

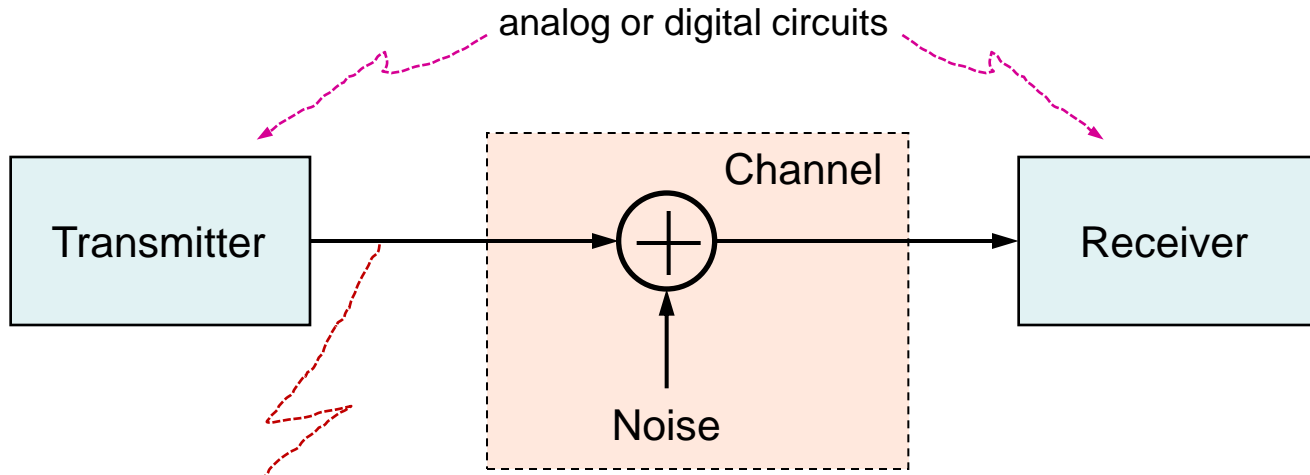
Introduction

by Erol Seke

For the course “[FPGA Structures for Digital Comm.](#)”



Electronic Communication

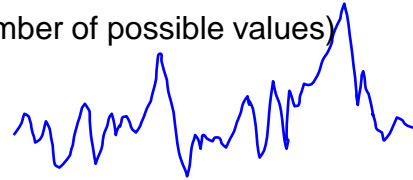


Transmitted Signal

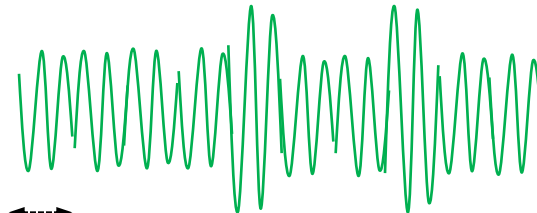
Analog Communication

Digital Communication

(infinite number of possible values)



Signal's all values are important at every point and cannot be completely repaired when damaged



T_s

Finite number of symbols represented by finite number of waveforms within T_s

Digital Communication

Advantages :

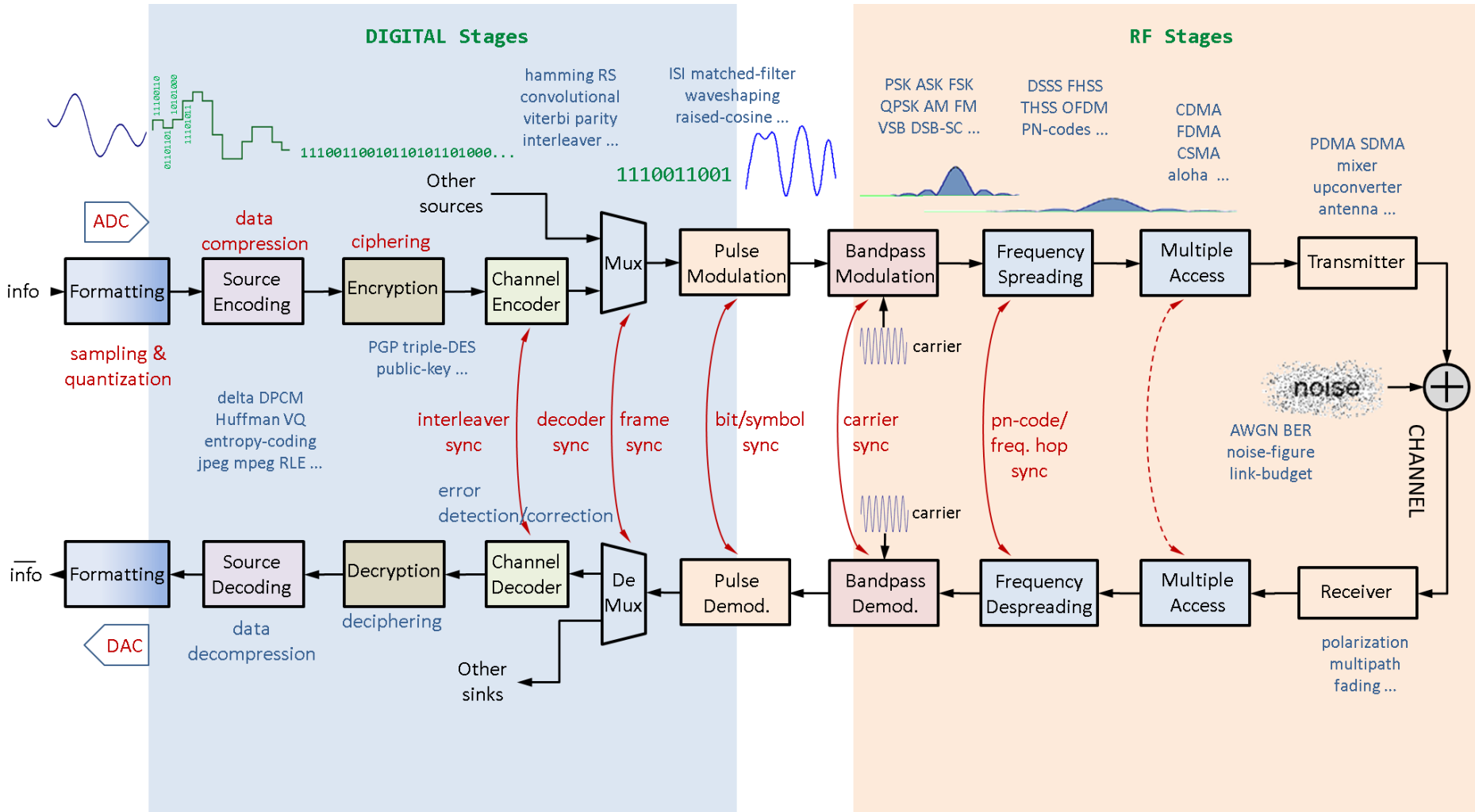
- Mathematical/Logical Processing on the data is possible
- Therefore : higher protection against noise
- More flexible when performed using reconfigurable / reprogrammable elements
- ?

Disadvantages :

- Complexity is higher
- Higher speed devices are required
- Analog signals need to be converted/deconverted using ADC/DAC
- ?

against analog communication

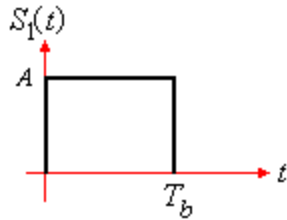
General Communication System



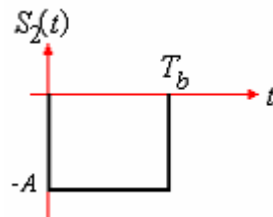
Simplest Waveforms

PAM : Pulse Amplitude Modulation (Amplitude of the signal carries the information)

Simplest PAM (binary antipodal signaling)



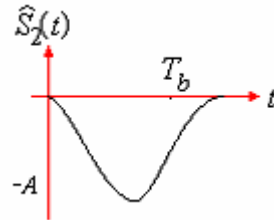
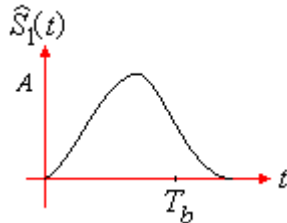
Binary 1



Binary 0

$T_b =$ Bit interval

Bit Rate = $1/T_b$



Spectrally more efficient waveform set
(Pulse shape determines the spectral characteristics)

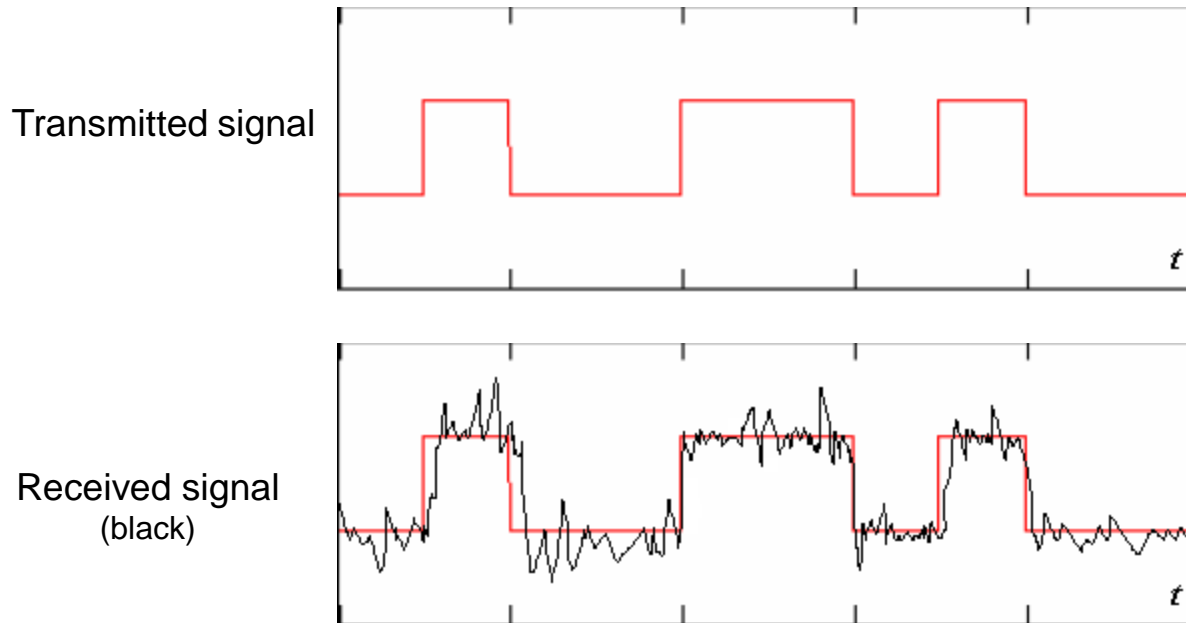
PAM signaling is generally used in *baseband channels* **Why?**

It would have been called ASK otherwise :)

Binary PAM

Example Data : 0100110100...

