TLP, SDS and Boolean, MapReduce and WSC

Thread Level Parallelism (OMP)

- Fork / Join Parallelism

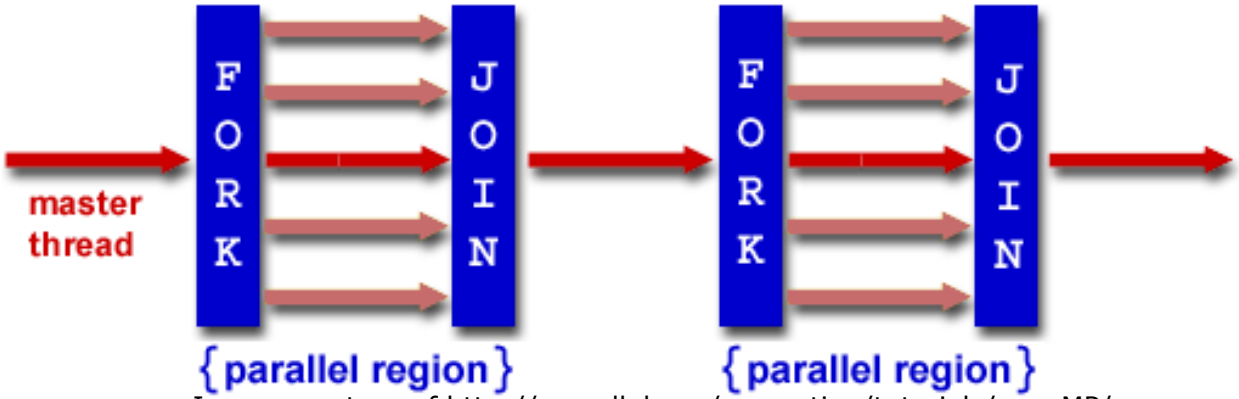


Image courtesy of http://www.llnl.gov/computing/tutorials/openMP/

- Speedup limited by linear portion
- Amdahl's Law, Speedup = 1 / [(1-F) + F/S]
- Synchronization wait time

Thread Level Parallelism (OMP)

- OpenMP: API allows us to code multi-threaded programs
- Share work across threads to increase performance
- Use **directives** to create multi-threaded code segments
- `#pragma omp parallel`
- Code inside block will be run on each thread
- Helper functions
  - `omp_get_num_threads()` = Number of Threads running
  - `omp_get_thread_num()` = Current Thread ID

Thread Level Parallelism (OMP)

- Duplication

```
#pragma omp parallel for
{
    for(int i=0; i<ARRAY_SIZE; i++)
        z[i] = x[i] + y[i];
}
```

- False Sharing

```
#pragma omp parallel
{
    int thread_id = omp_get_thread_num();
    int num_threads = omp_get_num_threads();
    for(int i=thread_id; i<ARRAY_SIZE; i+=num_threads)
        z[i] = x[i] + y[i];
}
```

- Data Race

```
#pragma omp parallel for
{
    for(int i=0; i<ARRAY_SIZE; i++)
        sum += x[i];
}
```

Thread Level Parallelism (OMP)

```
int values[100000];
#pragma omp parallel
{
        int i = omp_get_thread_num();
        int n = omp_get_num_threads();
        for (; i < 100000; i += n) {
                values[i] = i;
                #pragma omp barrier
        }
}
```

#pragma omp barrier is a directive that requires all threads to wait until other threads reach the barrier before proceeding.

Suppose each core has a 32 KiB direct mapped data cache with 64-byte blocks. (Write-back, write-allocate)

- If the snippet is run with one thread, how many cache misses for the values array? 6250
- If the snippet is run with two threads, how many cache misses for the values array? 100,000
- Name the phenomenon that explains the difference between these? False Sharing
- Using two threads and removing the barrier, would the number of cache misses decrease? Why?

  Yes, one thread could get far enough ahead and start working on a different cache block.

Thread Level Parallelism (OMP)

Without changing the #pragma statement, alter this code to correctly take advantage of multithreading. (Ignore edge cases)

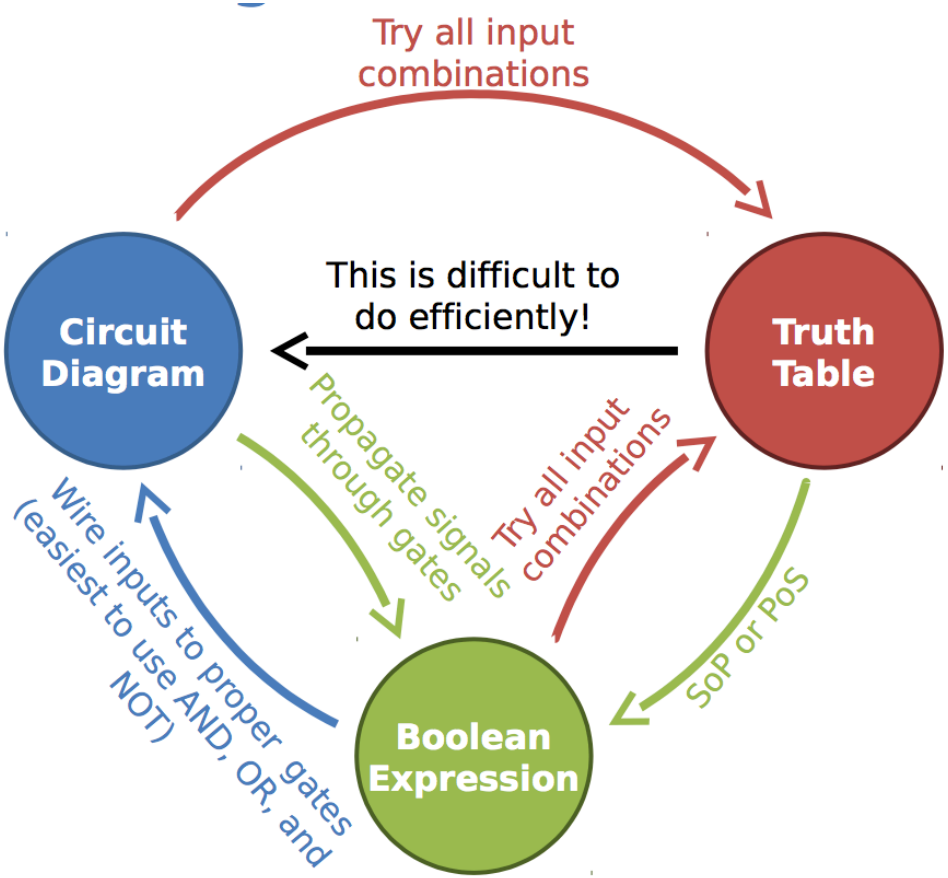
```
int values[100000];
#pragma omp parallel
{
    int i = omp_get_thread_num();
    int n = omp_get_num_threads();
    for (; i < 100000; i += n)
        values[i] = i;
}
```

```
int values[100000];
#pragma omp parallel
{
    int i = omp_get_thread_num();
    int n = omp_get_num_threads();
    for (100000/n*i; i < 100000/n*(i+1); i += 1)
        values[i] = I;
}
```

