Fast Frequency Function Generator v0.5

Marcus Jansson

May 26, 2011

```
                ___
         __    |   |    __
    _   |  |   |   |   |  |   _
_._| |__|  |___|   |___|  |__| |_._

  Fast  Frequency  Function  Generator
_ _   __    ___     ___    __   _ _
 , |_|  |  |   |   |   |  |  |_| ,
        |__|   |   |   |__|
               |___|
```

Contents

1 Introduction **3**

2 Materials and methods **3**
2.1 Hardware . . . 3
2.2 Hardware setup . . . 3
2.2.1 Development board . . . 3
2.2.2 MCU Fuses . . . 4
2.3 Software tools . . . 4
2.4 Firmware source code . . . 5
2.4.1 Initialization . . . 5
2.4.2 Main loop . . . 5
2.4.3 ADC module . . . 5
2.4.4 GPIO module . . . 6
2.4.5 Timer0 module . . . 6
2.4.6 Timer1 module . . . 6
2.5 Power considerations . . . 6
2.6 Compilation and programming . . . 7

3 Results **7**

4 Discussion **7**
4.1 MCU . . . 7
4.2 Oscilloscope . . . 7
4.3 Development board . . . 7
4.4 Timer0 . . . 8
4.5 Timer1 . . . 8
4.6 ADC . . . 8
4.7 Problems . . . 9
4.8 Hardware . . . 9
4.9 Software . . . 9

5 Conclusion **9**

6 References **10**

7 Future development **10**

1 Introduction

This is the documentation for the Fast Frequency Function Generator (F3G) v0.5 built around an AVR ATtiny45 microcontroller (MCU).

The F3G produce a variable stable square wave signal in the range of 15.4 Hz - 5.3 MHz which can be used in a variety of applications.

The goal for the F3G was to make a signal generator to test the new/old analog oscillator in possession of the author.

2 Materials and methods

2.1 Hardware

During the development the following materials were used:

- Atmel STK500 development board.
- A custom control board with pin head connector, potentiometer and button.
- AVR ATtiny45 MCU DIP8 chip.
- TRiO CS-1022 20 MHz oscilloscope.
- PC for development of source code and programming of the MCU.
- Datasheets and manuals, see References.

2.2 Hardware setup

2.2.1 Development board

The STK500 was wired as follows:

- ISP6PIN -> SPROG1
- PORTE.RST -> PORTB.PB5
- PORTE.XT1 -> PORTB.PB3
- The custom control board was attached to the STK500 Vtarg, GND, PORTB.PB2 (button), PORTB.PB4 (potentiometer).
- The oscilloscope probe tip and ground clip was connected to the STK500 PORTB.PB1 (OCR1A) and GND, respectively.
- The AVR ATtiny45 was attached to the STK500 socket SCKT3400D1 with pin 1 in the correct orientation.

- The STK500 RS-232 CTRL port was connected to a laptop PC through a RS-232 <-> USB converter, appearing as /dev/ttyUSB0 on the GNU/Linux system.
- The STK500 was set to run at 3.3V by using avrdude terminal option for vtarg.

```
#$ avrdude -pt45 -cstk500v2 -P/dev/ttyUSB0 -t
...
# vtarg 3.3
# q
#$
```

2.2.2 MCU Fuses

The fuses of the AVR ATtiny45 was set to their factory defaults by using avrdude, giving a $\mathrm{clk}_{I/O}$ of 1 MHz:

Fuse	Factory default value
lfuse	0x64
hfuse	0xdf
efuse	0xff

```
#$ avrdude -pt45 -cstk500v2 -P/dev/ttyUSB0 -U lfuse:w:0x64:m
#$ avrdude -pt45 -cstk500v2 -P/dev/ttyUSB0 -U hfuse:w:0xdf:m
#$ avrdude -pt45 -cstk500v2 -P/dev/ttyUSB0 -U efuse:w:0xff:m
```

2.3 Software tools

These are the software tools that were used during development:

- GNU/Linux was used on a PC development system.
- GNU AVR toolchain 4.3.5 was used for compilation of the C source code.
- GNU make 3.81 was used for the makefile system.
- avrdude 5.10 was used for programming the MCU.
- geany 0.20 was used to write the C source code.
- lyx 1.6.7 was used to write the documentation.

A GNU makefile system was created and C source code was written. Debugging and adaption of the custom contol board was done simultaneously, each function added and tested separately.

