4 on
0x0A – Rotating (1-2-4-3)
0x0B – Blinking*
0x0C – Slow Blinking*
0x0D – Alternating (1+4-2+3)
*Does the pattern and then goes back to previous pattern
Currently the only patterns that are programmed corrected are
ALLBLINKING
ROTATING
FLASHON1
ON1
FLASHON2
ON2
FLASHON3
ON3
FLASHON4
ON4
Rotating and Blinking, FlashOn1 and On1, FlashOn2 and On2, FlashOn3 and On3, FlashOn4 and On4 each share a pattern to simplify the display to the user since a single LED is used and not a 4 LED ring like the XBOX controller itself.
Example:

LEDPatternSelect(FLASHON1);

bootloaderJump

bootloaderJump()

This function provides an easy built-in way for the user to jump to the bootloader on the TeensyLC. This can be used if the teensy will be hidden within a device and the bootloader button cannot be pressed easily. The user can check for a specific combination of inputs or even their own hardwired button and make a call to this function to jump to the bootloader.
Example:

bootloaderJump();