Topic 8

FIFO

Synchronous FIFO
FIFO
delinks
producer and consumer
PRODUCER (WRITER)

CONSUMER (READER)

FIFO
Figure 1 - Connecting data with different bit-widths is like trying to connect a small garden hose to a fire hydrant. It doesn't work without a proper adapter.
Say 8 location FIFO

Initially

Producer

Consumer

a little after

Should we move use it to location 7 and increment WP to make it 0?

a little after

points to location to be read

points to location to be written

WP → 7

WP → 0
Depth = \( WP - RP \) (\# of filled locations)

FIFO ( = a Circular buffer)
FIFO initially empty
(D = WP-RP = 0 - 0 = 0)

FIFO depth = 4
(D = WP-RP = 4 - 0 = 4)

FIFO depth = 1
(D = WP-RP = 4 - 3 = 1)

FIFO depth = 7
(D = WP-RP = 2 - 3 = (2 - 3)_{mod 8} = 7)

© Copyright Gandhi Puvvada 2002
FIFO depth = 8
(D = WP-RP = 3 - 3 = (3 - 3)_{mod8} = 0 = \sim 8 !)

FIFO empty
(D = WP-RP = 3 - 3 = (3 - 3)_{mod8} = 0)

WP - RP = 0 \iff \text{EMPTY (if most recently it was almost empty)}
\iff \text{FULL (if most recently it was almost full)}

Example: \text{Almost Empty if depth is less than or equal to 2 and almost Full if depth is more than or equal to 6.}

we need a \text{flip-flop} for each of these (almost empty and almost full) as otherwise when WP = RP you can not tell if it is more than 6 or less than 2.
8-location FIFO

Depth values:
0, 1, 2, 3, 4, 5, 6, 7, 8

\(|WP - RP| \mod 8 =

0, 1, 2, 3, 4, 5, 6, 7

EMPTY

FULL
MOD SUBTRACTOR

ORDINARY SUBTRACTOR uses a

POSTED BORROW TECHNIQUE

\[
\begin{array}{c}
4002 \\
-1234 \\
\hline
\end{array}
\quad \begin{array}{c}
4002 \\
-1234 \\
\hline
\end{array}
\]

Summary

An ordinary \textit{subtractor} with its borrow output ignored is a mod \textit{2} \textit{r} \textit{n-bit} \textit{subtractor}.

\[
\begin{array}{c}
4002 \\
-7234 \\
\hline
\end{array}
\]
MOD SUBTRACTOR

ORDINARY SUBTRACTOR

uses a Posted Borrow Technique

\[
\begin{array}{cccc}
3 & 10 & 10 & 10 \\
4 & 0 & 0 & 2 \\
\end{array}
\]

\[
\begin{array}{cccc}
9 & 9 & 10 & 10 \\
0 & 10 & 10 & 10 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
-1 & 2 & 3 & 4 \\
\end{array}
\]

\[
\begin{array}{cccc}
2 & 7 & 6 & 8 \\
2 & 7 & 6 & 8 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 10 & 10 & 10 \\
4 & 0 & 0 & 2 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 10 & 10 & 10 \\
7 & 2 & 3 & 4 \\
\end{array}
\]

\[
\begin{array}{cccc}
6 & 7 & 6 & 8 \\
& & & \\
\end{array}
\]

Summary

An ordinary subtractor with its borrow output ignored is a mod 2^n subtractor.
SOFTWARE QUEUE (FIFO)

CSCI 102L → DATA STRUCTURE

TAIL ← WP

HEAD ← RP

Ticket Counter
FIFO

Pinout

Flags

WEN

FULL

REN

EMPTY

AF

(Almost full)

AE

(Almost empty)

WD

CLK

RD
Synchronous FIFO (common clock for WRITE & READ domains)
Synchronous FIFO (Common clock for WRITE & READ)
Register array acting as **DUAL PORT memory**
(a **WO** and a **RO**)

- **WA**: Write Address
- **RA**: Read Address
- **WENQ**: Write Enable
- **RENQ**: Read Enable
- **WD**: Write Data
- **RD**: Read Data
- **8X4**
- **DUAL PORT RAM**
- **MEMORY**
- **SIMMODEL=VHDL**
- **VHDL=MEM8X4**
- **REN**
- **CLK**

*Write only Read only*
\[ x - y = x + y' + 1 \]

\[ 0 \text{WA}_2 \text{WA}_1 \text{WA}_0 - 0 \text{RA}_2 \text{RA}_1 \text{RA}_0 \]

By ignoring \( A_4, B_4, S_3, C_4 \), we made it a 3-bit subtractor.

By ignoring \( B_4 \) (and Borrow-3) we made it a mod-8 subtractor.

