Shift registers

Circuit for simple shift register

Basic applications

- Ring counters
- Johnson counters
- Pseudo-random binary sequences and encryption

Ready-made shift registers are available as integrated circuits, such as the ’165

Conversion of data from serial to parallel and *vice versa*

- Large-scale devices such as ‘universal asynchronous receiver transmitters’ (UARTs) are based on shift registers
- Same functions available in microcontrollers (‘shift’ and ‘rotate’ instructions)
A basic shift register is simply a chain of $D$ flip-flops with a common clock.

Each flip-flop transfers its $D$ input to its $Q$ output at a clock transition.

- The effect is to transfer data along the register, one flip-flop per clock cycle.

This type of register is called a serial input-serial output (SISO).
A basic shift register is simply a chain of $D$ flip-flops with a common clock.

The table shows the contents of the register after successive clock transitions. The assumption is that the register is initially clear.

- The number of clock pulses needed to fill the register is equal to the number of flip-flops used to make the register.
- This is a 4 bit register.
The pattern in successive flip-flops moves to the right with each clock cycle to shift the pattern into and out of the register.
Timing for a shift register

- **Clock**: Represents the clock input to the shift register.
- **Input**: Represents the input signal to the shift register.
- **Q_A**, **Q_B**, **Q_C**, **Q_D**: Output signals of the shift register.
- **t_{pd}**: Propagation delay time for the shift register.

The table shows the timing sequence of the shift register's outputs based on the input:

<table>
<thead>
<tr>
<th>Input</th>
<th>Q_A</th>
<th>Q_B</th>
<th>Q_C</th>
<th>Q_D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Applications of a basic shift register

1. Delay line — $N$ stages delay the signal by $N$ clock cycles

2. Multiplication and division by powers of 2, because this just requires a shift of the binary number (like multiplication or division by 10 in decimal)
   Example: decimal $3 \times 4 = 12$ becomes $11 \times 100 = 1100$ in binary
   The arithmetic logic unit (ALU) of a computer processor uses a shift register for this purpose.

   **Warning**: the ‘sense’ of a shift — left or right — is usually based on its effect on binary numbers written in the usual way. For example, $11 \rightarrow 1100$ is called a **left shift**. This is clearer if both numbers are written with 8-bits as $00000011 \rightarrow 00001100$. Similarly, dividing by 2 such as $00010110 \rightarrow 00001011$ is a **right shift**.

   This is the opposite of what we usually draw in a counter circuit, with the least significant bit (LSB) on the left. **Take care!**

   There is a ‘rotate’ operation where the output from the shift register is fed back to the beginning, usually through the ‘carry bit’.
A shift register with its output fed back to its input forms a **ring counter**.

This can be used to generate an arbitrary binary pattern of length \( N \), where \( N \) is the number of stages in the ring counter. It must be preloaded with the sequence desired, which then rotates around the counter indefinitely.

One application is to divide down the clock frequency for a slower part of a digital system, while keeping everything synchronous. Modern computers have several ‘buses’ running at different speeds, where a ring counter is used to create the clocks for the various buses.

It is much harder to multiply a given frequency to obtain a higher frequency signal. A **phase locked loop (PLL)** is often used.
A ring counter with the complement of its output fed back is a Johnson counter.

This generates longer sequences than a simple ring counter.

For example, a ring counter with 3 stages produces a cycle of 3 states — a waste as there are $2^3 = 8$ states in all.

A Johnson counter with 3 stages has a cycle of 6 and a separate cycle of 2. It is important to ensure that it follows the correct one!
A ring counter with the complement of its output fed back is a Johnson counter.
Pseudo-random sequences of 1s and 0s have many applications, notably in encryption. They appear to be random over ‘short’ times but the sequence eventually repeats, hence the more accurate term ‘pseudo-random’.

Also, they can be reproduced perfectly if you know both:

- the method used to generate the sequence
- the state in the sequence at which to start

This is an important feature! — see next sheet.

The circuit above has a period of $2^4 - 1 = 15$
(the missing state is 0000 —why?).
Pseudo-random binary sequences and encryption

sender

transmit data over insecure link

exclusive or

looks like binary 'noise' — apparently random

receiver

pseudo-random binary sequence

your data (plain text)
Pseudo-random binary sequences and encryption

your data (plain text) \[\rightarrow\] pseudo-random binary sequence

sender

transmit data over insecure link

exclusive or

looks like binary ‘noise’ — apparently random

receiver

same pseudo-random binary sequence

exclusive or

if you exclusive-or a bit with the same value (0 or 1) twice, you get the initial value back again.

your data in plain text again
Pseudo-random binary sequences and encryption

Your data (plain text) transmit data over insecure link

Sender

Exclusive or

Looks like binary ‘noise’ — apparently random

Receiver

Exclusive or

Same pseudo-random binary sequence

If you exclusive-or a bit with the same value (0 or 1) twice, you get the initial value back again.