Synchronous Digital Systems

Combinational Logic

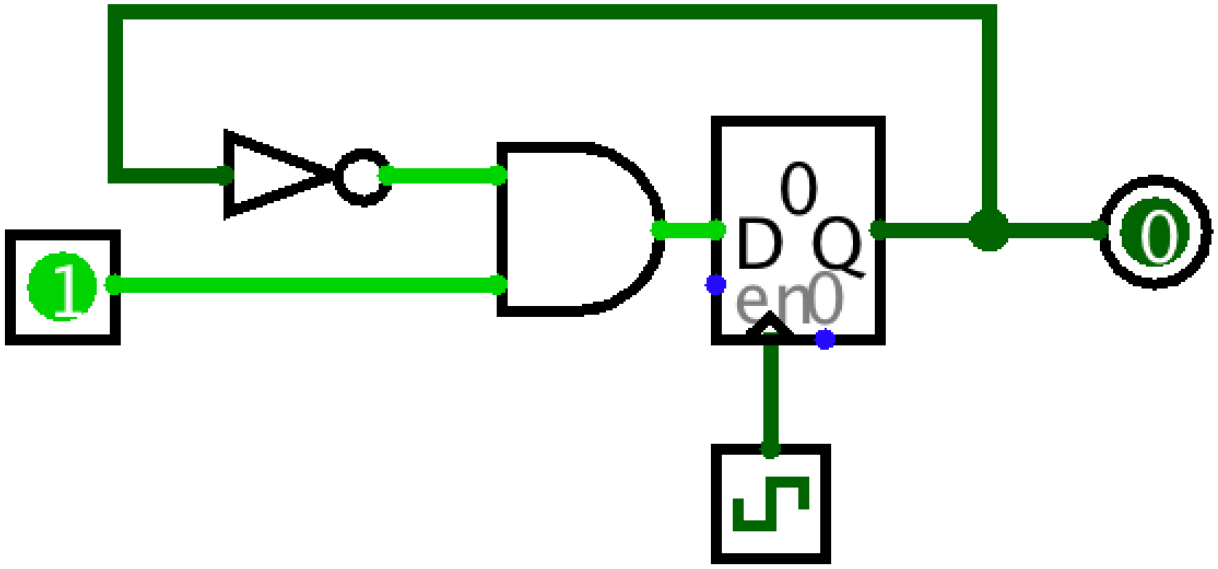
- Circuit Diagram, Truth Table, Boolean Expression



Synchronous Digital Systems

Sequential Logic

- State elements, circuits that "remember"
- Controls flow of information between CL blocks



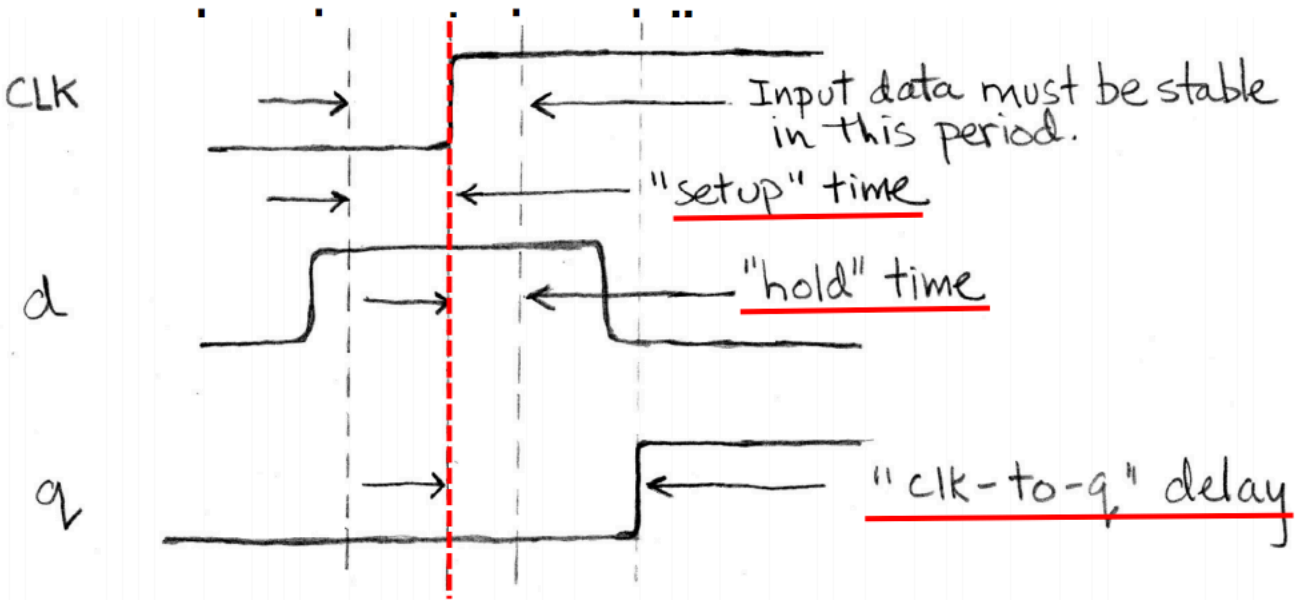
Synchronous Digital Systems

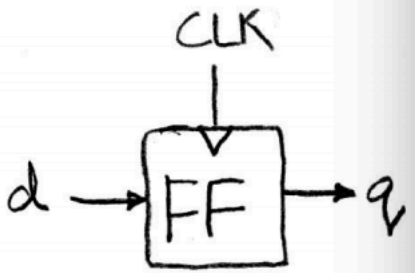
Sequential Logic Timing

- Setup Time: Input must be stable *before* the CLK trigger for proper read
- Hold Time: Input must be stable *after* the CLK trigger for proper read
- "CLK-to-Q" Delay: Takes time for output to change *after* the CLK trigger
- Must design stateful circuits with these in mind

Synchronous Digital Systems

Sequential Logic Timing





Synchronous Digital Systems

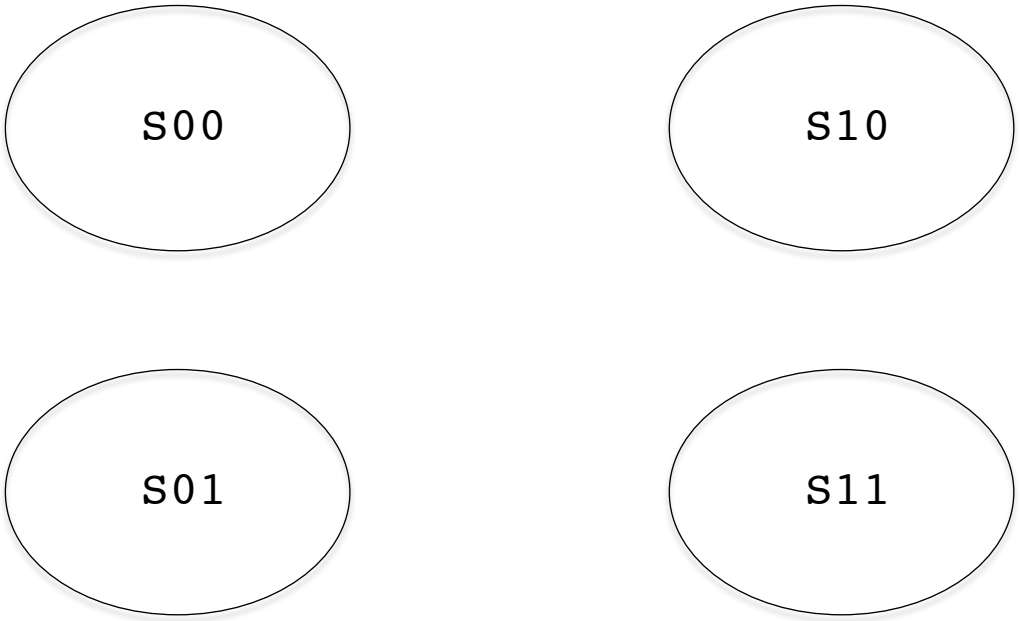
Garcia Spring 2007 F2

We are trying to implement an FSM that outputs a 1 iff the last 3 input bits form a palindrome. Assume that the input was all 1s before the circuit begins.

```
Example:
I: . . . 1 1 1 0 0 1 1 1 0 0 0 1 0 1 1 0 0
O: . . . 1 1 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0
```

