Stephen Hall and Razik Ahmed
TA - Harshad Desai

Objectives) Develop software debugging techniques, performance debugging techniques (dynamic or real time), and profiling techniques (detection and visualization of program activity). Pass data using a FIFO queue, learn how to use the logic analyzer and observe critical sections.

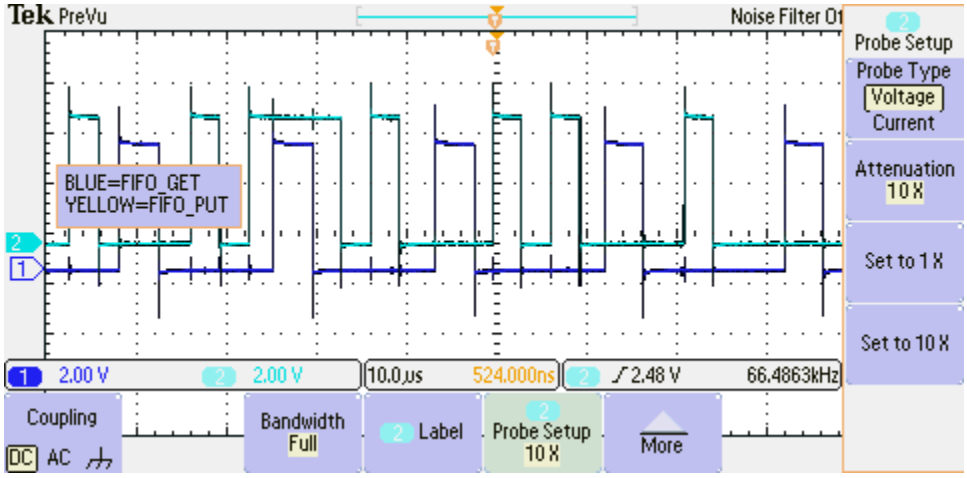
Prep Part 3)

```
Function: Fifo_Get
 0000 3b          [2]     PSHD
  63:    if(PutI == GetI ){
 0001 fc0000      [3]     LDD   PutI
 0004 bc0000      [3]     CPD   GetI
 0007 2603        [3/1]   BNE   *+5 ;abs = 000c
  64:      return(FIFOFAIL); // Empty if PutI=GetI
 0009 c7          [1]     CLRB
 000a 2017        [3]     BRA   *+25 ;abs = 0023
  65:    }
  66:    *datapt = Fifo[GetI&(FIFOSIZE-1)];
 000c f60000      [3]     LDAB  GetI:1
 000f c41f        [1]     ANDB  #31
 0011 ce0000      [2]     LDX   #Fifo
 0014 a6e5        [3]     LDAA  B,X
 0016 ee80        [3]     LDX   0,SP
 0018 6a00        [2]     STAA  0,X
  67:    GetI++;  // Success, update
 001a fe0000      [3]     LDX   GetI
 001d 08          [1]     INX
 001e 7e0000      [3]     STX   GetI
  68:    return(FIFOSUCCESS);
 0021 c601        [1]     LDAB  #1
 0023 87          [1]     CLRA
  69:  }
 0024 30          [3]     PULX
 0025 3d          [5]     RTS
  91:      while(Fifo_Get(&ForeData)==FIFOFAIL){
 0022 cc0000      [2]     LDD   #ForeData
 0025 160000      [4]     JSR   Fifo_Get
 0028 0454f7      [3]     TBEQ  D,*-6 ;abs = 0022
```

50 cycles per call
50/24*10^6 = 2.08333333 us per call

Part B) The lab 2 starter project had two threads, a foreground and a background. The foreground would set PT1 when it called Fifo_Get() and clear it when it returned. The interrupt would set PT0 when it started and clear it when it returned. The interrupt would fire at an incrementing amount of cycles (23) so that it was not fundamentally periodic. Because of this, the Fifo_Get() would sometimes get interrupted, and sometimes not. The screenshot below shows the reactions of PT0 and PT1. Since the

oscilloscope inverted the colors, the green waveform is Fifo_Get and the blue waveform is Fifo_Put (the interrupt).



