Phase Locked Loop

Internally the Philips version of the ARM Cortex M4 processor runs at 168 MHz which is a cycle time of just 5.952 nsec. This would imply that a very fast external interface bus would be needed to communicate with the outside world. For many embedded controller systems this high speed interface is unnecessary. The solution to this problem is to add a phase locked loop into the hardware which allows the CPU to run at 168 MHz and to simultaneously communicate with a synchronized 8 MHz system on the outside. Thus, the STM32F407VG requires only an 8 MHz crystal and from the outside, the user sees it as an 8 MHz system although inside it is actually executing instructions 21 times faster.

A system diagram for a phase locked loop (PLL) is shown below. An 8 MHz crystal oscillator runs the external system busses and clocks. Internally a voltage controlled oscillator runs at about 168 MHz. The VCO is adjusted to run in synchronization with the 12 MHz oscillator by comparing the phase of the 8 MHz oscillator with that of the 168 MHz oscillator divided by 21. The 168 MHz oscillator therefore runs in synchronization with the 8 MHz oscillator and is just as precise. Effectively, the PLL acts as a frequency multiplier.

Memory Map

There are no separate I/O instructions on this processor. All I/O is memory mapped. For example, I/O port A can be addressed at address 0x40020014. Figure 2 shows the memory map for the processor. From the memory map we see that the STM32F407G has space for 512 Mbytes of Flash ROM and 512 Mbytes of SRAM. For the Discovery board implementation we have 1024K of flash programmable memory and 192K of SRAM.
Figure 2
Memory map for the ARM Cortex M4 processor.

System Clock
Three different clock sources can be used to drive the system clock (SYSCLK):
- HSI oscillator clock
- HSE oscillator clock
- Main PLL (PLL) clock
The devices have the two following secondary clock sources:
- 32 kHz low-speed internal RC (LSI RC) which drives the independent watchdog and, optionally, the RTC used for Auto-wakeup from the Stop/Standby mode.
- 32.768 kHz low-speed external crystal (LSE crystal) which optionally drives the RTC clock (RTCCLK)

The STM32F407VG processor can make use of three independent oscillators. These are:
- HSI Clock – High Speed Internal Clock is generated from an internal 16 MHz RC oscillator so that the system can be operated without a crystal albeit not as accurately.
- HSE Clock – High Speed External Clock is generated externally.
- PLL Clock – Phase Locked Loop Clock is generated from the phase locked loop at 168 MHz.

Figure 3 shows a block diagram of the clock generation signals. For the board that we are using in class, an 8 MHz crystal has been added and we will used the PLL Clock exclusively.

In addition to the main system clock there are two low speed clocks:
- LSI clock – Low Speed Internal Clock generated from a 32 KHz internal RC oscillator
- LSE clock – Low Speed External Clock generated from a 32.768 KHz external crystal and built in oscillator.

The low speed clocks can be used by the built in random number generator, the watchdog timer, or as the basis for a real time clock. Note that \(32768 = 2^{15}\)
May 15, 2015

Figure 3
Clock generation for the ARM Cortex STM32F407G processor.

A Programmer's Model of the STM32F407VG ARM Processor
From a C programmer's point of view the STM32F407VG consists of a CPU plus several I/O modules. The CPU has 16 general purpose registers which are accessible in assembly language. These are shown in Figure 4. The I/O modules include:

Digital peripherals:
82-GPIO pins can be used as edge and level sensitive interrupt sources. High-current output driver (25 mA) on one pin in or out. GPIO is 5 volt tolerant.

Timers
Twelve 16-bit and two 32-bit general purpose counter/timers

Analog peripherals:
Three 12-bit ADCs and two 12-bit DACs

Serial interfaces:
Four USARTs plus two UARTs
Three SPI channels and two I2C channels
Two CAN bus interfaces

These I/O ports are bit programmable (each bit can be independently programmed). Each of the I/O modules consists of several special function registers that define their operation. The definitions for all of the special function registers is in stm32f407vg.h and this file needs to be included in all C code. You can look at this file to see the assigned register names using any text editor. Figure 5 gives a block diagram for the processor. Figure 6 give the pin assignments for the Discovery board with the STM32F407VG chip. See the data sheet at http://www.st.com/web/en/resource/technical/document/datasheet/DM00037051.pdf pp. 46-58 for a complete definition of all of the pin functions for the LQFP100 package.