\( \text{DIFF} \)

\( \text{WP} \)

\( \text{RP} \)
\[ 11X = \overset{\text{RAW almost full}}{\text{RAF}} \]

\[ 01X = \overset{\text{RAW almost empty}}{\text{RAE}} \]

Notice that "000" is excluded from activating RAF or RAE.
We need to record (register) the RAF and RAE to form registered AF and registered AE.

AF = Almost Full = MRSAF =
Most Recently Seen the FIFO running Almost Full

AE = Almost Empty = MRSAE =
Most Recently Seen the FIFO running Almost Empty

We need a 2-state state machine, which can be implemented using either one FF (using the encoded state assignment method) or two FFs (using the one-hot coded state assignment method).

We will show both of them.
The state diagram is the same irrespective of the implementation method.

Please notice that the FIFO may be half-full and it is possible that you had seen most recently the depth crossing (lingering around) the low-threshold or the high-threshold. This (in itself) does not create any ambiguous situation. We make use of AE/AF information when the ambiguous depth situation (WP-RP = 0) arises. Before the FIFO becomes FULL, it would be almost full a little before. Before the FIFO becomes empty, it would be almost empty a little before.
Using a JK FF and the encoded state assignment method. The NSL is simple and heuristically arrived. Just connect RAF to J and RAE to K.

Glitches in RAF and RAE die down by the end of the clock.

Almost FULL

Registered values remain "as is" when WP=RP.

Almost EMPTY

RAW Almost Full

RAW Almost Empty

Can have glitches
Encoded state assignment method using a D Flip-Flop

Heuristic design procedure: Say the D-FF output is labeled as AF. Then initially under reset, the flip-flop should be cleared so as to allow the interpretation of the initial situation of (WP==RP) as indicating "Empty". Also the contents of the Flip-flop should be retained at the current value if neither RAE nor RAF is true using a recirculating mux. If one of the two is true it should be set if RAF is true and reset if RAE is true. So if RAF is lead to the D-input of the FF, the D-FF will set if RAF is true otherwise it will reset. So effectively, the RAF connection to the D input takes care of RAE also.
Encoded state assignment method using a D Flip-Flop

State transition Table method

<table>
<thead>
<tr>
<th>Current state</th>
<th>Next State RAE,RAF = 0, 0</th>
<th>RAE,RAF = 0, 1</th>
<th>RAE,RAF = 1, 1</th>
<th>RAE,RAF = 1, 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Symbolic</td>
<td>Coded (Q*)</td>
<td>Symbolic</td>
<td>Coded (Q*)</td>
</tr>
<tr>
<td>AE</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AF</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q* =
Encoded state assignment method using a D Flip-Flop

State transition Table method

<table>
<thead>
<tr>
<th>Current state</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RAE,RAF = 0, 0</td>
</tr>
<tr>
<td>Symbolic</td>
<td>Coded (Q)</td>
</tr>
<tr>
<td>AE</td>
<td>0</td>
</tr>
<tr>
<td>AF</td>
<td>1</td>
</tr>
</tbody>
</table>
Encoded state assignment method using a D Flip-Flop

Two different designs or are they essentially the same? Can you show that the NSL arrived at heuristically same as the one arrived using the State transition Table method?

\[ Q^* = ? \]

\[ Q^* = Q \cdot \overline{RAE} + RAF \]
\[ Q^* = Q \cdot (\overline{RAF + RAE}) + RAF \cdot (\overline{RAF + RAE}) \]

\[ = Q \cdot \overline{RAE} \cdot \overline{RAF} + RAF \]

\[ = Q \cdot \overline{RAE} + RAF \]

\[ \text{Note: } RAF \cdot RAE \text{ is never true.} \]

\[ \text{Note: } x + \overline{x} \cdot y = x + y \]

Yes! they are the same!
WP = RP

\[ WP - RP = 0 \]

\( \text{EMPTY} \) (if most recently it was almost empty)

\( \text{FULL} \) (if most recently it was almost full)
Now let us use one-hot method of state assignment to implement the 2-state state machine.
Interestingly, we can label the /Q output of the D-FF and manage to work with just one D-FF!

Normally no one uses one-hot state assignment method for a 2-state state machine!

Either

Or
Instead of remembering weather most recently the depth was lingering around the almost empty threshold or the almost full threshold, in order to disambiguate the ambiguous situation caused by WP-RP = 0, we can actually avoid the ambiguous situation totally as follows.

For the 8-location FIFO, we used 3-bit pointers for the WP and the RP. And the 3-bit subtraction (WP-RP)mod_8 produced an 8-valued result \([0, 1, 2, 3, 4, 5, 6, 7]\), where as the depth has nine values, namely \([0, 1, 2, 3, 4, 5, 6, 7, 8]\). This led to the ambiguous situation.

Now suppose we use deliberately 4-bit pointers for the WP and the RP. Then the 4-bit subtraction (WP-RP)mod_16, which can potentially produce 16 values (0-15), will produce all the 9 legal values (0-8) and will never produce the 7 illegal values (9 through 15). There is no ambiguity to be resolved now!

