libDaisy

Contents

Core

Memory Section Macros

Macro for area of memory that is configured as cacheless This should be used primarily for DMA buffers, and the like.

THE DTCM RAM section is also non-cached. However, is not suitable for DMA transfers. Performance is on par with internal SRAM w/ cache enabled.

Helpful Functions

Cube

GPIO

Enums and a simple struct for defining a hardware pin on the MCU These correlate with the stm32 datasheet, and are used to configure the hardware.

dsy_pin

Helper for creating pins from port/pin combos easily

dsy_pin_cmp

Helper for testing sameness of two dsy_gpio_pins

returns 1 if same, 0 if different

```
FORCE_INLINE uint8_t dsy_pin_cmp(dsy_gpio_pin *a, dsy_gpio_pin *b)
```

Hardware related defines. keyboard switches shift register UART for MIDI via TRS jacks on Field CD4051 Select Pin controls enums for controls, etc. All knobs connect to ADC1_INP10 via CD4051 mux Init Daisy Seed Init Switches Init Gate Input Init Gate Output Init LED Driver 2x PCA9685 addresses 0x00, and 0x01 TODO: add multidriver support Init Keyboard Switches TODO: add cd4021 with parallel data support Init ADC (currently in daisy_seed). Set up mux pin Set up CV inputs Init all 5 channels Setup Knob/CV Analog Controls Mapped to ADCs Start timer

Daisy Patch BSP

Description

Class that handles initializing all of the hardware specific to the Daisy Patch Board.

Helper funtions are also in place to provide easy access to built-in controls and peripherals.

Credits

Author: Stephen Hensley **Date Added:** November 2019

Data Types

Ctrl

Enum of Ctrls to represent the four CV/Knob combos on the Patch

```
enum Ctrl
{
    CTRL_1,
    CTRL_2,
    CTRL_3,
    CTRL_4,
    CTRL_LAST,
};
```

Init

Initializes the daisy seed, and patch hardware.

```
void Init();
```

Audio Block size defaults to 48. Change it using this function before StartingAudio Hardware Accessors

Public Members

These are exposed for the user to access and manipulate directly

Helper functions above provide easier access to much of what they are capable of. TODO: Add class for Gate output daisy::Switch switches[SW_LAST];

enums for controls, etc. All knobs connect to ADC1_INP10 via CD4051 mux Hardware Accessors TODO breakout to individuals with an array of pointers. // Init Daisy Seed // TODO: decide if this should be a part of the bsp init, // or if users should have to init seed and board. daisy_seed_init(&p->seed);

// Init Switches uint8_t sw_pins[SW_LAST] = { SW_1_PIN, SW_2_PIN, SW_3_PIN, SW_4_PIN, SW_5_PIN, SW_6_PIN, SW_7_PIN}; dsy_gpio_port sw_ports[SW_LAST] = {SW_1_PORT, SW_2_PORT, SW_3_PORT, SW_4_PORT, SW_5_PORT, SW_6_PORT, SW_7_PORT}; for(uint8_t i = 0; i < SW_LAST; i++) { p->switches[i].Init({sw_ports[i], sw_pins[i]}, 1000.0f, daisy::Switch::TYPE_TOGGLE, daisy::Switch::POLARITY_INVERTED, daisy::Switch::PULL_UP); }

// Encoder // TODO Add Encoder support

// Init LED Driver // 2x PCA9685 addresses 0x00, and 0x01 uint8_t addr[2] = {0x00, 0x01}; dsy_led_driver_init(&p->seed.LED_DRIVER_I2C, addr, 2);

// ADC uint8_t channel_order[KNOB_LAST + CV_LAST] = { DSY_ADC_PIN_CHN10, DSY_ADC_PIN_CHN17, DSY_ADC_PIN_CHN15, DSY_ADC_PIN_CHN5, DSY_ADC_PIN_CHN7, DSY_ADC_PIN_CHN3, DSY_ADC_PIN_CHN4, // Expression 0-5V }; p->seed.adc_handle.channels = KNOB_LAST + CV_LAST; for(uint8_t i = 0; i < 5; i++) { p->seed.adc_handle.active_channels[i] = channel_order[i]; } dsy_adc_init(&p->seed.adc_handle);

Daisy Pod BSP

Description

Class that handles initializing all of the hardware specific to the Daisy Patch Board.

Helper funtions are also in place to provide easy access to built-in controls and peripherals.

Credits

Author: Stephen Hensley **Date Added:** November 2019 Functions Init related stuff. Audio Block size defaults to 48. Change it using this function before StartingAudio Hardware Accessors Public Members.

Daisy Seed

This is the higher-level interface for the Daisy board.

All basic peripheral configuration/initialization is setup here.

Initialization and Startup

Configure

configures the settings for all internal peripherals, but does not initialize them.

This allows for modification of the configuration handles prior to initialization.

Defaults listed below:

TODO: Add defaults

```
void Configure();
```

Init

Initializes the Daisy Seed and the following peripherals: SDRAM, QSPI, 24-bit 48kHz Audio via AK4556, Internal USB, as well as the built-in LED and Testpoint.

ADCs, DACs, and other special peripherals (such as I2C, SPI, etc.) can be initialized using their specific initializers within libdaisy for a specific application.

```
void Init();
```

Helpers and More

GetPin

Returns the gpio_pin corresponding to the index 0-31. For the given GPIO on the Daisy Seed (labeled 1-32 in docs).

```
dsy_gpio_pin GetPin(uint8_t pin_idx);
```

StartAudio

Begins the audio for the seeds builtin audio. the specified callback will get called whenever new data is ready to be prepared.

```
void StartAudio(dsy_audio_callback cb);
```

SetLed

Sets the state of the built in LED

```
void SetLed(bool state);
```

SetTestPoint

Sets the state of the test point near pin 10

```
void SetTestPoint(bool state);
```

AudioSampleRate

Returns the audio sample rate in Hz as a floating point number.

```
float AudioSampleRate();
```

SetAudioBlockSize

Sets the number of samples processed per channel by the audio callback.

```
void SetAudioBlockSize(size_t blocksize);
```

Public Members

While the library is still in heavy development, most of the configuration handles will remain public.

```
dsy_sdram_handle sdram_handle;
dsy_qspi_handle  qspi_handle;
dsy_audio_handle audio_handle;
dsy_sai_handle   sai_handle;
dsy_i2c_handle   i2c1_handle, i2c2_handle;
AdcHandle        adc;
```

```
dsy_dac_handle   dac_handle;
UsbHandle        usb_handle;
```

Codec AK4556

Description

Driver for the AK4556 Stereo Codec

Init

Resets the AK4556

reset_pin should be a dsy_gpio_pin that is connected to the RST pin of the AK4556

```
void codec_ak4556_init(dsy_gpio_pin reset_pin);
```

LED Driver

Description

Device driver for PCA9685 16-channel 12-bit PWM generator

TODO:

- Move color usage to util_color
- Add a function to dump out how long it takes to write all assigned LEDs.
- Fix update function so that it doesn't only write one device per call

Defines

Data

Preset Colors

```
enum
{
    LED_COLOR_RED,
    LED_COLOR_GREEN,
    LED_COLOR_BLUE,
    LED_COLOR_WHITE,
    LED_COLOR_PURPLE,
    LED_COLOR_CYAN,
    LED_COLOR_GOLD,
    LED_COLOR_OFF,
    LED_COLOR_LAST
};
```

Color

Simple color struct

Different from util_color only in type (0-4095 vs 0-1)

This could easily be migrated to work with those instead.

```
typedef struct
{
    uint16_t red, green, blue;
} color;
```

Functions

Init

Initializes the LED Driver(s) on the specified I2C Bus

- dsy_i2c should be any dsy_i2c_handle with pins and speed configured.
- addr is either a pointer to 1 device address, or an array of addresses for multiple devices
- addr_cnt is the number of addresses passed in (use '1' for a single device)

```
void   dsy_led_driver_init(dsy_i2c_handle *dsy_i2c,
                           uint8_t *       addr,
                           uint8_t         addr_cnt);
```

Update

Updates the LED Driver with the values set using the set function

Currently only updates one driver at a time due to the time it takes to update all of the devices.