2.4 Firmware source code

The F3G system roughly consist of these modules:

- Initialization
- Main loop
- Timer0 module
- Timer1 module
- ADC module
- GPIO module

A brief explanation of each module follows:

2.4.1 Initialization

Initialization of the system is done by these steps:

1. Turn off power to unused peripherals in the MCU.
2. Initialization of ADC, GPIOs, timers and PLL.
3. Enable global interrupts.

2.4.2 Main loop

1. Main loop enter idle sleep mode.
2. When a timer0 or pin change interrupt occur the MCU is woken from sleep and an interrupt service routine (ISR) is executed.
3. When the ISR is done the main loop resume execution.
4. Status of the button is checked and action is taken accordingly.
5. The main loop enter idle sleep mode again.

2.4.3 ADC module

The ADC module keeps the ADC peripheral turned off when not used to save power.

1. Initialization
   (a) The ADC clock is initialized to run at 500 kHz. The ADC use the Vcc as voltage reference. No autotrigger is used.
2. ADC ISR
   (a) The ADC ISR is responsible for reading the 10-bit ADC value and setting the timer1 frequency.

2.4.4 GPIO module

The GPIO module is responsible for operating GPIOs.

The GPIO module uses the pin change interrupt (PCINT) to detect if a button have been pressed. PCINT0 ISR sets a flag whenever the button is pressed or released which others can use by the *isButtonPressed()* function.

2.4.5 Timer0 module

Timer0 is the overall system timer that control the firmware. It consists roughly of these parts:

1. Initialization of timer0.
   (a) The timer0 waveform is set to phase corrected pulse width modulation (PWM) to get as slow overflow (OVF) interrupt frequency as possible from timer0.
2. Timer0 OVF ISR:
   (a) Start a ADC conversion.
   (b) Toggle a LED to indicate that the system is alive.
   (c) Update the global systick variable for time keeping.
   (d) Run the jitterReduction() function to reduce jitter from ADC noice.

2.4.6 Timer1 module

Timer1 handle the PLL PWM signal and output to OCR1A. It doesnt generate any interrupts, as that would increase time out of sleep mode.

When the button is pressed the timer1 changes its base frequency prescaler value in a fixed cycle.

$$Frequency_{base} = \frac{PCK}{2^N}$$

where PCK is the PLL frequency, 64 MHz, and N={0..14}

2.5 Power considerations

The ADC peripheral make the MCU consume 5% more current in active mode and 25% more current in idle sleep mode. Therefore it is important to turn the ADC peripheral off when it is not used. The drawback is that turning ADC on and doing a single ADC conversion takes 25 ADC clock cycles instead of 13 ADC clock cycles as from an already running ADC. This extra time adds very litte to the overall consumption, given the ADC is turned off during idle sleep mode.

2.6 Compilation and programming

Compiling the source code is done by invoking make in the f3g directory:

```
#$ make
```

Programming the firmware to the MCU is done by:

```
#$ make program
```

3 Results

A stable square signal is generated on PORTB.PB1 (OCR1A). The signal is variable in the range 15.4 Hz - 5.3 MHz by a potentiometer and button. The normally noisy ADC potentiometer reading is stabilized by the C program without affecting the resolution of the potentiometer setting.

The firmware uses approximately 25% of the available flash space in the AVR ATtiny45 MCU.

4 Discussion

4.1 MCU

The MCU AVR ATtiny45 was choosen for its phase lock loop (PLL) capability that can generate a clock signal @ 64 MHz from the internal 8 MHz RC oscillator. And also because it has an analog to digital converter (ADC) peripheral. The 8-bit MCU make development fast and simple.

The AVR ATtiny45 can run at 5V instead of 3.3V. The choise to run on 3.3V was to make it directly compatible with other 3.3V MCU systems.

4.2 Oscilloscope

The analog 20 MHz oscilloscope was suitable for analysis of the produced square wave signal.

4.3 Development board

The STK500 development board was used to speed up development instead of building a full custom board for the MCU.

4.4 Timer0

The settings of timer0 was choosen to have the slowest possible overflow interrupt frequency in order to keep the MCU in idle sleep mode as much as possible. This is important since it is the timer0 overflow interrupt that wake and start the ADC to read the potentiometer.

This was realised by a phase corrected PWM waveform and highest possible prescaler, $clk_{I/O}/1024$. This gives a timer0 overflow interrupt frequency of:

$$F_{timer0_{ovf}} = \frac{clk_{I/O}}{Prescale*2*count_{ovf}}$$

Where the *2 factor is due to the phase corrected PWM counting both up and down.