Figure 4
CPU Registers for the ARM Cortex STM32F407VG
Figure 5
Block diagram for the ARM Cortex STM32F407VG processor.
Digital I/O
Digital I/O is referred to as General Purpose I/O or GPIO in the reference manuals. The block diagram of Figure 5 shows that there are 9 GPIO ports labeled PA through PI
where PA through PH are 16-bit ports and PI is a 12-bit port. This totals 140 pins but the package that we are using is only a 100 pin package so many of these I/O pins are not available. The Discovery board has two connectors each consisting of two columns of pins. Connector P1 is on the left in Figure 6 and P2 is on the right. P1 and P2 each have 50 pins in two 25-pin columns for a total of 100 pins so that all pins of the LQFP 100 package are brought out. See Table 5 in the User's Manual at http://www.st.com/st-web-ui/static/active/cn/resource/technical/document/user_manual/DM00039084.pdf for a complete mapping of all of the package pins to the board pins. For the I/O ports the Table 1 gives the pins are available for the Discovery Board:

### Table 1

<table>
<thead>
<tr>
<th>Port</th>
<th>Pins on Discovery Board</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA</td>
<td>PA0 to PA10 plus PA13 to PA15*</td>
</tr>
<tr>
<td>PB</td>
<td>PB0 to PB15</td>
</tr>
<tr>
<td>PC</td>
<td>PC0 to PC15</td>
</tr>
<tr>
<td>PD</td>
<td>PD0 to PD15</td>
</tr>
<tr>
<td>PE</td>
<td>PE0 to PE15</td>
</tr>
<tr>
<td>PF</td>
<td>None</td>
</tr>
<tr>
<td>PG</td>
<td>None</td>
</tr>
<tr>
<td>PH</td>
<td>PH0-PH1</td>
</tr>
<tr>
<td>PI</td>
<td>None</td>
</tr>
</tbody>
</table>

PA11 and PA12 are not brought out to board pins.

Each port has ten associated registers which configure the port since each pin is capable of multiple functions. The ten registers and their function is shown in Table 2.

### Table 2

<table>
<thead>
<tr>
<th>Register</th>
<th>Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>GPIOA_MODER</td>
<td>Sets pin to input, output, analog, or alternate function.</td>
</tr>
<tr>
<td>Pull up/Pull down</td>
<td>GPIOA_PUPDR</td>
<td>Sets pin to have a pull up, a pull down, or floating.</td>
</tr>
<tr>
<td>Speed</td>
<td>GPIOA_OSMODE</td>
<td>If the pin is output it can be set to low, medium, fast, or high speed.</td>
</tr>
<tr>
<td>Output type</td>
<td>GPIOA_OTYPE</td>
<td>If the pin is output, it can be set to push-pull or open drain.</td>
</tr>
<tr>
<td>Input Data Register</td>
<td>GPIOA_IDR</td>
<td>Holds input data</td>
</tr>
<tr>
<td>Output Data Register</td>
<td>GPIOA_ODR</td>
<td>Holds output data</td>
</tr>
<tr>
<td>Output bit set/reset register</td>
<td>GPIOA_BSRR</td>
<td>Writing a one to this register allows individual port bits to be set or reset.</td>
</tr>
<tr>
<td>Alt Function Register low</td>
<td>GPIOA_AFRL</td>
<td>Sets pins for alternate functions such as UART, Timers, CAN, SPI, etc.</td>
</tr>
<tr>
<td>Alt Function Register high</td>
<td>GPIOA_AFRH</td>
<td></td>
</tr>
<tr>
<td>Lock register</td>
<td>GPIOX_LCKR</td>
<td>Allows a bit to be locked – it cannot be changed except by a reset.</td>
</tr>
</tbody>
</table>
All of the ports are synchronous to a clock. This means that the user must first enable the clock for the port before it can be used for input or output. There are three clock registers related to peripherals: RCC_AHB1ENR, RCC_AHB2ENR, and RCCAHB3ENR. All of the GPIO ports are on RCC_AHB1ENR. Bits 0, 1, 2, 3, and 4 enable/disable the clock for GPIO ports A, B, C, D, and E respectively. A 1-bit enable the clock and a 0-bit disables the clock.