This can likely be set up to use DMA so that the function doesn't block for so long.

```
void   dsy_led_driver_update();
```

set_led

sets the LED at the index to the specified brightness (0-1)

Index is sequential so device 0 will have idx 0-15, while device 1 will have idx 16-31, etc.

```
void   dsy_led_driver_set_led(uint8_t idx, float bright);
```

color__by__name

Passing in one of the preset colors will return a pointer to a color struct

```
color *dsy_led_driver_color_by_name(uint8_t name);
```

SDRAM

Description

SDRAM for 32MB AS4C16M16SA (and 64MB equivalent).

Thanks to whoever this awesome person is:

http://main.lv/writeup/stm32f4_sdram_configuration.md

The Init function is basically a copy and paste.

He has references to timing, etc.

RAM is configured at 100MHz (fastest possible on the MCU).

To use these the .sdram_data/_bss sections must be configured correctly in the LINKER SCRIPT. using BSS is advised for most things, since the

Errata:

- Data section init not properly set up, as SDRAM is not initialized until after startup code.

```
MEMORY
{
    SDRAM (RWX) : ORIGIN = 0xC0000000, LENGTH = 64M
}

SECTIONS
{





    } > SDRAM
}
```

Section Attributes

SDRAM_DATA Attribute

As mentioned above, this does not currently initialize correctly (startup

The variables placed here will also need to fit inside of the flash in order to initialize.

Usage: E.g. int DSY_SDRAM_DATA initialized_var = 1;

```
#define DSY_SDRAM_DATA __attribute__((section(".sdram_data")))
```

SDRAM_BSS

Variables placed here will not be initialized.

Usage E.g. int DSY_SDRAM_BSS uninitialized_var;

```
#define DSY_SDRAM_BSS __attribute__((section(".sdram_bss")))
```

SDRAM STATUS

```
enum
{
    DSY_SDRAM_OK,
    DSY_SDRAM_ERR,
};
```

SDRAM State

Determines whether chip is initialized, and activated.

```
typedef enum
{
    DSY_SDRAM_STATE_ENABLE,
    DSY_SDRAM_STATE_DISABLE,
    DSY_SDRAM_STATE_LAST,
} dsy_sdram_state;
```

Pin Config

For now this is the only supported pinswap

The Chipselect/bank could likely be changed, but we haven't had any hardware that does that yet.

Pins that have functions that cannot be moved to another pin will be hard-coded into the driver.

This is PH5 on Daisy

```
typedef enum
{
    DSY_SDRAM_PIN_SDNWE,
    DSY_SDRAM_PIN_LAST,
} dsy_sdram_pin;
```

Sdram Handle

Configuration struct for passing to initialization

```
typedef struct
{
    dsy_sdram_state state;
    dsy_gpio_pin    pin_config[DSY_SDRAM_PIN_LAST];
} dsy_sdram_handle;
```

Functions

init

Initializes the SDRAM peripheral

```
uint8_t dsy_sdram_init(dsy_sdram_handle *dsy_hsdram);
```

CD4021 Input Shift Register

Description

Device driver for the CD4021

Bit-banged serial shift input.

Defines

Fixed maximums for parallel/daisychained use

These could be expanded, but haven't been tested beyond this

```
#define SR_4021_MAX_PARALLEL 2
#define SR_4021_MAX_DAISYCHAIN 1
```

Data

Pin Config

Pins that need to be configured to use

DATA2 only needs to be set if num_parallel is > 1

```
enum
{
    DSY_SR_4021_PIN_CS,
    DSY_SR_4021_PIN_CLK,
    DSY_SR_4021_PIN_DATA,
    DSY_SR_4021_PIN_DATA2,//optional
    DSY_SR_4021_PIN_LAST,
};
```

sr_4021_handle

configuration strucutre for 4021

pin config is used to initialize the dsy_gpio

num_parallel is the number of devices connected that share the same clk/cs, etc. but have independent data num_daisychained is the number of devices in a daisy-chain configuration

```
typedef struct
{
    dsy_gpio_pin pin_config[DSY_SR_4021_PIN_LAST];
    uint8_t      num_parallel, num_daisychained;
    dsy_gpio   cs, clk, data[SR_4021_MAX_PARALLEL];
    uint8_t      states[8 * SR_4021_MAX_DAISYCHAIN * SR_4021_MAX_PARALLEL];
} dsy_sr_4021_handle;
```

Init

Initialize CD4021 with settinsgs from sr_4021_handle

```
void    dsy_sr_4021_init(dsy_sr_4021_handle *sr);
```

Update

Fills internal states with CD4021 data states.

```
void dsy_sr_4021_update(dsy_sr_4021_handle *sr); // checks all 8 states per confi
```

State

Returns the state of a pin at a given index.

```
uint8_t dsy_sr_4021_state(dsy_sr_4021_handle *sr, uint8_t idx);
```

ShiftRegister595

Descriptioin

Device Driver for 8-bit shift register

CD74HC595 - 8-bit serial to parallel output shift

Credits

Author: shensley

Date Added: May 2020

Data

Consts

Maximum Number of chained devices

Connect device's QH' pin to the next chips serial input

```
const size_t kMaxSr595DaisyChain = 16;
```

Pins

The following pins correspond to the hardware connections to the 595.

- LATCH corresonds to Pin 12 "RCLK"
- CLK corresponds to Pin 11 "SRCLK"
- DATA corresponds to Pin 14 "SER"
- *SRCLR* is not added here, but is tied to 3v3 on test hardware.

```
enum Pins
{
    PIN_LATCH,
    PIN_CLK,
    PIN_DATA,
    NUM_PINS,
};
```

Functions

Init

Initializes the GPIO, and data for the ShiftRegister

Arguments:

- *pin__cfg is an array of dsy__gpio__pin corresponding the the Pins enum above.
- num__daisy__chained (default = 1) is the number of 595 devices daisy chained together.

```
void Init(dsy_gpio_pin *pin_cfg, size_t num_daisy_chained = 1);
```

Set

Sets the state of the specified output.

The index starts with QA on the first device and ends with QH on the last device.

a true state will set the output HIGH, while a false state will set the output LOW.

Write

Writes the states of shift register out to the connected devices.

Audio

Description

Audio Driver

Configures Audio Device and provides callback for signal processing.

Many of the hard-coded values here will change (increase), and/or be replaced by configurable options

Defines

Defines for generic maximums

While 'Audio Channels Max' is set to 2, this is per-SAI

4x4 Audio I/O is possible using the dsy_audio_mc_callback

Hard-coded samplerate is calculated from original clock tree.

The new clock tree has less than 0.01% error for all supported samplerates

```
#define DSY_AUDIO_CHANNELS_MAX 2

#ifndef DSY_AUDIO_SAMPLE_RATE
#define DSY_AUDIO_SAMPLE_RATE 48014.0f
#endif
```

Enum for specifying SAI Block

Internally, there are two separate 'audio blocks' that can be configured together or separately

```
    enum
    {
        DSY_AUDIO_INTERNAL,
        DSY_AUDIO_EXTERNAL,
        DSY_AUDIO_LAST,
    };
```

Audio Handle

Simple config struct that holds peripheral drivers.

```
typedef struct
{
    size_t          block_size;
    dsy_sai_handle *sai;
    dsy_i2c_handle *dev0_i2c;
    dsy_i2c_handle *dev1_i2c;
} dsy_audio_handle;
```

Callback Definitions

These are user-defineable callbacks that are called when audio data is ready to be received/transmitted.

