# Sequential Circuits

Madhav Desai

February 14, 2018

## 1 Sequential Functions

A *one-sided sequence* from a finite set $A$ is a set of elements

$$\{x(0), x(1), x(2), \ldots\}$$

and is represented by $\{x(k)\}$. A sequential function $f$ maps sequences to sequences:

$$f(\{x(k)\}) = \{y(k)\}$$

$f$ is a causal sequential function if, given that

$$\begin{aligned} f(\{x_1(k)\}) &= \{y_1(k)\} \\ f(\{x_2(k)\}) &= \{y_2(k)\} \end{aligned}$$

and that $x_1(k) = x_2(k)$ for $k \leq M$, it is also true that $y_1(k) = y_2(k)$ for $k \leq M$. Simply put, the output at instant $k$ should not depend on inputs at instants greater than $k$.

## 2 Sequential Functions: Finite State Machines

Clearly, we can implement a sequential function only if it is causal. Further, we can only implement a finite amount of memory. Thus, we need a general model for sequential systems which has these two properties.

- We are interested in those implementations of causal sequential functions that need a finite amount of memory.
- A *Mealy* machine $(Q, \Sigma, \Lambda, \delta, \lambda, q_0 \in Q)$ is a system with an input sequence $\{x(k)\}$ with $x(k) \in \Sigma$, and output sequence $\{y(k)\}$ with $y(k) \in \Lambda$, an internal state sequence $\{q(k)\}$ with $q(k) \in Q$, and the following relations between the sequences $\{x(k)\}$, $\{y(k)\}$ and $\{q(k)\}$.

  $$\begin{aligned} q(0) &= q_0 \\ q(k+1) &= \delta(x(k), q(k)) \\ y(k) &= \lambda(x(k), q(k) \end{aligned}$$

  Here, $\Sigma,\ \Lambda,\ Q$ are finite sets and $\delta,\ \lambda$ are functions from $\Sigma \times Q \to Q$ and $\Sigma \times Q \to \Lambda$ respectively.

# 3 State Minimization of a Mealy Machine

In many cases, if we are given a Mealy FSM, we can reduce the number of states without changing the behaviour of the state machine.

For an integer $m \geq 0$, two states $u, v$ are said to be $m$-compatible if it is not possible to distinguish them using input sequences of length $m$. That is, for $k \geq 0$, the sequence of outputs $y(k), y(k+1), \ldots y(k+m-1)$ produced by an input sequence $x(k), x(k+1), \ldots x(k+m-1)$ is the same if $q(k)$ is either $u$ or $v$.

Then, the minimization procedure (described in Kohavi) proceeds as follows:

- Find the sets of 1-compatible states. This is easy, because $u$ and $v$ are 1-compatible if $\lambda(u, \sigma) = \lambda(v, \sigma)$ for all $\sigma \in \Sigma$. We generate a collection of subsets of 1-compatible states, which is a partition of the state space.
- Now for $k > 1$, we use induction. Assuming that we know the set of 1-compatibles and the set of $(k-1)-$compatibles, the collection of $k$-compatible sets can be built by noting that $u$ and $v$ are $k$-compatible if they are 1-compatible, and further for each $\sigma \in \Sigma$, the states $\lambda(u, \sigma)$ and $\lambda(v, \sigma)$ are $(k-1)$-compatible.
- As soon as the set of $k$-compatibles converges (that is the set of $k$-compatibles is the same as the set of $(k-1)$-compatibles), we can stop. This convergence will happen for some $k \leq (|Q| - 1)$ (why?). At the convergence point, we have a collection of sets of compatible states.
- The reduced state machine can be constructed easily: The number of states in the reduced state space is the size of the collection of compatible sets. The transitions are built up in the obvious manner (how?).

One of the consequences of this procedure is that two incompatible states in a Mealy FSM can be distinguished by an input sequence of length at most $|Q| - 1$.

# 4 Implementing a Mealy Machine

Implementation of a Mealy machine using logic gates is easy.

