ECE/CS 5780/6780: Embedded System Design

Scott R. Little

Lecture 9: FIFOs

Administrivia

- CodeWarrior IDE compiler optimization configuration.
- No labs are scheduled next week. Please contact your TA if you would like to meet them.
**Introduction to FIFOs**

- FIFO circular queue is useful for a buffered I/O interface.
- This order-preserving data structure temporarily saves data created by a producer before being processed by a consumer.
- Decouples the producer from the consumer.
- Use statically allocated global memory, so they can be shared by threads, but must be accessed carefully.

---

**FIFO with Infinite Memory**

---

Scott R. Little (Lecture 9: FIFOs)  ECE/CS 5780/6780  3 / 20

---

Scott R. Little (Lecture 9: FIFOs)  ECE/CS 5780/6780  4 / 20
Basic Idea of a FIFO

```c
char static volatile *PutPt; // put next
char static volatile *GetPt; // get next
// call by value
int Fifo_Put(char data){
    *PutPt = data; // Put
    PutPt++; // next
    return(1); // true if success
}
// call by reference
int Fifo_Get(char *datapt){
    *datapt = *GetPt; // return by reference
    GetPt++; // next
    return(1); // true if success
}
```

Two-Pointer FIFO

![Diagram of Two-Pointer FIFO](image-url)
Initialization of a Two-Pointer FIFO

```c
#define FIFOSIZE 10 /* can hold 9 */
char static volatile *PutPt; /* Pointer to put next */
char static volatile *GetPt; /* Pointer to get next */
/* FIFO is empty if PutPt=GetPt */
/* FIFO is full if PutPt+1=GetPt (with wrap) */
char static Fifo[FIFOSIZE];
void Fifo_Init(void) {
    unsigned char SaveSP;
    asm tpa
    asm staa SaveSP
    asm sei /* make atomic, entering critical */
    PutPt=GetPt=&Fifo[0]; /* Empty when PutPt=GetPt */
    asm ldnn SaveSP
    asm tap /* end critical section */
}
```
Put for a Two-Pointer FIFO

```c
int Fifo_Put(char data) { char *Ppt; /* Temp put pointer */
    unsigned char SaveSP;
    asm tpa
    asm staa SaveSP
    asm sei /* make atomic, entering critical */
    Ppt=PutPt; /* Copy of put pointer */
    *(Ppt++)=data; /* Try to put data into fifo */
    if (Ppt == &Fifo[FIFOSIZE]) Ppt = &Fifo[0]; /* Wrap */
    if (Ppt == GetPt) {
        asm ldaa SaveSP
        asm tap /* end critical section */
        return(0); /* Failed, fifo was full */
    } else {
        PutPt=Ppt;
        asm ldaa SaveSP
        asm tap /* end critical section */
        return(1); } /* Successful */
}
```

Put for a Two-Pointer FIFO Example

```c
int Fifo_Put(char data) {
    char *Ppt;
    unsigned char SaveSP;
    asm tpa
    asm staa SaveSP
    asm sei
    Ppt=PutPt;
    *(Ppt++)=data;
    if (Ppt == &Fifo[FIFOSIZE])
        Ppt = &Fifo[0];
    if (Ppt == GetPt) {
        asm ldaa SaveSP
        asm tap
        return(0);
    } else {
        PutPt=Ppt;
        asm ldaa SaveSP
        asm tap
        return(1);
    }
}
```

```text
data = 0x04

<table>
<thead>
<tr>
<th></th>
<th>0xXX</th>
<th>0xXX</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetPt</td>
<td>0x01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x02</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x03</td>
<td></td>
</tr>
<tr>
<td>PutPt</td>
<td></td>
<td>0x0X</td>
</tr>
<tr>
<td></td>
<td>0xXX</td>
<td></td>
</tr>
</tbody>
</table>
```
Put for a Two-Pointer FIFO Example

```
int Fifo_Put(char data) {
    char *Ppt;
    unsigned char SaveSP;
    asm tpa
    asm staa SaveSP
    asm sei
    Ppt=PutPt;
    *(Ppt++)=data;
    if (Ppt == &Fifo[FIFOSIZE])
        Ppt = &Fifo[0];
    if (Ppt == GetPt ) {
        asm ldaa SaveSP
        asm tap
        return(0);
    } else {
        PutPt=Ppt;
        asm ldaa SaveSP
        asm tap
        return(1);
    }
}
```