This function is called at samplerate/blocksize (e.g. 1kHz when

Interleaved Stereo callback

Function to define for using a single Stereo device for I/O

audio is packed as: { LEFT | RIGHT | LEFT | RIGHT }

typical example:

```
void AudioCallback(float *in, float *out, size_t size)
{
}
```

```
typedef void (*dsy_audio_callback)(float*, float*, size_t);
```

Non-Interleaved Multichannel callback

Defaults to 4 channels, and is fixed for now. (still works for stereo, but will still fill buffers)

audio is packed as:

{ LEFT | LEFT + 1 | . . . | LEFT + SIZE | RIGHT | RIGHT + 1 | . . . | RIGHT + SIZE } typical example:

```
void AudioCallback(float **in, float **out, size_t size)
{
}
```

```
    typedef void (*dsy_audio_mc_callback)(float**, float**, size_t);
```

Functions

Init

Initializes the Audio Engine using configurations set to the sai_handle

i2c_handles can be set to NULL if not needed.

```
    void dsy_audio_init(dsy_audio_handle* handle);
```

set_callback

Sets the user defined, interleaving callback to be called when audio data is ready.

intext is a specifier for DSY_AUDIO_INT/EXT (which audio peripheral to use).

When using this, each 'audio block' can have completely independent callbacks.

```
    void dsy_audio_set_callback(uint8_t intext, dsy_audio_callback cb);
```

set_mc_callback

Sets the user defined, non-interleaving callback to be called when audio data is ready.

This will always use both DSY_AUDIO_INT and DSY_AUDIO_EXT blocks together.

To ensure clean audio you'll want to make sure the two SAIs are set to the same samplerate

```
    void dsy_audio_set_mc_callback(dsy_audio_mc_callback cb);
```

set_blocksize

Sets the number of samples (per-channel) to be handled in a single audio frame.

```
void dsy_audio_set_blocksize(uint8_t intext, size_t blocksize);
```

start

Starts Audio Engine, callbacks will begin getting called.

When using with dsy_audio_mc_callback (for 4 channels), this function should be called for both audio blocks

```
void dsy_audio_start(uint8_t intext);
```

stop

Stops transmitting/receiving audio on the specified audio block.

```
void dsy_audio_stop(uint8_t intext);
```

enter bypass

If the device supports hardware bypass, enter that mode.

Only minimally tested with WM8731 codec.

exit bypass

If the device supports hardware bypass, exit that mode.

Only minimally tested with WM8731 codec.

```
void dsy_audio_exit_bypass(uint8_t intext);
```

Default Callbacks

A few useful stereo-interleaved callbacks

passthru

Passes the input to the output

```
    void dsy_audio_passthru(float* in, float* out, size_t size);
```

silence

sets outputs to 0 without stopping the Audio Engine.

```
    void dsy_audio_silence(float* in, float* out, size_t size);
```

AnalogControl

Description

Hardware Interface for control inputs

Primarily designed for ADC input controls such as potentiometers, and control voltage.

Credit:

Author: Stephen Hensley **Date Added:** November 2019

General Functions

Init

Initializes the control *adcptr is a pointer to the raw adc read value – This can acquired with dsy_adc_get_rawptr(), or dsy_adc_get_mux_rawptr()

sr is the samplerate in Hz that the Process function will be called at.

slew_seconds is the slew time in seconds that it takes for the control to change to a new value.

flip determines whether the input is flipped (i.e. 1.f - input) or not before being processed.

invert determines whether the input is inverted (i.e. -1.f * input) or note before being processed.

```
void Init(uint16_t *adcptr,
          float     sr,
          bool      flip         = false,
          bool      invert       = false,
          float     slew_seconds = 0.002f);
```

Init_bipolar_cv

This Initializes the AnalogControl for a -5V to 5V inverted input

All of the Init details are the same otherwise

```
void InitBipolarCv(uint16_t *adcptr, float sr);
```

Process

filters, and transforms a raw ADC read into a normalized range.

this should be called at the rate of specified by samplerate at Init time.

Default Initializations will return 0.0 -> 1.0

Bi-polar CV inputs will return -1.0 -> 1.0

```
float Process();
```

Value

Returns the current stored value, without reprocessing

```
inline float Value() const { return val_; }
```

Encoder

Description

Generic Class for handling Quadrature Encoders

Files

hid_encoder.*

Credits

Author: Stephen Hensley

Date: December 2019

Inspired/influenced by Mutable Instruments (pichenettes) Encoder classes

General Functions

Init

Initializes the encoder with the specified hardware pins.

Update rate should be the rate at which Debounce() gets called in Hertz.

```
void
Init(dsy_gpio_pin a, dsy_gpio_pin b, dsy_gpio_pin click, float update_rate);
```

Debounce

Called at update_rate to debounce and handle timing for the switch.

In order for events not to be missed, its important that the Edge/Pressed checks be made at the same rate as the debounce function is being called.

```
void Debounce();
```

Increment

Returns +1 if the encoder was turned clockwise, -1 if it was turned counter-clockwise, or 0 if it was not just turned.

```
inline int32_t Increment() const { return inc_; }
```

RisingEdge

Returns true if the encoder was just pressed.

```
inline bool RisingEdge() const { return sw_.RisingEdge(); }
```

FallingEdge

Returns true if the encoder was just released.

```
inline bool FallingEdge() const { return sw_.FallingEdge(); }
```

Pressed

Returns true while the encoder is held down.

```
inline bool Pressed() const { return sw_.Pressed(); }
```

TimeHeldMs

Returns the time in milliseconds that the encoder has been held down.

```
inline float TimeHeldMs() const { return sw_.TimeHeldMs(); }
```

GateIn

Description

Generic Class for handling gate inputs through GPIO.

Files

hid_gatein.h

Credits

Author: Stephen Hensley

Date: March 2020

General Functions

Init

Initializes the gate input with specified hardware pin

```
void Init(dsy_gpio_pin *pin_cfg);
```

Trig

Checks current state of gate input. Returns FALSE if pin is low, and TRUE if high

```
bool Trig();
```

Led

Description

LED Class providing simple Software PWM ability, etc.

Eventually this will work with hardware PWM, and external LED Driver devices as well.

Credit

Author: shensley

Date Added: March 2020

TODO:

- Get this set up to work with the dev_leddriver stuff as well
- Setup Hardware PWM for pins that have it

Init

Initializes an LED using the specified hardware pin.

invert will set whether to internally invert the brightness due to hardware config.

samplerate sets the rate at which 'Update()' will be called (used for software PWM)

```
void Init(dsy_gpio_pin pin, bool invert, float samplerate=1000.0f);
```

Set

Sets the brightness of the Led.

val will be cubed for gamma correction, and then quantized to 8-bit values for Software PWM.

8-bit is for more flexible update rate options, as 12-bit or more would require faster update rates.

```
void Set(float val);
```

Update

This processes the pwm of the LED sets the hardware accordingly.

```
void Update();
```

MidiHandler

Description

Simple MIDI Handler

Parses bytes from an input into valid MidiEvents.

The MidiEvents fill a FIFO queue that the user can pop messages from.

Credit

author: shensley *date added*: March 2020

Data

MidiMessageType

Parsed from the Status Byte, these are the common Midi Messages that can be handled. At this time only 3-byte messages are correctly parsed into MidiEvents.

```
enum MidiMessageType
{
    NoteOff,
    NoteOn,
    PolyphonicKeyPressure,
    ControlChange,
    ProgramChange,
    ChannelPressure,
    PitchBend,
    MessageLast, // maybe change name to MessageUnsupported
};
```

NoteOnEvent

Struct containing note, and velocity data for a given channel.

Can be made from MidiEvent

```
struct NoteOnEvent
{
    int     channel;
    uint8_t note;
    uint8_t velocity;
};
```

ControlChangeEvent

Struct containing control number, and value for a given channel.