The ARM processor does not support the bit data type so there are no instructions that directly operate on bits. You can use the AND and OR operators or you can use the bit set/reset register.

**Using AND and OR operators to do bit I/O**

Suppose we want to set bit 3 on Port A to a 1. We could do this like this:

```
GPIOA_ODR = GPIOA_ODR | 0x4;
```

Which can be written as:

```
GPIOA_ODR |= 0x4;
```

Or it can be written as:

```
GPIOA_ODR |= (1 << 3);  //<< is the shift operator in C
```

Likewise, we can use the AND operator to clear a bit to zero. To clear bit 3 of Port A to zero we can write any of the following:

```
GPIOA_ODR = GPIOA_ODR & 0xFFFB;  //0xFFFB = 1111 1111 1111 1011
GPIOA_ODR &= 0xFFFB;
GPIOA_ODR &= ~(1 << 3);
```

**Using the bit set/reset register to do I/O**

Each port has a bit set/reset register named GPIOx_BSRR. Each port is only 16-bits long but the GPIOx_BSRR register is 32 bits long. Placing a zero in the BSRR register has no effect. Placing a 1 in bit-\( n \) in the BSRR register where \( 0 \leq n \leq 15 \) causes bit-\( n \) in the ODR register to be a 1. Placing a 1 in bit-\( n \) in the BSRR register where \( 16 \leq n \leq 31 \) causes bit \( n - 16 \) to be a zero in the ODR register.

Suppose we want to set bit 3 on Port A to a 1. We could do this like this:

```
GPIOA_BSRR |= (1 << 3);
```

If we want to clear bit 3 on Port A we could do this:

```
GPIOA_BSRR |= (1 << 19);  //19 = 16 + 3 so bit 3 is cleared.
```

If you simultaneously set a bit to both zero and a one the one will win out.

The program on the next page shows how to do digital I/O.
May 15, 2015

/*GPIOInOut.c                 May 15, 2015
   This program toggle pin PA7. It also inputs from pin PB2 and output
   to PA9 as fast as possible
*/
#include "stm32f407vg.h"

int main()
{
    int i, tmp;
    //Clock bits
    RCC_AHB1ENR |= 1;  //Bit 0 is GPIOA clock enable bit
    RCC_AHB1ENR |= 2;  //Bit 1 is GPIOB clock enable bit
    //I/O bits PA7
    GPIOA_MODER |= 0x4000;  //Bits 15-14 = 01 for digital output on PA7
    GPIOA_OTYPER register resets to 0 so it is push/pull by default
    GPIOA_OSPEEDER |= 0xC000;  //Bits 15-14 = 11 for high speed on PA7
    //PUPDR defaults to no pull up no pull down
    //I/O bits PA9
    GPIOA_MODER |= 0x40000; //Bits 18-19 = 01 for digital output on PA9
    GPIOA_OTYPER register resets to 0 so it is push/pull by default
    GPIOA_OSPEEDER |= 0xC0000; //Bits 18-19 = 11 for high speed on PA9
    //PUPDR defaults to no pull up no pull down
    //I/O bits PB2
    GPIOB_OSPEEDER |= 0xC;     //PB2 to high speed
    //All bits are input by default.
    //Main program loop
    tmp = 0;
    while(1)
    {
        if(tmp == 0)
            GPIOA_ODR &= ~(1 << 7);  //Toggle PA7 depending on tmp
        else
            GPIOA_ODR |= (1 << 7);
        tmp = ~tmp;
        //Input bit PB2 and shift it 7 places to output to PA9
        if((GPIOB_IDR & 0x4) == 0)
            GPIOA_ODR |= (1 << 9);
        else
            GPIOA_ODR &= ~(1 << 9);
        //for(i=0;i<10000;i++);  //Put this in to slow down toggle
    }
}