Using hardware and software to make new stuff

Search

type, hit enter

Categories

- 3D Printing (1)
- Affinity Photo (1)
- Aide-memoir (1)
- Analog (1)
- Ansible (3)
- Dates to Remember (3)
- Docker (1)
- Electronics (144)
- ESP8266 (12)
- FPGA (4)
- Garden (1)
- General (8)
- Home Built CPU (6)
- Informatica (1)
- Internet of Things (8)
- iOS (2)
- KiCad (4)
- MSP430 (2)
- Netduino (49)
- NuttX (9)
- Photography (3)
- Pico (10)
- Raspberry Pi (17)
- Silverlight (6)
- Software Development (88)
- STM32 (5)
- STM8 (54)
- Tips (5)

Archives

- July 2024
- June 2024
- May 2024
- April 2024
- March 2024
- February 2024
- January 2024
- December 2023
- November 2023
- October 2023

- April 2023
- February 2020
- January 2020
- July 2019
- February 2018
- July 2017
- June 2017
- April 2017
- March 2017
- February 2017
- October 2016
- September 2016
- July 2016
- June 2016
- May 2016
- April 2016
- March 2016
- January 2016
- December 2015
- November 2015
- October 2015
- August 2015
- June 2015
- May 2015
- April 2015
- March 2015
- February 2015
- January 2015
- December 2014
- October 2014
- September 2014
- August 2014
- July 2014
- June 2014
- May 2014
- March 2014
- February 2014
- January 2014
- December 2013
- November 2013
- October 2013
- September 2013
- August 2013
- July 2013
- June 2013
- May 2013
- April 2013
- March 2013
- January 2013
- November 2012
- October 2012
- September 2012
- August 2012
- July 2012
- June 2012
- May 2012

- October 2011
- September 2011
- July 2011
- June 2011
- May 2011
- April 2011
- March 2011
- February 2011
- January 2011
- December 2010
- April 2010

Configuring the STM8S System Clock – The Way of the Register

This post continues the low level use of the STM8S registers and shows you how to configure the STM8S system clock. In particular we will configure the system clock to use the internal high speed clock.

So let us start with a little background. The STM8S can be driven by three clock sources:

- High Speed Internal (HSI) RC oscillator running at 16MHz
- Low Speed Internal (LSI) RC oscillator running at 128KHz
- External clock (1-24 MHz) which can be either an external crystal or user supplied clock.

We will be using the internal oscillator as this will be satisfactory for our purposes. An external clock may be more accurate but for most of the work I am doing I am happy to sacrifice the accuracy for the lower component count and greater simplicity of my designs.

At start up, the STM8S will be using HSI with a prescalar to bring the initial clock speed down to 2MHz. We will be resetting the clock prescalar to allow the chip to run at the full 16MHz.

The switching algorithm we will use is as follows:

1. Reset all of the clock registers to the power on state
2. Select the clock we wish to use
3. Enable the switching mechanism
4. Wait until the STM8S indicates that the clock has stabilised

Clock Registers

The description of the clock registers start around page 89 of the STM8S Reference Manual. In this article we will run through the registers concentrating on the values we will be setting. For a fuller description you should refer to the STM8S Reference Manual.

CLK_ICKR – Internal Clock Register

The only bits we will really be interested in for this example bits 1 and 0, this allows us to select the HSI and see if it is ready.

Register Name	Description
REGAH	Control the actions of the internal voltage regulator when the chip enters Active-halt mode.
LSIRDY	Set by hardware, this bit determines if the LSI is stabilised and ready. 0 indicates the oscillator is not ready, 1 indicates it is stable.

	enabled. 0 = disabled, 1 = enabled
HSIRDY	Set and reset by hardware, this bit determines if the HSI is stable and ready for use.
HSIEN	Set and cleared by the user program, this bit determines if the HSI has been selected as the clock source for the chip. This can also be set by hardware to indicate that the HSI is required. 0 indicates HSI is not selected, 1 indicates HSI is selected.

Our example here will be resetting the clock source on the STM8S chip to use the internal oscillator using HSIEN and using the HSIRDY bit to determine when the clock has been setup and is stable.

CLK_ECKR – External Clock Register

This register is used to provide information about the state of the external clock. We will not be using this register for anything other than resetting the register to the power on state.

Register Name	Description
HSERDY	External high speed clock state, 0 = not ready, 1 = ready.
HSEEN	Enable/disable the external high speed clock. 1 = enabled, 0 = disabled

CLK_CMSR – Clock Master Status Register

This register is set and cleared by hardware and indicates which clock source is currently selected as the master clock source.