Transmitted signal is a sum of the corresponding waveforms at appropriate positions



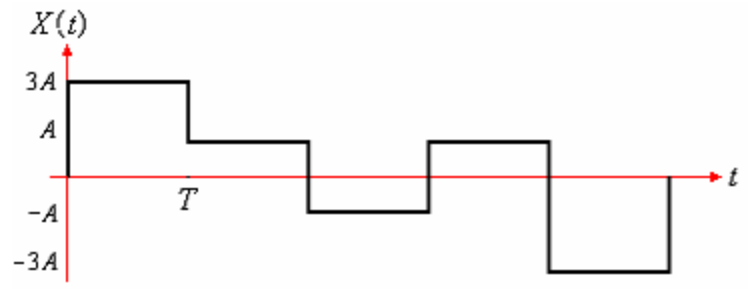
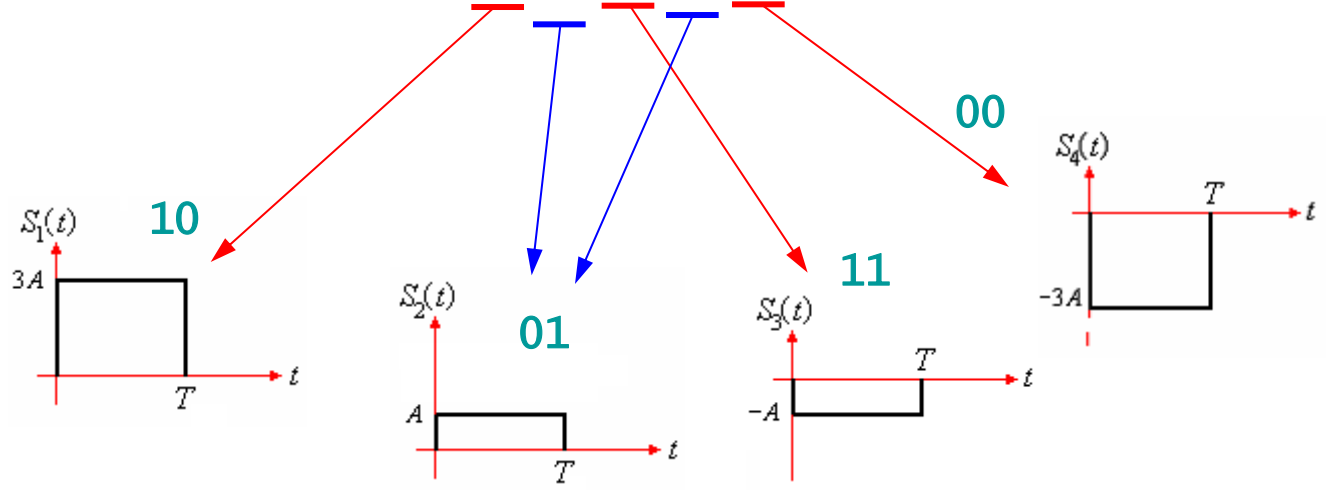
Problem of the receiver : Determine the 1-0 data sequence from the noisy signal

M-ary PAM

Instead of 2 levels (binary) M levels (M-ary) can be used. M is selected so that $M = 2^k$

Example : $k=2$

... 1001110100 ... Symbols are 2 bits

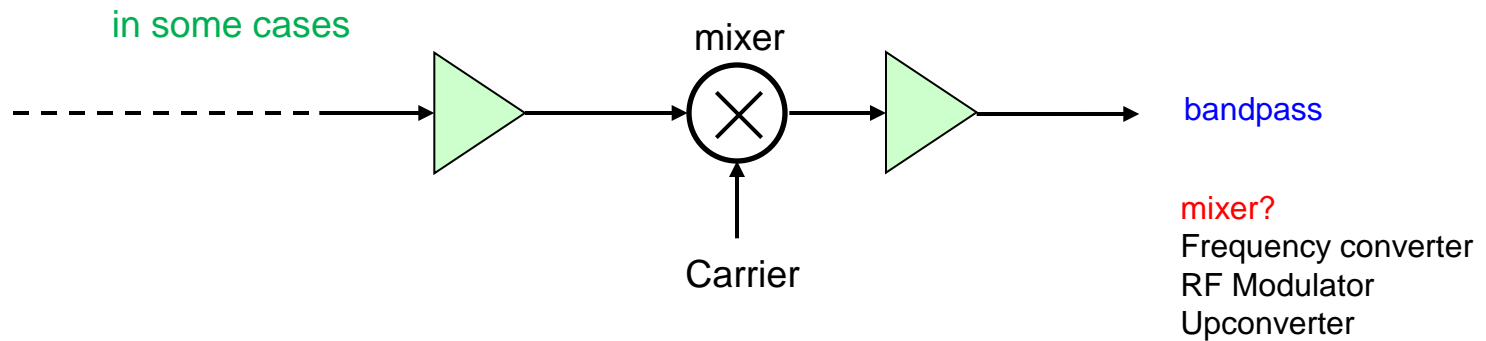
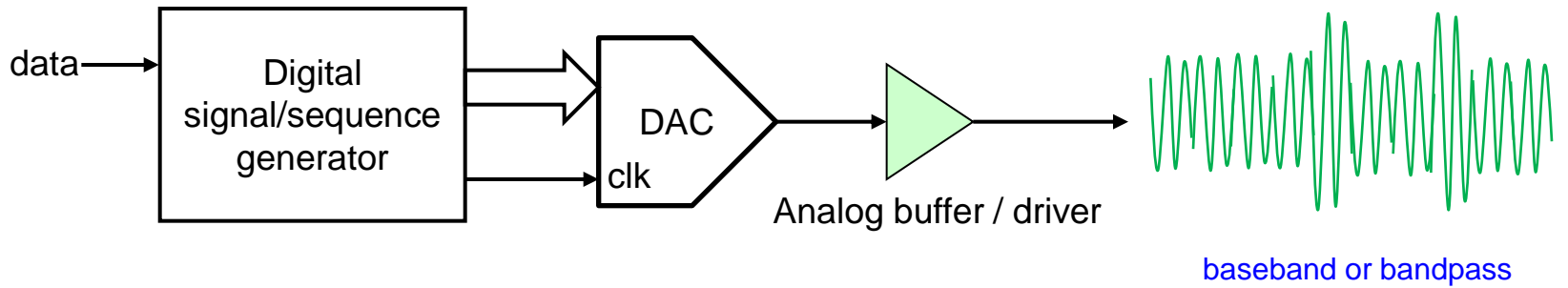
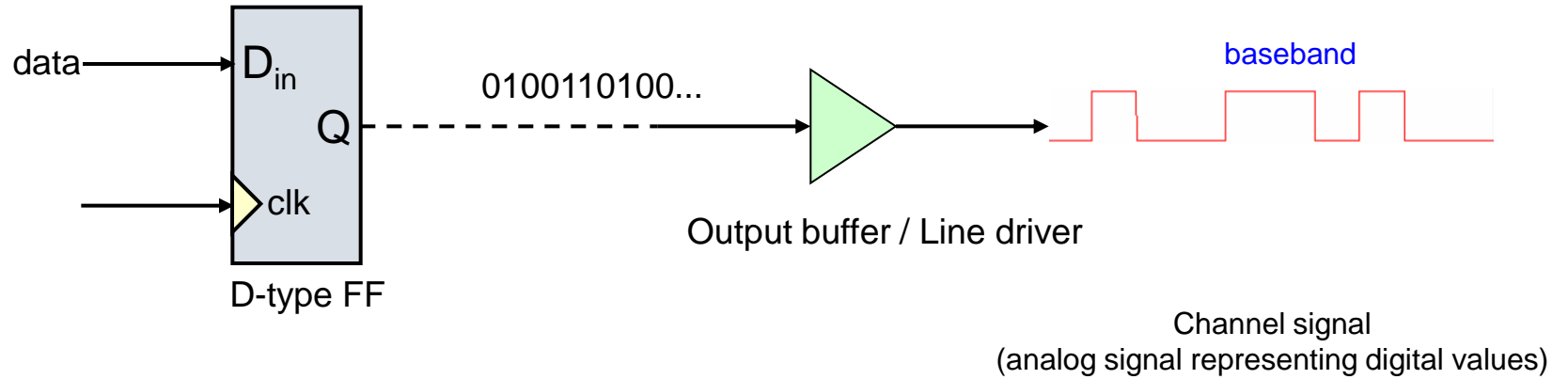


Symbol interval = 2 Bit interval
since 1 change transfers 2 bits

For M-ary case
 $T = kT_b$

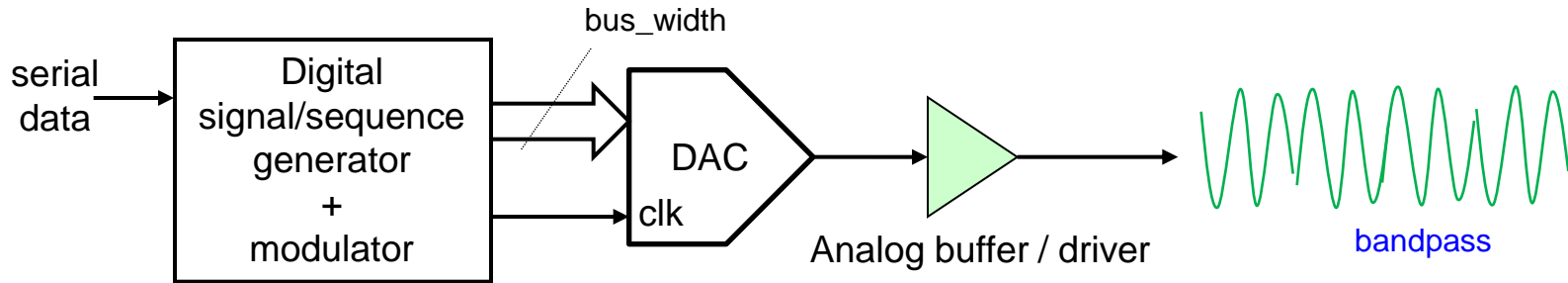
2 bits are transmitted at each T now, but the receiver's job is more complicated

Signal Generation for Driving Channel



Example

1 MHz BPSK signal is to be generated without using frequency upconversion.



Let the design requirements be

Carrier frequency (center freq. of BPSK signal) = 1 MHz

Samples per carrier period ≥ 10

Carrier periods per bit ≥ 5

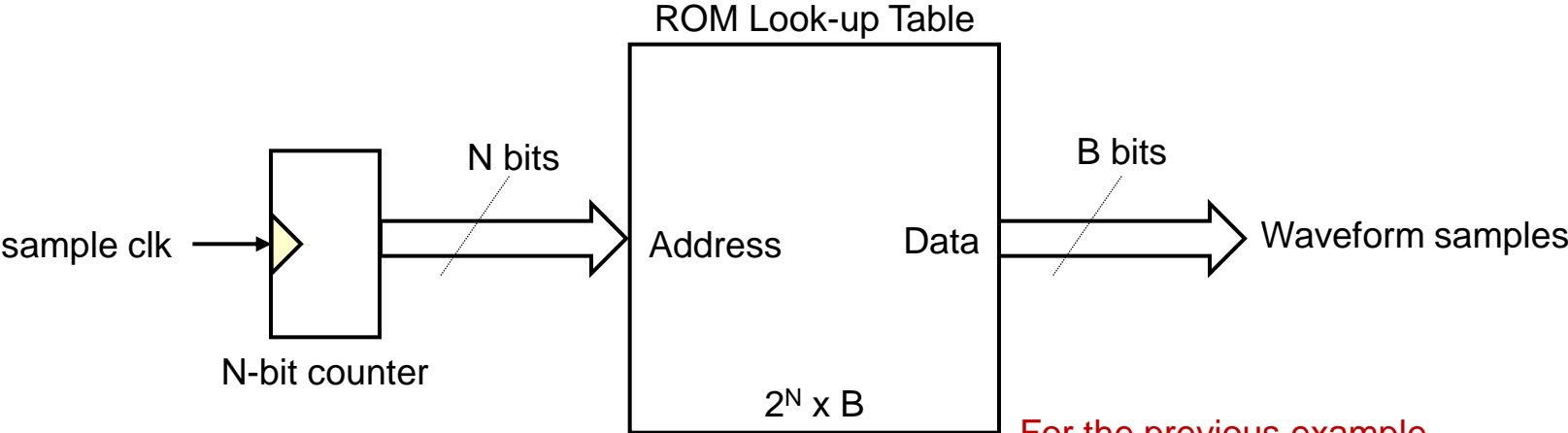
Calculations

Sample rate = $10 \times 1 \text{ MHz} = 10 \text{ Msps}$ (min for DAC and generator)