Part C) The first version takes 48 cycles to execute. The second one takes 10336 cycles. The third one takes 85 cycles.

Part D) Using the scope, we get 2.13 us. The first Fifo_Get function is 48 cycles/24000000 cycles per second = 2 us. The measurement error for setting and clearing PT0 is 160 ns = .16 us. So the advantages of using the TCNT method are that we can store it in memory and it is very accurate, but storing TCNT takes some execution time. The advantage of the scope is that it introduces no unnecessary execution time. The disadvantage of the scope method is that it introduces error in the total measurement. However if the error is compensated then the scope method gives 1.97 us which is only a 1.5% difference between methods.

Part E)

Index	timeBuf	placeBuf
0	0	0
1	251	1
2	305	4
3	0	0
4	530	4
5	775	4
6	0	0
7	1189	3
8	0	0
9	1235	1
10	1337	4
11	1519	2
12	1801	1
13	0	0

14	1899	2
15	1988	4
16	2181	1
17	2279	2
18	2352	4
19	2561	1
20	2659	2
21	2735	4
22	2941	1
23	3039	2
24	3149	4
25	3321	1
26	3419	2
27	3572	4
28	3799	2
29	3895	1
30	4025	4
31	4179	2
32	4275	1
33	4373	2
34	4501	4
35	4655	1
36	4753	2
37	4849	3
38	4895	1
39	4999	4
40	5180	2
41	5276	1
42	5374	2
43	5521	1
44	0	0
45	5720	4
46	5944	4
47	6126	2
48	6190	4
49	6459	1
50	0	0
51	6750	4
52	6974	1
53	7067	4
54	7258	2

