Homework 2 Report  
 (1Hz to 50MHz range signal generation and sampling)

Submitted by: Kripa Varma  
Date : 5/11/2015

# Summary

Files : main.c, makefile, Homework2\_Report.docx  
 Implemented the procedures for gpio port0.pin8 toggling and sampling:

1. Continuous generation of 50MHz ( a 3\* half cycle interleaved every 20 cycles due to the branch instruction)
2. Continuous generation of 50MHz ( a 1.5\* half cycle interleaved every 20 cycles due to the branch instruction)
3. Continuous generation of 16.667MHz signal with 50-50 duty cycle.
4. Continuous generation of user input freq in the range 0.5 MHz to 1 Hz signal with 50-50 duty cycle.
5. Sampling at 100MHz
6. Sampling at 50MHz (“”)
7. Sampling at user input freq in the range 0.5 MHz to 1 Hz signal.

All functions tested and results of few sample input frequencies are recorded in this report.

# Learned

1. Details of pipelining – Stalling effects of **Load,Branch**.
2. Using **inline functions** in arm
3. Adding **inline assembly** safely as follows:

First: Push all registers to be used in assembly section

Next: Add the assembly here ..

Last : Pop the registers that were push in the beginning

1. **Branch Target Forwarding :** Feature of ARM, that allows an Unconditional Branch instruction Execution cycle to be pipelined with InstructionFetch from the target address.

A simple unconditional branch like:

B offset

can have the target program counter calculated in the decode stage and start the program fetch early.

1. str - one cycle, no stall , due to the use of **Write Buffer/Store Buffer**.

STR Rx,[Ry,#imm] is always one cycle. This is because the address generation is performed

in the initial cycle, and the data store is performed at the same time as the next instruction

is executing. If the store is to the store buffer, and the store buffer is full or not enabled,

the next instruction is delayed until the store can complete.

1. ldr - takes 2 cycles. And Data Phase can be pipelined with Address Phase of next load/store, unless next instruction uses address from Data Phase of ldr.
2. Use of **SysTickTimer** to implement delay:

* wait\_us() has a +/-0.5us error and was yielding inconsistent sampling.
* SysTickTimer has a Reload register, when loaded with value 99, it will count down to 0 in 100 cpu cycles = 100 \* 10ns = 1us
* To account for the systick register loading instructions, value 91 is used to get a delay of 1 us.

# Procedures

1. **toggle\_gpio\_50MHz()**

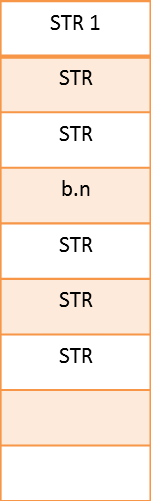
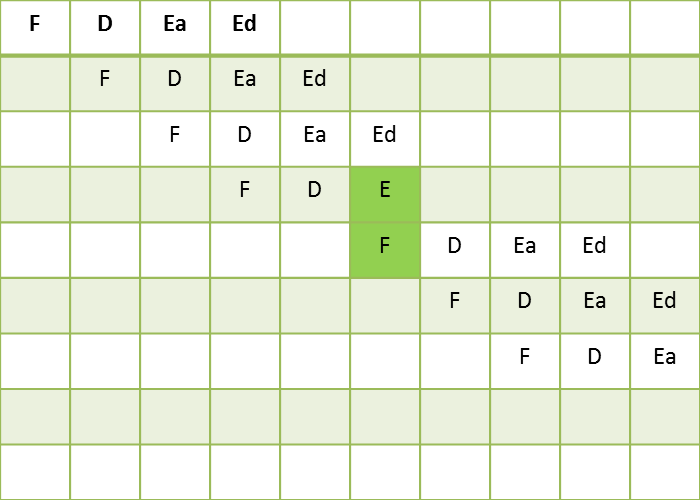
50 MHz signal generation: ( time period = 20ns) = At 96 MHz this is Max possible toggling speed.