Timer0 have a 8-bit counter giving a $count_{ovf} = 256$. With the I/O clock at 1 MHz and prescale factor of 1024 the interrupt frequency is:

$$F_{timer0_{ovf}} = \frac{1MHz}{1024 * 2 * 256} \approx 1.9Hz$$

To save even more power the potentiometer could be read only after N successive timer0 overflow interrupts. However, this will have a negative influence on the responsiveness. To take this to the extreme the potentiometer could be read only by pressing a button simultaneously. Since these options would not be user friendly they were discarded.

4.5 Timer1

Timer1 runs from the PLL @ 64 MHz and generate the square wave output on OCR1A. It is important that no interrupts due to timer1 occur, since the interrupts would probably not be served fast enough by the much slower CPU. This would keep the MCU off from idle sleep mode and consume more power.

4.6 ADC

To increase precision in the ADC measurement the ADC noice reduction sleep mode could have been used. However, this sleep mode stop all timers. Stopping timer1 is not desired, since timer1 produces the square wave output on OCR1A.

No autotrigger of the ADC is used, eventhough timer0 OVF autotrigger could be used and resulting in less code execution and more time in idle sleep mode.

The ADC clock is initialized to run at 500 kHz. This is outside the recommended 50 kHz - 200 kHz range for accurate ADC measurement. But as the ADC noice make precision poor anyway this higher frequency was choosen to allow more time in idle sleep mode. 500 kHz is still below 1 MHz, which is the maximum recommended sampling rate.

The ADC ISR is responsible for setting the timer1 frequency. This is done so the timer1 frequency will be given the current ADC value, instead of the old value, which would be the case if timer0 ISR or main loop is responsible for the frequency setting while the ADC conversion is still running.

4.7 Problems

4.8 Hardware

A better potentiometer and better custom control board would reduce ADC noice and potentially increase the range and stability of the produced square wave signal.

When connecting the button a too large resistor was tried at first, resulting in the signal not getting low enough. Changing to a lower resistance made the button work better.

The button have alot of contact switch bouncing, which have to be handled by software.

4.9 Software

There are a few limitations to the code:

- The ADC readings are very noicy, making the variable signal jittering. A jitter reduction functionality had to be added. This jitter reduction does however not produce a stable square wave during the time the potentiometer is twisted. Only after a short period of time after the potentiometer is set to a value the jitter reduction will come into effect. This is intentional to allow as much fine tuning of the potentiometer setting as possible. But makes the system somewhat flakey.
- Some hardcoded values. For instance the setting of the prescaler, selectable by clicking the button, is not remembered from each power on/off cycle of the MCU.
- To speed up development a copy/paste bug could have been avoided.

5 Conclusion

A variable stable square wave signal generator with a signal range of 15.4 Hz - 5.3 MHz have been constructed. The signal generator consist of a AVR ATtiny45, a potentiometer and a button.

6 References

1. 8-bit AVR Microcontroller with 2/4/8KB In-System Programmable Flash, ATtiny45, Rev. 2586N-AVR-04/11.
2. AVR STK500 User guide, Rev. 1925C-AVR-3/03.
3. AVR On-line help [http://support.atmel.no/knowledgebase/avrstudiohelp/]
4. AVR STK500 User guide [http://support.atmel.no/knowledgebase/avrstudiohelp/mergedProj
5. On-line fuse calculator [http://www.engbedded.com/fusecalc]

7 Future development

Here are suggestions for future development of the F3G:

- Put some of the now hardcoded setting values into EEPROM, to make the MCU remember last user setting. Make the setting values use a circular buffer to reduce EEPROM wear.
- Make a ADC value averaging funtionality, to reduce noise and jitter of the square wave signal.
- Build hardware low-pass filter for the potentiometer.
- Increase the signal range read from the potentiometer.
- Add more buttons and potentiometers and add various functionality to these. E.g. a second potentiometer control the duty cycle of the square wave.
- Enhance the hardware. Make a nice case. Lable the connections.
- Bootloader for easy firmware upgrade.
- TWI communication with LCD display units, other MCUs etc.
- Add watch dog timer to reset the MCU if something goes wrong.
- Investigate running the MCU on 128 kHz WDT clock.
- Investigate running the MCU on 8 MHz system clock.
- Autotrigger ADC on timer0 OVF.
- Better software debounce of the button.
- Run the F3G on solar cell and/or battery.
- Optimize the code in assembler.
- Structure the code.