BoardGen.pl : a board.h generator

requirement

In order to understand this doc, you need to know how STM32 MCU pin multiplexer works, and how ChibiOS manage to use it for initial (static) configuration.

ChibiOS way of defining Port Access Layer initial state for each pin

When a ChibiOS application start, it has to call halInit() early in the main, this will configure all the pins needed by the application. What does halInit, among initialing all peripherals, is configuring each of the pins of the MCU, taking data in a file named board.h which contain numerous macro that give information for all the pins for all the parameters :

- input or output or adc, or alternate function
- opendrain or pushpull
- pullup or pulldown resistor
- slew rate,
- etc etc

Because each pin of each port of a given MCU is described, not only the one you actually need in your application, modifying this file by hand is not recommended, for some MCU, it’s for now a 115Kb file with 2000 lines full of C macro, a nightmare to edit

This board.h is used by a board.c file which initialize all the available ports for a given MCU, and can do a little more : it’s here that you have to implement some specific board function (like the function which verify that a sdcard ou mmc card is present by checking if a presence switch is closed)

ChibiOS has his own board.h generator which use as source some xml definition of board, in file usually called board.chcfg

All examples that come with ChibiOS use an adapted board.h file.

Goals

STMicroElectronics offer a free multi-platform tool, named STM32CubeMX, which is a graphical tool that can help to configure an MCU or a known development board. I encourage people to use it, it helps a lot to understand how alternative function multiplexer works, (and the clock configuration panel is a must see too).

There is already numerous project, to generate board.h, or board.chcfg from STM32CubeMX output.

So is there a need for Yet Another Board Generator tool ?

It depends of your usage, if you have few boards with static configuration that barely change, you can live with existing tools. On the other way, if you experiment new wiring often, you want to avoid launching heavy graphical tools to change a pin parameter, you use ChibiOS as a teaching platform where student have to frequently change pin parameters, this tool helps.

The idea is to edit a simple, text based, relaxed syntax file where you just have to declare the pin you need, the other one will be generated with default parameters.

ChibiOS build system is makefile based, so we have to had a rule in the makefile which call boardGen.pl to generate board.h from board.cfg

highlights :

- use stm32cubemx MCU description file to make check that a function can be assignated to a particular pin
- find alternate function number, user just have do bind the peripheral name with the pin
- warn is you in case of suspect configuration :
  - use non 5 volts tolerant pin configured in input
  - outpull pushpull with pullup/pulldown activated
  - same function routed to more than one pin
  - ...
- shortcut for function that are commonly used (all peripheral managed by chibios, plus some like LED,SYS(tem),OSC(illaror), etc

Install

On ***linux*** machine, perl is installed, so there is just need to install needed modules

For ubuntu, by example :

- modules that are ubuntu (or debian packaged) :
  - `sudo apt install libmodern-perl-perl libxml-libxml-perl`
- modules that are not packaged by debian (or ubuntu) but are in CPAN :
  - `sudo perl -MCPAN -e 'install String::LCSS'`

on ***macosx***, perl is installed, but there is no distribution package manager, so all packages should be installed with cpan :

- `sudo perl -MCPAN -e 'install Modern::Perl'`
- `sudo perl -MCPAN -e 'install XML::LibXML'`
- `sudo perl -MCPAN -e 'install String::LCSS'`

on ***windows*** perl is not installed

- install perl : active perl is the best berl available on windows
- use cpan, ot the graphical package manager which come with active perl to install theses 3 packages : `Modern::Perl, ML::LibXML, String::LCSS`

Usage

There is two kind of usage :