Can be made from MidiEvent

```
struct ControlChangeEvent
{
    int     channel;
    uint8_t control_number;
    uint8_t value;
};
```

MidiEvent

Simple MidiEvent with message type, channel, and data[2] members. Newer ish.

AsNoteOn

Returns the data within the MidiEvent as a NoteOnEvent struct.

```
    NoteOnEvent     AsNoteOn()
```

AsNoteOn

Returns the data within the MidiEvent as a NoteOnEvent struct.

```
    ControlChangeEvent AsControlChange()
```

MidiHandler

Midi IO Modes

Input and Output can be configured separately Multiple Input modes can be selected by OR'ing the values.

```
enum MidiInputMode
{
    INPUT_MODE_NONE    = 0x00,
    INPUT_MODE_UART1   = 0x01,
    INPUT_MODE_USB_INT = 0x02,
    INPUT_MODE_USB_EXT = 0x04,
};
enum MidiOutputMode
{
    OUTPUT_MODE_NONE    = 0x00,
    OUTPUT_MODE_UART1   = 0x01,
    OUTPUT_MODE_USB_INT = 0x02,
    OUTPUT_MODE_USB_EXT = 0x04,
};
```

Functions

Init

Initializes the MidiHandler

```
void Init(MidiInputMode in_mode, MidiOutputMode out_mode);
```

StartReceive

Starts listening on the selected input mode(s). MidiEvent Queue will begin to fill, and can be checked with

Parse

Feed in bytes to state machine from a queue.

Populates internal FIFO queue with MIDI Messages

For example with uart: midi.Parse(uart.PopRx());

```
void Parse(uint8_t byte);
```

HasEvents

Checks if there are unhandled messages in the queue

```
    bool HasEvents() const { return event_q_.readable(); }
```

PopEvent

Pops the oldest unhandled MidiEvent from the internal queue

OLED Display

Description

Human Interface Driver for using an OLED Display (SSD1309)

For all `bool on` arguments: true is on, false is off.

Credits

Credit to Aleksander Alekseev (github.com/afiskon/stm32-ssd1306) on github for a great starting point.

adapted for SSD1309 and H7 by shensley, 2020

SSD1309 width in pixels

Pins

GPIO Pins that need to be used independent of peripheral used.

```
enum Pins
{
    DATA_COMMAND,
    RESET,
    NUM_PINS,
};
```

Init

TODO:

- add I2C Support.
- add configuration for specific spi/i2c peripherals (currently only uses SPI1, w/ hardware controlled chip select.
- re-add support for SSD1306 displays

Takes an argument for the pin cfg *pin_cfg should be a pointer to an array of OledDisplay::NUM_PINS dsy_gpio_pins

```
void Init(dsy_gpio_pin* pin_cfg);
```

Fill

Fills the entire display with either on/off.

```
void Fill(bool on);
```

Draw Pixel

Sets the pixel at the specified coordinate to be on/off.

```
void DrawPixel(uint8_t x, uint8_t y, bool on);
```

WriteChar

Writes the character with the specific FontDef to the display buffer at the current Cursor position.

```
char WriteChar(char ch, FontDef font, bool on);
```

WriteString

Similar to WriteChar, except it will handle an entire String.

Wrapping does not happen automatically, so the width of the string must be kept within the dimensions of the screen.

```
char WriteString(char* str, FontDef font, bool on);
```

SetCursor

Moves the 'Cursor' position used for WriteChar, and WriteStr to the specified coordinate.

```
void SetCursor(uint8_t x, uint8_t y);
```

Update

Writes the current display buffer to the OLED device using SPI or I2C depending on how the object was initialized.

```
void Update();
```

Parameter

```
Simple parameter mapping tool that takes a 0-1 input from an hid_ctrl.
```

parameter class

Data Types

Curve

Curves are applied to the output signal

```
enum Curve
{
    LINEAR,
    EXPONENTIAL,
    LOGARITHMIC,
    CUBE,
    LAST,
};
```

init

initialize a parameter using an hid_ctrl object.

hid_ctrl input - object containing the direct link to a hardware control source.

min - bottom of range. (when input is 0.0)

max - top of range (when input is 1.0)

curve - the scaling curve for the input->output transformation.

```
void Init(AnalogControl input, float min, float max, Curve curve);
```

process

processes the input signal, this should be called at the samplerate of the hid_ctrl passed in.

returns a float with the specified transformation applied.

```
float Process();
```

value

returns the current value from the parameter without processing another sample. this is useful if you need to use the value multiple times, and don't store the output of process in a local variable.

```
    inline float Value() { return val_; }
```

RGB Led

Description

3x LEDs configured as an RGB for ease of use.

Init

Initializes 3x GPIO Pins as red, green, and blue elements of an RGB LED

Invert will flip polarity of LED.

```
void
Init(dsy_gpio_pin red, dsy_gpio_pin green, dsy_gpio_pin blue, bool invert);
```

Set

Sets each element of the LED with a floating point number 0-1

```
void Set(float r, float g, float b);
```

SetColor

Sets the RGB using a Color object.

```
void SetColor(Color c);
```

Update

Updates the PWM of the LED based on the current values.

Should be called at a regular interval. (i.e. 1kHz/1ms)

```
void Update();
```

Switch

Description

Generic Class for handling momentary/latching switches

Files

hid_switch.*

Credits

Author: Stephen Hensley

Date: December 2019

Inspired/influenced by Mutable Instruments (pichenettes) Switch classes

Data Types

Type

Specifies the expected behavior of the switch

```
enum Type
{
    TYPE_TOGGLE,
    TYPE_MOMENTARY,
};
```

Polarity

Specifies whether the pressed is HIGH or LOW.

```
enum Polarity
{
    POLARITY_NORMAL,
    POLARITY_INVERTED,
};
```

Pull

Specifies whether to use built-in Pull Up/Down resistors to hold button at a given state when not engaged.

```
enum Pull
{
    PULL_UP,
    PULL_DOWN,
    PULL_NONE,
};
```

General Functions

Init

Initializes the switch object with a given port/pin combo.

Parameters:

- pin: port/pin object to tell the switch which hardware pin to use.
- update_rate: the rate at which the Debounce() function will be called. (used for timing).
- t: switch type – Default: TYPE_MOMENTARY
- pol: switch polarity – Default: POLARITY_INVERTED
- pu: switch pull up/down – Default: PULL_UP

```
void
Init(dsy_gpio_pin pin, float update_rate, Type t, Polarity pol, Pull pu);

void Init(dsy_gpio_pin pin, float update_rate);
```

Debounce

Called at update_rate to debounce and handle timing for the switch.

In order for events not to be missed, its important that the Edge/Pressed checks be made at the same rate as the debounce function is being called.

```
void Debounce();
```

RisingEdge

Returns true if a button was just pressed.

```
inline bool RisingEdge() const { return state_ == 0x7f; }
```

FallingEdge

Returns true if the button was just released

```
inline bool FallingEdge() const { return state_ == 0x80; }
```

Pressed

Returns true if the button is held down (or if the toggle is on).

```
inline bool Pressed() const { return state_ == 0xff; }
```

TimeHeldMs

Returns the time in milliseconds that the button has been held (or toggle has been on)

```
inline float TimeHeldMs() const
```

hid_usb

Description

Interface for initializing and using the USB Peripherals on the daisy

Credits

Author: Stephen Hensley

Date Added: December 2019

Data Types

UsbPeriph

Specified which of the two USB Peripherals to initialize.