Value	Clock Source
0xe1	HSI
0xd2	LSI
0xb4	HSE

CLK_SWR – Clock Master Switch Register

This register is written to by the user program and is used to select the master clock source. The register cannot be written to whilst a clock source switch is on-going. You can test for the switch using the SWBSY in the CLK_SWCR register.

Value	Clock Source Selected
0xe1	HSI
0xd2	LSI
0xb4	HSE

Clock Switch Control Register – CLK_SWCR

The Clock Switch Control Register is used to control how and when the clock switch is performed.

Register Name	Description
SWBSY	This bit indicates if a clock switch is in progress, 1 = clock switch is on-going, 0 indicates that no switch is in progress.

SWIF	Indicates if a clock switch is taking place (interrupts enabled) or is the target clock is ready (interrupts disabled)

In our case we will be using this register to start the clock switch process.

CLK_CKDIVR – Clock Divider Register

The system clock can be divided down to allow for lower clock frequencies to be used. This register allows for the HSI prescalar (divider) and the CPU prescalar to be set.

Register Name	Description
HSIDIV	HSI prescalar
CPUDIV	CPU Prescalar

The following values can be used for the HSI prescalar:

Value	Prescalar
0	HSI frequency
1	HSI frequency divided by 2
2	HSI frequency divided by 4
3	HSI frequency divided by 8

The following values are allowed for the CPU prescalar:

Value	Prescalar
0	Master clock frequency
1	Master clock frequency divided by 2
2	Master clock frequency divided by 4
3	Master clock frequency divided by 8
4	Master clock frequency divided by 16
5	Master clock frequency divided by 32
6	Master clock frequency divided by 64
7	Master clock frequency divided by 128

In this case we will want the CPU to be running at full speed and so no prescalars will be applied to either the master clock or the HSI.

CLK_PCKENR1 & CLK_ PCKENR2 – Peripheral Clock Gating Registers

These two registers enable (set to 1 and disable (set to 0) the clocks used to control the peripherals used in the application. I have fallen foul of these values a few times. If you set these registers or zero then you will turn off the peripheral clocks. This has the effect of turning off the feature you may be trying to use. If in doubt set both of these registers to 0xff. You can always refine the values later by restricting the clock signals to only those which you are using.

Register Name	Peripheral	Description
CLK_PCKENR1	TIM1	Enable/disable timer 1
	TIM3	Enable/disable timer 2
	TIM2/TIM5	Enable/disable timer 2/5 (product dependent)
	TIM4/TIM6	Enable/disable timer 4/6 (product dependent)

	ADC	Enable/.disable the analogue to digital converter
	AWU	Enable/disable the watchdog service

Enabling the service will use the frequency of the master clock. Disabling the clock to the service will turn it off.

The <iosstm8s103f3.h> file does not contain definitions for these bits, you need to define your own constants for these. I have recently taken to just turning them all on as I'm not too worried about the power requirements for the applications I am working on.

CLK_CSSR – Clock Security System Register

We will not be using this register in this example apart for setting the default power on value. Please see the STM8S Reference Manual for more information.

CLK_CCOR – Configurable Clock Output Register

We will not be using this register in this example apart for setting the default power on value. Please see the STM8S Reference Manual or future examples for more information.

CLK_HSITRIMR – HSI Clock Calibration Trimming Register

We will not be using this register in this example apart for setting the default power on value. Please see the STM8S Reference Manual for more information.

CLK_SWIMCCR – SWIM Clock Control Register

We will not be using this register in this example apart for setting the default power on value. Please see the STM8S Reference Manual for more information.

Software

As previously mentioned, this example will simply set the system clock to the maximum speed the HSI can deliver. We will then toggle a GPIO pin and hook this up to a scope to get an idea of the output. We will never see the full 16MHz clock speed of the HSI but we should be able to see that the output frequency changes when we change the clock dividers.