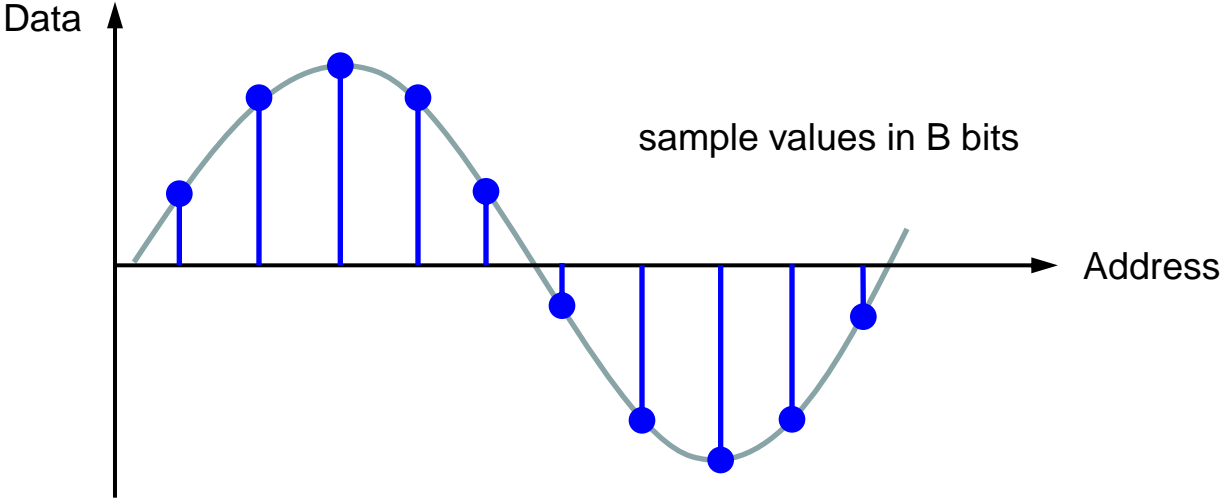
Serial Data rate = $1 \text{ MHz} / \text{samples_per_period} / \text{carrier_periods_per_bit} = 1 \text{ MHz} / (10 \times 5) = 20 \text{ kbps}$ (max)

Corollary : You need high speed digital equipment even for low data rates.
Constraints are tighter at the receiver side.

Primitive (not optimized) Sequence Generator

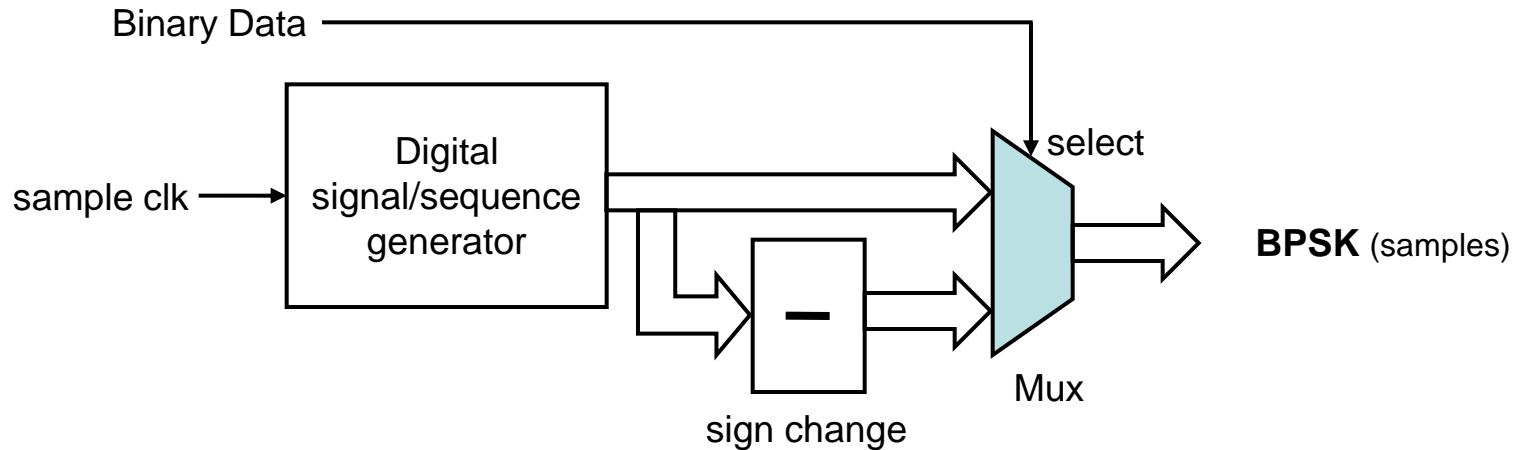


For the previous example
Samples per carrier period = 2^N



since 10 is not an integer power of 2, a 16 x B ROM is used (remaining 6 locations are not used) and the counter should count from 0 to 9 for the previous example (rethink efficiency/design)

BPSK on Sinusoidal Sequence



All these circuits, except DAC/ADC, can be implemented by digital circuits.

FPGAs are full of digital circuit primitives. Therefore these circuits can be implemented on FPGAs.

Advantages of FPGAs ? :

- reconfigurable
- small, power efficient
- short development time

What are FPGAs ?

Field Programmable Gate Array :

We have a bunch of digital circuit primitives with ~~user~~ programmable connections
designer

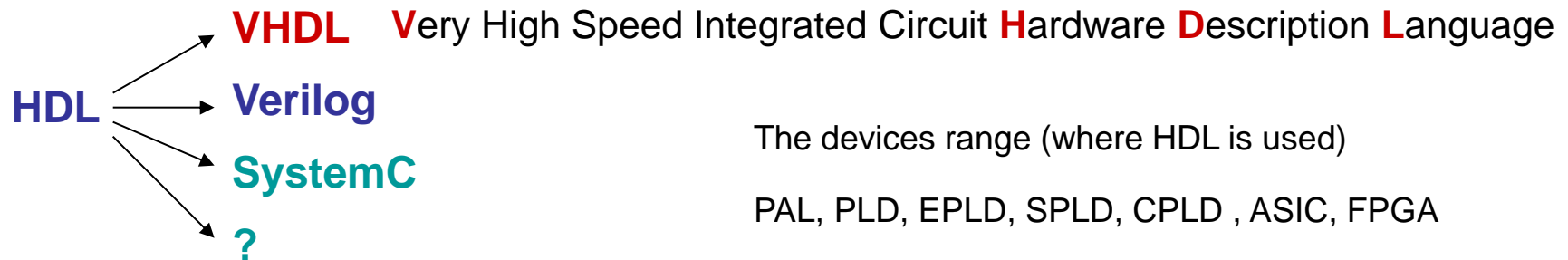
There are several ways to design digital circuits on FPGAs

One option is to use a **HDL**

Hardware Description Language :

We describe the circuits in plain text just like a programming language.

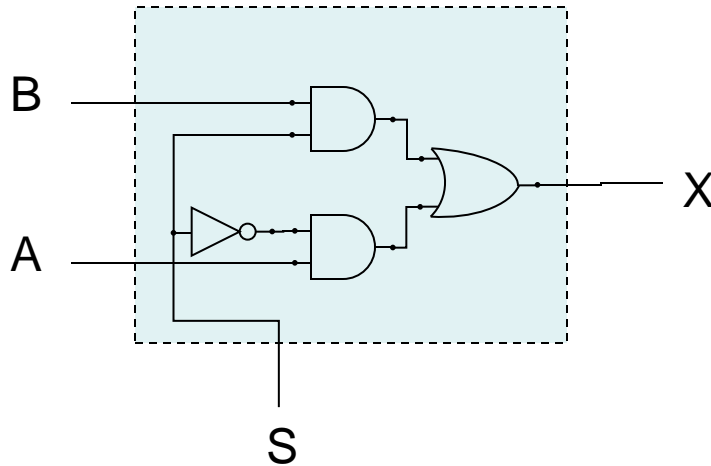
But it is not a programming language! it is a description language.



In this course, we will be using **VHDL**

Start with a Simple Digital Example

Consider the following combinatorial digital circuit and truth table



S	A	B	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

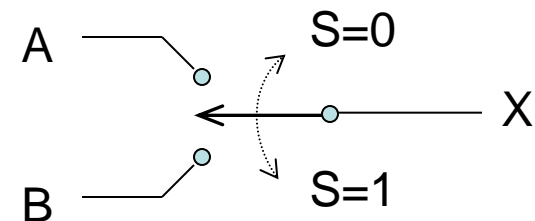
We can describe the function as

$X = A$ when $S=0$, B when $S=1$

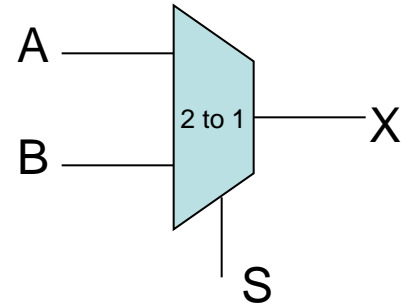
or

$X = (A \text{ and not } S) \text{ or } (B \text{ and } S)$

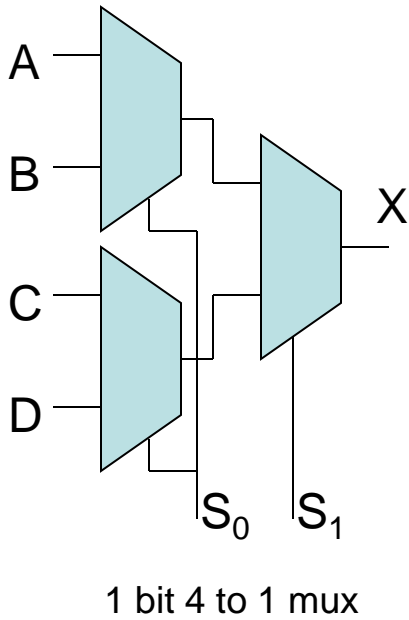
or, with a switch analogy



It is a 1-bit 2-to-1 multiplexer as we know

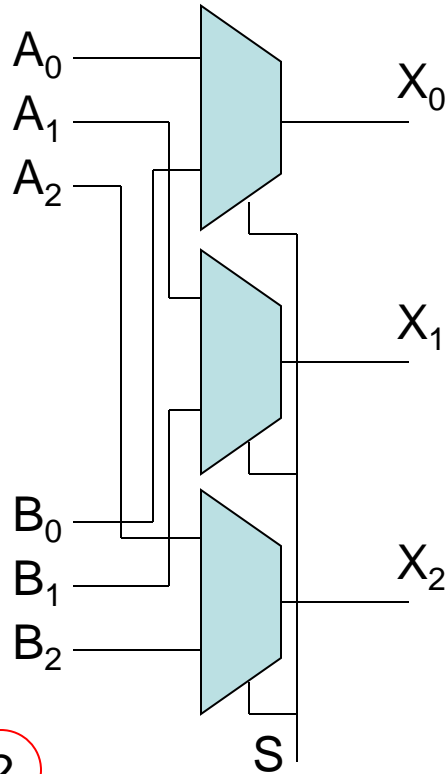


We can make other multiplexers using this basic mux.



