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

Submitted by: Kripa Varma  
Date : 4/20/2015

# Summary

Files : main.c, makefile, Homework2\_Report.docx  
 Implemented the procedures for gpio 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.

Only 1 -25 Hz toggling could be tested so far, using on board LED – 1 -25 Hz , just by observing changes in speed of blinking.

Teste sampling functions were tested using pre-initialized buffer values.

Todo :

Test procedures above with another mbed board.

# 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.

# Background info:

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 Data Phase can be **pipelined** with Address Phase of next load/store, unless 2nd inst uses address from Data Phase of 1st Inst.
2. **Branch Target Forwarding** : Especially useful in Unconditional Branch

a simple unconditional branch like:

B offset

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

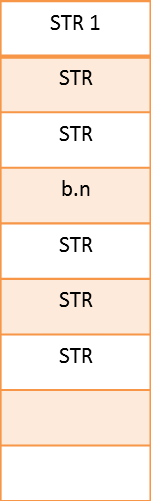
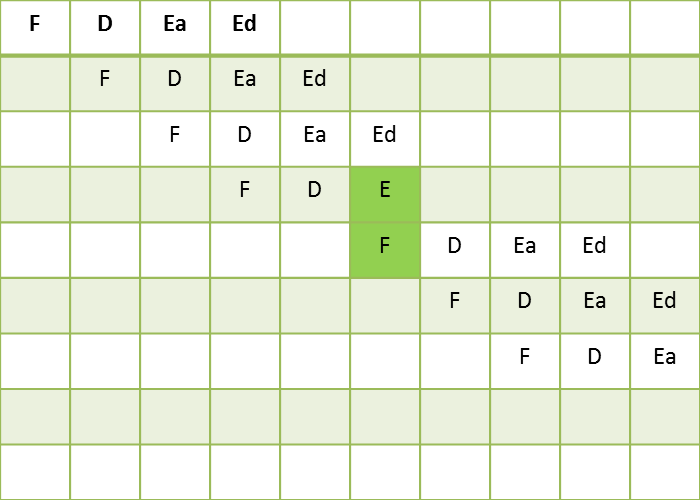
# Procedures:

1. 50 MHz signal generation: ( time period = 20ns)

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.

30 ns



Branch target forwarding

1. 25 MHz signal generation

|  |
| --- |
| 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]  .  .  .  } |

By repeating the above 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.

1. 16.667 MHz signal generation

|  |
| --- |
| while (1)  {  \*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]  } |

The above GPIO set clear loop,

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

Total 60ns time period = 16.6667 MHz frequency.

1. 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 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. 100 MHz sampling – using 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.

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. 50 MHz sampling – By interleaving buffer stores in between load of GPIO to register.

|  |
| --- |
| 246: 7d08 ldrb r0, [r1, #20]  248: 3308 adds r3, #8  24a: f803 0c08 strb.w r0, [r3, #-8]  reg2 = LPC\_GPIO0->FIOPIN0;  24e: 7d08 ldrb r0, [r1, #20]  buf[1] = reg2;  250: f803 0c07 strb.w r0, [r3, #-7]  reg3 = LPC\_GPIO0->FIOPIN0;  254: 7d08 ldrb r0, [r1, #20]  buf[2] = reg3;  256: f803 0c06 strb.w r0, [r3, #-6]  reg4 = LPC\_GPIO0->FIOPIN0;  25a: 7d08 ldrb r0, [r1, #20] |

1. Compute frequency from stored samples and sampling freq.

# Generic syntax of instructions used in the program

# Hours:

4 hours : GPIO related reading, coding, makefile.  
10-12 hrs : Study instructions encoding and documenting the decoding of machine code.

# References:

1. UM10360 LPC176x/5x User manual ; Section 9.5 – GPIO Registers
2. ARM Architecture Reference Manual ; Section A4, A5