- generate board.h from board.cfg :
  - `boardGen.pl [options] cfgFile.cfg boardFile.h`
    - `if boardFile.h is '-', stdout is used`
    - `options :`
      - `--no-pp-pin : don't prepend pin name with port letter and pin number`
        - `if you have a pin described (cf next chapter board.cfg syntax) like this :`
          `"PC00 LED1 OUTPUT PUSHPULL SPEED_VERYLOW FLOATING LEVEL_HIGH"`
          By default, and because main use is for education (teacher have to find quickly student mistake), the generated name for the pin will be PC00_LED1, so teacher can verify easily that LED1 is actually wired to PC00.
          On the other hand, if the tool is used to generate file for a project, one want to change pin use without having to change everywhere in the source the pin name, in this case use –`no-pp-pin` and the generated name wont mention port and pin number, it will be LED1
      - `--no-pp-pin :` same as above, but for line macros
        Without option, line macro will be LINE_C00_LED1, with option, it will be LINE_LED1
      - --no-adcp-in : do not permit to use non 5V tolerant pins for input or alternate, unless explicitly promising in the .cfg that voltage won't exceeds 3.3v (cf syntax)
      - --no-error : do not abort when pin check is not OK
- interactively find on which pin a fonction is available on the MCU selected in the board.cfg (that can save you opening the doc or launching STM32CubeMX)
  - boardGen –find regexp board.cfg
    - examples :
      - boardGen.pl --find 'UART4' board.cfg
        UART4_TX on PA00 has AF8
        UART4_RX on PA01 has AF8
        UART4_TX on PC10 has AF8
        UART4_RX on PC11 has AF8

- boardGen.pl --find 'ADC._IN11' board.cfg
  ADC1_IN11 on PC01 is not alternate
  ADC2_IN11 on PC01 is not alternate
  ADC3_IN11 on PC01 is not alternate
- all available function : boardGen.pl --find . board.cfg
  - that can be used to know all available functions on a given pin :
    - boardGen.pl --find . board.cfg | grep PA10
      - DCMI_D1 on PA10 has AF13
        USB_OTG_FS_ID on PA10 has AF10
        USART1_RX on PA10 has AF7
        TIM1_CH3 on PA10 has AF1

- interactively find the dma parameters for a given function, in a form that is directly embeddable in mcuconf.h. This is an help if you manage to directly use DMA using ChibiOS macros and functions, or using a driver that I have made public : http://www.chibios.com/forum/viewtopic.php?t=4596&p=32570#p32570, or using your own driver.
  - boardGen.pl --dma=regexp board.cfg
  - examples:
    - boardGen.pl --dma=USART1_TX board.cfg

```
USART1_TX : [stream(2, 7) C4]
#define STM32_USART1_TX_DMA_STREAM          STM32_DMA_STREAM_ID(2, 7)
#define STM32_USART1_TX_DMA_CHANNEL          4
#define STM32_USART1_TX_DMA_IRQ_PRIORITY     6
#define STM32_USART1_TX_DMA_PRIORITY         2
```

    - boardGen.pl --dma='USART1_[TR]X' board.cfg

```
USART1_TX : [stream(2, 7) C4]
#define STM32_USART1_TX_DMA_STREAM          STM32_DMA_STREAM_ID(2, 7)
#define STM32_USART1_TX_DMA_CHANNEL          4
#define STM32_USART1_TX_DMA_IRQ_PRIORITY     6
#define STM32_USART1_TX_DMA_PRIORITY         2

USART1_RX : [stream(2, 2) C4]
#define STM32_USART1_RX_DMA_STREAM          STM32_DMA_STREAM_ID(2, 2)
#define STM32_USART1_RX_DMA_CHANNEL          4
#define STM32_USART1_RX_DMA_IRQ_PRIORITY     6
#define STM32_USART1_RX_DMA_PRIORITY         2

USART1_RX : [stream(2, 5) C4]
#define STM32_USART1_RX_DMA_STREAM          STM32_DMA_STREAM_ID(2, 5)
#define STM32_USART1_RX_DMA_CHANNEL          4
#define STM32_USART1_RX_DMA_IRQ_PRIORITY     6
#define STM32_USART1_RX_DMA_PRIORITY         2
```

In the case there a numerous possibilities, you'll have to pick just one.