Toggles at 50MHz for 20 cycles and then there is a 3 times long half cycle

Introduced by branch delay. Also, branch target forwarding allows Fetch of Branch target address to be pipelined with Execution of Branch.

|  |
| --- |
| while (1)  {  \*FIOSET2 = mask;  1aa: 700b strb r3, [r1, #0]  \*FIOCLR2 = mask;  1ac: 7013 strb r3, [r2, #0]  \*FIOSET2 = mask;  1ae: 700b strb r3, [r1, #0]  \*FIOCLR2 = mask;  1b0: 7013 strb r3, [r2, #0] .  .  .  } |

30 ns



Branch target forwarding

10 ns

1. **toggle\_gpio\_25MHz()**

25 MHz signal generation:

By repeating the below code 10 times within one iteration of while loop,

generated the pattern **ten** **40ns cycles** that ends with a **30ns gap** introduced by branch instruction.

Thus generating 25MHz with a 30ns gap every ten cycles.

|  |
| --- |
| while (1)  {  // CYCLE 1  \*FIOSET2 = mask;  216: 7013 strb r3, [r2, #0]  \*FIOSET2 = mask;  218: 7013 strb r3, [r2, #0]  \*FIOCLR2 = mask;  21a: 700b strb r3, [r1, #0]  \*FIOCLR2 = mask;  21c: 700b strb r3, [r1, #0]  .  .  .  } |

1. **toggle\_gpio\_16MHz()-** At 96MHz this is Max possible freq. with 50% duty cycle

16.667 MHz signal generation:

Using the below GPIO set/clear loop,

GPIO is set for 30 ns, And clear for 10ns + 20ns branch delay

Total 60ns time period = 16.6667 MHz frequency

|  |
| --- |
| while (1)  { // set for 3 cycles  // clr for one cycles , followed by 2 cycles branch delay  \*FIOSET2 = mask;  282: 7013 strb r3, [r2, #0]  \*FIOSET2 = mask;  284: 7013 strb r3, [r2, #0]  \*FIOSET2 = mask;  286: 7013 strb r3, [r2, #0]  \*FIOCLR2 = mask;  288: 700b strb r3, [r1, #0]  } |

1. **toggle\_gpio\_delay()**

Arbitrary Freq. signal generation (0.5 MHz to 1Hz)

wait\_us() function of mbed library was used to add delay between consecutive write to GPIO pin. The delay in us between toggling was computed at run time from user input freq using the simple formula:

delay\_us\_f = 1000000/(freq\*2);

delay\_us\_i = (unsigned int) delay\_us\_f;

1. **sample\_gpio\_100MHz(**), this is Max possible sampling freq, one sample per 10ns cpu clock cycle.

100 MHz sampling – using 10 register variables

|  |
| --- |
| 33a: 4918 ldr r1, [pc, #96] ; (39c <main+0x1b4>)  33c: f203 30ff addw r0, r3, #1023 ; 0x3ff  340: 9005 str r0, [sp, #20]  reg1 = LPC\_GPIO0->FIOPIN0;  342: f891 9014 ldrb.w r9, [r1, #20]  reg2 = LPC\_GPIO0->FIOPIN0;  346: f891 8014 ldrb.w r8, [r1, #20]  reg3 = LPC\_GPIO0->FIOPIN0;  34a: f891 c014 ldrb.w ip, [r1, #20]  reg4 = LPC\_GPIO0->FIOPIN0;  34e: f891 e014 ldrb.w lr, [r1, #20]  reg5 = LPC\_GPIO0->FIOPIN0;  352: 7d0f ldrb r7, [r1, #20]  reg6 = LPC\_GPIO0->FIOPIN0;  354: 7d0e ldrb r6, [r1, #20]  reg7 = LPC\_GPIO0->FIOPIN0;  356: 7d0d ldrb r5, [r1, #20]  reg8 = LPC\_GPIO0->FIOPIN0;  358: 7d0c ldrb r4, [r1, #20]  reg9 = LPC\_GPIO0->FIOPIN0;  35a: 7d08 ldrb r0, [r1, #20] |

ldrb takes total 4 cycles (**F,D,Ea,Ed)** – or 2 exec cycles.

However, Nth load’s Ed can be pipelined with N+1th Ea ( since address of N+1 is not based on Data Phase of N)

A sample is taken every 10ns (one clock cycle) and hence 100MHz sampling is achieved.