- The sets $\Sigma$, $\Lambda$, $Q$ are encoded with bit-vectors (note: there are many possible encodings). After this step, $x(k)$, $y(k)$ and $q(k)$ are viewed as bit-vectors. Thus, $\delta$ and $\lambda$ become Boolean functions which we know how to implement as combinational functions. We implement the following

$$\begin{aligned} y(k) &= \lambda(x(k), q(k)) \\ nq(k) &= \delta(x(k), q(k)) \end{aligned}$$

  where $nq(k)$ is introduced to represent the next state.

- We need a concept of a sequential delay element to implement

$$q(k+1) \;=\; nq(k)$$

- Some questions:
  - How are the instants $k$ defined?
  - Using logic gates, how can we construct the delay element?
  - Under what conditions will a logic circuit faithfully implement the equations describing a Mealy machine?

## 4.1 Defining the time instants: a clock

To define the time instants $k$, we can use a periodic waveform, for example a square wave, and associate the rising edges of the square wave with the time-instants. The simplest clock is a square wave with 50% duty-cycle. We denote the period of the clock by $T$.

If $x$ is a wire in a logic circuit, then $x(k)$ is the voltage on the wire at the $k^{th}$ rising edge on the clock.

## 4.2 A delay element: the data flip-flop (DFF)

The data (or delay) flip-flop is a logic circuit which

- has two inputs: a data input $d$, a clock input $clk$, one output $q$.
- The $d$ input is sampled at the rising edge of clock, and sampled value appears at $q$ after a delay ($d_{clk \to q}$).
- The data input must be stable for a period $S$ before the sampling clock edge and a period $H$ after the sampling clock edge (the setup time $S$ and the hold time $H$).

## 4.3 Implementation of a Mealy machine

The implementation of a Mealy machine then proceeds as follows:

- Encode the states in $Q$ using a set of state variables $\mathbf{s} = (s_0 s_1 \ldots s_k)$. You will need at least $\log_2 |Q|$ state variables.
- Encode the input and output symbol sets $\Sigma$ and $\Lambda$ using variables $\mathbf{x} = (x_0 x_1 \ldots x_m)$ and $\mathbf{y} = (y_0 y_1 \ldots y_p)$ repectively.
- Implement the next-state and output functions $\lambda$ and $\delta$ using logic gates.
- Use one flip-flop for each state variable and connect the next-state variables to the flip-flop inputs (outputs are connected to state variables).

The structure of the resultant circuit is shown in Figure 1

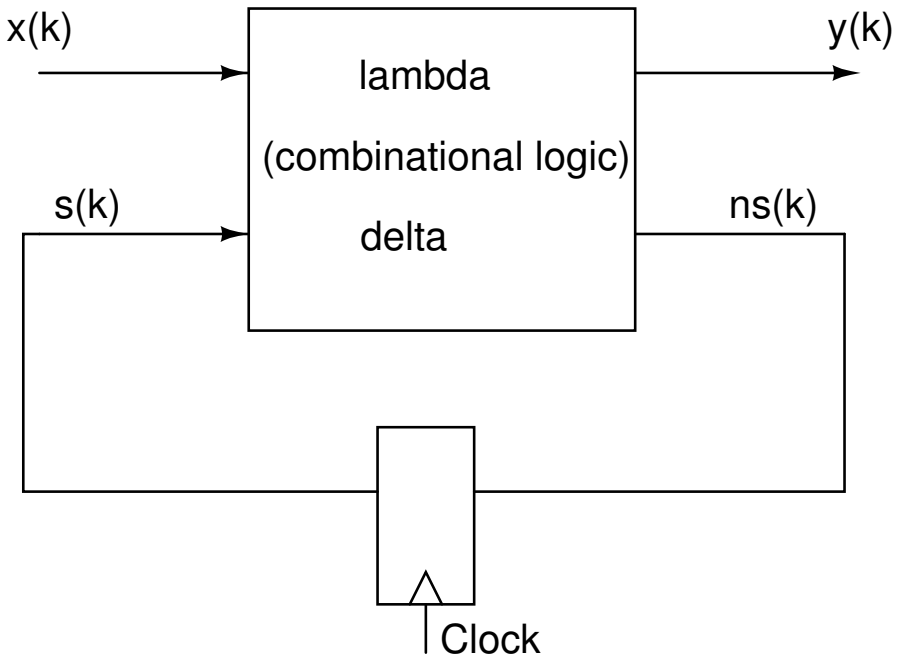


Figure 1: Mealy machine circuit structure

### 4.4 Timing correctness of a Mealy machine

For the implementation in Figure 1 to be faithful to the original model, we must ensure that the D-flipflops will correctly sample the next-state values. That is, the computation of the next-state values must be completed so that the correct next-state value is stable during the flip-flop setup and hold window.