Synchronous Digital Systems

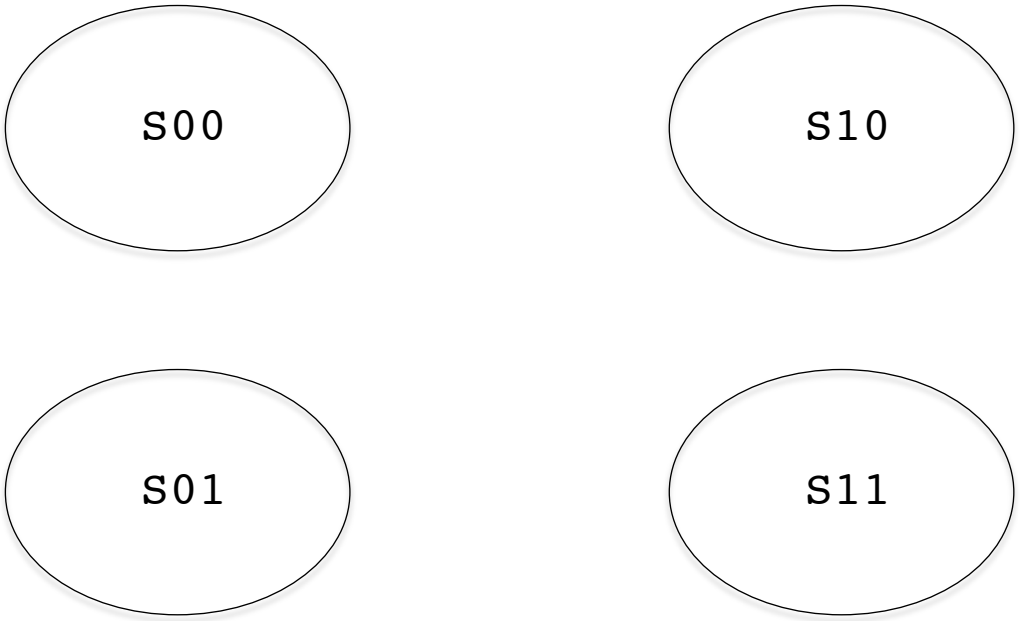
Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Synchronous Digital Systems

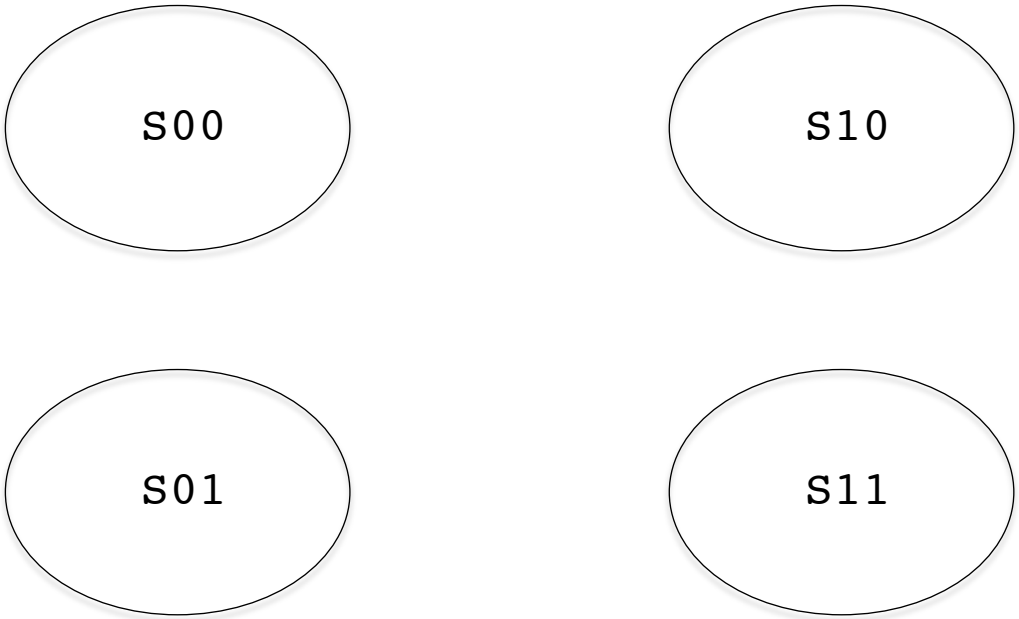
Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Synchronous Digital Systems

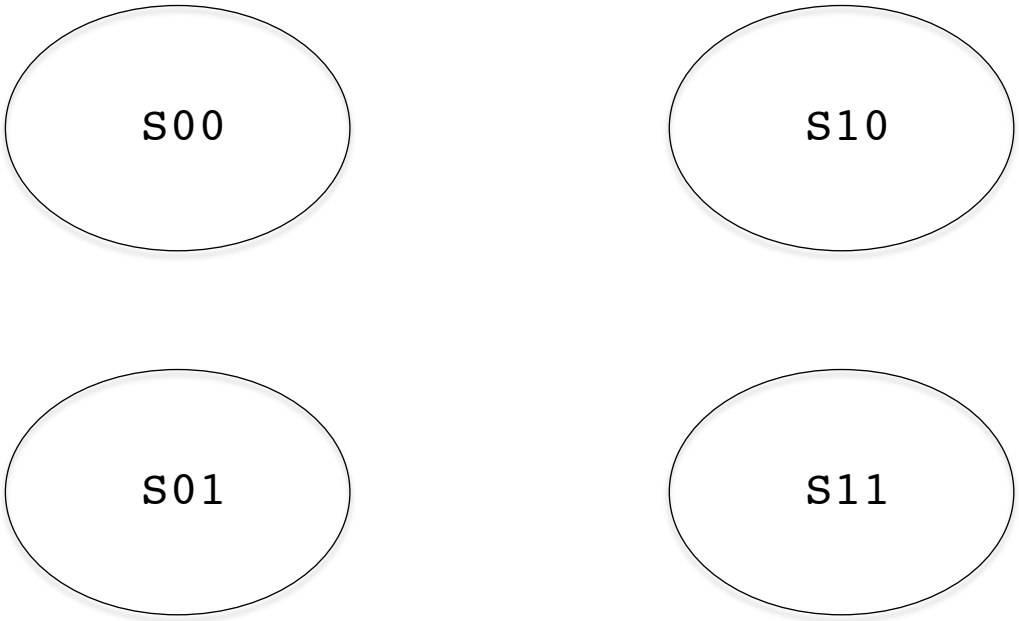
Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Synchronous Digital Systems

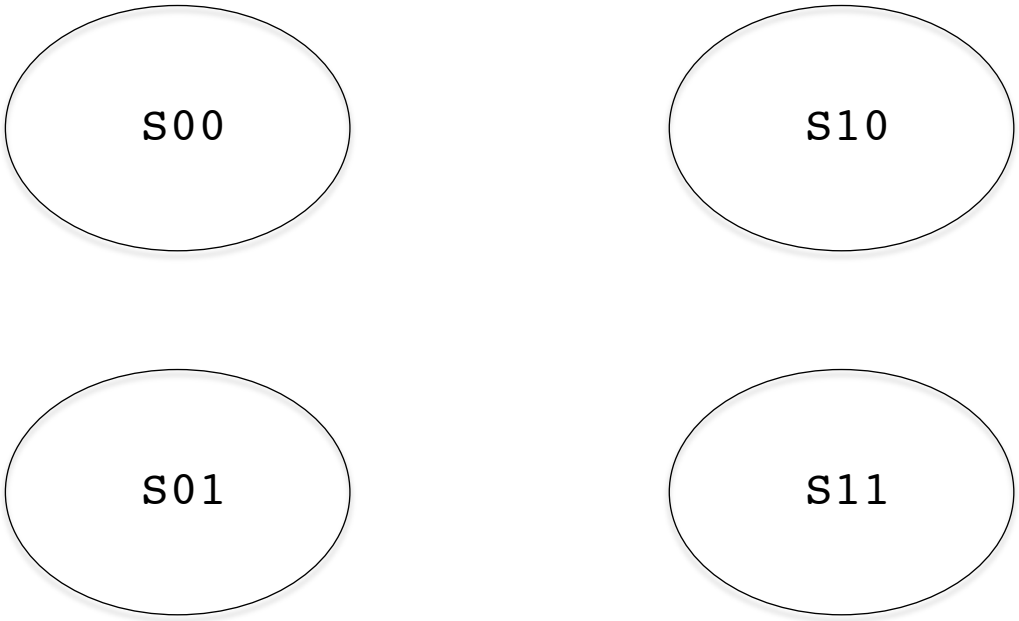
Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Synchronous Digital Systems

