10.7. Multiple Access Circular Queues

source put read sink
producer MACQ consumer

used for data flow problems source to sink
digital filters and digital controllers
fixed length
order preserving
MACQ is always full

*source process (producer)*
places information into the MACQ
oldest data is discarded when new data is entered

*sink process (consumer)*

"I think you've carried the successive refinement of that module far enough."
can read any data
MACQ is not changed by the read operation.

\[
\begin{array}{c|c|c}
\text{MACQ before} & \text{MACQ after} \\
\hline
v[0] & v[0] \\
\end{array}
\]

Figure 10.7. A multiple access circular queue stores the most recent set of measurements.

Perform a 60Hz notch filter on a measured signal.
$v[0]$ $v[1]$ $v[2]$ and $v[3]$ are the most recent data sampled at 360 Hz.

![Frequency response graph](attachment:image.png)

\[
\text{filtered output} = \frac{v[0] + v[3]}{2}
\]

```
unsigned char v[4];
unsigned char samp(void) {
    v[3] = v[2];
    v[2] = v[1];
    v[1] = v[0];
    v     rmb 4
    org $0800
    org $F000
    samp movb v+2,v+3
    movb v+1,v+2
```
v[0] = Ad_In(2);

return (v[0]+v[3])/2;
}

movb v,v+1
ldaa #2
jsr AD_In
staa v
adda v+3 9-bit
rora (v[0]+v[3])/2
rts

10.8. First in first out queue and double buffers

source process / producer

Put

sink process / consumer

Get

FIFO or double buffer

Figure 10.8. FIFO queues and double buffers can be used to pass data from a producer to a consumer.

Table 11.1. Producer-consumer examples.

<table>
<thead>
<tr>
<th>Source/Producer</th>
<th>Sink/Consumer</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyboard input</td>
<td>program that interprets</td>
</tr>
<tr>
<td>program with data</td>
<td>printer output</td>
</tr>
<tr>
<td>program sends message</td>
<td>program receives message</td>
</tr>
<tr>
<td>microphone and ADC</td>
<td>program that saves sound data</td>
</tr>
<tr>
<td>program that has sound data</td>
<td>DAC and speaker</td>
</tr>
</tbody>
</table>

Table 11.1. Producer-consumer examples.
**Figure 11.1. A data flow graph showing two FIFOs that buffer data between producers and consumers.**

* Reg A is data to put into the FIFO

**RxFifo_Put**
- `ldx RxPutPt`
- `staa 1,X+`  
  store into FIFO
- `stx RxPutPt`  
  update pointer
- `rts`

* Reg A returned with byte from FIFO

**RxFifo_Get**
- `ldx RxGetPt`
Program 11.1. Code fragments showing the basic idea of a FIFO.

three modifications that are required to these functions.

If FIFO full when `RxFifo_Put` is called
then the subroutine should return a full error.
If the FIFO is empty when `RxFifo_Get` is called,
then the subroutine should return an empty error.
A finite number of bytes will be permanently allocated

Start with empty FIFO
Put 8 elements
Get 7 elements

Figure 11.3. The FIFO Put operation showing the pointer wrap.
Figure 11.4. The FIFO \texttt{Get} operation showing the pointer wrap.

\begin{verbatim}
RXFIFO_SIZE equ 10
RxPutPt       rmb 2
RxGetPt       rmb 2
RxFifo        rmb RXFIFO_SIZE

Program 11.2. Global structures for a two-pointer FIFO.
\end{verbatim}

\begin{verbatim}
RxFifo_Init  ldx #RxFifo
             stx RxPutPt
             stx RxGetPt
             rts

Program 11.3. Initialize both pointers to the beginning of the FIFO.
\end{verbatim}
Figure 11.5. Flowcharts of the put and get operations.

* Input RegA data to put
* Output RegB \(-1=\text{OK}, \ 0=\text{full}\)

**RxFifo_Put**

```
ps hx
ldx RxPutPt Temporary
staa 1,x+ Try to put
cpx \#RxFifo+RXFIFO_SIZE
bne skip
ldx \#RxFifo Wrap
skip cl rb
```

```
cpx RxGetPt Full if same
beq ok
comb \(-1\) means OK
```
Program 11.4. Data is saved at \texttt{RxPutPt}.

* Input  none  
* Output RegA data from Get  
* RegB \(-1=\text{ok}, 0=\text{empty}\)

\texttt{RxFifo\_Get}

\begin{verbatim}
  pshy
  clrb
  ldy  RxGetPt
  cpy  RxPutPt  Empty?
  beq  done
  comb           -1=OK
  ldaa 1,y+       Data
  cpy  #RxFifo+RXFIFO\_SIZE
  bne  no        wrap?
  ldy  #RxFifo   yes
  no   sty  RxGetPt
  done puly
  rts
\end{verbatim}

Program 11.5. Data is removed from \texttt{RxGetPt}.

\begin{verbatim}
  stx  RxPutPt
  ok   pulx
  rts
\end{verbatim}