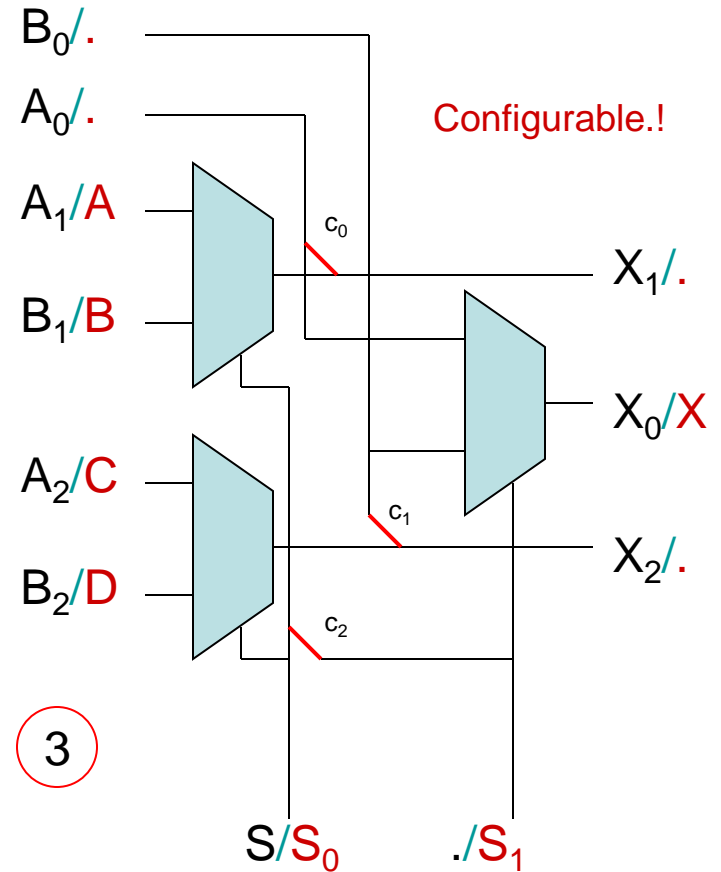
1 bit 4 to 1 mux

1



3 bits 2 to 1 mux

2



3

Programmable / Configurable devices basically work just like that

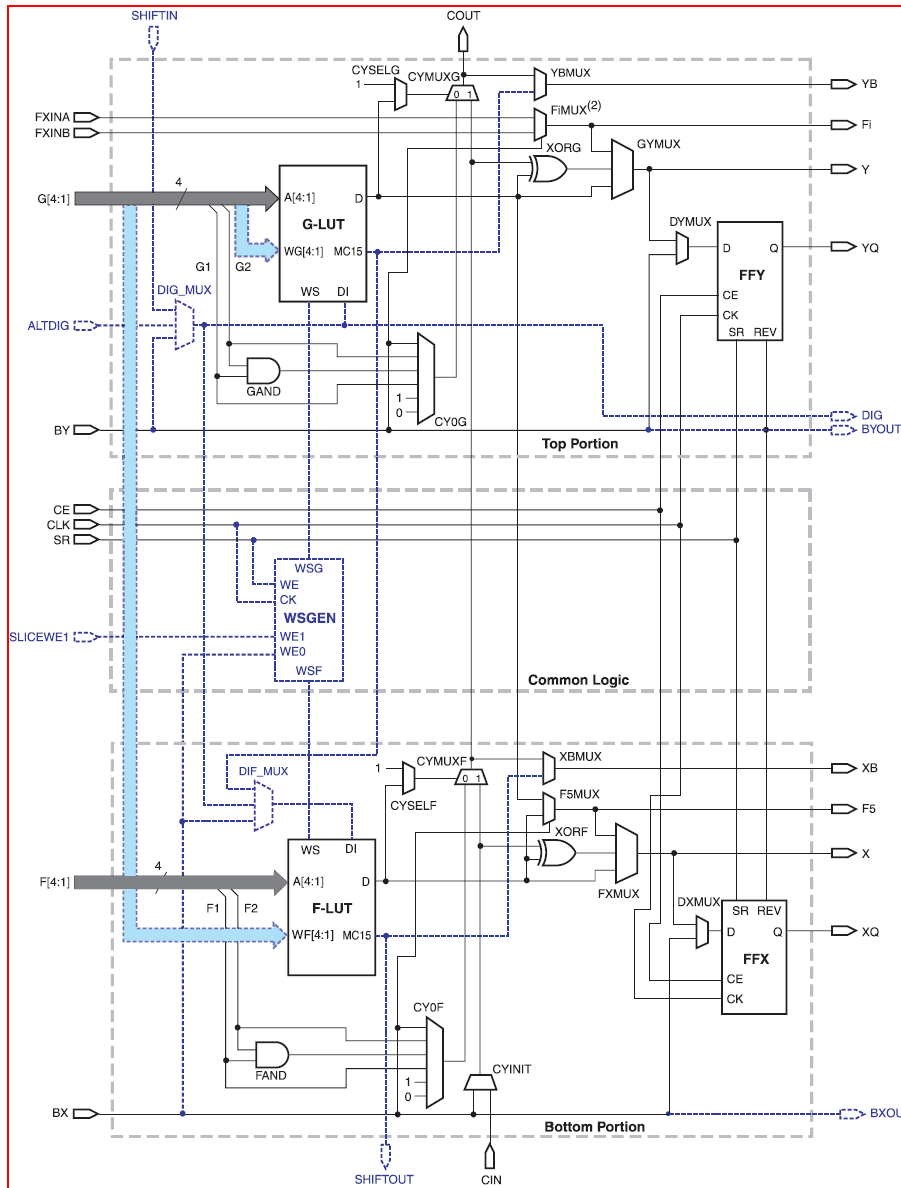
In a device, we have a finite number of

1. Flip-Flops, Registers
2. RAMs
3. Look Up Tables (LUTs)
4. Gates
5. Arithmetic Units
6. MUXs
7. Other (clock managers, buses, I/O blocks etc)

that we can interconnect them as we wish and design the digital circuit needed

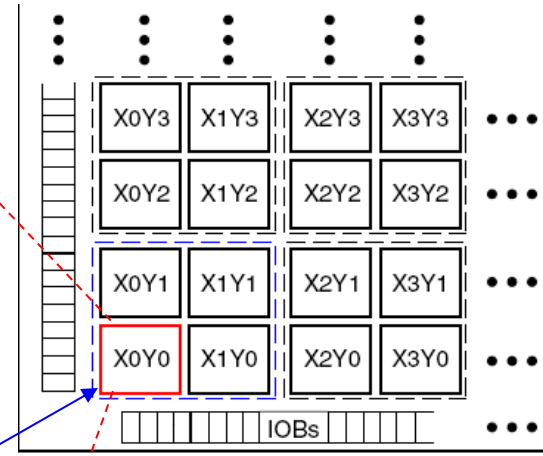
or we can use a **HDL** and let a compiler/synthesizer do the design and optimization for the resource/performance balance.

Example (Xilinx-Spartan3E structure)



Slice

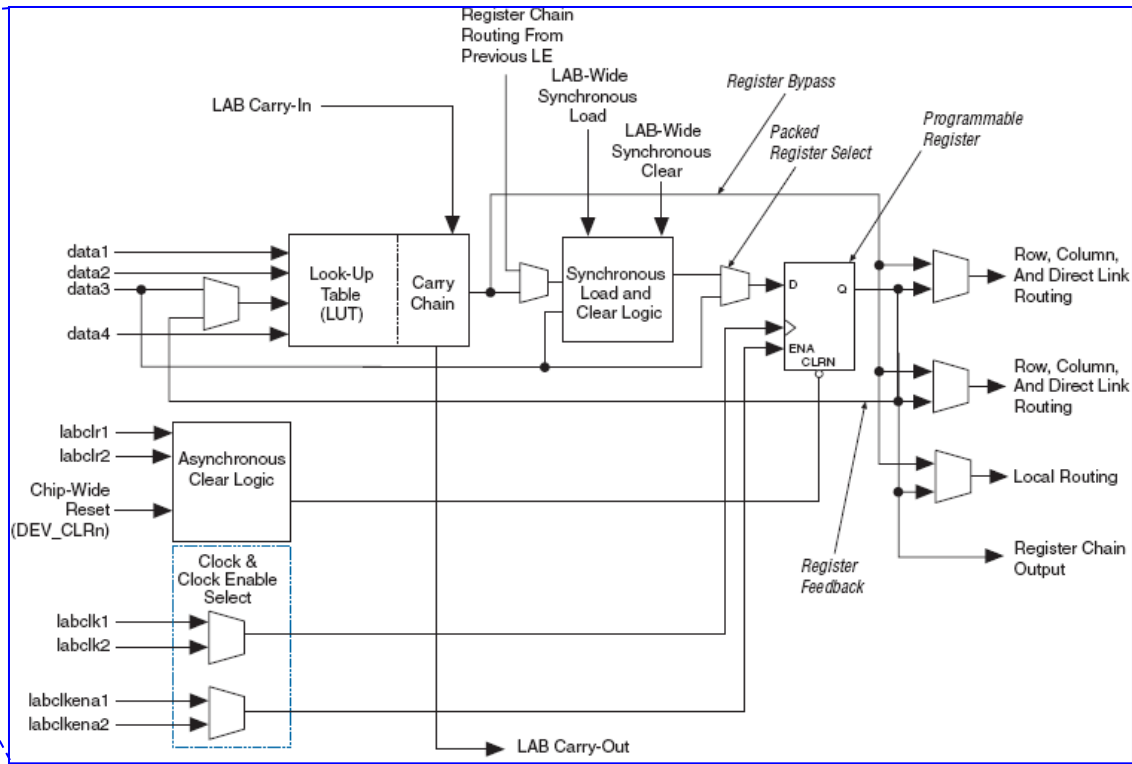
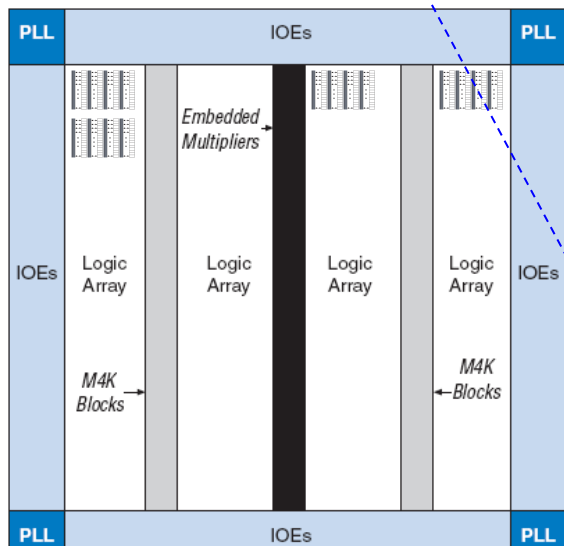
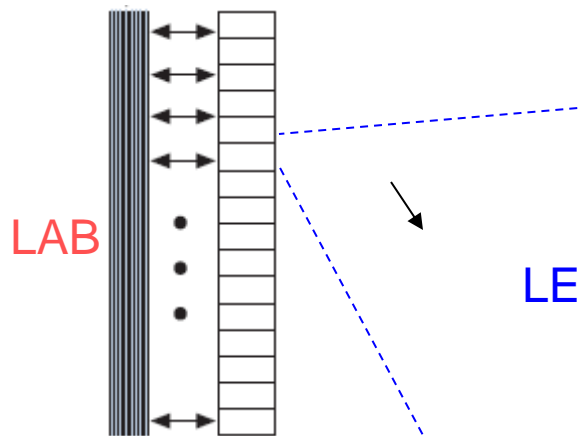
CLB