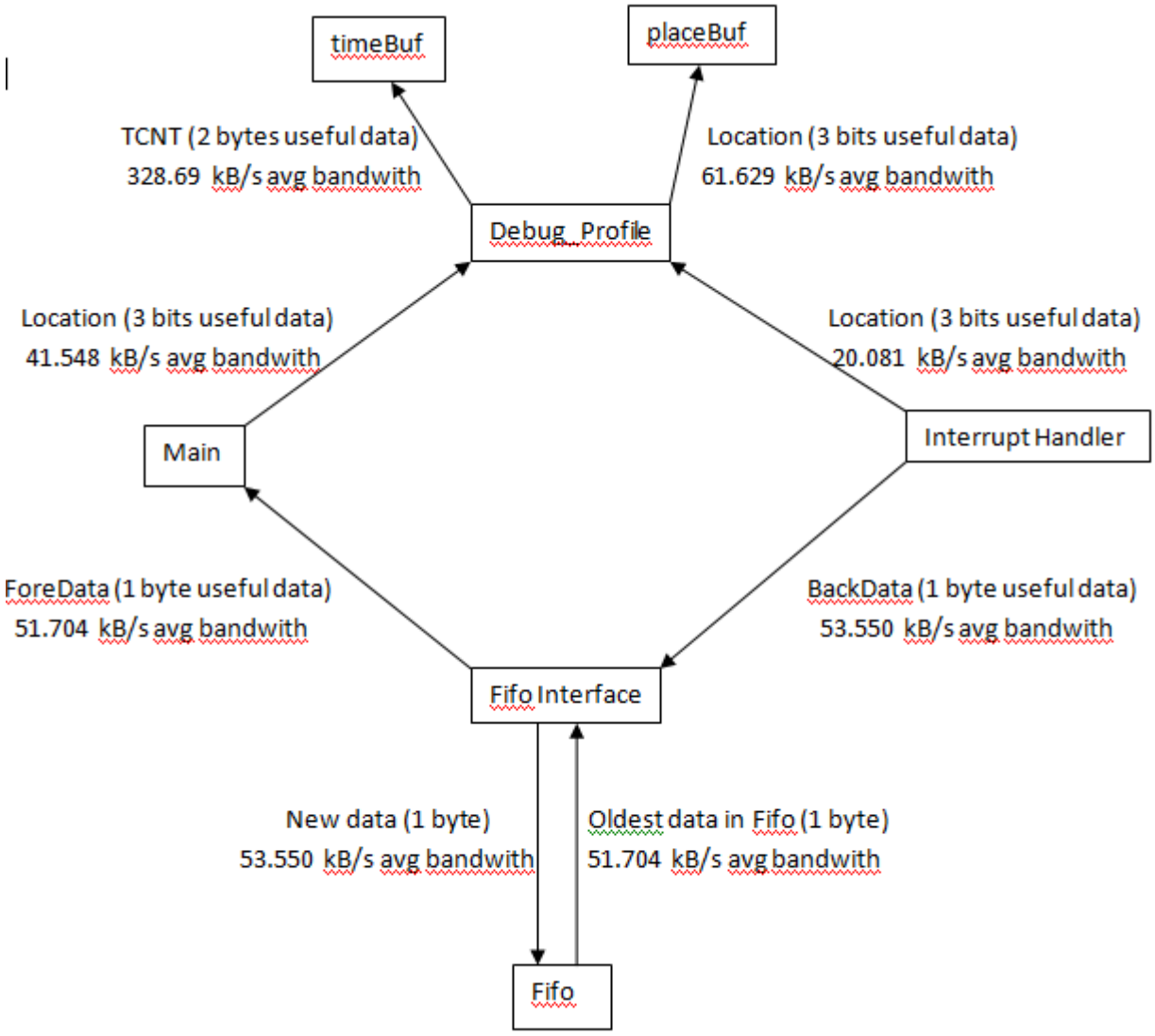
55	7405	1
56	0	0
57	7638	2
58	7766	4
59	7920	1
60	8018	2
61	8148	4
62	8300	1
63	8398	2
64	8494	1
65	8555	4
66	8778	2
67	8874	1
68	8985	4
69	9158	2
70	9254	3
71	9300	1
72	9584	2
73	0	0
74	9680	1
75	9778	2
76	10060	1
77	0	0
78	10158	2
79	10254	1
80	10352	2
81	10413	4
82	10635	1
83	10733	2
84	10829	1
85	10936	4
86	11136	4
87	11299	2
88	11360	4
89	11604	4
90	11767	1
91	11874	4
92	12051	2
93	12165	4
94	12333	1
95	12482	2

96	0	0
97	12713	1
98	12820	4
99	12997	2

The execution beginning is 1,4,4,4,3,1,4,2 which is slightly wrong. There were a few errors between the first 1 and first 3 which probably had a 2 in there, but it got overwritten.

```
void main(void){
  PLL_Init();          // running at 24MHz
  DDRT |= 0x03;        // debugging outputs
  PTT &= ~0x03;
  Debug_Profile(0);    // 0 means initialization phase
  Fifo_Init();         // Initialize fifo
  OC0_Init();          // variable rate interrupt
  ForeExpected = 0;    // expected data
  for(;;){
    Debug_Profile(1); // 1 means start of foreground waiting
    PTT_PTT1 = 0;      // falling edge of PT1 means start of fore waiting
    while(Fifo_Get(&ForeData)==FIFOFAIL){
    }
    Debug_Profile(2); // 2 means foreground has new data
    PTT_PTT1 = 1;      // rising edge of PT1 means start of fore processing
    if(ForeExpected != ForeData){
      Errors++;                   // critical section found
      ForeExpected = ForeData+1; // resych to lost/bad data
    }
    else{
      ForeExpected++;  // sequence is 0,1,2,3,...,254,255,0,1,...
    }
    if((ForeData%10)==0){
      Debug_Profile(3); // 3 means foreground has 10th data
    }
  }
}
interrupt 8 void OC0Han(void){ // periodic interrupt
  Debug_Profile(4);  // 4 means background thread active
  PTT_PTT0 = 1;      // rising edge of PT0 means start of interrupt
  TFLG1 = 0x01;      // acknowledge OC0
  TC0 = TC0 +BackPeriod;   // varies from 10us to 1ms
  if(Fifo_Put(BackData)==FIFOFAIL){
    NumLost++;
  }
  BackData++; // sequence is 0,1,2,3,...,254,255,0,1,...
  if(BackPeriod > 500){
    BackPeriod = 200;
  } else{
    BackPeriod = BackPeriod+23;
  }
  NumInterrupts++;
  PTT_PTT0 = 0;       // falling edge of PT0 means end of interrupt
}
```