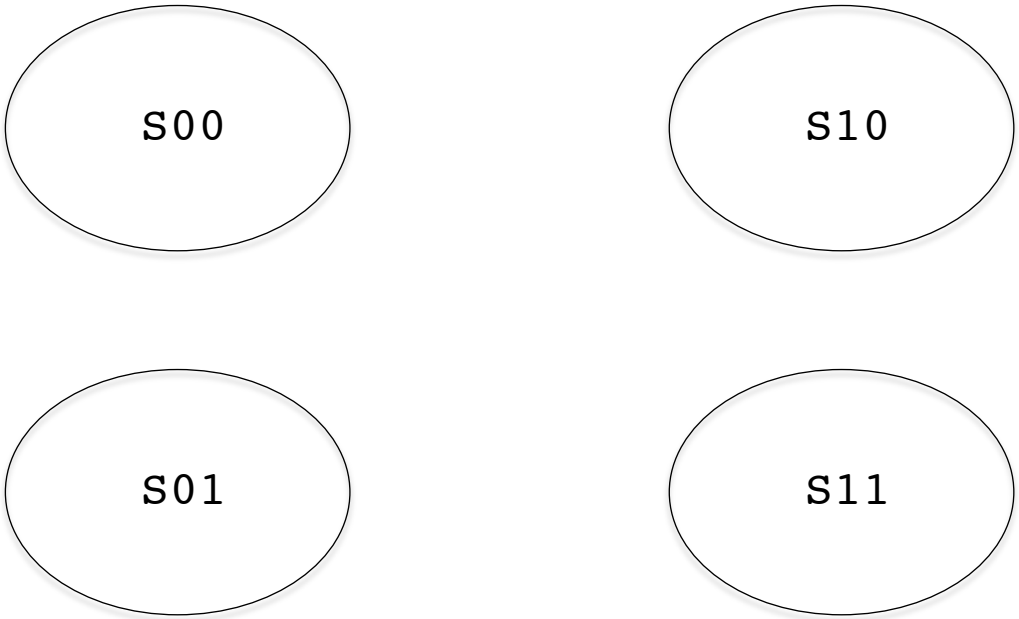
Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Synchronous Digital Systems

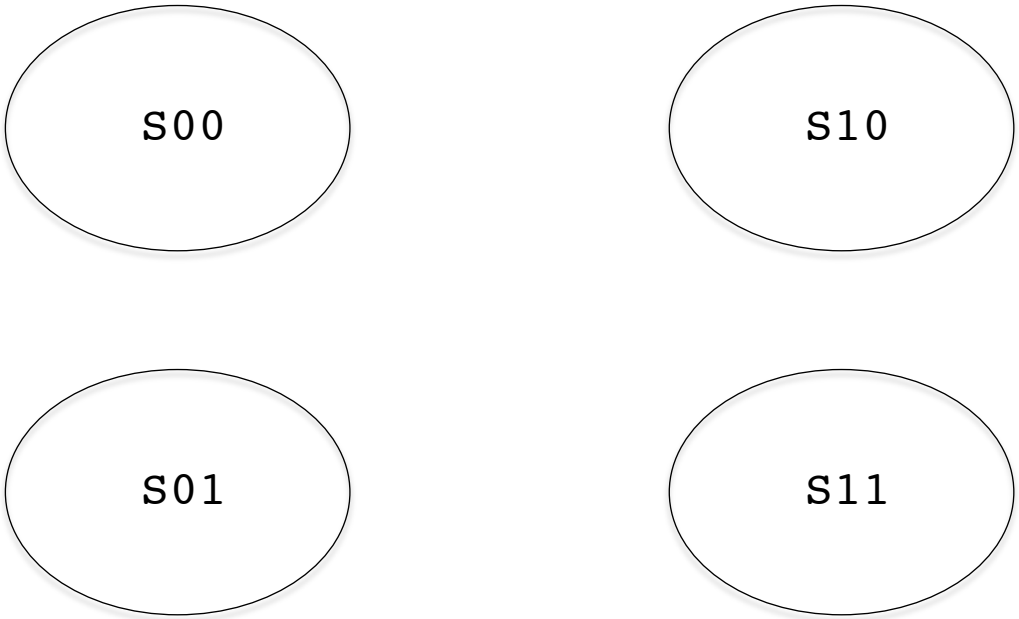
Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1			
1	1	0			
1	1	1			

Synchronous Digital Systems

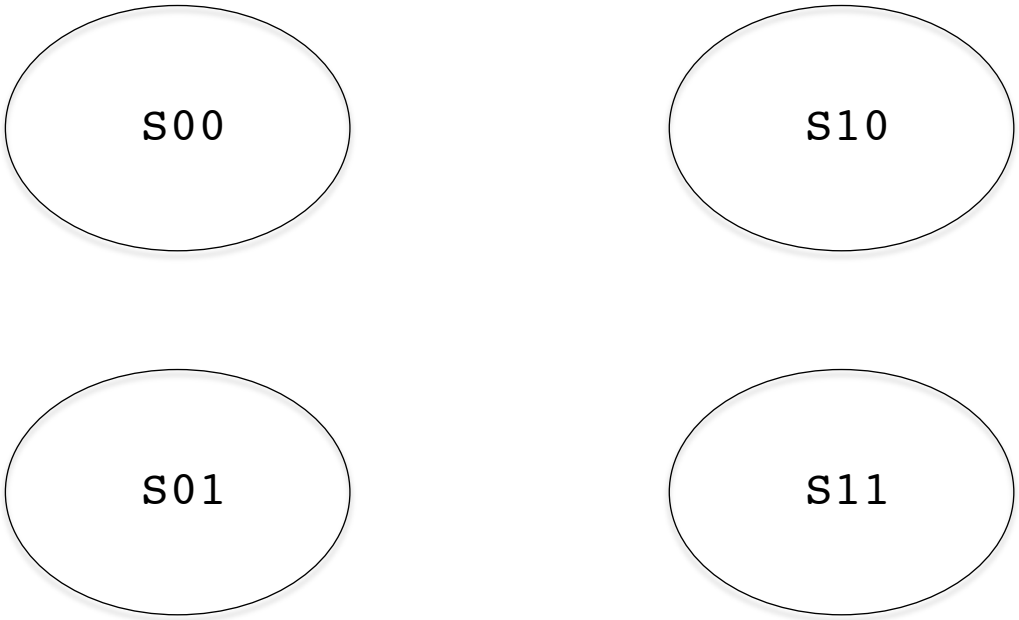
Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0			
1	1	1			