Device	RAM Blocks	#bits
XC3S100E	4	73,728
XC3S250E	12	221,184
XC3S500E	20	368,640
XC3S1200E	28	516,096
XC3S1600E	36	663,552

Device	CLB Total	Slices	LUTs / Flip-Flops	Equivalent Logic Cells	RAM16 / SRL16	Distributed RAM Bits
XC3S100E	240	960	1,920	2,160	960	15,360
XC3S250E	612	2,448	4,896	5,508	2,448	39,168
XC3S500E	1,164	4,656	9,312	10,476	4,656	74,496
XC3S1200E	2,168	8,672	17,344	19,512	8,672	138,752
XC3S1600E	3,688	14,752	29,504	33,192	14,752	236,032

Example (Altera-Cyclone II structure)



Feature	EP2C5	EP2C8	EP2C15	EP2C20	EP2C35	EP2C50	EP2C70
LEs	4,608	8,256	14,448	18,752	33,216	50,528	68,416
M4K RAM blocks (4 Kbits plus 512 parity bits)	26	36	52	52	105	129	250
Total RAM bits	119,808	165,888	239,616	239,616	483,840	594,432	1,152,000
Embedded multipliers	13	18	26	26	35	86	150
PLLs	2	2	4	4	4	4	4

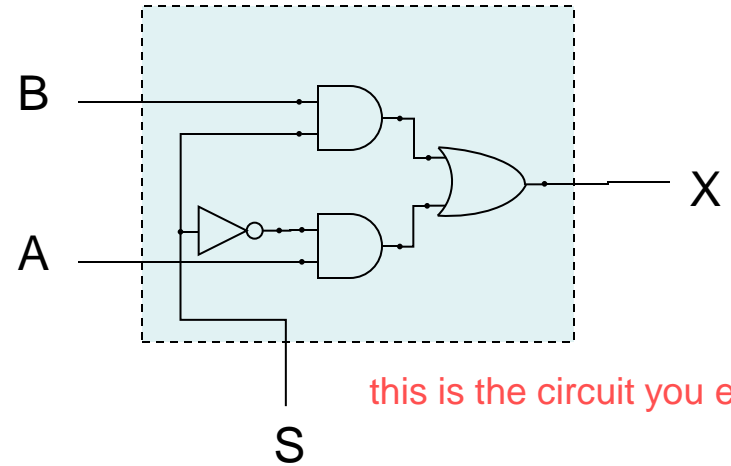
Device	M4K Blocks	Total RAM Bits
EP2C5	26	119,808
EP2C8	36	165,888
EP2C15	52	239,616
EP2C20	52	239,616
EP2C35	105	483,840
EP2C50	129	594,432
EP2C70	250	1,152,000

It may not be what it looks like

Combinatorial functions are usually implemented with look-up tables

this is the function you want

$$X = (A \text{ and not } S) \text{ or } (B \text{ and } S)$$

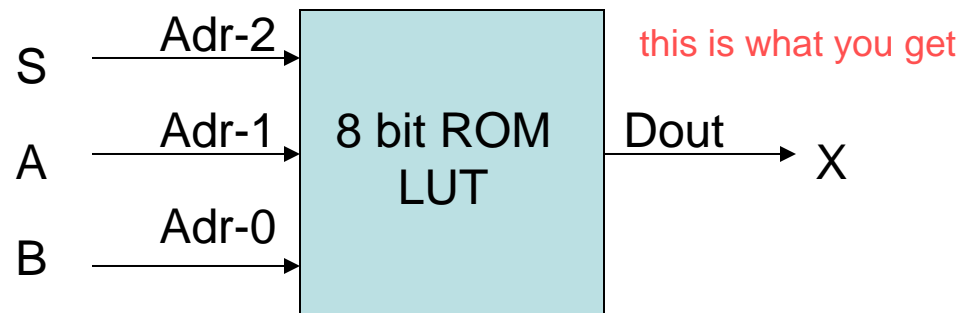


this is the circuit you expect

this is the truth table implemented

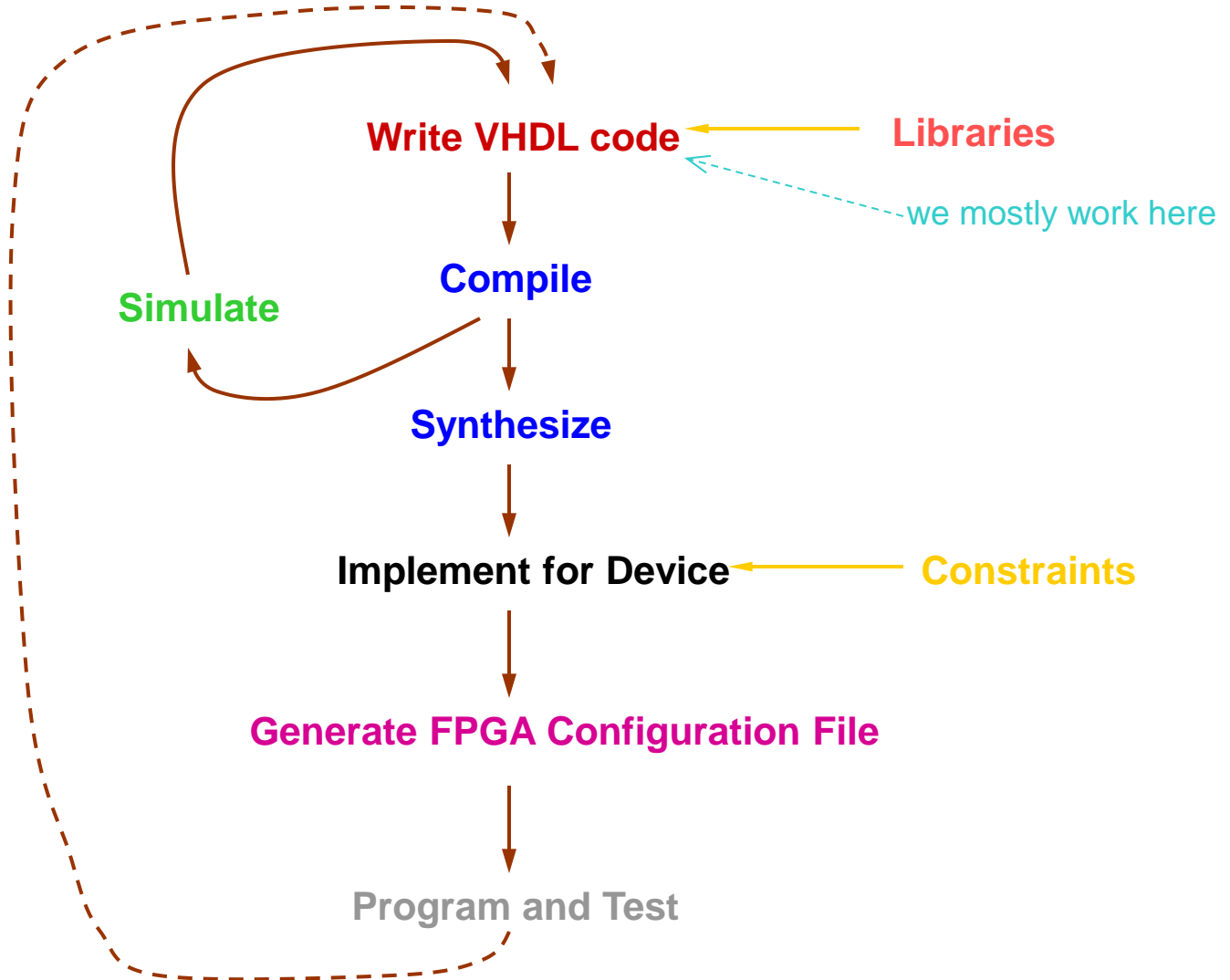
S	A	B	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Grouping the first four rows as 'A' and the last four rows as 'B'.



So, no matter how complex your function is it is as simple as a look-up rom with 2^N addresses with N being the number of variables in the function.

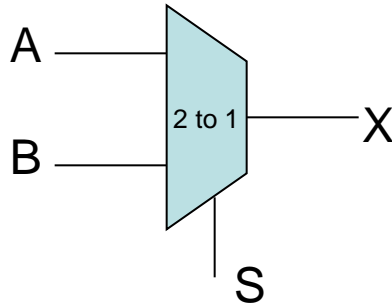
Steps of VHDL Design Flow



Since we have Spartan3E kits in the lab, we will be referring Xilinx ISE tool from now on, remembering that other vendors/manufacturers provide similar tools too. Tools for the steps mentioned here are mostly device/vendor specific.

Hello World

Consider the MUX



≡

```
entity mux2to1 is
  Port ( A : in  BIT;
         B : in  BIT;
         S : in  BIT;
         X : out BIT);
end mux2to1;
```

BIT type signals can assume one of two values : 0 or 1

How about the behavior of the mux box?

remember

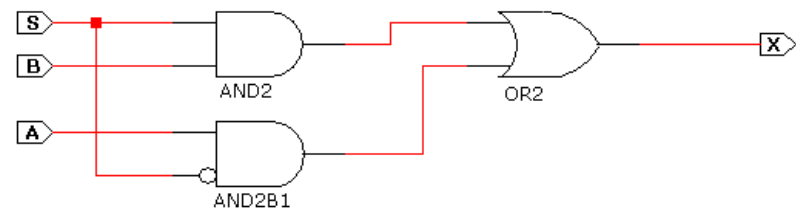
$X = (A \text{ and not } S) \text{ or } (B \text{ and } S)$

```
architecture Behavioral of mux2to1 is
begin
```

$X <= (A \text{ and not } S) \text{ or } (B \text{ and } S);$

```
end Behavioral;
```

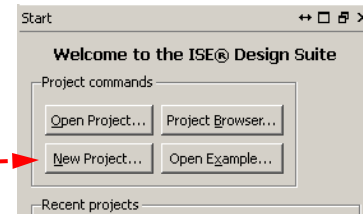
after synthesizing we get



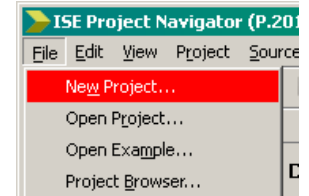
How we do it using ISE tool (ver 14.7)

