DaisySP

Contents

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 the 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 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; }
```

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

Sample Rates are not correclty supported. We'll get there. Sets clock speeds,

etc.