The computation of the next state value at time $k$ starts from the clock instant $k-1$, and will take a certain amount of time to complete. The paths of interest start from clock, proceed through a D-FF (the launch D-FF), continue through combinational logic and end at the input of a D-FF (the capture D-FF). Let $m$ and $M$ be the minimum and maximum possible delays of paths of interest (including the $Clock \rightarrow Q$ delay of the launch flip-flop.

The circuit will correctly implement the Mealy machine if

$$\begin{aligned} m &\geq H \\ M &\leq T - S \end{aligned}$$

If the clock at the capture flip-flop is delayed by an amount $\Delta$ relative to the clock at the launch flip-flop, then

$$\begin{aligned} m &\geq H + \Delta \\ M &\leq T - (S - \Delta) \end{aligned}$$

### 4.5 Summary: design procedure for a Mealy machine

- Choose input, state and output encodings.
  - Some common encodings: binary and variants (e.g. Gray codes), one-hot encodings, k-hot encodings etc.
- Implement the $\lambda$ and $\delta$ function blocks.
- Introduce the delay elements to close the loop.
- Confirm that the circuit works correctly at some clock period (and calculate the minimum clock period and circuit setup and delay times).

## 5 Signoff: Specification of a CMOS positive-edge-triggered Sequential Circuit Block

To finish the design process, you must characterize your implementation and find:

- The input set-up time relative to the rising edge of the clock (could be different for each input).
- The minimum and maximum delays from an input of the system to an input of a flip-flop (the input-to-flop delay, which could be different for each input).
- The minimum and maximum values of the clock to output delay (could be different for each output bit, both minimum and maximum delays should be specified).
- The minimum and maximum values of the clock to flip-flop input delay.
- The minimum and maximum values of the input to output delay.
- For CMOS circuits, the input capacitance at all inputs (including clock) and the maximum load capacitances that can be connected at the outputs.

## 6 Problem set

Implement an FSM with the following specification:

- The set of input symbols is $\Sigma = \{a,\ b\}$.
- The set of output symbols is $\Lambda = \{Y, N\}$.

The behaviour of the FSM is as follows:

- The FSM outputs a $Y$ at time instant $k$ only if the last 4 inputs were either

```
abab
baba
```

(that is, either $x(k) = a, x(k-1) = b, x(k-2) = a$, $x(k-3) = b$ or $x(k) = b, x(k-1) = a, x(k-2) = b$, $x(k-3) = a$). Otherwise the FSM outputs $N$.

Suppose you are given the following building blocks:

- Inverters, NAND2, NOR2 gates each with a delay of 2 units.
- D-flipflops with $clock \rightarrow q$ delay of 3 units, set-up time of 2 units and hold-time of 2 units.

1. Design an abstract Mealy machine (identify a potential set of states and next-state, output functions) which implements the required behaviour.
2. Minimize the set of states by finding compatible subsets of states.
3. Encode the input, output and state symbols using binary encoding.
   - Try two state encodings: the one-hot encoding and a compact encoding using $log_2|Q|$ state variables.
4. Complete the logic network for the FSM based on your encodings. Don't forget to add the reset signal. You may assume that the output is a dont-care when the reset signal is applied.
5. Confirm that your implementations are correct. Check the following test-case

```
k   0      1 2 3 4 5 6 7 8 9
x   reset  a b b a b a b a a
y   _      N N N N N Y Y Y N
```

6. Characterize the timing of your FSM implementations (find the minimum and maximum values of the input to flop, flop to flop, flop to output and input to output delays).

```
input to flop: x to nq
flop to flop: clock of q to nq
flop to output: clock of q to y
Input to output: x to y
```