Sequential Circuits
Why have sequential circuits?

- Sequential systems are time sequential devices - many systems are time sequential
- Examples
  - Memory controllers
  - Bit sequential arithmetic operators
  - Finite State Machines
- Filters, Finite string recognisers, digital combination locks and counters
- Serial line transmitter/receivers - another application for the deserialiser.
Sequential Devices: What is sequential?

In a **sequential** device the result is only **transferred to the output** at the **active** edge of a clock.
Modelling sequential circuits in Verilog

```verilog
always @(posedge clk or A) begin
    // Non-blocking IO
    X <= Y;
    Y <= A;
end
```

Sequential code assignment at the posedge clk

```verilog
always @(A or B) begin
    // Blocking IO
    D = B;
    C = A;
end
```

Combinational code
All values settle, clock period >> settling time
Deserialiser Animation!!!
VERILOG TIP 1: Unwanted latches in VERILOG Synthesis

➤ A reg keeps the value of its last assignment until it is assigned to again. If a reg is assigned to on some path of execution through an always block but not on all paths, it behaves as a latch.

➤ Make sure that non-latch reg variables are assigned to through every path - both arms of ifs, and all arms of cases. A case should have a default even if all possible inputs match some label.

➤ Also be aware that unintentional feedback in combinational circuits produces latches - see HLAB2 at the end.

always @(*) data = data

Or..

always @(*) data <= data
VERILOG TIP 2: Incomplete Event Control Lists

➤ Make sure that the necessary variables appear in *always* block sensitivity lists. If an input is missing from the sensitivity list, its lack of influence will induce unexpected behavior.

➤ One solution: In combinational *always* blocks use *always @(*)*.

```
always @(A) begin
  X = A;
  Y = B;
end
```
The essential feature of sequential circuits is that they have memory.

Example: a burglar alarm must remember whether it was tripped...

![Diagram of a burglar alarm circuit with a sensor, memory element, and alarm.]
Schmitt Trigger
Schmitt Trigger

- VTC with hysteresis
- Restores signal slopes
Bistable Latch

- A latch is a level controlled memory device.
- There are two stable states for this device. As it has no inputs, the one it assumes depends on its power-up phase.
Basic SR Latch

![SR Latch Diagram]

R

S

Q

\overline{Q}

Basic SR Latch

R

Q

S

\overline{Q}
SR Latch Truth Table

- **A Race condition** occurs if $S = 1$ and $R = 1$

<table>
<thead>
<tr>
<th>$S$</th>
<th>$R$</th>
<th>$Q(t+1)$</th>
<th>$Q(t+1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$Q(t)$</td>
<td>$Q(t)$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>reset</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>set</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
SR Latch Timing

![SR Latch Timing Diagram]

---

ENGN3213: Digital Systems and Microprocessors L#9-10
What is wrong with the following circuit?

```verilog
module RSlatch_nor_comb(S, R, Q);
    input S;
    input R;
    output Q;
    wire Qa;
    wire Qb;
    assign Qa = ~(S | Qb);
    assign Qb = ~(R | Qa);
    assign Q = Qa;
endmodule
```
What is wrong with the following circuit?

module dff (D, Clk, Q);
    input D, Clk;
    output Q;

    wire  D, Clk;
    reg   Q;

    always @(posedge Clk) Q <= D;
endmodule
What is wrong with the following circuit?

```
always @(Clk or D) begin

//This works...
    Q = Q2a;
    Q2a = ~(S2 & Q2b);
    Q2b = ~(R2 & Q2a);

    S2 = ~(Q1a & Clkp);
    R2 = ~(Q1b & Clkp);

    Q1a = ~(S1 & Q1b);
    Q1b = ~(R1 & Q1a);

    S1 = ~(So & Clkn);
    R1 = ~(Ro & Clkn);
    So = D;
    Ro = ~D;
    Clkp = Clk;
    Clkn = ~Clk;

end
```
What is wrong with the following circuit?

```verilog
always @ (Clk or D) begin

    // This does not work
    Q    = Q2a;
    Q2a  = ~(S2 & Q2b);
    Q2b  = ~(R2 & Q2a);
    #1S2  = ~(Q1a & Clk);
    #1R2  = ~(Q1b & Clk);
    Q1a  = ~(S1 & Q1b);
    Q1b  = ~(R1 & Q1a);
    S1   = ~(D & ~Clk);
    R1   = ~(~D & ~Clk);

end
```

18 ENGN3213: Digital Systems and Microprocessors L#9-10
D Latch and Gated D Latch

➤ The SR latch has memory and could therefore serve the role of an alarm control - but it is **transparent**
➤ Also there is the (1,1) ambiguity

➤ → Gated D-latch
Gated D Latch

➤ Note that the CLOCK still a level control.
➤ Undesirable (1,1) condition does not arise.