Data Flow Graph



Part F) Four Pins are changed during different stages of the code execution. PTT is changed to 0x01 when entering and leaving the interrupt routine. PT1 is toggled before calling Fifo_Get() and after calling it. PT2 is toggled inside of Fifo_Get() and PT3 is toggled inside of Fifo_Put(). So when PTT = 0x01, the interrupt is occurring, and when PTT = 0x09 the interrupt has called Fifo_Put(). When PTT = 0x02, the program is in the foreground, but is not calling Fifo_Put(). When PTT = 0x06, the foreground is calling Fifo_Put().

Part G) The expected value of Fifo_Get() was supposed to be 163. However, the Fifo_Put() had gotten so far ahead that it overwrote 163 with 195.

ForeData 195 unsigned char
ForeExpected 163 unsigned char
Fifo.c::Fifo <32> array[32] of unsigned char
[0] 192 unsigned char

[1]	193 unsigned char
[2]	194 unsigned char
[3]	195 unsigned char
[4]	164 unsigned char
[5]	165 unsigned char
[6]	166 unsigned char
[7]	167 unsigned char
[8]	168 unsigned char
[9]	169 unsigned char
[10]	170 unsigned char
[11]	171 unsigned char
[12]	172 unsigned char
[13]	173 unsigned char
[14]	174 unsigned char
[15]	175 unsigned char
[16]	176 unsigned char
[17]	177 unsigned char
[18]	178 unsigned char
[19]	179 unsigned char
[20]	180 unsigned char
[21]	181 unsigned char
[22]	182 unsigned char
[23]	183 unsigned char
[24]	184 unsigned char
[25]	185 unsigned char
[26]	186 unsigned char
[27]	187 unsigned char
[28]	188 unsigned char
[29]	189 unsigned char
[30]	190 unsigned char
[31]	191 unsigned char

Analysis and Discussion)

1. We did not get the same result with the three ways because each method of storing or showing the number of cycles also takes a certain amount of cycles. Sending the info to the computer, for instance, take milliseconds while the actual function call only takes 2 microseconds.
2. The execution speed will vary the most if the SCIO port is used because it may have to wait before it can send the next byte of data. This wide range will give different values for the min, max, average. Since the other functions take the same time consistently, the min, max, and average values will all be nearly the same.
3. Periodically updating an array of a profile in the debugger while simultaneously printing it out to a computer that runs at a very small baud rate.
4. Minimally intrusive is the ability for a debugging tool to take very little time compared to the overall execution time of the code it is debugging.
5. A profile gives both a time and place where the profile was called.
6. The critical section in the bad FIFO is this piece of code.

```
*datapt = *(RxGetPt);  // return by reference
RxGetPt++;             // removes data from fifo
```

```
if(RxGetPt == &RxFifo[RXFIFOSIZE]){
   RxGetPt = &RxFifo[0];  // wrap
}
return(1);
```

If the interrupt fires right after RxGetPt++ is called and before the function returns, then the data pointed to by datapt will be overwritten.