Synchronous Digital Systems

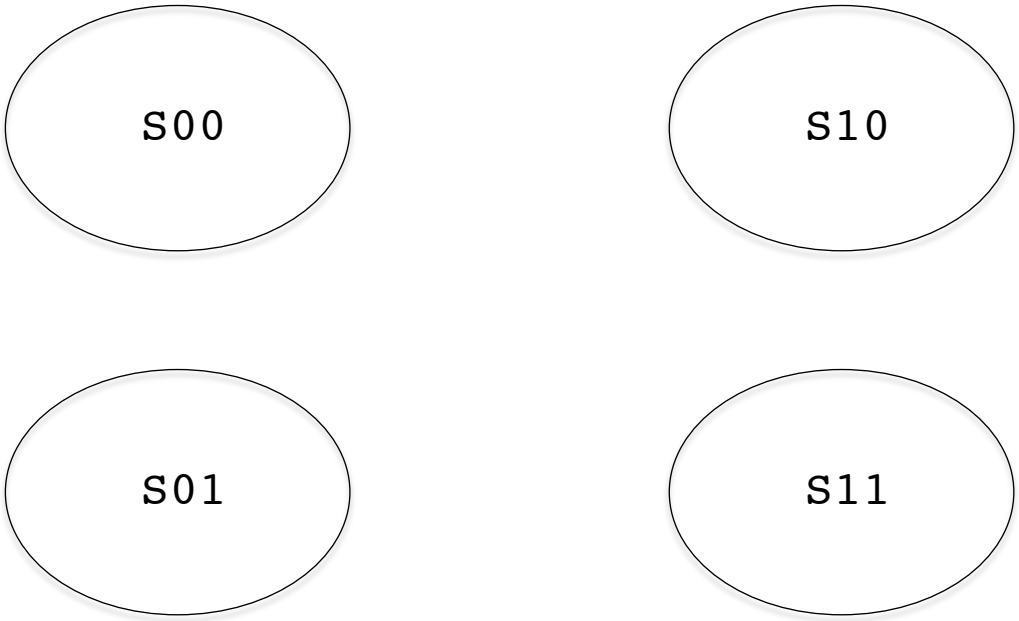
Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1			

Synchronous Digital Systems

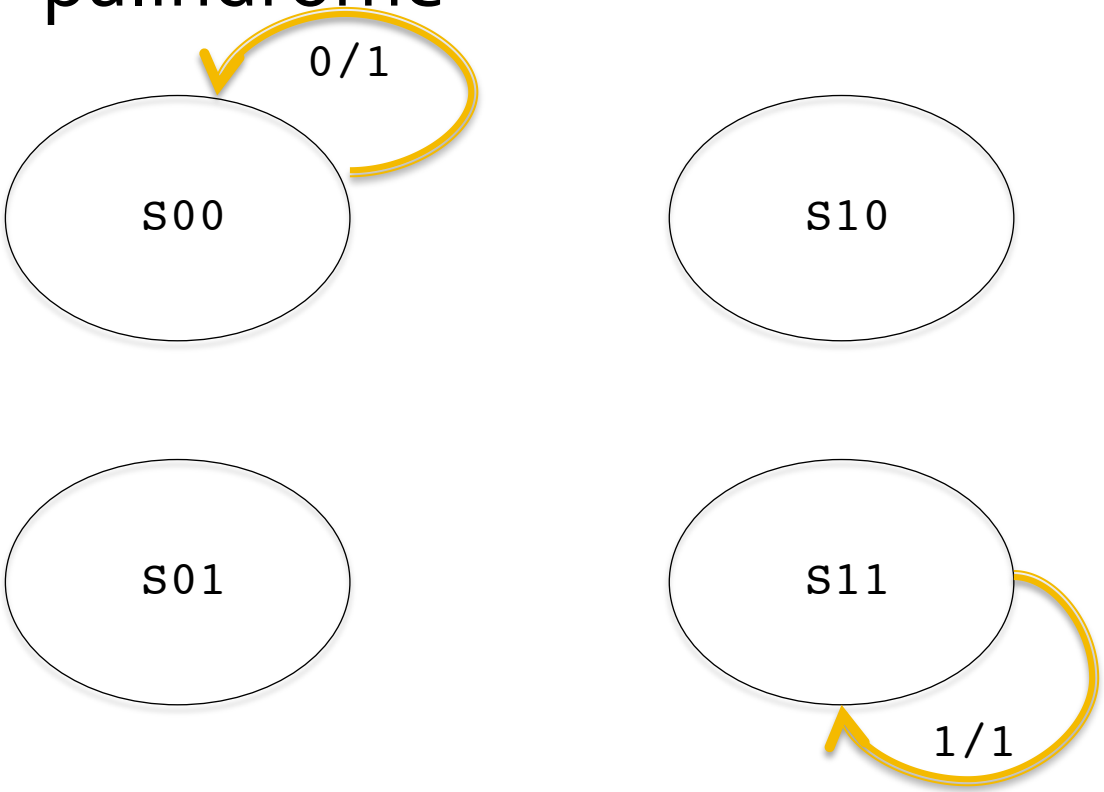
Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	1	1	1

Synchronous Digital Systems

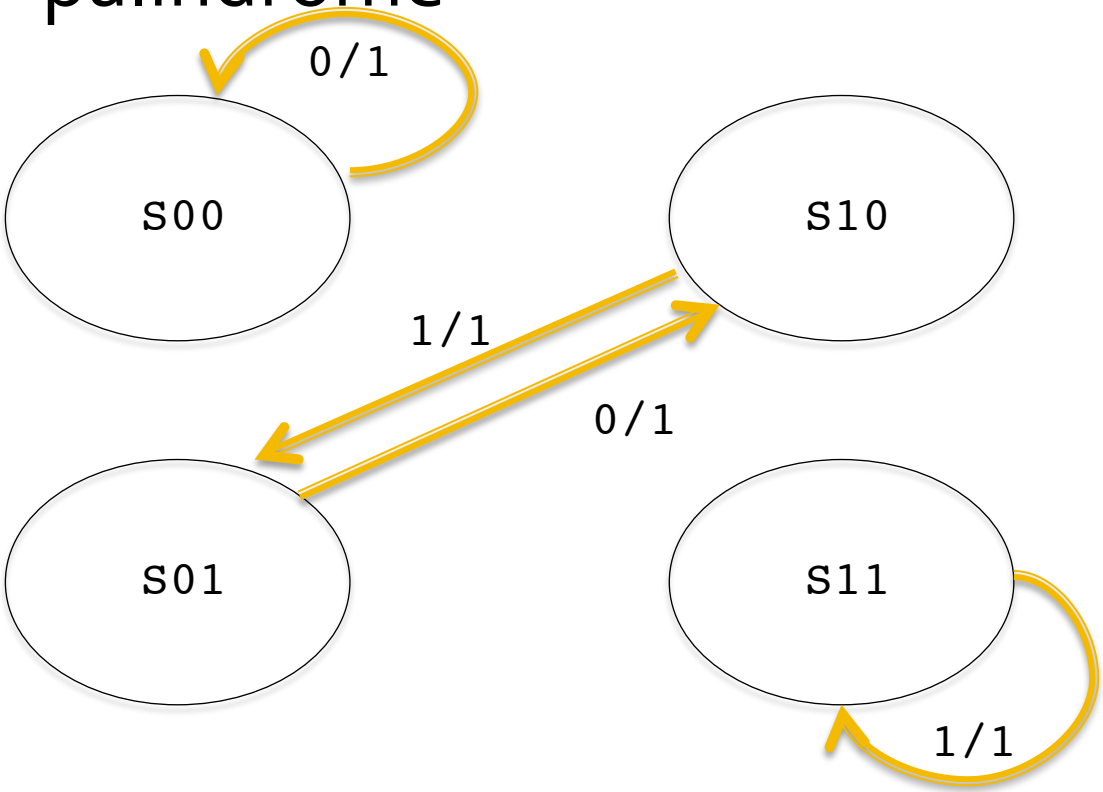
Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	1	1	1