FS External D- pin is Pin 37 (GPIO31)

FS External D+ pin is Pin 38 (GPIO32)

```
        FS_INTERNAL,
        FS_EXTERNAL,
        FS_BOTH,
```

ReceiveCallback

Function called upon reception of a buffer

```
    typedef void (*ReceiveCallback)(uint8_t* buff, uint32_t* len);
```

General Functions

Init

Initializes the specified peripheral(s) as USB CDC Devices

```
    void Init(UsbPeriph dev);
```

TransmitInternal

Transmits a buffer of ‘size’ bytes from the on board USB FS port.

```
void TransmitInternal(uint8_t* buff, size_t size);
```

TransmitExternal

Transmits a buffer of ‘size’ bytes from a USB port connected to the external USB Pins of the daisy seed.

```
void TransmitExternal(uint8_t* buff, size_t size);
```

SetReceiveCallback

sets the callback to be called upon reception of new data

```
void SetReceiveCallback(ReceiveCallback cb);
```

TODO:

- Add support for other USB classes (currently only CDC is supported)
- DMA setup

WavPlayer

Description

Wav Player that will load .wav files from an SD Card, and then provide a method of accessing the samples with double-buffering.

Current Limitations: - 1x Playback speed only - 16-bit, mono files only (otherwise fun weirdness can happen). - Only 1 file playing back at a time. - Not sure how this would interfere with trying to use the SDCard/FatFs outside of

WavFileInfo

Struct containing details of Wav File.

TODO: add bitrate, samplerate, length, etc.

```
struct WavFileInfo
{
    WAV_FormatTypeDef raw_data;
    char              name[WAV_FILENAME_MAX];
};
```

WavPlayer

Class for handling playback of WAV files.

TODO: - Make template-y to reduce memory usage.

Init

Initializes the WavPlayer, loading up to max_files of wav files from an SD Card.

```
    void Init();
```

Open

Opens the file at index sel for reading.

```
    int Open(size_t sel);
```

Close

Closes whatever file is currently open.

```
int Close();
```

Stream

Returns the next sample if playing, otherwise returns 0

```
int16_t Stream();
```

Prepare

Collects buffer for playback when needed.

```
void Prepare();
```

Restart

Resets the playback position to the beginning of the file immediately

```
void Restart();
```

SetLooping

Sets whether or not the current file will repeat after completing playback.

```
inline void SetLooping(bool loop) { looping_ = loop; }
```

GetLooping

Returns whether the WavPlayer is looping or not.

```
inline bool GetLooping() const { return looping_; }
```

GetNumberFiles()

Returns the number of files loaded by the WavPlayer

```
inline size_t GetNumberFiles() const { return file_cnt_; }
```

GetCurrentFile()

Returns currently selected file.

```
    inline size_t GetCurrentFile() const { return file_sel_; }
```

AdcChannelConfig

Configuration Structure for a given channel

While there may not be many configuration options here, using a struct like this allows us to add more configuration later without breaking existing functionality.

```
struct AdcChannelConfig
```

InitSingle

Initializes a single ADC pin as an ADC.

```
    void InitSingle(dsy_gpio_pin pin);
```

InitMux

Initializes a single ADC pin as a Multiplexed ADC.

Requires a CD4051 Multiplexor connected to the pin

Internal Callbacks handle the pin addressing.

channels must be 1-8

```
    void InitMux(dsy_gpio_pin adc_pin,
                 dsy_gpio_pin mux_0,
                 dsy_gpio_pin mux_1,
                 dsy_gpio_pin mux_2,
                 size_t       channels);
```

Init

Initializes the ADC with the pins passed in.

- *cfg: an array of AdcChannelConfig of the desired channel
- num_channels: number of ADC channels to initialize
- ovs: Oversampling amount - Defaults to OVS_32

```
    void Init(AdcChannelConfig *cfg, size_t num_channels, OverSampling ovs=OVS_32)
```

Start

Starts reading from the ADC

```
void Start();
```

Stop

Stops reading from the ADC

```
void Stop();
```

Accessors

These are getters for a single channel

```
uint16_t  Get(uint8_t chn);
uint16_t *GetPtr(uint8_t chn);
float     GetFloat(uint8_t chn);
```

These are getters for multiplexed inputs on a single channel (up to 8 per ADC input).

```
uint16_t  GetMux(uint8_t chn, uint8_t idx);
uint16_t *GetMuxPtr(uint8_t chn, uint8_t idx);
float     GetMuxFloat(uint8_t chn, uint8_t idx);
```

DAC

Description

Driver for the built in DAC on the STM32

The STM32 has 2 Channels of independently configurable

TODO:

- Add Interrupt/DMA modes for block transfers, etc.
- Add configuration for Buffer state, etc.
- Add independent settings (bitdepth, etc.) for each channel

Data

Mode

Currently only Polling is supported.

```
typedef enum
{
    DSY_DAC_MODE_POLLING,
    DSY_DAC_MODE_LAST,
} dsy_dac_mode;
```

Bitdepth

Sets the bit depth of the DAC output

This can be set independently for each channel.

```
typedef enum
{
    DSY_DAC_BITS_8,
    DSY_DAC_BITS_12,
    DSY_DAC_BITS_LAST,
} dsy_dac_bitdepth;
```

Channel

Sets which channel(s) are initialized with the settings chosen.

```
typedef enum
{
    DSY_DAC_CHN1,
    DSY_DAC_CHN2,
    DSY_DAC_CHN_LAST,
    DSY_DAC_CHN_BOTH,
} dsy_dac_channel;
```

dsy_dac_handle

Configuration structure for DAC initialization and settings.

pin_config must be filled out. However, the DACs are pretty

```
typedef struct
{
    dsy_dac_mode     mode;
    dsy_dac_bitdepth bitdepth;
    dsy_gpio_pin     pin_config[DSY_DAC_CHN_LAST];
} dsy_dac_handle;
```

Functions

Init

Initializes the specified channel(s) of the DAC

```
void dsy_dac_init(dsy_dac_handle *dsy_hdac, dsy_dac_channel channel);
```

Start

Turns on the DAC and turns on any internal timer if necessary.

```
void dsy_dac_start(dsy_dac_channel channel);
```

Write

Sets the specified channel of the dac to the value (within bitdepth) resolution.

When set to 8-bit, val should be 0-255

When set to 12-bit, val should be 0-4095

```
void dsy_dac_write(dsy_dac_channel channel, uint16_t val);
```

GPIO

Description

General Purpose IO driver

Data

Gpio Mode

Sets the mode of the GPIO

```
typedef enum
{
    DSY_GPIO_MODE_INPUT,
    DSY_GPIO_MODE_OUTPUT_PP, // Push-Pull
    DSY_GPIO_MODE_OUTPUT_OD, // Open-Drain
    DSY_GPIO_MODE_ANALOG,
    DSY_GPIO_MODE_LAST,
} dsy_gpio_mode;
```

Pull

Configures whether an internal Pull up or Pull down resistor is used

```
typedef enum
{
    DSY_GPIO_NOPULL,
    DSY_GPIO_PULLUP,
    DSY_GPIO_PULLDOWN,
} dsy_gpio_pull;
```

dsy_gpio

Struct for holding the pin, and configuration

```
typedef struct
{
    dsy_gpio_pin pin;
    dsy_gpio_mode mode;
    dsy_gpio_pull pull;
} dsy_gpio;
```

Functions

init

Initializes the gpio with the settings configured.

```
void dsy_gpio_init(dsy_gpio *p);
```

deinit

Deinitializes the gpio pin

```
void dsy_gpio_deinit(dsy_gpio *p);
```

read

Reads the state of the gpio pin

returning 1 if the pin is HIGH, and 0 if the pin is LOW

```
uint8_t dsy_gpio_read(dsy_gpio *p);
```

write

Writes the state to the gpio pin

Pin will be set to 3v3 when state is 1, and 0V when state is 0

```
void dsy_gpio_write(dsy_gpio *p, uint8_t state);
```

toggle

Toggles the state of the pin so that it is not at the same

```
void dsy_gpio_toggle(dsy_gpio *p);
```

PER_I2C

Driver for controlling I2C devices

TODO:

- Add DMA support
- Add timing calc based on current clock source freq.
- Add discrete rx/tx functions (currently other drivers still need to call ST HAL functions).

