

# DaisySP

## Contents

<b>Parameter</b>	<b>11</b>
curve settings . . . . .	11
parameter class . . . . .	11
init . . . . .	11
process . . . . .	11
value . . . . .	11
<b>PER_I2C</b>	<b>12</b>
Types . . . . .	12
dsy_i2c_periph . . . . .	12
dsy_i2c_pin . . . . .	12
dsy_i2c_speed . . . . .	12
dsy_i2c_handle . . . . .	13
Functions . . . . .	13
dsy_i2c_init . . . . .	13
dsy_i2c_hal_handle . . . . .	13
<b>per_tim</b>	<b>15</b>
init . . . . .	15
start . . . . .	15
tick . . . . .	15
get_tick . . . . .	15
delay_tick . . . . .	15
ms . . . . .	15
get_ms . . . . .	15
delay_ms . . . . .	16
us . . . . .	16
get_us . . . . .	16
delay_us . . . . .	16

Maps Daisy interface to STM32 HAL – I’d like to get all of this stuff tucked away somewhere inbetween the HAL, and User level So that I can start to slowly replace HAL stuff over time. Also I don’t like that this throws a warning for every library file that doesn’t use it... Possible solution: Move this to `dsy_hal_interface.h` – and then explicitly include it in the lower-level files that need it. 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 Leaving non-cplusplus ver-

sion in place below so as not to break examples yet... Order of ADC Channels  
for accessing dsy\_adc.h Mapping of LEDs via dsy\_leddriver.h Hardware related

defines. Switches Encoder Extra Peripherals enums for controls, etc. All knobs connect to ADC1\_\_INP10 via CD4051 mux This is a Board Specific File I don't

think it actually belongs in the library. Any new piece of hardware can just have their own board file. This will allow minor pin changes, etc. not to require changing the library in a million places. Specifies whether generic initialization will be done within the `daisy_seed_init`, or not. Allows for more selective init Probably should move this to a `dsy_handle.h` So that it can be used in the other peripheral initializations, etc. (E.g. Audio needs both SAI, and I2C for most devices.) THIS BREAKS WHEN ITS INLINED? WM8371 Codec support. SDRAM

for 32MB AS4C16M16SA (and 64MB equivalent). Thanks to whoever this awesome person is: [http://main.lv/writeup/stm32f4\\_sdram\\_configuration.md](http://main.lv/writeup/stm32f4_sdram_configuration.md) The Init function is basically a copy and paste. He has references to timing, etc. for now we're configuring the RAM to run at 108MHz 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 DATA section must also fit in flash in order to be initialized. Determines whether chip is initialized, and activated. This is only the pins that can change on a board-to-board basis. Pins that have functions that cannot be moved to another pin will be hardcoded into the driver.

- SDNWE is the only pin that i've seen move, Fixed maximums for paral-

lel/daisychained use These could be expanded TODO Fix hard coding of



these parameters Usage: Using the `dsy_switch_state()`, will work with

no setup other than init. For edge, and time based functions, you'll have to call `debounce()` at a regular interval (i.e. 1ms) In order not to miss those events, the rising/falling edge checks should be made at the same frequency as the `debounce()` function.

## Parameter

Simple parameter mapping tool that takes a 0-1 input from an hid\_ctrl.

TODO: Move init and process to .cpp file - i was cool with them being in the h file until math.h got involved for the log stuff.

### curve settings

Curves are applied to the output signal

```
enum
{
    PARAM_CURVE_LINEAR,
    PARAM_CURVE_EXP,
    PARAM_CURVE_LOG,
    PARAM_CURVE_CUBE,
    PARAM_CURVE_LAST,
};
```

### parameter class

#### 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.

```
inline void init(hid_ctrl input, float min, float max, uint8_t 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.

```
inline 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; }
```

## 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

## Types

### **dsy\_i2c\_periph**

Specifies 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);
```

### **dsy\_i2c\_hal\_handle**

Returns a pointer to the HAL I2C Handle, for use in device drivers.

```
I2C_HandleTypeDef *dsy_i2c_hal_handle(dsy_i2c_handle *dsy_hi2c);
```

Currently Sample Rates are not correctly supported. We'll get there.

## **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);
```



Sets clock speeds, etc.