Synchronous Digital Systems

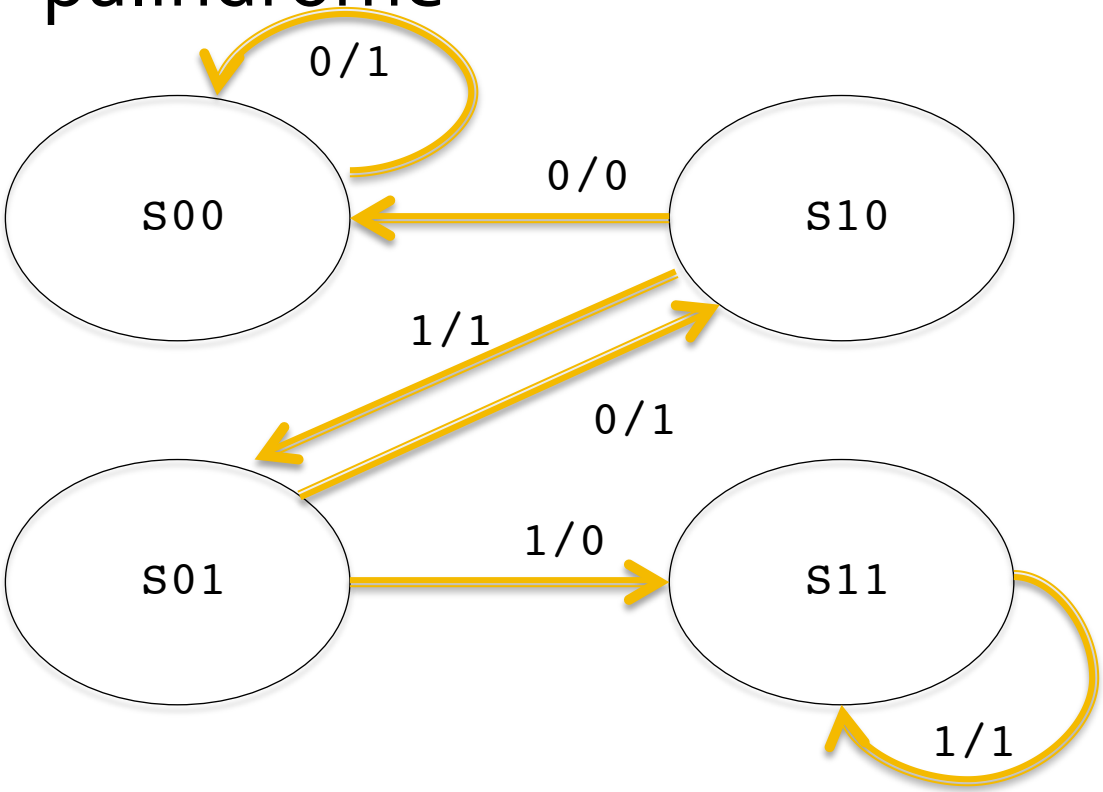
Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	1	1	1

Synchronous Digital Systems

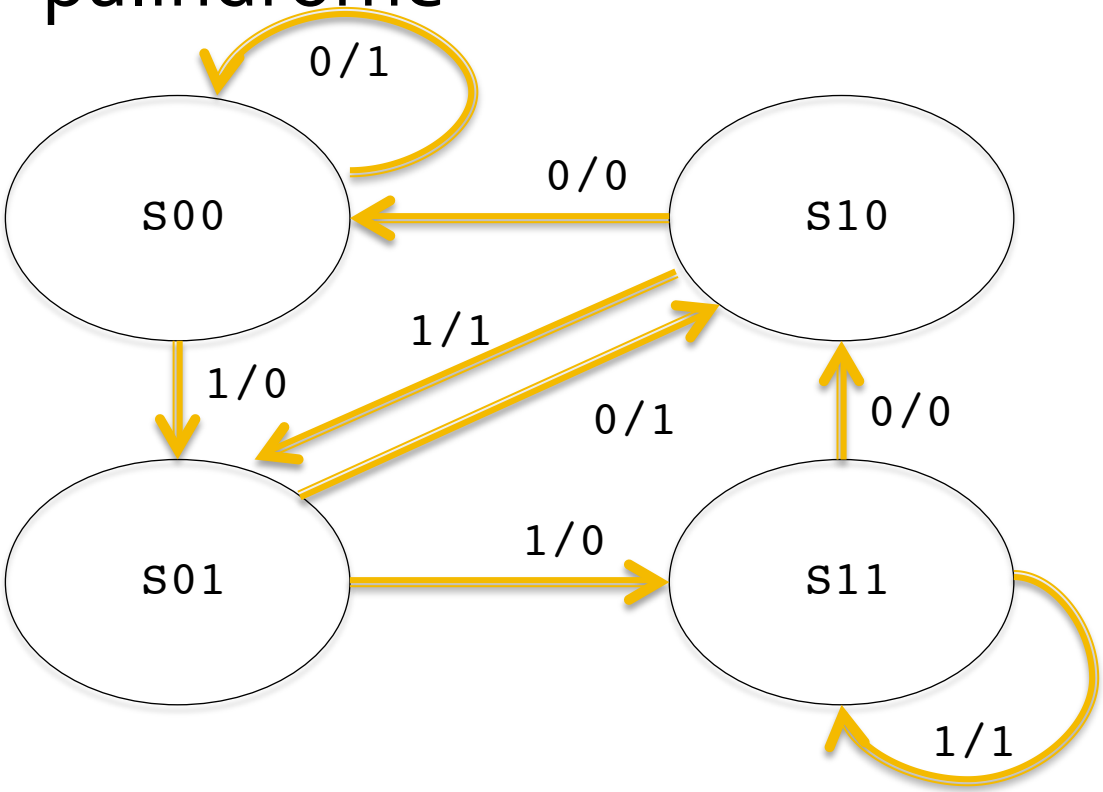
Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	1	1	1

Synchronous Digital Systems

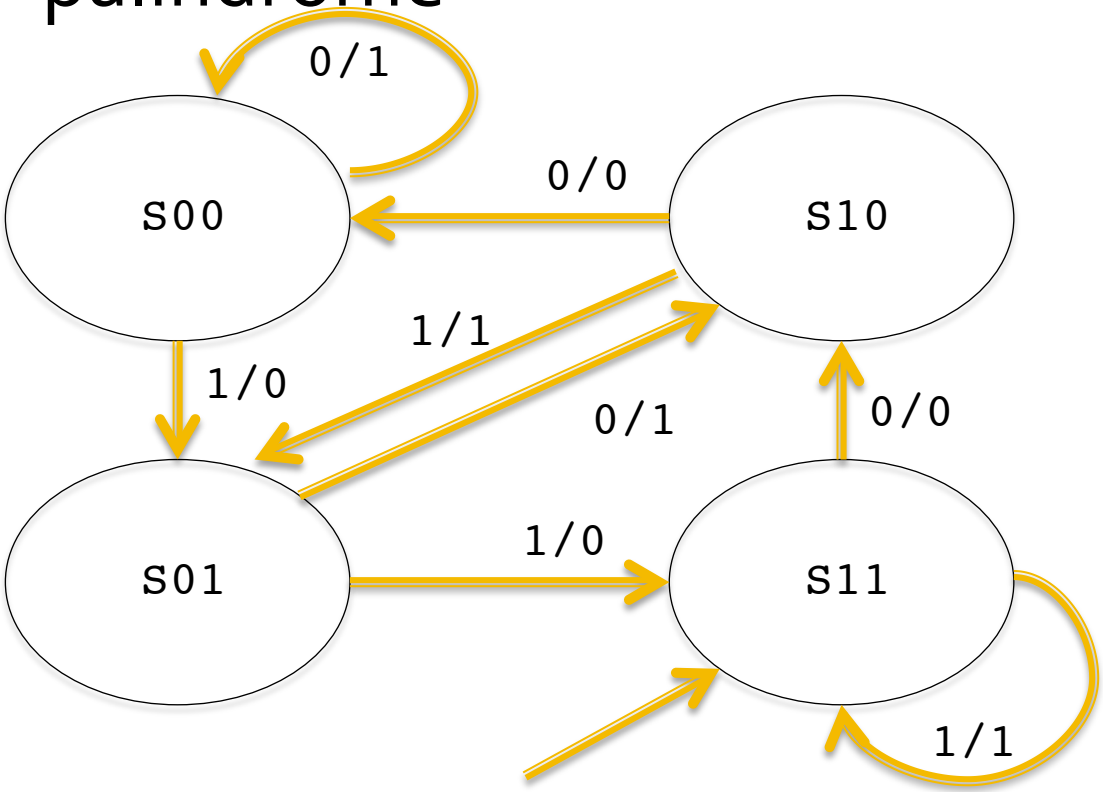
Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	1	1	1