Errata:

- 1MHZ (FastMode+) is currently only 886kHZ (should get remeasured with latest clock tree).

Types

dsy_i2c_periph

Specifices the internal peripheral to use (these are mapped to different pins on the hardware).

```
typedef enum
{
    DSY_I2C_PERIPH_1,
    DSY_I2C_PERIPH_2,
    DSY_I2C_PERIPH_3,
    DSY_I2C_PERIPH_4,
} dsy_i2c_periph;
```

dsy_i2c_pin

List of pins associated with the peripheral. These must be set in the handle's pin_config.

```
typedef enum
{
    DSY_I2C_PIN_SCL,
    DSY_I2C_PIN_SDA,
    DSY_I2C_PIN_LAST,
} dsy_i2c_pin;
```

dsy_i2c_speed

Rate at which the clock/data will be sent/received. The device being used will have maximum speeds.

1MHZ Mode is currently 886kHz

```
typedef enum
{
    DSY_I2C_SPEED_100KHZ,
    DSY_I2C_SPEED_400KHZ,
    DSY_I2C_SPEED_1MHZ,
    DSY_I2C_SPEED_LAST,
} dsy_i2c_speed;
```

dsy_i2c_handle

this object will be used to initialize the I2C interface, and can be passed to dev_ drivers that require I2C.

```
typedef struct
{
    dsy_i2c_periph periph;
    dsy_gpio_pin   pin_config[DSY_I2C_PIN_LAST];
    dsy_i2c_speed  speed;
} dsy_i2c_handle;
```

Functions

dsy_i2c_init

initializes an I2C peripheral with the data given from the handle.

Requires a dsy_i2c_handle object to initialize.

```
void dsy_i2c_init(dsy_i2c_handle *dsy_hi2c);
```

QSPI

Description

Driver for QSPI peripheral to interface with external flash memory.

Currently supported QSPI Devices:

Error codes

```
#define DSY_MEMORY_OK ((uint32_t)0x00)
#define DSY_MEMORY_ERROR ((uint32_t)0x01)
```

Section Attributes

used for reading memory in memory_mapped mode.

```
#define DSY_QSPI_TEXT __attribute__((section(".qspiflash_text")))
#define DSY_QSPI_DATA __attribute__((section(".qspiflash_data")))
#define DSY_QSPI_BSS __attribute__((section(".qspiflash_bss")))
```

Data

Pins

List of Pins used in QSPI (passed in during Init)

```
    typedef enum
    {
        DSY_QSPI_PIN_IO0,
        DSY_QSPI_PIN_IO1,
        DSY_QSPI_PIN_IO2,
        DSY_QSPI_PIN_IO3,
        DSY_QSPI_PIN_CLK,
        DSY_QSPI_PIN_NCS,
        DSY_QSPI_PIN_LAST,
    } dsy_qspi_pin;
```

Modes

Modes of operation. / Memory Mapped mode: QSPI configured so that the QSPI can be Indirect Polling mode: Device driver enabled.

```
typedef enum
{
    DSY_QSPI_MODE_DSY_MEMORY_MAPPED,
    DSY_QSPI_MODE_INDIRECT_POLLING,
    DSY_QSPI_MODE_LAST,
} dsy_qspi_mode;
```

Device

Flash Devices supported. (Both of these are more-or-less the same,

```
typedef enum
{
    DSY_QSPI_DEVICE_IS25LP080D,
    DSY_QSPI_DEVICE_IS25LP064A,
    DSY_QSPI_DEVICE_LAST,
} dsy_qspi_device;
```

dsy_qspi_handle

Configuration structure for interfacing with QSPI Driver.

```
typedef struct
{
    dsy_qspi_mode   mode;
    dsy_qspi_device device;
    dsy_gpio_pin    pin_config[DSY_QSPI_PIN_LAST];
} dsy_qspi_handle;
```

Init

Initializes QSPI peripheral, and Resets, and prepares memory for access.

hqspi should be populated with the mode, device and pin_config before calling this function.

Returns DSY_MEMORY_OK or DSY_MEMORY_ERROR

```
int dsy_qspi_init(dsy_qspi_handle* hqspi);
```

Deinit

Deinitializes the peripheral This should be called before reinitializing QSPI in a different mode.

Returns DSY_MEMORY_OK or DSY_MEMORY_ERROR

```
int dsy_qspi_deinit();
```

writepage

Writes a single page to to the specified address on the QSPI chip.

For IS25LP* page size is 256 bytes.

Returns DSY_MEMORY_OK or DSY_MEMORY_ERROR

```
int dsy_qspi_writepage(uint32_t adr, uint32_t sz, uint8_t* buf);
```

write

Writes data in buffer to to the QSPI. Starting at address to address+size

```
int dsy_qspi_write(uint32_t address, uint32_t size, uint8_t* buffer);
```

erase

Erases the area specified on the chip. Erasures will happen by 4K, 32K or 64K increments.

Smallest erase possible is 4kB at a time. (on IS25LP*)

```
int dsy_qspi_erase(uint32_t start_adr, uint32_t end_adr);
```

erasesector

Erases a single sector of the chip.

TODO: Document the size of this function.

```
int dsy_qspi_erasesector(uint32_t addr);
```

SAI (Serial Audio Interface)

Description

Driver for the SAI peripheral

Supports SAI1 and SAI2 with several configuration options

TODO

- Fix hard-coding of SAI2 Initialization
- Clean up this and hid_audio (the lines between them are a bit blurry).
- Variable Samplerate setting (should be fine with the new clock tree)
- Fix intermingling of both SAI. they should be able to be independently initialized (like i2c, etc.)
- Add 32-bit bitdepth (for devices that may need that)
- Move the 'device' chunk of this to hid_audio. given that they sometimes need other pins or i2c, etc.

DataTypes

sai

selects which SAI (or both/none) to initialize

```
typedef enum
{
    DSY_AUDIO_INIT_SAI1,
    DSY_AUDIO_INIT_SAI2,
    DSY_AUDIO_INIT_BOTH,
    DSY_AUDIO_INIT_NONE,
    DSY_AUDIO_INIT_LAST,
} dsy_audio_sai;
```

samplerate

Currently Sample Rates are not correclty supported. All audio is currently run at 48kHz

```
typedef enum
{
    DSY_AUDIO_SAMPLERATE_32K,
    DSY_AUDIO_SAMPLERATE_48K,
    DSY_AUDIO_SAMPLERATE_96K,
    DSY_AUDIO_SAMPLERATE_LAST,
}dsy_audio_samplerate;
```

bitdepth

Specifies the bitdepth of the hardware connected to the SAI peripheral

```
typedef enum
{
    DSY_AUDIO_BITDEPTH_16,
    DSY_AUDIO_BITDEPTH_24,
    DSY_AUDIO_BITDEPTH_LAST
} dsy_audio_bitdepth;
```

sync

Setting for each SAI that sets whether the processor is generating the MCLK signal or not.

```
typedef enum
{
    DSY_AUDIO_SYNC_MASTER,// No Crystal
    DSY_AUDIO_SYNC_SLAVE, // Crystal
    DSY_AUDIO_SYNC_LAST
} dsy_audio_sync;
```

direction

Each SAI has two datalines, they can independently be configured as inputs or outputs.

```
typedef enum
{
    DSY_AUDIO_RX,
    DSY_AUDIO_TX,
} dsy_audio_dir;
```

sai_pin

List of the pins that need to be initialized

SIN/SOUT is a bit misleading, and should be turned into A/B since it is possible to configure two inputs or two outputs on a single SAI.

```
typedef enum
{
    DSY_SAI_PIN_MCLK,
    DSY_SAI_PIN_FS,
    DSY_SAI_PIN_SCK,
    DSY_SAI_PIN_SIN,
    DSY_SAI_PIN_SOUT,
    DSY_SAI_PIN_LAST,
} dsy_sai_pin;
```

device

List of devices with built in support Devices not listed here, will need to have initialization done externally.

```
typedef enum
{
    DSY_AUDIO_NONE, // For unsupported, or custom devices.
    DSY_AUDIO_DEVICE_PCM3060,
    DSY_AUDIO_DEVICE_WM8731,
    DSY_AUDIO_DEVICE_AK4556,
    DSY_AUDIO_DEVICE_LAST,
} dsy_audio_device;
```

