Jonathan W. Valvano April 17, 2009, 10:00 to 10:50am

(15) Question 1. A CAN system with 3 nodes has a baud rate of 50,000 bits/sec.

Part a) First think about the problem, as it is. 50,000 bps is 20μs/bit. The “does it work” issue involves the average bandwidth. Peak bandwidths can be solved with Fifo queues. Each frame is 11+32+36 = 79 bits, which is 1.58 ms per frame, each node sends one so it takes 4.74 ms to send all three. 4.74ms is less than 20 ms, so it works. Let x be the CAN baud rate in bps. It will work if

$3*79/x < 0.02$sec

$50*3*79 < x$

11850 bps $< x$

Part b) Bandwidth means data (real information) transfer rate. When selling a communication system to the public, they want to transfer information, and do not care how many overhead bits you need or how fast you can send overhead bits. (4 bytes/node*3nodes)/20ms = 600 bytes/sec

Part c) Stuff bits cause transitions in signals with long sequences of 0’s or 1’s. They allow the receivers to synchronize to the transmitter, reducing the requirement that the baud rates in each node match exactly. Stuff bits are not used for error checking. They do however prevent errors.

(20) Question 2. Consider a 256-point FFT calculated on 12-bit ADC data sampled at 10 Hz.

Part a) $f = k*f_s/n = 64*10\text{Hz}/256 = 2.5$ Hz

Part b) The phase is 45^o, or $\pi/4$ (could also have been 225^o)

Part c) The FFT output at k and 256-k are complex conjugate pairs of each other (because the input is real).

Part d) The first term of the FFT is the sum of all the input data, so the real part is 2048*256 = 524,288, and the imaginary part will be zero.

(15) Question 3. Consider the following 16-bit FIFO implementation.

```
//*********Fifo_AlmostFull**********
// check to see if FIFO is almost full
// Input:  none
// Output: true if the fifo is more than 75% full
#define FIFO75 ((3*FIFOSIZE)/2)
// 75% full in bytes
int Fifo_AlmostFull(void){
  unsigned short size;
  size = (unsigned short)PutPt – (unsigned short)GetPt;
    if(PutPt < GetPt ){
      size = size +2*FIFOSIZE;
  }
  return size>FIFO75;
}
```

(25) Question 4. There are multiple threads that need to update a shared LCD display.

```
void Display(int line, unsigned short num){
asm ldx #Free
asm clra        // new value for Free
asm minm 0,x    // atomic test and set
asm bcc  busy   //C=1, if Free went from 1 to 0
   LCD_GoTo(line,1);
   LCD_OutDec(num);
 asm busy:
}
```

(25) Question 5. Implement the following fork and join synchronization.

```
Sema4Type Done;
void master(void){
  for(;;){
    fun1();
    OS_InitSemaphore(&Done,-1);
    OS_AddThread(&slave3,50,3);
    OS_AddThread(&slave4,50,3);
    fun2();
    OS_Wait(&Done); // both done
    fun5();
  }
}
```

```
void slave3(void){
// none
  fun3();
  OS_Signal(&Done); // done
  OS_Kill();
}
```

```
void slave4(void){

  fun4();
  OS_Signal(&Done); // done
  OS_Kill();
}
```

Part 0) List the semaphore(s) needed

```
  Sema4Type Done;   // define this in shared global space
```

Part a) Give the C code labeled **`segment1A`** for the master to execute between **`fun1`** and **`fun2`**

```
    OS_InitSemaphore(&Done,-1);
    OS_AddThread(&slave3,50,3);
    OS_AddThread(&slave4,50,3);
```

Part b) Give the C code labeled **`segment1B`** for the master to execute between **`fun2`** and **`fun5`**

```
    OS_Wait(&Done); // both done
    // when both slave 3 and 4 are done, done=1
```

Part c) Give the C code labeled **`segment3A`** for the slave3 to execute before **`fun3`**

None

Part d) Give the C code labeled **`segment3B`** for the slave3 to execute after **`fun3`**

```
    OS_Signal(&Done);  // slave 3 is done
    OS_Kill();
```

Part e) Give the C code labeled **`segment4A`** for the slave4 to execute before **`fun4`**

None

Part f) Give the C code labeled **`segment4B`** for the slave3 to execute after **`fun4`**

```
    OS_Signal(&Done);  // slave 3 is done
    OS_Kill();
```