1. **sam****ple\_gpio\_ 25MHz**()

25MHz sampling – By interleaving buffer stores in between load of GPIO to register. 32 samples are taken back to back and stored to buffer.

|  |
| --- |
| \*SYS\_TICK\_LOAD = 500 ;//- overhead;  \*SYS\_TICK\_CTRL = 0x5;  \*SYS\_TICK\_LOAD = 0;  406: 781a ldrb r2, [r3, #0]  buf[0] = reg1;  408: 7002 strb r2, [r0, #0]  reg1 = \*FIOPIN1;  40a: 781a ldrb r2, [r3, #0]  buf[1] = reg1;  40c: 7042 strb r2, [r0, #1]  reg1 = \*FIOPIN1;  40e: 781a ldrb r2, [r3, #0]  buf[2] = reg1;  410: 7082 strb r2, [r0, #2]  reg1 = \*FIOPIN1;  412: 781a ldrb r2, [r3, #0]  buf[3] = reg1;  414: 70c2 strb r2, [r0, #3]  -  -  -  printf ("SYS\_TICK\_VAL = %u\r\n", \*SYS\_TICK\_VAL); |

Initially this function was thought to be sampling at 50MHz, after reading Load-Store timing section of arm technical reference. And the function sampled 16MHz accurately too. However, when sampling 4MHz the computation was way off and showed 8MHz, thus indicating that the function was sampling twice as slow as it was thought to be. So sys\_tick\_timer was used to measure the time taken for 32 load-store pairs. It showed that 32 ldr-ltr pairs took 120 cycles, as opposed to the expected execution time of 64 cycles. So the function was thus sampling at only 25MHz.

Now the perplexing thing was how still the freq. estimation of 16MHz input was correct, if sampling was only at 25 MHz while computation was based on 50MHz. So the table below was made to explain this effect.As s

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 16 MHz signal |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1  100MHz | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1  50MHz |  | 1 |  | 0 |  | 1 |  | 1 |  | 0 |  | 1 |  | 1 |  | 0 |  | 1 |  | 1 |  | 0 |  | 1 |  | 1 |  | 0 |  | 1 |  | 1 |  | 0 |  | 1 |  | 1 |  | 0 |  |
| 1  25MHz |  |  |  | 0 |  |  |  | 1 |  |  |  | 1 |  |  |  | 0 |  |  |  | 1 |  |  |  | 1 |  |  |  | 0 |  |  |  | 1 |  |  |  | 1 |  |  |  | 0 |  |

.

As seen from the table above, when 16MHz is sampled at 50MHz or at 25 MHz the samples collected would be the same. (011011011 pattern)

50MHz : 110110110110…..   
25MHz : 10110110110….

1. sample\_gpio\_delay()

Arbitrary Freq. signal generation (1Hz to 500,000 Hz )

SysTickTimer is used to insert delay in multiples of microseconds. The delay in us between consecutive samples was computed at run time from user entered freq using the formula:

delay\_us\_f = 1000000/(freq);

delay\_us\_i = (unsigned int) delay\_us\_f;

1. compute\_input\_freq ()

Compute frequency from stored samples.

Sets of 10/32 samples are collected into a 4098 size buffer for 100/50MHz respectively.

For 1Hz to 500,000 range, 4098 samples are collected 100 times, and **calculated delay is inserted using SysTickTimer**.

**Leading bits are discarded**, and samples are counted from the first toggle onwards.

The max even number of toggles recorded are considered towards freq. estimation and **trailing bits are discarded.**

A simple **average** is performed, to compute frequency based on **sampling time period , number of valid samples and number of cycles recorded .**

Freq in Hz= (sampling time period in ns \* number of valid samples \*1000000000)/number of cycles