SAI Index

Index for the several arrays in the sai_handle struct below.

```
enum
{
    DSY_SAI_1,
    DSY_SAI_2,
    DSY_SAI_LAST,
};
```

dsy_sai_handle

Configuration structure for SAI

contains all above settings, and passes them to internal structure for hardware initialization.

```
typedef struct
{
    dsy_audio_sai        init;
    dsy_audio_samplerate samplerate[DSY_SAI_LAST];
    dsy_audio_bitdepth   bitdepth[DSY_SAI_LAST];
    dsy_audio_dir        a_direction[DSY_SAI_LAST];
    dsy_audio_dir        b_direction[DSY_SAI_LAST];
    dsy_audio_sync       sync_config[DSY_SAI_LAST];
    dsy_audio_device     device[DSY_SAI_LAST];

    dsy_gpio_pin         sai1_pin_config[DSY_SAI_PIN_LAST];
    dsy_gpio_pin         sai2_pin_config[DSY_SAI_PIN_LAST];
} dsy_sai_handle;
```

init

Intializes the SAI peripheral(s) with the specified settings.

pinlists should be arrays of DSY_SAI_PIN_LAST elements

```
void dsy_sai_init(dsy_audio_sai        init,
                  dsy_audio_samplerate sr[2],
                  dsy_audio_bitdepth   bitdepth[2],
                  dsy_audio_sync       sync_config[2],
                  dsy_gpio_pin *       sai1_pin_list,
                  dsy_gpio_pin *       sai2_pin_list);
```

init_from_handle

uses the data within *hsai to initialize the peripheral(s)

```
void dsy_sai_init_from_handle(dsy_sai_handle *hsai);
```

SDMMC

Description

Configuration for interfacing with SD cards.

Currently only supports operation using FatFS filesystem

TODO:

- Implement configuration (currently all settings are fixed).

Data Structures

Mode

Operating Mode

Currently only FatFS is supported.

```
enum SdmmcMode
{
    SDMMC_MODE_FATFS,
};
```

BitWidth

Sets whether 4-bit mode or 1-bit mode is used for the SDMMC

```
enum SdmmcBitWidth
{
    SDMMC_BITS_1,
    SDMMC_BITS_4,
};
```

Speed

Sets the desired clock speed of the SD card bus.

Initialization is always done at or below 400kHz, and then the user speed is set.

```
enum SdmmcSpeed
{
    SDMMC_SPEED_400KHZ,
    SDMMC_SPEED_12MHZ,
};
```

HandlerInit

Structure for setting the options above.

Used to intiailize SdmmcHandler

```
struct SdmmcHandlerInit
{
    SdmmcBitWidth bitdepth;
    SdmmcSpeed    speed;
};
```

SdmmcHandler

Init

Initializes the SD Card Interface

For now all settings are fixed (See todo at top of section)

```
    void Init();
```

SPI

TODO: - Add documentation - Add configuration - Add reception - Add IT - Add DMA

per_tim

General purpose timer for delays and general timing.

TODO:

- Add configurable tick frequency – for now its set to the APB1 Max Freq (200MHz)
- Add ability to generate periodic callback functions

init

initializes the TIM2 peripheral with maximum counter autoreload, and no prescalers.

```
void dsy_tim_init();
```

start

Starts the timer ticking.

```
void dsy_tim_start();
```

tick

These functions are specific to the actual clock ticks at the timer frequency which is currently fixed at 200MHz

get_tick

Returns a number 0x00000000-0xffffffff of the current tick

```
uint32_t dsy_tim_get_tick();
```

delay_tick

blocking delay of cnt timer ticks.

```
void     dsy_tim_delay_tick(uint32_t cnt);
```

ms

These functions are converted to use milliseconds as their time base.

get_ms

returns the number of milliseconds through the timer period.

```
uint32_t dsy_tim_get_ms();
```

delay_ms

blocking delay of cnt milliseconds.

```
void     dsy_tim_delay_ms(uint32_t cnt);
```

us

These functions are converted to use microseconds as their time base.

get_us

returns the number of microseconds through the timer period.

```
uint32_t dsy_tim_get_us();
```

delay_us

blocking delay of cnt microseconds.

```
void     dsy_tim_delay_us(uint32_t cnt);
```

UART

Description

Uart Peripheral

Credit

Written by: shensley Date Added: March 2020

TODO:

- Add flexible config for:
  - data size, stop bits, parity, baud, etc.
  - dma vs interrupt (or not).
- Error handling
- Transmit function improvements.
- Other UART Peripherals (currently only handles USART1 in UART mode.
- Overflow handling, etc. for Rx Queue.

Functions

Init

Initializes the UART Peripheral

```
void Init();
```

PollReceive

Reads the amount of bytes in blocking mode with a 10ms timeout.

```
int PollReceive(uint8_t *buff, size_t size, uint32_t timeout);
```

StartRx

Starts a DMA Receive callback to fill a buffer of specified size.

Data is populated into a FIFO queue, and can be queried with the functions below. Maximum Buffer size is defined above.

If a value outside of the maximum is specified, the size will be set to the maximum.

```
int StartRx(size_t size);
```

RxActive

Returns whether Rx DMA is listening or not.

FlushRx

Flushes the Receive Queue

```
int FlushRx();
```

PollTx

Sends an amount of data in blocking mode.

```
int PollTx(uint8_t *buff, size_t size);
```

PopRx

Pops the oldest byte from the FIFO.

```
uint8_t PopRx();
```

Readable

Checks if there are any unread bytes in the FIFO

```
size_t  Readable();
```

CheckError

Returns the result of HAL_UART_GetError() to the user.

```
int CheckError();
```

DMA

Description

Initializes the Direct Memory Access Peripheral used by many internal elements of libdaisy.

dsy_dma_init

Initializes the DMA (specifically for the modules used within the library)

```
void dsy_dma_init(void);
```

System

Description

Low level System Configuration

Functions

dsy_system_init

Initializes Clock tree, MPU, and internal memories voltage regulators.

This function *must* be called at the beginning of any program using libdaisy

Higher level daisy_ files call this through the DaisySeed object.

```
void dsy_system_init();
```

dsy_system_jumpto

Jump to an address within the internal memory

This may not work correctly, and may not be very useful with the single sector of memory on the stm32h750

```
void dsy_system_jumpto(uint32_t addr);
```

dsy_system_jumptoqspi()

Jumps to the first address of the external flash chip (0x90000000)

If there is no code there, the chip will likely fall through to the while() loop

Documentation/Loader for using external flash coming soon.

```
void     dsy_system_jumptoqspi();
```

dsy_system_getnow

Returns a uint32_t value of milliseconds since the SysTick started

```
uint32_t dsy_system_getnow(); // returns HAL_GetTick()
```

Blocking Delay that uses the SysTick (1ms callback) to wait.

```
void dsy_system_delay(uint32_t delay_ms);
```

BSP SD DiskIO

Description

Functions for handling DiskIO via SDMMC

These are usually configured through the FatFS driver/interface, and won't need to be accessed directly often.