Makefile integration :

```
$(OBJS): board.h

board.h: board.cfg Makefile
	boardGen.pl --no-adcp-in	$< $@
```

board.cfg syntax

There is 3 sections

section 1 :

- MCU_MODEL : one has to use exact name listed on STM32CubeMX utility, (you can launch STM32CubeMX or examine the dierctory where STM32CubeMX store MCU description : on rootdir/db/mcu/
- CHIBIOS_VERSION : either 2.6, or 3.0 for any version of chibios >= 3.0 (it will works with 18.2 by example)

section 2 :

begin with keyword HEADER, and ends with keyword CONFIG

this section is a plain C header where you have to define ChibiOS macro, and you can also define your own set of macro for use in your application or librairies

ChibiOS that have to be defined are :

STM32_LSECLK if LSE is used

STM32_HSECLK if HSE is used

STM32_VDD : board voltage

MCU type as defined in the ST header (by example #define STM32F407xx)

Often, you can copy section 2 from one of the numerous board.h in the ChibiOS archive (of course only if you use the same mcu as the example) If you have your own board, you can start from a given board.h and adapt values according to your custom board.

Section 3 : pin definition

First you have to define DEFAULT configuration, all the pins which are not defined after will take this configuration :

```
DEFAULT                 INPUT   PUSHPULL SPEED_VERYLOW PULLDOWN LEVEL_LOW  AF0
```

then, all the pins of your board :

example :

```
PC00    LED1          OUTPUT  PUSHPULL   SPEED_VERYLOW FLOATING LEVEL_HIGH 0_VOLT
```

The order of the 3 first fields must be respected :

- PC00 : which port, which pin number in the port
- LED1 : the name of the pin. Depending on option when boardGen is invoked, it will generate macro #define LED1 0U or #define PC00_LED1 0U and
  #define LINE_C00_LED1 PAL_LINE(GPIOC, 0) or #define LINE_LED1 PAL_LINE(GPIOC, 0)
  you will then be able to to palSetLine(LINE_LED1 or LINE_C00_LED1) in your code.
- OUTPUT : configuration of the pin : can be OUTPUT, INPUT, ALTERNATE, ANALOG , or a shortcut that will be described later.

The other fields are attributes, they are optional, if not given, the value of the DEFAULT is used, and can be given in any order, can also be specified more than once, in this case the later win.

Attributes are :

- pullup, pulldown resistor activation : FLOATING, PULLUP, PULLDOWN
- output mode : OPENDRAIN, PUSHPULL
- slew rate : SPEED_VERYLOW, SPEED_LOW, SPEED_MEDIUM, SPEED_HIGH
- initial output level : LEVEL_HIGH, LEVEL_LOW
- alternate function multiplexer
  - in numeric form (not recommended) the syntax is AFx where x is the alternate function number (see stm32 reference manual for your MCU)
  - in named form (recommended, the script will be able to check that the function is available on the pin) the syntax is AF:function_name by example : AF:USART1_TX.
    Name are as they are spelled by STM32CubeMX tool, or can be listed with
    boardGen –find . board.cfg for your actual cpu
- the maximum voltage that can be sent on pin : 0_VOLT or 3.3_VOLTS or 5_VOLTS,
  - not needed on pin configured in output pushpull, or pin that are 5 volts tolerant.
- on specific MCU (STM32L4 by example) there is 2 more fields :
  - ASCR (connection to analog circuitry) : ASCR_DISABLED or ASCR_ENABLED
  - LOCKR (pin function cannot be dynamically modified) : LOCKR_DISABLED or LOCKR_ENABLED

Shortcuts are : SYS OSC ADC COMPIN COMPOUT OPAMP DAC PWM ICU I2C SPI UART OTG ETH FSMC SDIO SDIOCK SWDIO SWCLK CAN DCMI LED PASSIVE.