# Results

1. Output signal **measurements using logic analyzer** (borrowed during class room break)

25 MHz (40ns) 🡪measured as 40ns with 50-50 duty cycle (loop unrolling implementation)

16 MHz (60ns) 🡪 measured as 55ns to 65ns ( branch instruction every signal cycle )

500,000kHz(2uS) 🡪measured as 2.1us ( delay implemented using sys\_tick\_timer

1. Frequency estimation using mbed2

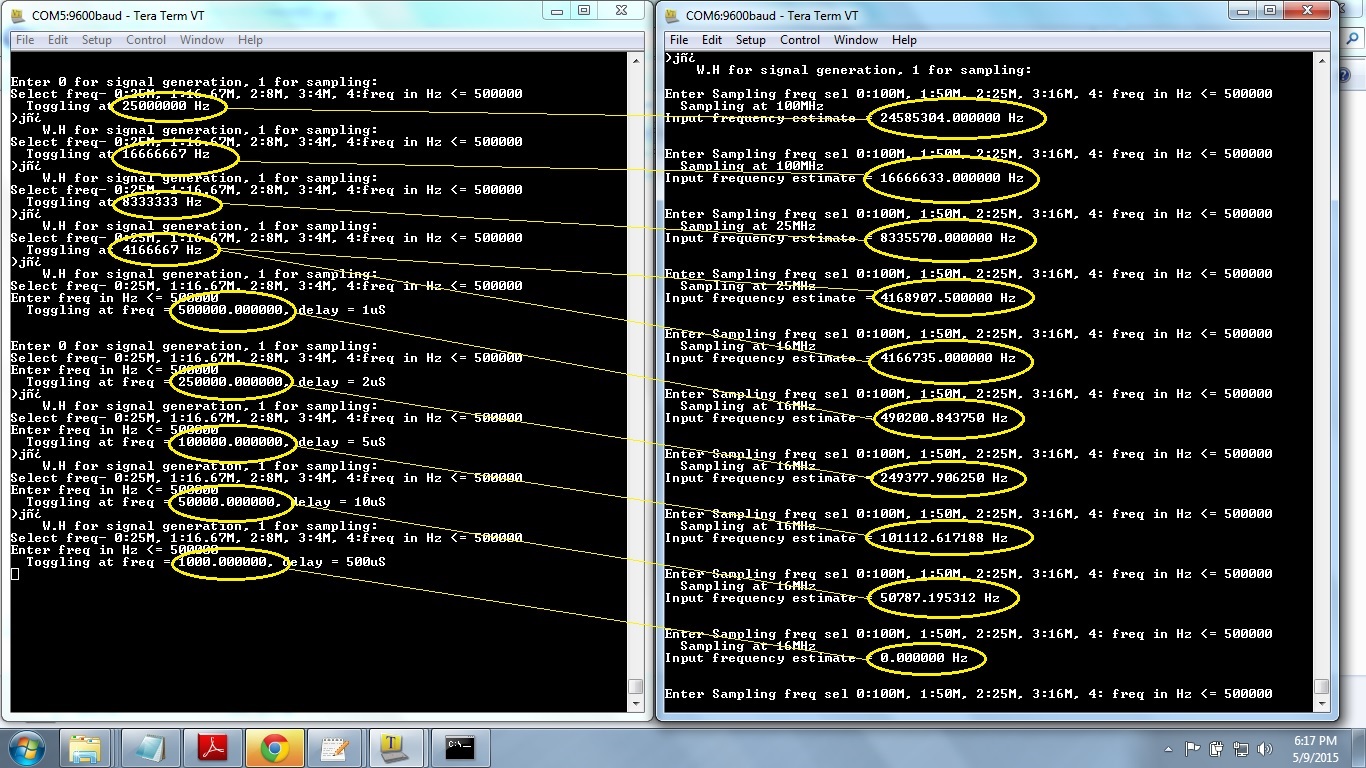
(25MHz could be estimated with <=2% Err)