DataStructures

This struct is identical to the struct provided as "HAL_SD_CardInfoTypeDef" I'm using this to allow users to link to the fatfs middleware without having to then link in the entire HAL to their project.

```
typedef struct
{
    uint32_t CardType; /*!< Specifies the card Type                         */

    uint32_t
        CardVersion; /*!< Specifies the card version                     */

    uint32_t Class; /*!< Specifies the class of the card class          */

    uint32_t RelCardAdd; /*!< Specifies the Relative Card Address           */

    uint32_t BlockNbr; /*!< Specifies the Card Capacity in blocks          */

    uint32_t BlockSize; /*!< Specifies one block size in bytes             */

    uint32_t
        LogBlockNbr; /*!< Specifies the Card logical Capacity in blocks   */

    uint32_t
        LogBlockSize; /*!< Specifies logical block size in bytes          */

    uint32_t CardSpeed; /*!< Specifies the card Speed                      */

} DSY_SD_CardInfoTypeDef;
```

SD Related Defines

```
#define BSP_SD_CardInfo DSY_SD_CardInfoTypeDef
#define MSD_OK ((uint8_t)0x00)
#define MSD_ERROR ((uint8_t)0x01)
#define MSD_ERROR_SD_NOT_PRESENT ((uint8_t)0x02)
#define SD_TRANSFER_OK ((uint8_t)0x00)
#define SD_TRANSFER_BUSY ((uint8_t)0x01)
#define SD_PRESENT ((uint8_t)0x01)
#define SD_NOT_PRESENT ((uint8_t)0x00)
#define SD_DATATIMEOUT ((uint32_t)100000000)
```

Functions internal for diskIO

```
uint8_t BSP_SD_Init(void);
uint8_t BSP_SD_ITConfig(void);
uint8_t BSP_SD_ReadBlocks(uint32_t *pData,
                          uint32_t  ReadAddr,
                          uint32_t  NumOfBlocks,
                          uint32_t  Timeout);
uint8_t BSP_SD_WriteBlocks(uint32_t *pData,
                           uint32_t  WriteAddr,
                           uint32_t  NumOfBlocks,
                           uint32_t  Timeout);
uint8_t
        BSP_SD_ReadBlocks_DMA(uint32_t *pData, uint32_t ReadAddr, uint32_t NumOfBl
uint8_t BSP_SD_WriteBlocks_DMA(uint32_t *pData,
                               uint32_t  WriteAddr,
                               uint32_t  NumOfBlocks);
uint8_t BSP_SD_Erase(uint32_t StartAddr, uint32_t EndAddr);
uint8_t BSP_SD_GetCardState(void);
void    BSP_SD_GetCardInfo(BSP_SD_CardInfo *CardInfo);
uint8_t BSP_SD_IsDetected(void);
```

These functions can be modified in case the current settings (e.g. DMA stream) need to be changed for specific application needs

```
void BSP_SD_AbortCallback(void);
void BSP_SD_WriteCpltCallback(void);
void BSP_SD_ReadCpltCallback(void);
```

Color

Description

Class for handling simple cololrs

TODO:

- Add Blend(), Scale(), etc.
- I'd also like to change the way the Color names are accessed.
- There's no way to change a color once its been created (without unintuitively reiniting it).

PresetColor

List of colors that have a preset RGB value

```
enum PresetColor
{
    RED,
    GREEN,
    BLUE,
    WHITE,
    PURPLE,
    CYAN,
    GOLD,
    OFF,
    LAST
};
```

Init

Initializes the Color with a given preset.

```
void Init(PresetColor c);
```

Initializes the Color with a specific RGB value

red, green, and blue should be floats between 0 and 1

```
void Init(float red, float green, float blue);
```

Accessors

Returns the 0-1 value for the given color

```
inline float Red() const { return red_; }
inline float Green() const { return green_; }
inline float Blue() const { return blue_; }
```

HAL Map

Description

global structs, and helper functions for interfacing with the stm32 HAL library while it remains a dependancy.

Usage

This file should only be included from source files (c/cpp)

Including it from a header within libdaisy would expose the entire HAL to the users.

This should be an option for users, but should not be required.

Global Structs

externs of HAL handles…

```
extern I2C_HandleTypeDef hi2c1;
extern I2C_HandleTypeDef hi2c2;
extern I2C_HandleTypeDef hi2c3;
extern I2C_HandleTypeDef hi2c4;
```

Functions

GPIO Map

These return a HAL GPIO_TypeDef and HAL GPIO Pin as used in the HAL from a dsy_gpio_pin input.

```
GPIO_TypeDef *dsy_hal_map_get_port(dsy_gpio_pin *p);
uint16_t      dsy_hal_map_get_pin(dsy_gpio_pin *p);
```

I2C Map

Returns the I2C_HandleTypeDef for a given dsy_i2c_handle

```
I2C_HandleTypeDef *dsy_hal_map_get_i2c(dsy_i2c_handle *p);
```

OLED Fonts

Description

Utility for displaying fonts on OLED displays

Credit

Migrated to work with libdaisy from stm32-ssd1306 by @afiskon on github.

Data Structures

FontDef

```
typedef struct
{
    const uint8_t   FontWidth;  /*!< Font width in pixels */
    uint8_t         FontHeight; /*!< Font height in pixels */
    const uint16_t *data;       /*!< Pointer to data font data array */
} FontDef;
```

Fonts

These are the different sizes of fonts (width x height in pixels per character)

```
extern FontDef Font_6x8;
extern FontDef Font_7x10;
extern FontDef Font_11x18;
extern FontDef Font_16x26;
```

RingBuffer

Description

Utility Ring Buffer

Credit

imported from pichenettes/stmlib

RingBuffer

Init

Initializes the Ring Buffer

```
inline void Init() { read_ptr_ = write_ptr_ = 0; }
```

capacity

Returns the total size of the ring buffer

```
inline size_t capacity() const { return size; }
```

writable

Returns the number of samples that can be written to ring buffer without overwriting unread data.

```
inline size_t writable() const
```

readable

Returns number of unread elements in ring buffer

```
inline size_t readable() const { return (write_ptr_ - read_ptr_) % size; }
```

Write

Writes the value to the next available position in the ring buffer

```
inline void Write(T v)
```

Overwrite

Writes the new element to the ring buffer, overwriting unread data if necessary.

Read

Reads the first available element from the ring buffer

```
inline T Read()
```

ImmediateRead

Reads next element from ring buffer immediately

```
inline T ImmediateRead()
```

Flush

Flushes unread elements from the ring buffer

```
inline void Flush() { write_ptr_ = read_ptr_; }
```

Swallow

Read enough samples to make it possible to read 1 sample.

```
inline void Swallow(size_t n)
```

ImmediateRead (Multiple elements)

Reads a number of elements into a buffer immediately

```
inline void ImmediateRead(T* destination, size_t num_elements)
```

Overwrite (Multiple elements)

Overwrites a number of elements using the source buffer as input.

```
inline void Overwrite(const T* source, size_t num_elements)
```

Util - Unique Id

Description

Returns 96-bit Unique ID of the MCU

Credit

Author: shensley

Date: May 2020

dsy_get_unique_id

fills the three pointer arguments with the unique ID of the MCU.

Wav Format

Description

Helper struct for handling the WAV file format

Data Structures

WAV_FormatTypeDef

```
typedef struct
{
    uint32_t ChunkId;
    uint32_t FileSize;
    uint32_t FileFormat;
    uint32_t SubChunk1ID;
    uint32_t SubChunk1Size;
    uint16_t AudioFormat;
    uint16_t NbrChannels;
    uint32_t SampleRate;
    uint32_t ByteRate;
    uint16_t BlockAlign;
    uint16_t BitPerSample;
    uint32_t SubChunk2ID;
    uint32_t SubCHunk2Size;
} WAV_FormatTypeDef;
```