Synchronous Digital Systems

Outputs a 1 iff the last 3 input bits form a palindrome



P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	1	1	1

Synchronous Digital Systems

Provide a *fully reduced* Boolean expression for the Output as a function of `P1`, `P0`, and `I`

P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	1	1	1

What is the name of this circuit?

Synchronous Digital Systems

Provide a *fully reduced* Boolean expression for the Output as a function of P1, P0, and I

P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	1	1	1

$$\overline{P1 \oplus I}$$

What is the name of this circuit?

Synchronous Digital Systems

Provide a *fully reduced* Boolean expression for the Output as a function of P1, P0, and I

P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	1	1	1

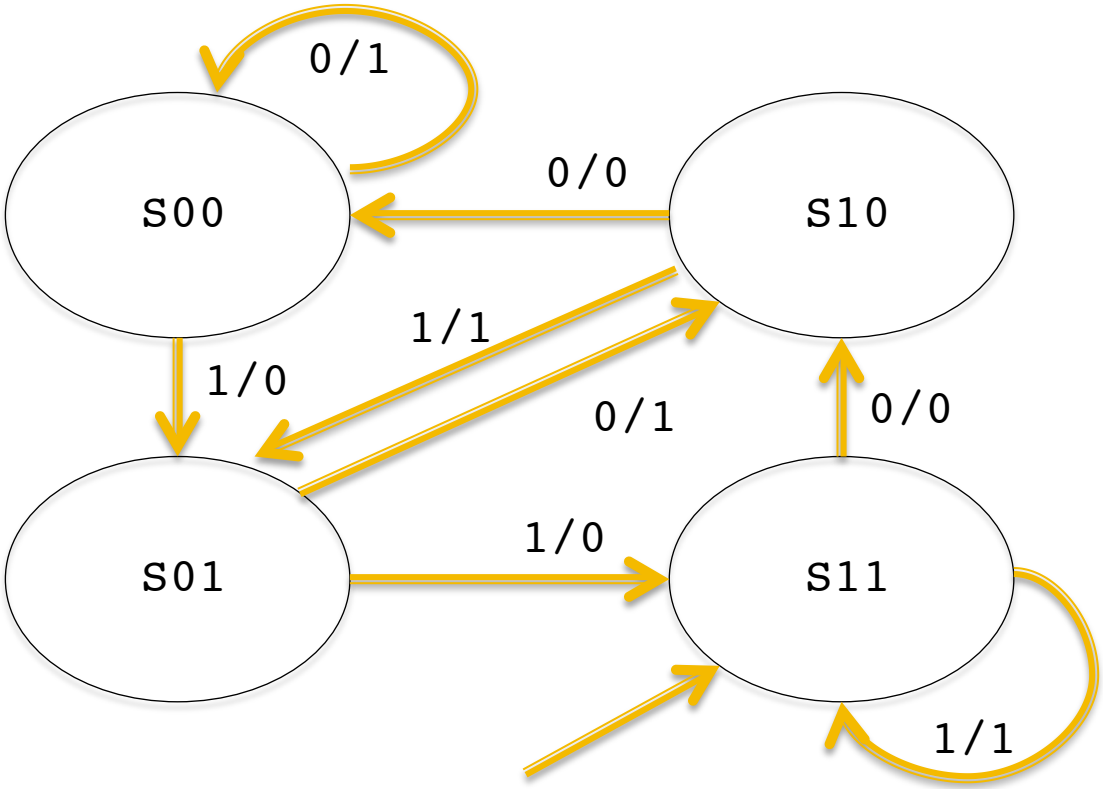
$$\overline{P1 \oplus I}$$

What is the name of this circuit?

XNOR

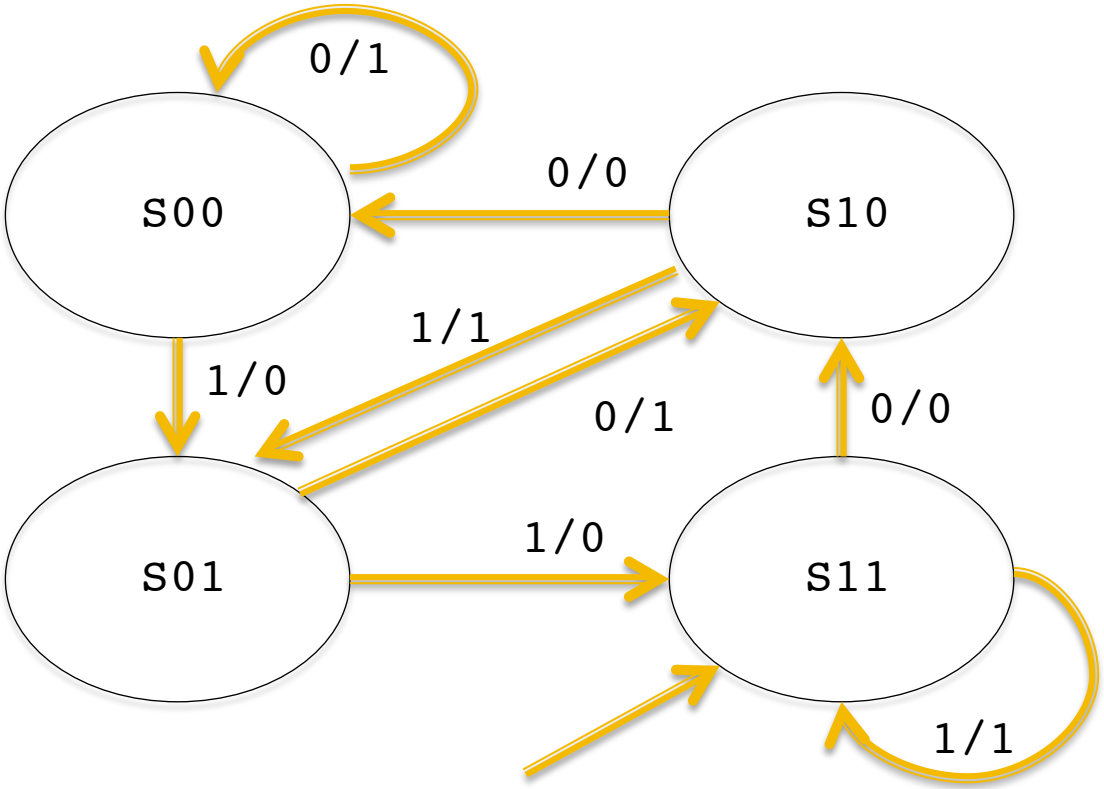
Synchronous Digital Systems

What is the shortest length input stream that can guarantee we test this circuit exhaustively?