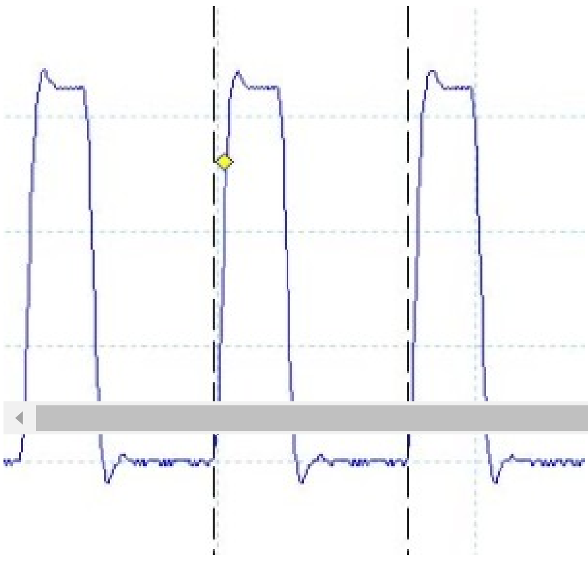
So first thing, let's get the system set up and running at the full 16 MHz and see what we the scope produces.

```c
#include <intrinsics.h>
#include <iostm8s103f3.h>

//
//  Setup the system clock to run at 16MHz using the internal oscillator.
//
void InitialiseSystemClock()
{
    CLK_ICKR = 0;                       //  Reset the Internal Clock Register
    CLK_ICKR_HSIEN = 1;                 //  Enable the HSI.
    CLK_ECKR = 0;                       //  Disable the external clock.
    while (CLK_ICKR_HSIRDY == 0);       //  Wait for the HSI to be ready for
    CLK_CKDIVR = 0;                     //  Ensure the clocks are running at
    CLK_PCKENR1 = 0xff;                 //  Enable all peripheral clocks.
    CLK_SWR = 0xe1;                     //  Use HSI as the clock source.
    CLK_SWCR = 0;                       //  Reset the clock switch control re
    CLK_SWCR_SWEN = 1;                  //  Enable switching.
    while (CLK_SWCR_SWBSY != 0);        //  Pause while the clock switch is b
}

//
//  Main program loop.
//
int main(void)
{
    __disable_interrupt();
    //
    //  Initialise Port D.
    //
    PD_ODR = 0;             //  All pins are turned off.
    PD_DDR_DDR4 = 1;        //  Port D, bit 4 is output.
    PD_CR1_C14 = 1;         //  Pin is set to Push-Pull mode.
    PD_CR2_C24 = 1;         //  Pin can run up to 10 MHz.
    //
    //  Now setup the system clock
    //
    InitialiseSystemClock();
    __enable_interrupt();
    while (1)
    {
        PD_ODR_ODR4 = 1;    // Turn Port D, Pin 4 on.
        PD_ODR_ODR4 = 0;    // Turn Port D, Pin 4 off.
    }
}
```

Running this code resulted in the following trace on the oscilloscope:



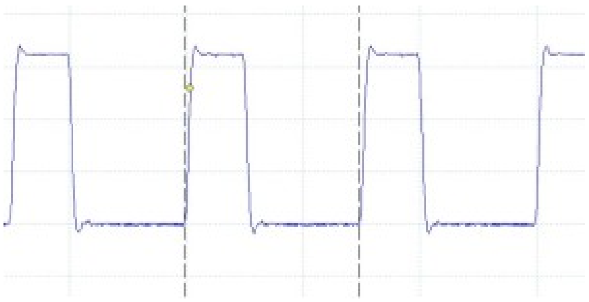
GPIO Output when the System Clock is set to HSI with no Divider.

The frequency of the out was 2.693Mhz.

to this:

```
CLK_CKDIVR = 0;                         //  Ensure the clocks are running at full
CLK_CKDIVR_HSIDEV = 1;                  //  Set the HSI divider to 2.
```

The result is the following trace:



GPIO Output when System Clock is set to HSI divided by 2

This looks remarkably like the trace above but the oscilloscope shows that the frequency of the signal is now 1.335MHz, or approximately half of the frequency of the first trace.

The full project for this article can be downloaded from here. This code has been tested with my reference platform, the Variable Labs Protomodule and the STM8S Discovery board.

Compatibility

System	Compatible?
STM8S103F3 (Breadboard)	✓
Variable Lab Protomodule	✓
STM8S Discovery	✓

Tags: Electronics, Software Development, STM8

Sunday, July 29th, 2012 at 4:19 pm • Electronics, Software Development, STM8 • RSS 2.0 feed Both comments and pings are currently closed.

Comments are closed.

Pages

- About
- Making a Netduino GO! Module
- The Way of the Register
- Making an IR Remote Control
- Weather Station Project
- NuttX and Raspberry Pi PicoW

Support This Site

Currently Reading