Jonathan W. Valvano March 6, 2002, 9:00am-9:50am

**(25) Question 1**. The status of the FIFO can tell if the system is I/O-bound or CPU-bound. I/O bound means the bandwidth is limited by the speed of the I/O device (a faster I/O device will improve bandwidth.) CPU bound means the bandwidth is limited by the speed of the computer/software (a faster computer or better compiler will improve bandwidth.)

Part a) In order to make them private (accessible only from within this file.)

Part b) Make it public

```
unsigned int TxFifo_Size(void){
  return(TxPutPt-TxGetPt+10)%10);
}
```

Part c)

```
void debugFIFO(void){
  Count[ TxFifo_Size()] ++;
}
```

Part d) If the system were I/O bound, then the FIFO would fill up and the software would usually have to wait for there to be room in the FIFO. The `Count[ 9]` would have a lot of entries. E.g.,

Count[0] = 1
Count[1] = 1
Count[2] = 1
Count[3] = 1
Count[4] = 1
Count[5] = 1
Count[6] = 1
Count[7] = 1
Count[8] = 100
Count[9] = 1000

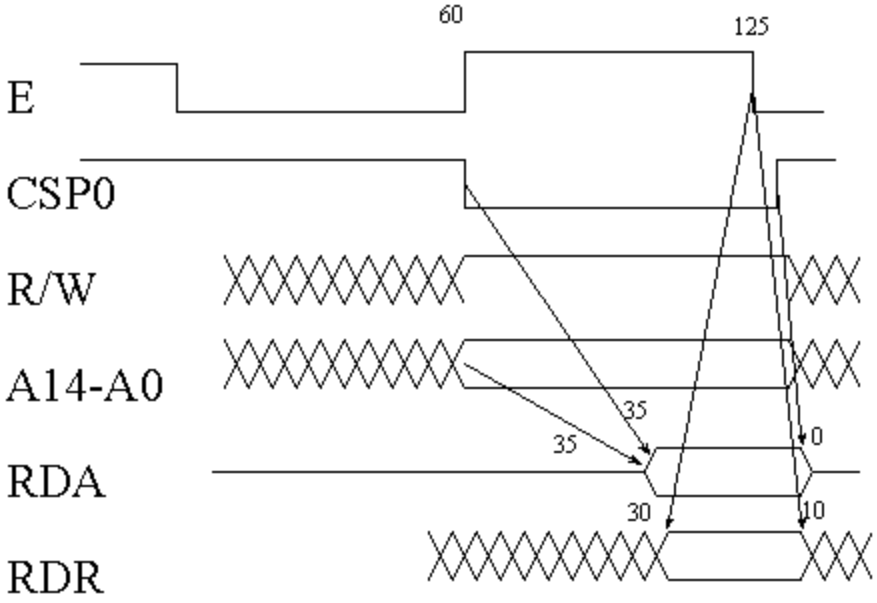
**(25) Question 2.** The goal is to find a fast-enough memory so that cycle stretching is not required.

Part a) Since OE is grounded the value of $t_{OE}$ doesn't matter.

60+ $t_{ACC} \leq$ 125 - 30 so $t_{ACC} \leq 35$

60+ $t_{CE} \leq$ 125 - 30 so $t_{CE} \leq 35$

Part b) Draw the read-cycle timing diagram for the new interface.



**(50) Question 3.** The memory for the TCB is dynamically allocated on the heap.

Part a) We will put back `ThreadId` and initialize it to 0. This function needs to be atomic

```
//******** OS_AddThread ***************
// add a foreground thread to the scheduler
// Inputs: pointer to a void/void foreground function
// Outputs: 1 if successful, 0 if this thread cannot be added
int OS_AddThread(void(*fp)(void)){  TCBPtr pt; unsigned char saveCCR;
  if(pt=malloc(sizeof(TCBType))){
    return 0;  // heap is full
  }
asm("tpa\n"                /* previous interrupt enable */
    "staa %SaveCCR\n"      /* save previous */
    "sei");                /* make atomic */
  if(RunPt){
    pt->Next = RunPt->Next; // place right after RunPt
    RunPt->Next = pt;
  }
  else{
    pt->Next = pt; // first one, linked to itself
    RunPt = pt;
  }
  pt->StackPt = &(pt->InitialCCR);
  pt->Id = ThreadId++;   // thread numbers go 0,1,2,3,...
  pt->InitialCCR = 0x40;
  pt->InitialPC = fp;
asm("ldaa %SaveCCR\n"  /* recall previous */
    "tap");            /* end critical section */
  return 1;
}
```

Part b) Similar to OS_Launch

```
//******** OS_Kill ***************
// kill this thread, launch a new thread
// Inputs: none
// Outputs: will not return
TCBPtr Killpt,PrevPt;
void OS_Kill(void){              // ***NO LOCAL VARIABLES*****
asm("  sei");                    // must be atomic
  KillPt = RunPt;                // to one to kill
  PrevPt = RunPt;                // search for previous
  while(PrevPt->Next != RunPt){  // quit when pt points to previous
    PrevPt = PrevPt->Next;
  }
  PrevPt->Next = RunPt->Next;  // unlink this thread
  RunPt = Killpt->Next;        // next one to run
asm("  ldx _RunPt\n"
    "  lds 2,x"):              // new valid stack
  free(Killpt);                // return TCB to heap
  TC3 = TCNT+TIMESLICE;
  TFLG1 = 0x08;                // Clear C3F
  PORTJ = RunPt->Id;
asm("  rti");                  // Launch next Thread
}
```