How do we ensure that both sender and receiver use the same pseudo-random binary sequence?

This is the basis of the method used to encrypt data sent over the internet (https) or with a digital mobile phone.
Transmission of data — serial format

Data often has to be transmitted from one computer to another, or from a computer to peripheral equipment (printer, modem, …). This can be done in:

- **serial format**, one bit at a time
- **parallel format**, several bits at a time (e.g. byte at a time, 8 bits)

Serial format is most commonly used because it is simpler. Only a few wires are needed:

- traditional **serial ‘COM’ ports (RS-232)** need only 3 wires (transmitted data, received data and ground — but more may be used for control)
- **universal serial bus** (USB, common on modern computers) uses 4 wires (two for differential data plus power and ground)

Traditional serial transmission was slow but modern systems use much faster rates (USB version 1 up to 12 Mbits per second, FireWire 1 up to 400 Mbits per second), version 2 of both even faster.

```
simple serial bit stream: 1 0 0 1 1 0 1 1 0 1 0 1
```
Parallel data

Where higher speed is required, several bits (usually a small number of bytes, each of 8 bits) may be moved at once. More complicated connections are needed — more wires. Common applications include:

- **inside the processor itself**, e.g. our microcontroller handles bytes
- **inside a computer system on the bus** (e.g. PCI) and interfaces to **disk drives** (e.g. e.g. SCSI or IDE)— but these are now mainly serial

Interfaces have changed to serial because it is hard to ensure that all bits on a parallel bus arrive at the same time at the high speed of modern systems.
Parallel data

Where higher speed is required, several bits (usually a small number of bytes, each of 8 bits) may be moved at once. More complicated connections are needed — more wires. Common applications include:

- **inside the processor itself**, e.g. our microcontroller handles bytes
- inside a computer system on the **bus** (e.g. PCI) and interfaces to **disk drives** (e.g. e.g. SCSI or IDE) — but these are now mainly serial

Interfaces have changed to serial because it is hard to ensure that all bits on a parallel bus arrive at the same time at the high speed of modern systems.

**How do you interface a serial device to a computer?**

How do we interface an external device that transmits serially with the bus of a computer that transfers one byte (8 bits) at a time?

- **Use a shift register.**

In practice this would almost certainly be buried inside a larger circuit called a **UART** (universal asynchronous receiver transmitter) or something similar.
Use of shift register to serialize data

Extra logic is added to the basic shift register so that all the flip-flops can be loaded in parallel (simultaneously), controlled by a shift/load input.

Once the data have been loaded, the clock is enabled and the values are shifted once per clock cycle. This causes the input data to be transferred to the output, one bit at a time — serial output (PISO).

The opposite process is used to read in serial data, fill up the shift register, and transfer it in parallel to a bus when the register is full (SIPO).

The register can also be parallel input – parallel output (PIPO).

Shift or rotate instructions can be used for the same process inside a microcontroller (if it doesn’t have a UART built in, which many do).
A shift register has 16 stages and is clocked at 1 MHz. How long does data take to pass through it?

We wish to multiply a binary number by 8. How can this be done with a shift register? In which direction should it operate?

Can division be done in the same way? Are there any problems?

A 4-bit ring counter is started with the pattern \( ABCD = 1100 \). What are its subsequent states after the clock is turned on?

Repeat the analysis for a 4-bit Johnson counter.

Prove that a bit returns to its original value, either 0 or 1, after two exclusive-or operations with the same value, whether this is 0 or 1.

Try to work out the sequence produced by the 4-stage pseudo-random number generator described in the lecture. Start at \( ABCD = 1000 \). (What would happen if you started with 0000?)
shift register - 移位寄存器
ring counter - 环形计数器
serial input - 串行输入
parallel input - 并行输入
serial output - 串行输出
parallel output - 平行输出
random number generator - 随机数生成程序
pseudo - 假的
universal asynchronous receiver transmitter - 通用异步收发两用机
universal serial bus - 通用串行总线
COM port - 串行通讯端口
phase locked loop - 锁相环路
exclusive-or - 异或

http://dict.youdao.com/