1 Start ISE

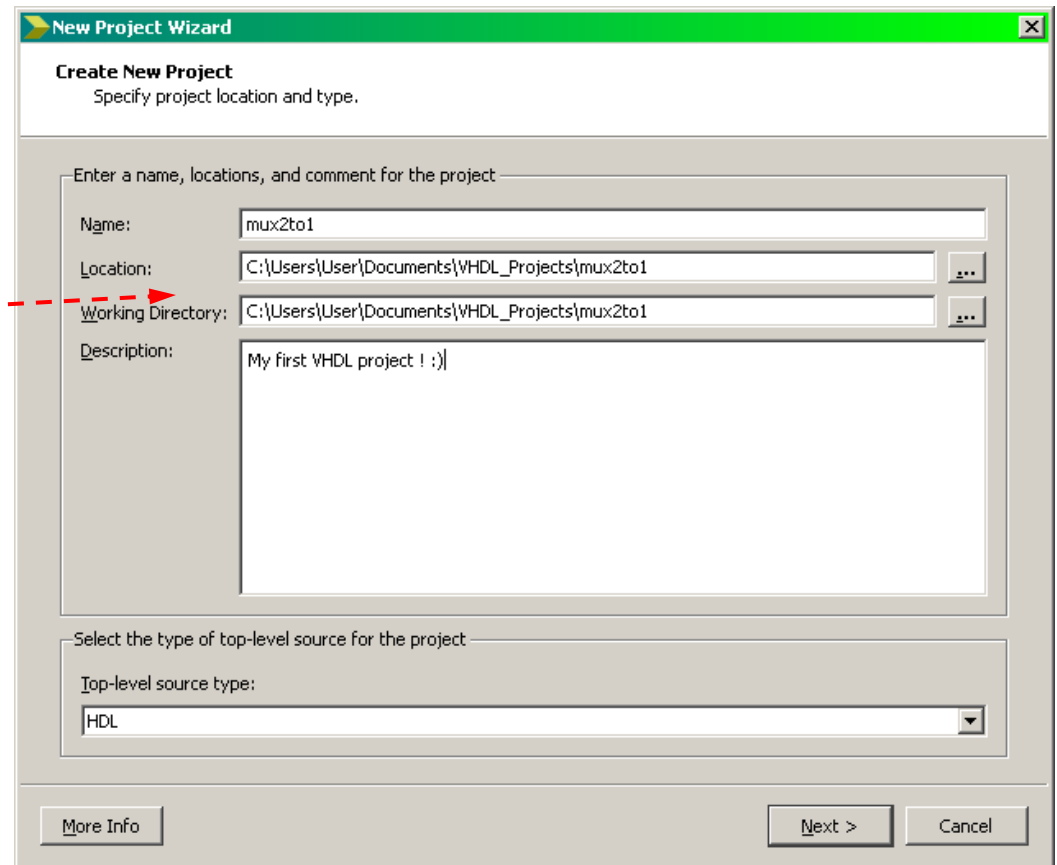
2 Create a new project named mux2to1



OR



Select your VHDL projects folder.
It will come up automatically
everytime you create a new project.



An ISE project file is a text file
consisting of names/references of
*.VHD source files, constraints files,
implementation specifiers etc.

Click Next

3

Select your device

If you cannot find your board in this list, select '*none specified*' and just select your FPGA chip

These should be as shown here

Property Name	Value
Evaluation Development Board	Spartan-3E Starter Board
Product Category	All
Family	Spartan3E
Device	XC35500E
Package	FG320
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

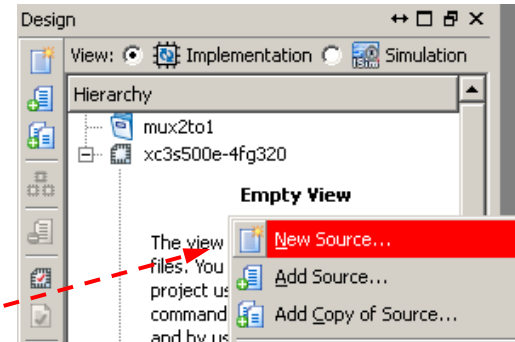
Click Next

and click '*Finish*' on the Project Summary dialog box

4 Create a source file

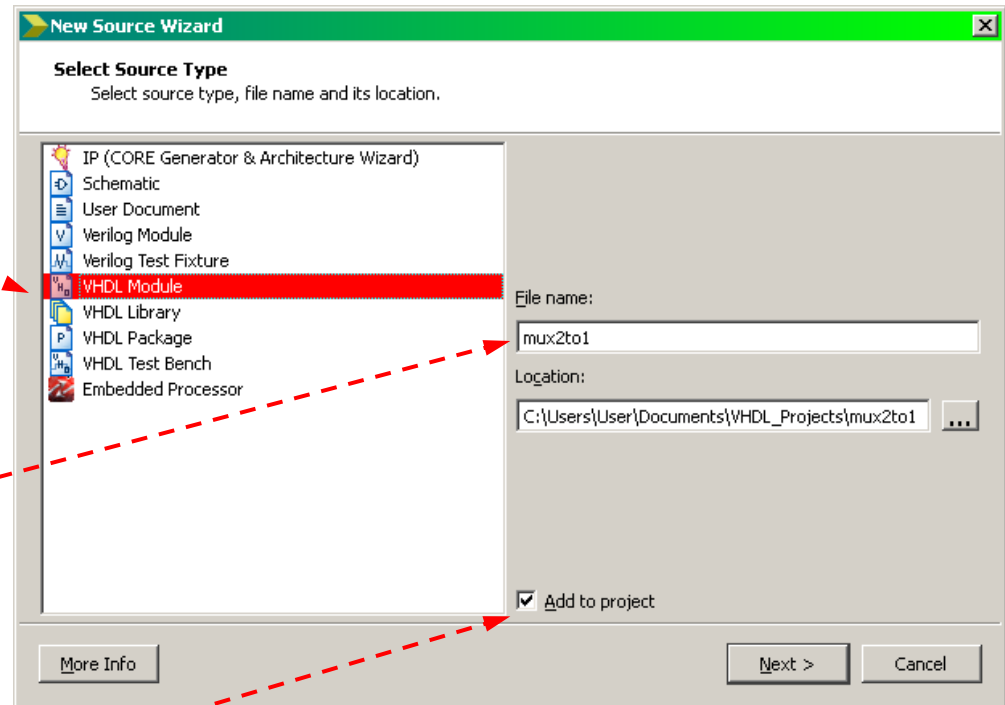
Source file is a text file with *.vhd extension where you put your VHDL code

Right-Click on an empty space in the 'Hierarchy' pane of the 'Implementation' view of the 'Design' tab. Click 'New Source...'



Since we are creating a VHDL source file, we should select 'VHDL Module' here.

Enter new source file name. Any valid file name is ok, but be wise.



this should be selected, otherwise you need to add the file to the project later.

Click *Next*

5 Define ports of the entity

The convention is to create a source file for each entity (circuit). Here you may define inputs/outputs of this entity.

Since source files are text files, many coders skip this step and insert/edit the port description by hand.

Default signal type is STD_LOGIC.

Port Name	Direction	Bus	MSB	LSB
A	in	<input type="checkbox"/>		
B	in	<input type="checkbox"/>		
S	in	<input type="checkbox"/>		
X	out	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		

```
entity mux2to1 is
    Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          S : in  STD_LOGIC;
          X : out STD_LOGIC);
end mux2to1;
```

Click *Next*

and click '*Finish*' on the Summary dialog box

We now have a source file editor window with entity description, some comments and library definitions and an empty architecture section. Architecture section is where you describe your circuit's behaviour.

6 Edit/Insert VHDL Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux2to1 is
    Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          S : in  STD_LOGIC;
          X : out STD_LOGIC);
end mux2to1;

architecture Behavioral of mux2to1 is

begin

Insert logical expressions here - - - - -> X <= (A and not S) or (B and S)
(between begin and end keywords of
Architecture section)

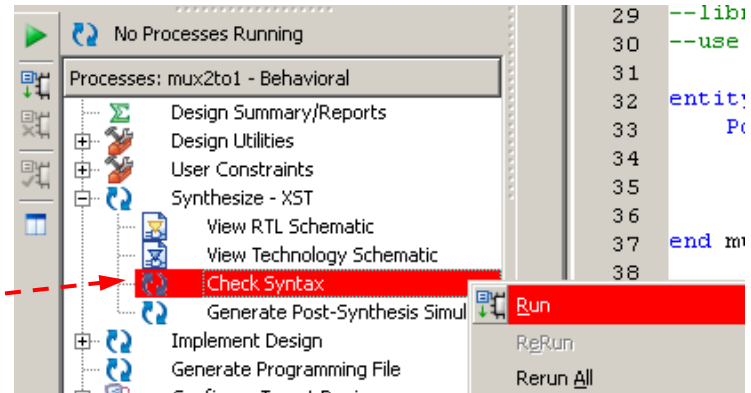
end Behavioral;
```

Click *Save icon*

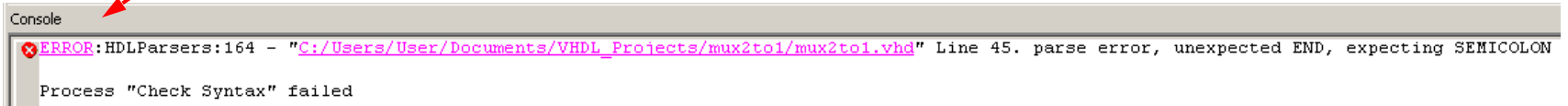
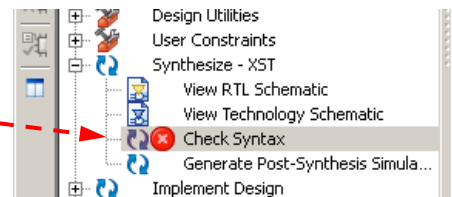
When saving, automatic **syntax check** is performed. Watch *Console* for error messages

7 Syntax Check

You can also check syntax by right clicking on the *Check Syntax* item in *Design Tab* and selecting *Run*, or double click on *Check Syntax*

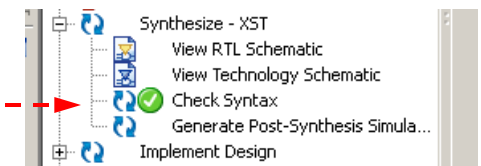


If there is an error you will see it on the *Check Syntax* item and in the *Console*.



This time, the syntax error is caused by a missing ; in `X <= (A and not S) or (B and S)` ;

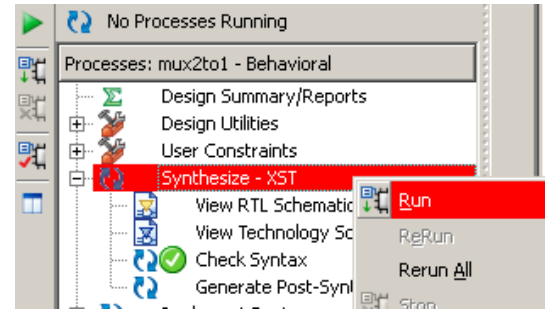
Do corrections and repeat *Syntax Check* until you see the green syntax validation checkmark



8

Synthesize

Synthesizing means that your code is realizable by logic components.
(but it does not mean that it is physically realizable within your device (FPGA) and VHDL rules)



You should see the green checkmark on *Synthesize* item too after synthesizing

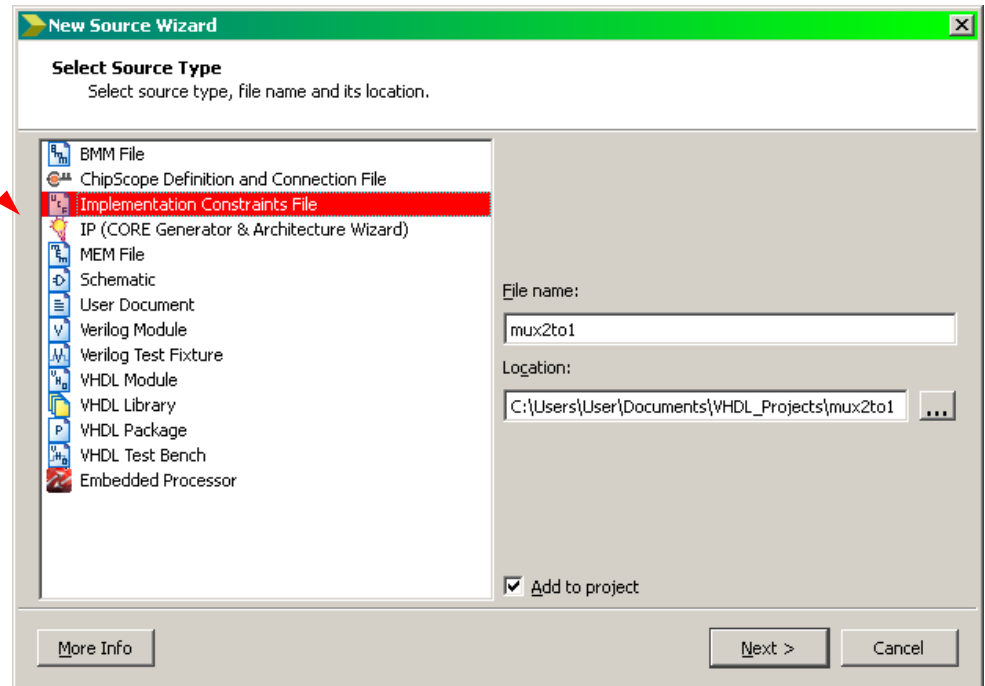
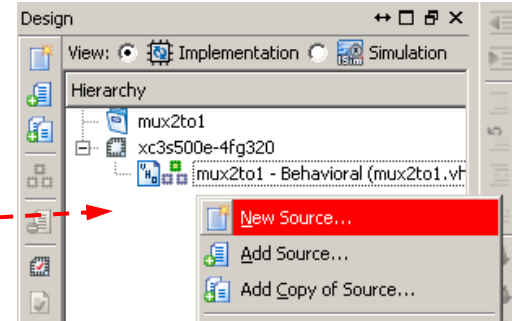
Now we need to implement a physical circuit for our selected device from this workable circuit description

It is imperative to define actual input output pins for a correct implementation as our design is a complete circuit and we need to test it by applying actual signals to the inputs and monitoring the outputs.

Therefore, we need to tell "which signal goes to which pin of the device" before this step. We do this by creating a constraint file.

9 Pin Connections

Create a new source as done before but this time select 'Implementation Constraints File' on the dialog box



Click Next

and click 'Finish' on the Summary dialog box

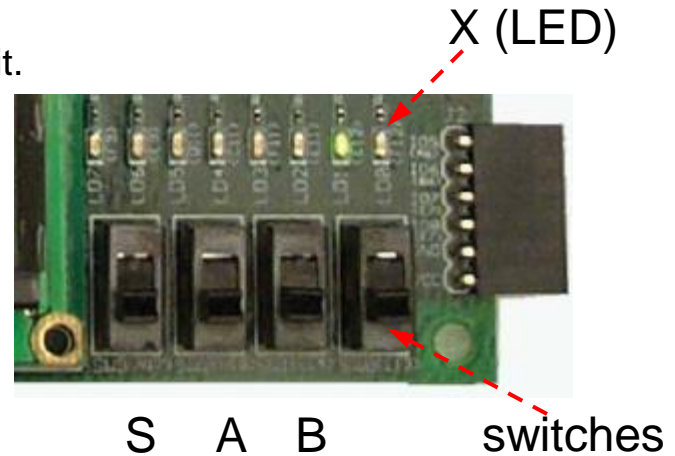
we will see a new editor window named as mux2to1.ucf (User Constraint File)

User Constraint File is a text file used for describing various constraints. There is a complete book on the possible contents of this file. This time we are just interested in pin connections.

Enter the following lines in the window and save it.