<table>
<thead>
<tr>
<th>Clock</th>
<th>D</th>
<th>$Q(t + 1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>×</td>
<td>$Q(t)$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
module Gated_D_latch(Clk, d, Q);
input Clk;
input d;
output Q;

/**
assign Q = Clk ? q : d;
***/

reg Q;

always @(*) begin
    if(Clk) Q = D;
end
endmodule
D Flip Flop

(a)

(b)

(c)
module D_flip_flop(Clk, d, Q);
    input Clk;
    input d;
    output Q;
    reg Q;

    always @(negedge Clk) begin
        Q <= D;
    end
endmodule
D Flip Flop with ASYNCHRONOUS Preset and Clear

(a)

(b)
module DFF_AR (D, Clock, Presetn, Q);

input D;
input Clock;
input Presetn;

output Q;
reg Q;

always @(negedge Presetn or negedge Clock)
    if (!Presetn)
        Q <= 0;
    else
        Q <= D;

endmodule
D Flip Flop with SYNCHRONOUS Preset and Clear

➤ The way to avoid generating glitches in sequential digital designs is by making all inputs to be **synchronous**

➤ Gating the input signal and the $\overline{\text{Clear}}$ signal through the AND gate eliminates glitches.
module DFF_SR (D, Clock, Resetn, Q);

  input D;
  input Clock;
  input Resetn;

  output Q;
  reg Q;

  always @(negedge Clock)
    if (!Resetn)
      Q <= 0;
    else
      Q <= D;

endmodule
ENGN3213 Digital Systems and Microprocessors

Examples of sequential circuits
DFF Timing parameters

DFF Timing parameters

DFF Timing parameters

CLOCK

D

Q

\( t_{su} \)

\( t_{h} \)

\( t_{w} \)

\( t_{p} \)
Flip flops-1: T-flip flop

<table>
<thead>
<tr>
<th>$T$</th>
<th>$Q(t + 1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$Q(t)$</td>
</tr>
<tr>
<td>1</td>
<td>$\overline{Q(t)}$</td>
</tr>
</tbody>
</table>

(a)

(b)
Flip flops-2: T-flip flop

T-latch???
Flip flops-3: T-flip flop

(a)

(b)
Flip-flops-4: J-K flip flop

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>Q(t + 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q(t)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>(\overline{Q(t)})</td>
</tr>
</tbody>
</table>

Diagram of J-K flip flop: [Diagram of J-K flip flop]
Flip-flops-5: J-K flip flop - Two implementations

(a) J-K flip flop schematic

(b) Master and Slave implementation

(c) J-K flip flop symbol
Applications of Registers

- PIPO - Parallel Input Parallel Output
- SIPO - Serial Input Parallel Output
- PISO - Parallel Input Serial Output
- SISO - Serial Input Serial Output
Counters: Asynchronous Binary Upcounter

![Diagram of an asynchronous binary upcounter]

- **Clock**
- **Q0**
- **Q1**
- **Q2**
Counters: Asynchronous Binary Upcounter or Ripple Counter

- T-flip flops: $Q \rightarrow \overline{Q}$ if $T = 1$, $Q \rightarrow Q$ if $T = 0$
- Works by changing only on the rising edge of the clock inputs.
- Asynchronous because the flip-flops do not toggle synchronously with the clock
- There is an increasing delay from flip-flop to flip-flop from left to right which leads to a constraint on size and speed.
- The propagation effect leads to the term: **ripple counter**.
- Eliminate this problem with a synchronous counter.
Counters: Synchronous Counter
Modelling counters

- Counters are simple state machines
- FSM model has the advantage that the outputs of the counter can be arbitrary (not just count up / count down)
Parallel to Serial and Serial to Parallel Conversion

- Conversion from serial communications channels or in low hardware resources situations one oftens uses **serial data**.
- Need to convert from parallel to serial and vice versa.
- Performed by **shift registers**
Serial-in Serial-out Shift Register