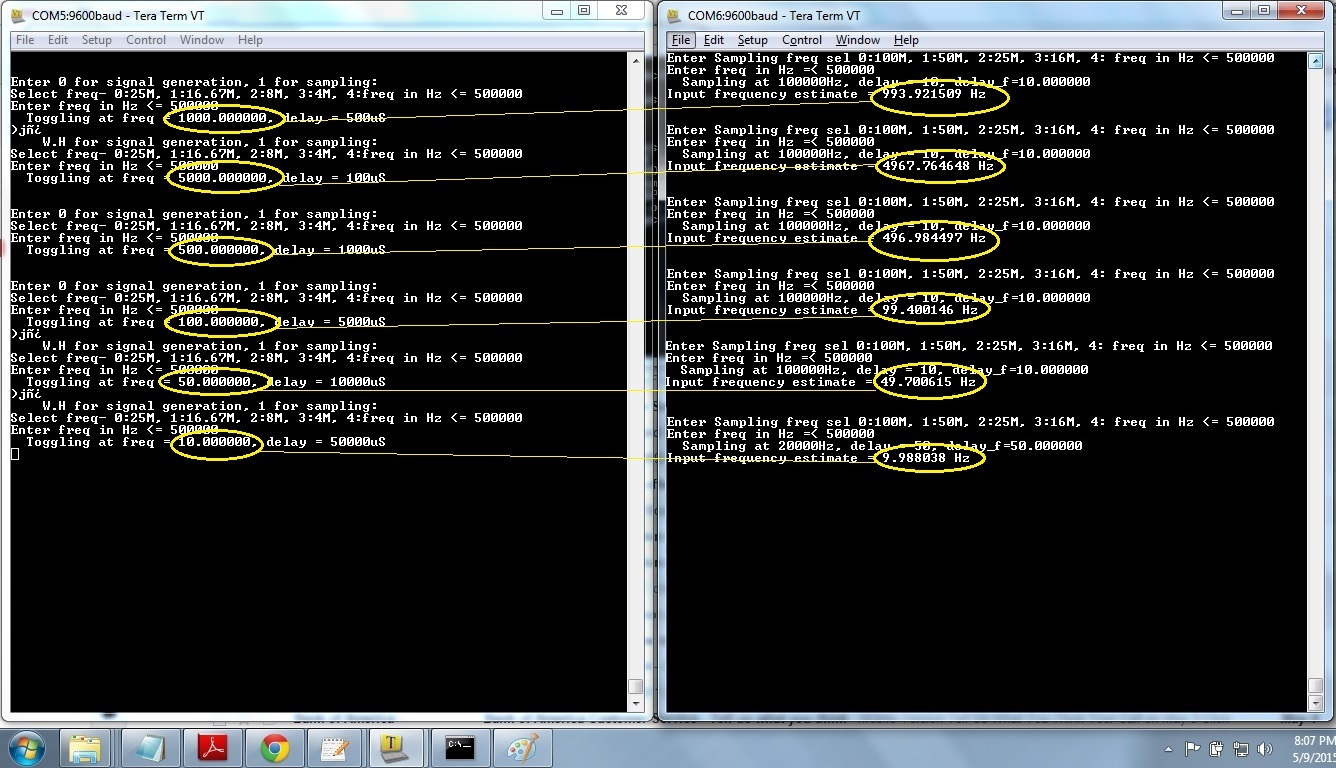
(10Hz to 16MHz could be estimated with < 0.5% Err)

The screen shots below, left side shows the freq. at which mbed1 is toggling gpio0.pin8. Right side shows corresponding freq. estimation done at mbed2.

First screen shot shows sampling at fast rates – 100MHz, 25MHz, 16MHz

Second screenshot shows results of sampling at rates <= 500,000Hz





# Issues

1. The loop for **16MHz sampling** ( one sample per 60ns )  was **sometimes taking 6 cycles and sometimes 7** ( as measured with the help of **sys\_tick\_timer**), depending on whether a debug print before the loop was added/removed.

Then I found out when it takes **7 cycles**, the loop's start address was **not word aligned**, but just 2bytes aligned.

1. Inconsistency in **100MHz sampling.** - sometimes it **measured 25MHz correctly, and sometimes as 40MHz**

Again it was found from objdump that, when the measurement was off,  the 32 bit   "ldr" instructions of 100MHz sampling was not word aligned.

This was resolved by using **gcc optimize options**: falign-loop=4, falign-function=4 and also \_\_attribute\_\_(noinline) in conjunction with –O3 level, to get consistent results over multiple compiles. ( -O1 would have solved this too , however, the inlining and other tight optimizations were helping the delay function run much faster, and hence I chose ot keep –O3 )

1. **50 MHz sampling** was giving some inconsistent results. It was correctly estimating 16MHz, but 4MHz was being estimated as 8MHz. Using **systick timer to measure** the 50MHz sampling function, showed that **actual sampling rate was only 25 MHz**, as explained [here](#Sampling25MHz).

# Further work

1. Study more about why/how 50 MHz sampling was not achieved.
2. Obtain a logic analyzer and study the output frequency to:
   1. Observe whether 16MHz is really generated at 50-50 duty cycle and study the effect of branch delay in conditional as well as unconditional branching.
   2. Fine tune the delay function

# Hours

2-3 Hrs : Searching literature for pipelining of instructions, SysTick timer.  
25 : Implementing functions to toggle GPIO at different frequencies, sample at different rates and compute the frequency based on stored samples.

4-5 Hrs : Preparing the document

# References

1. Cortex™-M3 Technical Reference Manual Revision r2p0
2. LPC17xx user manual