```
NET "A" LOC = "H18" ;  
NET "B" LOC = "L14" ;  
NET "S" LOC = "N17" ;  
NET "X" LOC = "F12" ;
```

It tells the implementor to connect the I/Os of our multiplexer to the physical switches and LEDs on our Spartan 3E Evaluation Board.



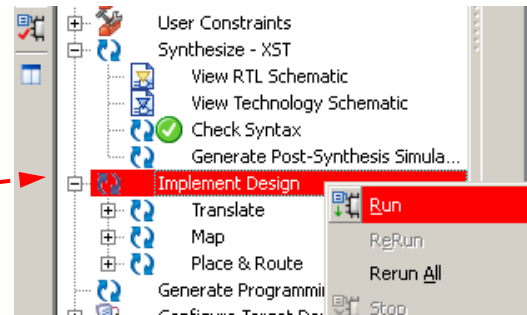
For example: A is the signal name, H18 is the pin number of the FPGA which is physically connected to the second switch on the board

Note : Instead of editing UCF as described above, you may also enter the following into the declaration part of the architecture section of the VHDL file. Differences will be mentioned later.

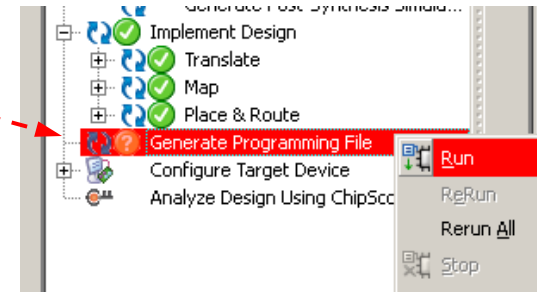
```
attribute LOC : string;  
attribute LOC of "A" : signal is "H18";  
attribute LOC of "B" : signal is "L14";  
attribute LOC of "S" : signal is "N17";  
attribute LOC of "X" : signal is "F12";
```

10 Implement Design

Now we can implement the design



and create the programming file



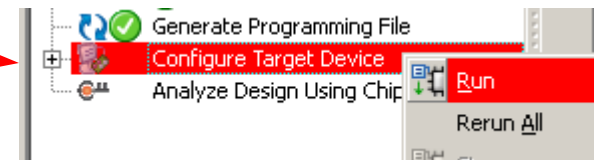
Programming File is a binary file with *.BIT extension.

This file will be loaded onto FPGA through FPGA's programming pins.

Our Spartan 3E Starter Board has a USB programming feature through which this file can be sent.

For this purpose, we will be using IMPACT program which can be initiated by *Configure Target Device* item.

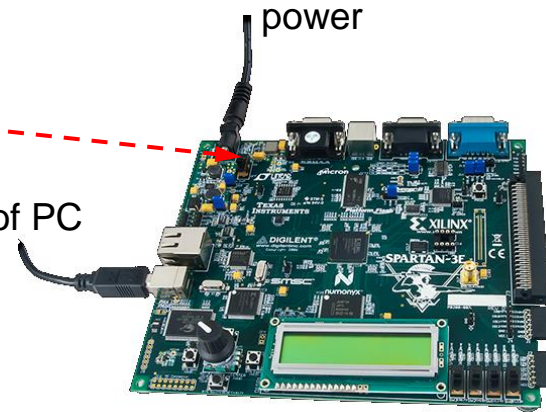
(IMPACT can also be started externally)



11 Connect your board to your PC using its USB cable

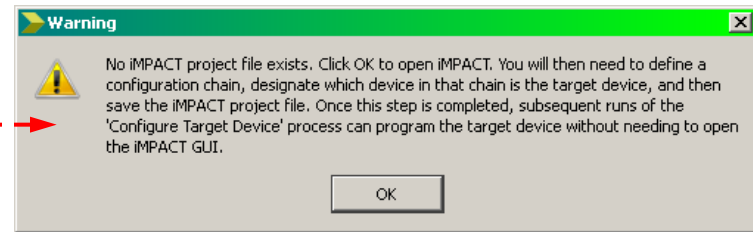
Turn on the board
and wait for the cable drivers to install

to USB port of PC

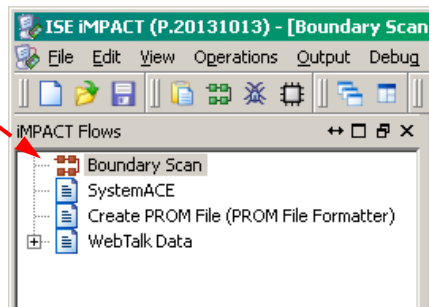


12 Putting Your Design Into FPGA

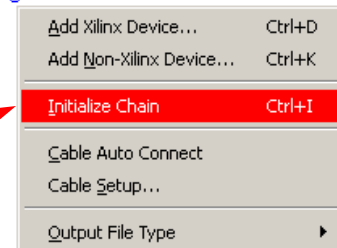
This warning is OK to
dismiss



Double Click on *Boundary Scan*



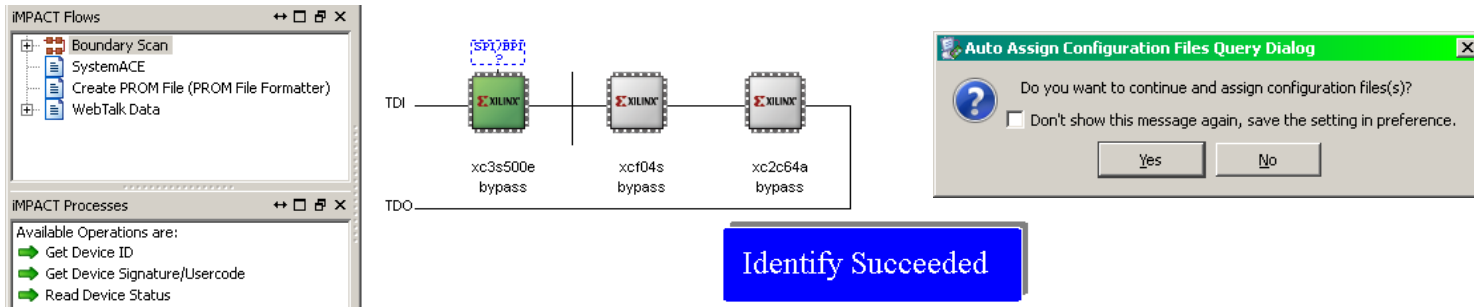
Right click to Add Device or Initialize JTAG chain



Right Click on the blank window and select *Initialize Chain* to search for devices on the board

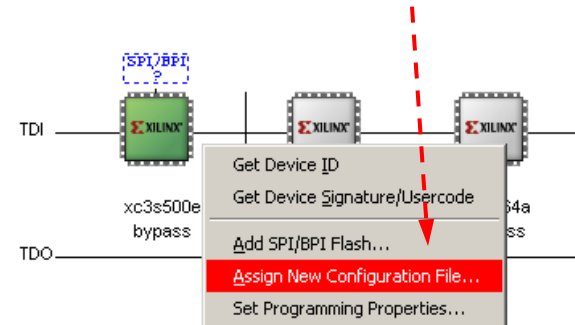
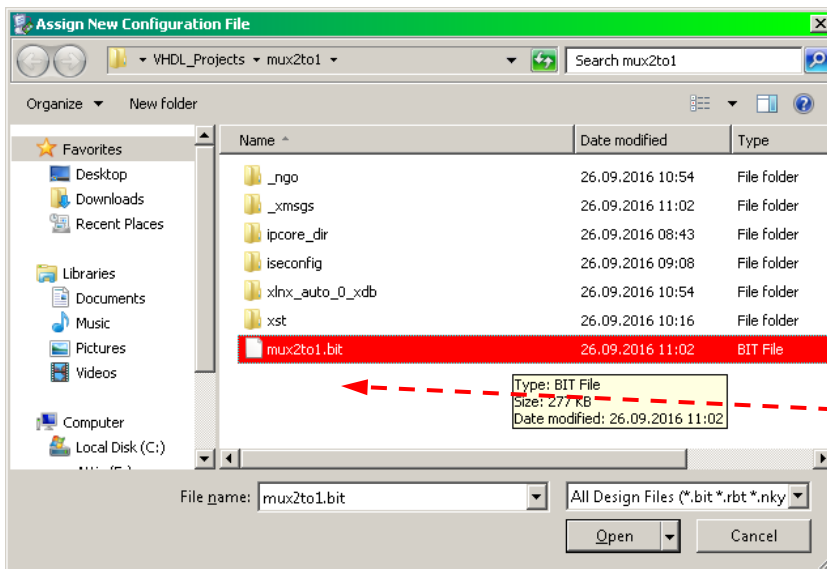
We should see the devices on the board in a chain configuration.

We also see a warning message about the configuration file(s). This time we will assign the configuration file manually, therefore, dismiss this dialog box and the next one.



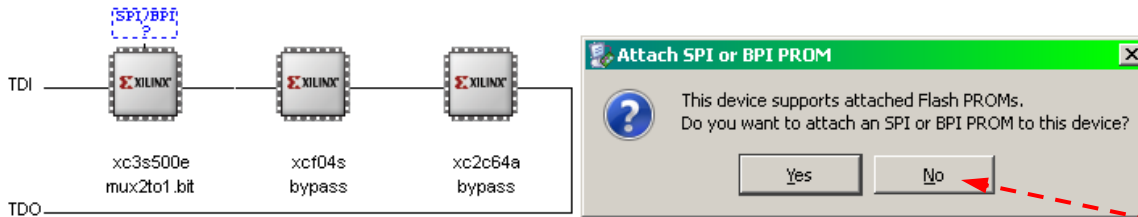
For the Spartan 3E Starter Board, there should be 3 programmable devices in the chain. The first one (xc3s500e) is the FPGA device and the one we are to program.

Click on xc3s500e to select device. Right click and select *Assign New Configuration File...*



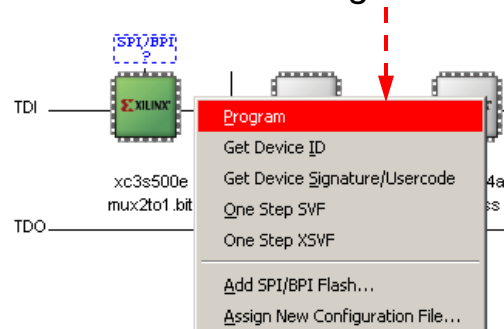
Find and select mux2to1.bit file and click *Open*

We should see the selected file name under the device name



Dismiss the dialog box about attachment of the SPI / PROM devices by clicking *No*

Select xc3s500e again, right click on the device and select *Program*



We should see the Program Succeeded message in the window

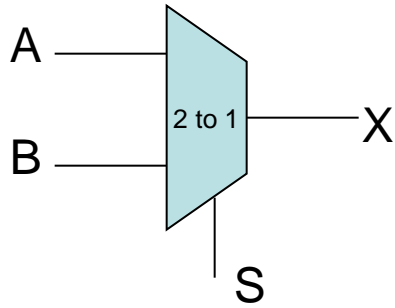
Program Succeeded

We can now test our multiplexer using switches and observing the LED

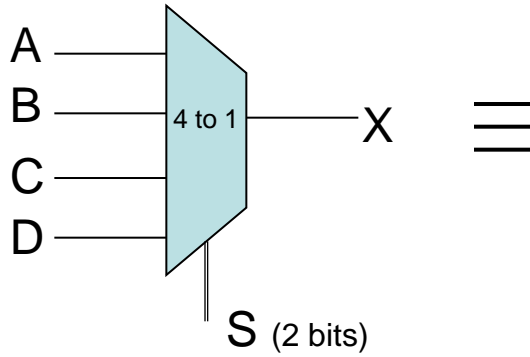
warning : Please **try not to** load files for xcf04s and xc2c64a devices and program them. This will destroy their original content and disables us to use simple test feature at the power up

Reusability

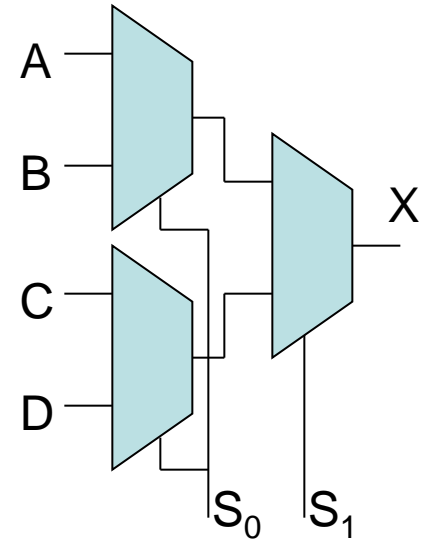
We have this



and want to have this



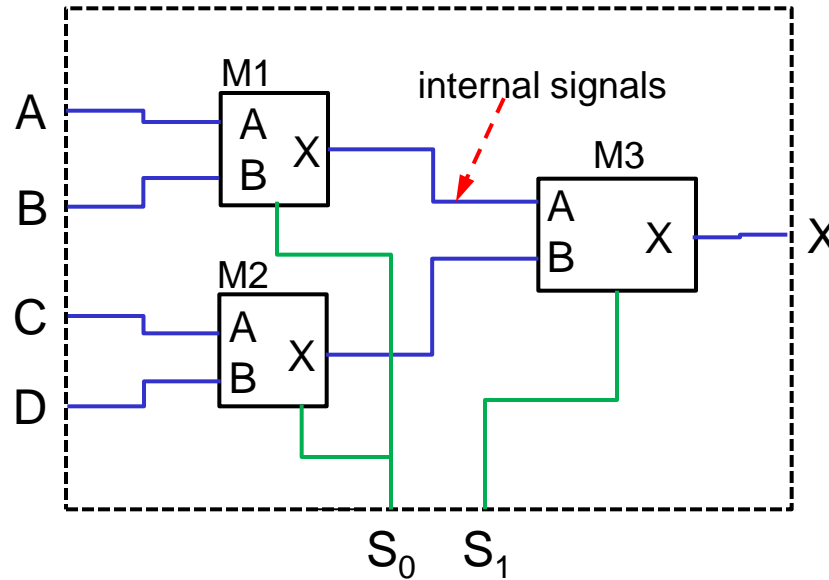
≡



1 bit 4 to 1 mux

1 bit 4 to 1 mux
created using 2 to 1 muxes

external connections



external connections

Back to Code