To understand the need to do \(\text{mod}_{16}\) subtraction, consider a slow producer and a fast consumer. The consumer would wait for the producer to deposit one item and in the next clock he would consume it. So the WP would be one step ahead of the RP for just one clock. Most of the time the WP is equal to the RP. A few example are given below.

<table>
<thead>
<tr>
<th>WP</th>
<th>RP</th>
<th>WP-RP</th>
<th>WP</th>
<th>RP</th>
<th>WP-RP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>1</td>
<td>15</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>0</td>
<td>15</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
WEN = Write Enable

WENQ = Write Enable Qualified
(qualified by FULL = 0)

REN = Read Enable

RENO = Read Enable Qualified
(qualified by EMPTY = 0)
Should we require that

The producer requests to write
(makes \texttt{WEN}=1) only after ascertaining that the fifo is not full
(\texttt{FULL}=0)

and similarly

The consumer requests to read
(makes \texttt{REN}=1) only after ascertaining that the fifo is not empty
(\texttt{EMPTY}=0)?

\textbf{Yes and No.}
Yes and No.

Because if FIFO is FULL it should not be written and if FIFO is EMPTY it should not be read.

Because we can do better in Timing Design.

The producer sends a request to write (WEN=1) if he has something to write without waiting to check to see if FULL=1. Later in the clock if he finds that FULL is actually true, he will retry same data write again.

**Critical Path**

At the beginning of a clock, DIFF is produced, FULL and EMPTY are updated by the FIFO. This info is to be relayed to the PRODUCER and CONSUMER who will then activate WEN and REN. Let us try to avoid this round trip.
FIFO to bridge separately clocked domains

Asynchronous (= Two Clock) FIFO

WCLK → FULL

2-clock FIFO

EMPTY → RCLK

Write CLK

Read CLK

WRITE CLOCK DOMAIN

READ CLOCK DOMAIN
Just can't do this subtraction $\text{WP-RP}$.

WP and RP are generated in different clock domains and as such, we can't say when both are stable and valid together.

We need **Synchronization**.

Depth = $\text{WP} - \text{RPSS}$

$\text{Depth} = \text{WPSS} - \text{RP}$$

Delay $\rightarrow$ Safe or Unsafe
Never ever try to synchronize a multibit data item

BAD IDEA

Say WP is changing from \(011\) to \(100\)
Can potentially be any value from 0 to 7!
For RANDOM Data
Have handshake Control signals To govern the transfers
For **sequentially** changing data such as WP and RP use **gray code**.

Since only one bit changes at most in WP, the WPS will either be the old WP or the new WP and will never be totally an absurd value!
Synchronization using GRAY Code

Delay \implies \text{Safe or Unsafe} \implies \text{Safe}

If the FIFO was empty and the \text{WRITER} just wrote a data item, \text{WP} takes 1 or 2 clocks to reach the \text{Read Clock domain} and until that time the reader continues to believe that the FIFO is still empty. So he will delay consuming the data and this is safe! As long as the \text{Consumer} does not consume from an \text{empty FIFO}, it is safe.
A 16-location 2-clock FIFO with 5-bit gray-code counters for WP_G and RP_G and 2 depth producing 5-bit subtractors (one in each of the two clock domains). Note: depth_wr and depth_rd can differ substantially tentatively because of the lag in pointer exchange, but it is all on safe side!
Mr. Bruin offers an alternative design. He says he would change all the 5-bit items to 4-bit (i.e. the counters, synchronizing registers, code converters, subtractors, etc.). In each of the two clock domain he will have a JK flip-flop (or something similar) to remember if most recently the FIFO was running almost empty (AE) or almost full (AF) and according interpret a depth of 0000 as zero (Empty) or sixteen (Full) similar to what was done in a single clock FIFO. He says that this is cheaper so why not use this alternative?

Mr. Bruin changed all [4:0] on the previous page to [3:0] as shown on the next page, and is about to add on each side one set of JK FF and related circuitry (similar to the one below from the single clock FIFO).
Mr. Trojan told him not to do so as it would create a dead-lock!

in a 2-CLK FIFO, don't do this

\[ WP = RP \]

\[ WP - RP = 0 \quad \leftrightarrow \quad \text{EMPTY (if most recently it was almost empty)} \]

\[ \quad \leftrightarrow \quad \text{FULL (if most recently it was almost full)} \]

Use \((n+1)\) bit counters for \( WP \) and \( RP \).
Why?

Well, suppose \( f_{\text{WCLK}} \gg f_{\text{RCLK}} \)

And say, the FIFO is empty.

In 1 blink of the reader (i.e. in 1 clock of the reader), the writer can write all \( 2^n \) locations (essentially go round the complete circle) and the reader would never know that writer did all this. Now the reader continues to believe that the FIFO is empty and would refuse to consume and the writer who knows that the FIFO is full will not write anymore. This causes **dead lock**!