Data → D1 → D → Q → Q1
Clock → Clk

D2 → Clk
D3 → Clk
Serial-in Serial-out Shift Register Timing

Clock

D1

Q1/D2

$\tau_h$

Q2/D3

$\tau_s$

Q3

$\tau_h$

$\tau_s$
Parallel-in Parallel-out Shift Register

74LS395 parallel-in/parallel-out shift register with tri-state output
Parallel Access Shift Register

➤ The parallel access shift register can be used for both serial to parallel and parallel to serial conversion.
Arithmetic Processors

- Can do operations in serial rather than parallel. E.G. Serial adder versus parallel adder (e.g. ripple carry adder)
- Good for saving hardware.
- Good for saving hardware. Bad for speed.
Serial Multiplier

➤ Have already seen ripple carry adders
➤ Can apply serial processing to multiplication as well
➤ Input b is in parallel and the bits of a in increasing significance are processed serially - same as textbook multiplication.
Asynchronous Communications: A sequential Device

➤ Asynchronous serial: very common communications protocol → many descendents: SPI (serial peripheral interface), RS485, I2C, packet radio (AX25)

➤ Used to be widely used by computers (e.g. PC, MAC, ...) but still dominant in microprocessors. Many micros have built in serial comms.

➤ Asynchronous means that the data clock is unimportant. The receiver assumes that its local clock is about the same as that of the transmitter.

➤ This is not good enough for some of the descendents. e.g. radio packet has a synchronising protocol in the PHY protocol layer called HDLC (High level data link control).

➤ C.F. A synchronous communications protocol where the receiver must lock onto the clock and synchronously decode the data. E.G. All wireless protocols are synchronous.

➤ Asynchronous serial comes as either RS232 ±(12 – 15)Volts or TTL levels (0-5 V for e.g.). You may remember RS232 from the “dial-up days”.

➤ Rather slow: 1200 baud, 19 Kbaud, 56 kBaud even 900 kBaud in specialised circumstances.

➤ In anycase all high speed Internet uses synchronous protocols e.g. Ethernet.
RS232 Signal traces

[Diagram showing RS232 signal traces with labels for Start, Stop, LSB, MSB, and data bits b0 to b7.]
RS232 pinouts (UART = Universal Asynchronous Receiver Transmitter)

Pin Connections

- **Rx**  - Data Receive (DTE)
- **Tx**  - Data Transmit (DTE)
- **RTS**  - Request to Send Data ...(DTE wants to Transmit)
- **CTS**  - Clear to Send ...............(DCE indicates OK to Receive)
- **DTR**  - Data Terminal Ready ......(DTE device present and ready)
- **DSR**  - Data Set Ready ............ (DCE device present and ready)
- **DCD**  - Data Carrier Detect
- **RI**  - Ring Indicator
- **GND**  - Common for All Above Signals

Diagram:

```
<table>
<thead>
<tr>
<th>PC</th>
<th>UART</th>
<th>MODEM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tx</td>
<td>Phone Line</td>
</tr>
<tr>
<td></td>
<td>Rx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RTS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CTS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DTR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DSR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DCD</td>
<td></td>
</tr>
</tbody>
</table>
```

3213: Digital Systems & Microprocessors: L#15

ENGN3213: Digital Systems and Microprocessors L#9-10
RS232 pinouts
RS232 pinouts
DVB-T Transmitter Block Diagram (ETSI EN 300 744 V1.4.1 (2001-01))

(ETSI = European Telecommunications Standards Institute)

Figure 1: Functional block diagram of the System
The system is defined as the functional block of equipment performing the adaptation of the baseband TV signals from the output of the MPEG-2 transport multiplexer, to the terrestrial channel characteristics. The following processes shall be applied to the data stream (see figure 1):

- transport multiplex adaptation and randomization for energy dispersal;
- outer coding (i.e. Reed-Solomon code);
- outer interleaving (i.e. convolutional interleaving);
- inner coding (i.e. punctured convolutional code);
- inner interleaving;
- mapping and modulation;
- Orthogonal Frequency Division Multiplexing (OFDM) transmission.

The system is directly compatible with MPEG-2 coded TV signals ISO/IEC 13818 [1].
To ensure adequate binary transitions, the data of the input MPEG-2 multiplex shall be randomized in accordance with the configurations depicted below.

Figure 2: Scrambler/descrambler schematic diagram

The polynomial for the Pseudo Random Binary Sequence (PRBS) generator shall be (see note):

$$14x^3 + x^2$$
DVB-T Inner Coder (ETSI EN 300 744 V1.4.1 (2001-01))

The system shall allow for a range of punctured convolutional codes, based on a mother convolutional code of rate 1/2 with 64 states.