Scott R. Little (Lecture 9: FIFOs)  ECE/CS 5780/6780  11 / 20
Put for a Two-Pointer FIFO Example

```c
int Fifo_Put(char data) {
    char *Ppt;
    unsigned char SaveSP;
    asm tpa
    asm staa SaveSP
    asm sei
    Ppt=PutPt;
    *(Ppt++)=data;
    if (Ppt == &Fifo[FIFOSIZE])
        Ppt = &Fifo[0];
    if (Ppt == GetPt ) {
        asm ldaa SaveSP
        asm tap
        return(0);
    } else {
        PutPt=Ppt;
        asm ldaa SaveSP
        asm tap
        return(1);
    }
}
```

Scott R. Little (Lecture 9: FIFOs)  ECE/CS 5780/6780  13 / 20

---

Put for a Two-Pointer FIFO Example

```c
int Fifo_Put(char data) {
    char *Ppt;
    unsigned char SaveSP;
    asm tpa
    asm staa SaveSP
    asm sei
    Ppt=PutPt;
    *(Ppt++)=data;
    if (Ppt == &Fifo[FIFOSIZE])
        Ppt = &Fifo[0];
    if (Ppt == GetPt ) {
        asm ldaa SaveSP
        asm tap
        return(0);
    } else {
        PutPt=Ppt;
        asm ldaa SaveSP
        asm tap
        return(1);
    }
}
```

Scott R. Little (Lecture 9: FIFOs)  ECE/CS 5780/6780  14 / 20
Get for a Two-Pointer FIFO

```c
int Fifo_Get(char *datapt) {
    unsigned char SaveSP;
    if (PutPt == GetPt) {
        return(0);    /* Empty if PutPt=GetPt */
    } else {
        asm tpa
        asm staa SaveSP
        asm sei        /* make atomic, entering critical */
        *datapt=*(GetPt++);
        if (GetPt == &Fifo[FIFOSIZE]) GetPt = &Fifo[0]; /* Wrap */
        asm ldaa SaveSP
        asm tap        /* end critical section */
        return(1);
    }
}
```

Initialization of a Two-Pointer/Counter FIFO

```c
#define FIFOSIZE 10    /* can hold 10 */
char static volatile *PutPt;    /* Pointer to put next */
char static volatile *GetPt;    /* Pointer to get next */
char Fifo[FIFOSIZE];
unsigned char Size;        /* Number of elements */
void Fifo_Init(void) {
    unsigned char SaveSP;
    asm tpa
    asm staa SaveSP
    asm sei        /* make atomic, entering critical */
    PutPt=GetPt=&Fifo[0]; /* Empty when Size==0 */
    Size=0;
    asm ldaa SaveSP
    asm tap        /* end critical section */
}
```
Put for a Two-Pointer/Counter FIFO

```c
int Fifo_Put(char data) {
    unsigned char SaveSP;
    if (Size == FIFOSIZE) {
        return(0);  /* Failed, fifo was full */
    } else {
        asm tpa
        asm staa SaveSP
        asm sei   /* make atomic, entering critical */
        Size++;
        *(PutPt++)=data; /* put data into fifo */
        if (PutPt == &Fifo[FIFOSIZE]) {
            PutPt = &Fifo[0]; /* Wrap */
        }
        asm ldaa SaveSP
        asm tap    /* end critical section */
        return(1);  /* Successful */
    }
}
```

Get for a Two-Pointer/Counter FIFO

```c
int Fifo_Get (char *datapt) {
    unsigned char SaveSP;
    if (Size == 0) {
        return(0);  /* Empty if Size=0 */
    } else {
        asm tpa
        asm staa SaveSP
        asm sei   /* make atomic, entering critical */
        *datapt=*(GetPt++);
        Size--;
        if (GetPt == &Fifo[FIFOSIZE]) {
            GetPt = &Fifo[0]; /* Wrap */
        }
        asm ldaa SaveSP
        asm tap    /* end critical section */
        return(1); }
```
FIFO Dynamics

- Rates of production/consumption vary dynamically.
- $t_p$ is time between Put calls, $r_p$ is arrival rate ($r_p = \frac{1}{t_p}$).
- $t_g$ is time between Get calls, $r_g$ is service rate ($r_g = \frac{1}{t_g}$).
- If $\min t_p \geq \max t_g$, FIFO is not necessary.
- If arrival rate can temporarily increase or service rate temporarily decrease, then a FIFO is necessary.
- If average production rate exceeds average consumption rate (i.e., $\bar{r}_p > \bar{r}_g$), then FIFO will overflow.
- A full error is serious because ignored data is lost.
- An empty error is usually not serious.

SCI Data Flow Graph with Two FIFOs