Synchronous Digital Systems

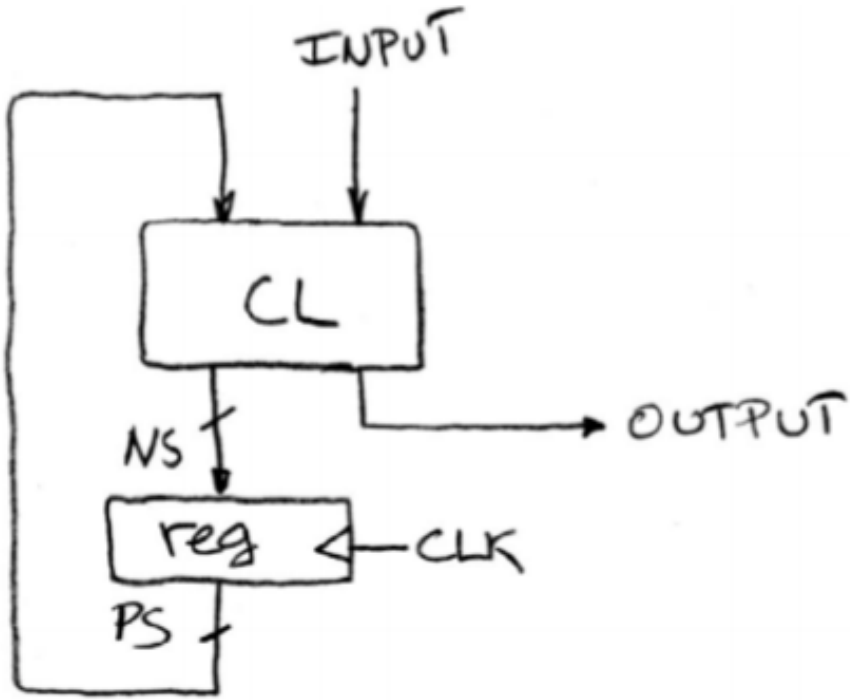
What is the shortest length input stream that can guarantee we test this circuit exhaustively?



Find a path through all transitions starting from S11:
8 (10001011)

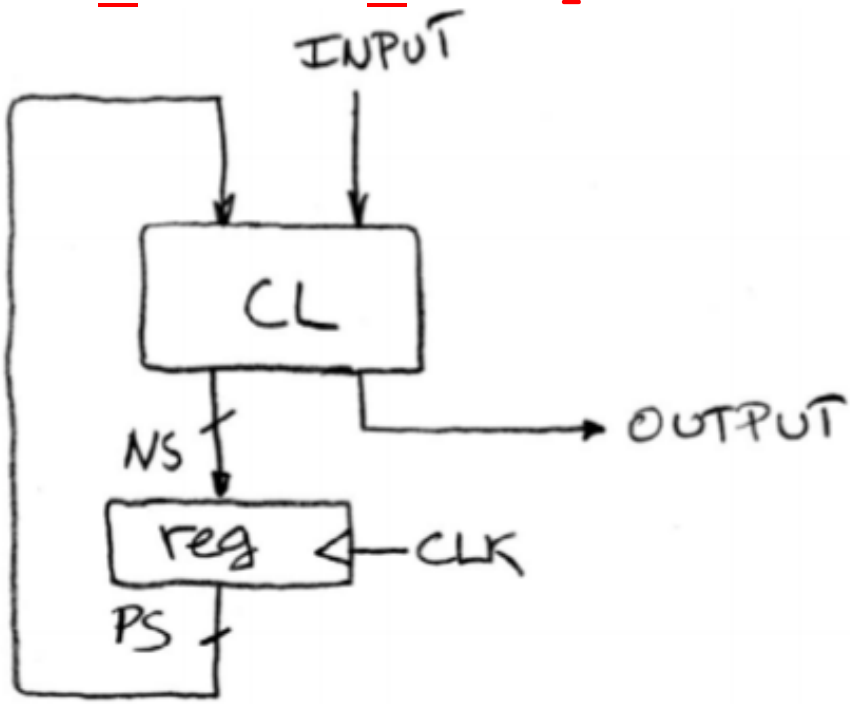
Synchronous Digital Systems

If we were to implement this circuit using our standard FSM model, what is the smallest clock period T we can drive our system with? Answer in terms of T_setup, T_hold, T_clk-to-q, and T_CL.



Synchronous Digital Systems

If we were to implement this circuit using our standard FSM model, what is the smallest clock period T we can drive our system with? Answer in terms of T_setup, T_hold, T_clk-to-q, and T_CL. `T = T_clk-to-q + T_CL + T_setup`



Warehouse Scale Computers

- Servers on a rack / rack part of cluster
- Process high volume of independent requests
- Computation partitioned across servers
- Web services rely on them

Warehouse Scale Computers

- Issues
  - Load balancing
  - Power usage vs compute load (1/2 peak power when idle)
  - Cope with failures gracefully (redundancy)
  - Elaborate hierarchy of memories

Warehouse Scale Computers

- Power Usage Efficiency
- PUE = Total Building Power / IT equipment power
- Where does power go?
  - Cooling
  - Power Distribution
  - Lighting
- 1.0 = Ideal maximum

MapReduce

- Application of WSC
- Scalable
  - Add more machines, increase performance
  - Strong: Speed increases with more processors
  - Weak: Speed increases with larger problem size

MapReduce

- Apply Map to user supplied key/value pairs
  - Data is sliced into "shards" and distributed
  - Emit intermediate key/value pairs
- Apply Reduce to all values that share the same key
  - Produces a set of merged output values
- Let MapReduce library handle details, focus on implementing Map and Reduce functions

MapReduce

- Implement Map and Reduce functions to calculate the number of unique words of certain lengths.
- Example input: "The quick brown fox jumped"
- Output: (3, 2) (5, 2) (6, 1)

```
Map (String title, String text):



Reduce (_____ key, list(____) values):
```

MapReduce

- Implement Map and Reduce functions to calculate the number of unique words of certain lengths.
- Example input: "The quick brown fox jumped"
- Output: (3, 2) (5, 2) (6, 1)

```
Map (String title, String text):



Reduce (Integer key, list(Integer) values):
```

MapReduce

- Implement Map and Reduce functions to calculate the number of unique words of certain lengths.
- Example input: "The quick brown fox jumped"
- Output: (3, 2) (5, 2) (6, 1)

```
Map (String title, String text):
    dict = {}
    for word in text:
        if !dict.get(word):
            emit(len(word), 1))
            dict.put(word)

Reduce (Integer key, list(Integer) values):
```

MapReduce

- Implement Map and Reduce functions to calculate the number of unique words of certain lengths.
- Example input: "The quick brown fox jumped"
- Output: (3, 2) (5, 2) (6, 1)

```
Map (String title, String text):
    dict = {}
    for word in text:
        if !dict.get(word):
            emit(len(word), 1))
            dict.put(word)

Reduce (Integer key, list(Integer) values):
    emit(key, sum(list))
```

MapReduce

- Calculate cost of customer's orders based on a list of transactions.
- Input: (CustomerID, ItemInfo{name, qty, price})

```
Map (Integer ID, ItemInfo info):



Reduce (_________ key, list(_________) values):
```

MapReduce

- Calculate cost of customer's orders based on a list of transactions.
- Input: (CustomerID, ItemInfo{name, qty, price})

```
Map (Integer ID, ItemInfo info):



Reduce (Integer key, list(Integer) values):
```

MapReduce

- Calculate cost of customer's orders based on a list of transactions.
- Input: (CustomerID, ItemInfo{name, qty, price})

```
Map (Integer ID, ItemInfo info):
    emit(ID, info.qty*info.price)

Reduce (Integer key, list(Integer) values):
```

MapReduce

- Calculate cost of customer's orders based on single transactions.
- Input: (CustomerID, ItemInfo{name, qty, price})

```
Map (Integer ID, ItemInfo info):
    emit(ID, info.qty*info.price)
```

```
Reduce (Integer key, list(Integer) values):
    emit(key, sum(values))
```