```
entity mux2to1 is
  Port ( A : in  STD_LOGIC;
        B : in  STD_LOGIC;
        S : in  STD_LOGIC;
        X : out STD_LOGIC);
end mux2to1;

architecture Behavioral of mux2to1 is
begin
  X <= (A and not S) or (B and S);
end Behavioral;
```

declare *components* here

declare intermediate *signals* here too

these are called instantiations

optional labels

you can shape text to your taste

You may obviously describe a 4 to 1 multiplexer in one entity
(it may be more efficient and readable too)

```
entity mux4to1 is
  Port ( A : in  STD_LOGIC;
        B : in  STD_LOGIC;
        C : in  STD_LOGIC;
        D : in  STD_LOGIC;
        S0 : in  STD_LOGIC;
        S1 : in  STD_LOGIC;
        X : out STD_LOGIC);
end mux4to1;

architecture mux4to1 of mux4to1 is
  component mux2to1 is Port ( -- component declaration
    A: in  STD_LOGIC;
    B: in  STD_LOGIC;
    S: in  STD_LOGIC;
    X: out STD_LOGIC);
  end component;

  signal X1, X2 : STD_LOGIC;

begin
  M1: mux2to1 port map ( -- component instantiation
    A => A,
    B => B,
    S => S0,
    X => X1
  );
  M2: mux2to1 port map (
    A => C, B => D, S => S0, X => X2
  );

  -- Complete the rest yourselves

end mux4to1;
```

Hmw : design a 4 to 1 mux

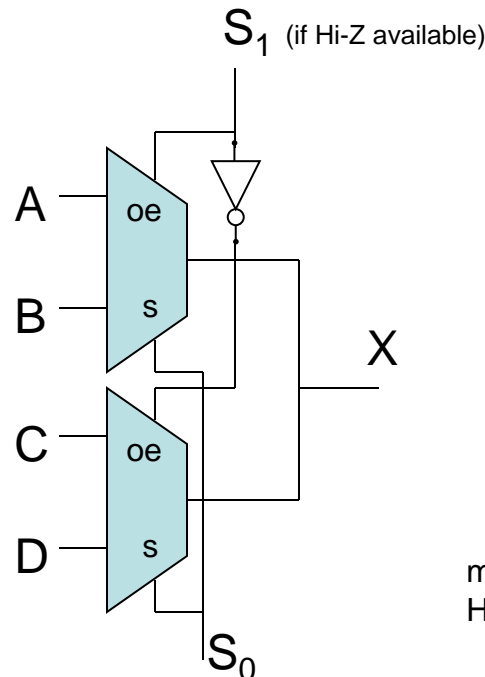
STD_LOGIC Type

STD_LOGIC types can take the following values

- 0 : logic 0
- 1 : logic 1
- Z : high impedance (Hi-Z)
- X : unknown
- W : weak unknown
- L : weak low
- H : weak high
- : don't care

oe	S	A	B	X
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1
0	-	-	-	Z

read for these at home



use of Z for a multiply driven signal

most FPGAs (including Spartan) do not have internal Hi-Z.
Hi-Z can be used at I/O ports (pins).

Hmw : design a 4 to 1 mux using two 2 to 1 mux with oe inputs ☺

STD_LOGIC_VECTOR Type

A collection of `STD_LOGIC` types

Example :

```
signal sel, sel2 : STD_LOGIC_VECTOR (0 to 3);  
signal LEDES : STD_LOGIC_VECTOR (7 downto 0);
```

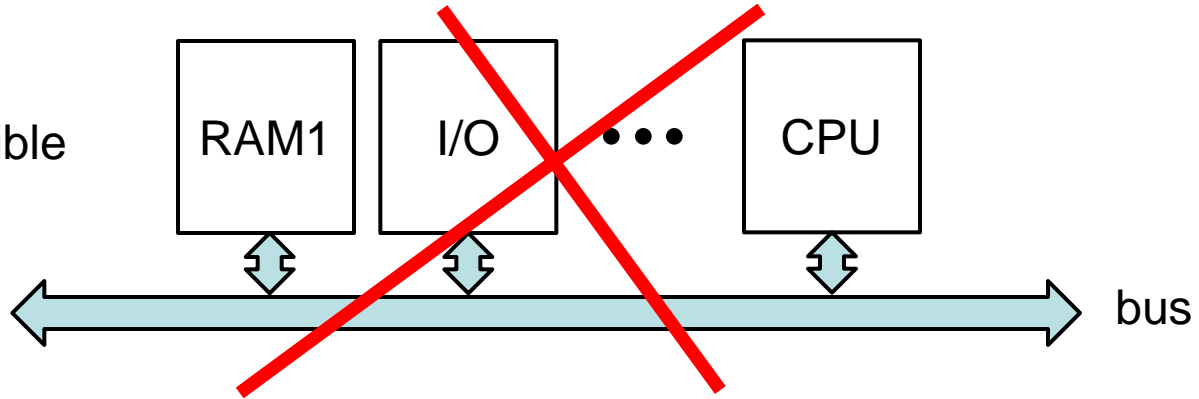
```
sel <= "0110";  
sel2(2 to 3) <= "01";  
LEDES <= (7=>'1', 6=>'0', others=>'Z');  
LEDES(4) <= '0'; -- notice single quotes
```

```
LEDES <= LEDES +1;  
-- requires use IEEE.STD_LOGIC_ARITH.ALL;
```

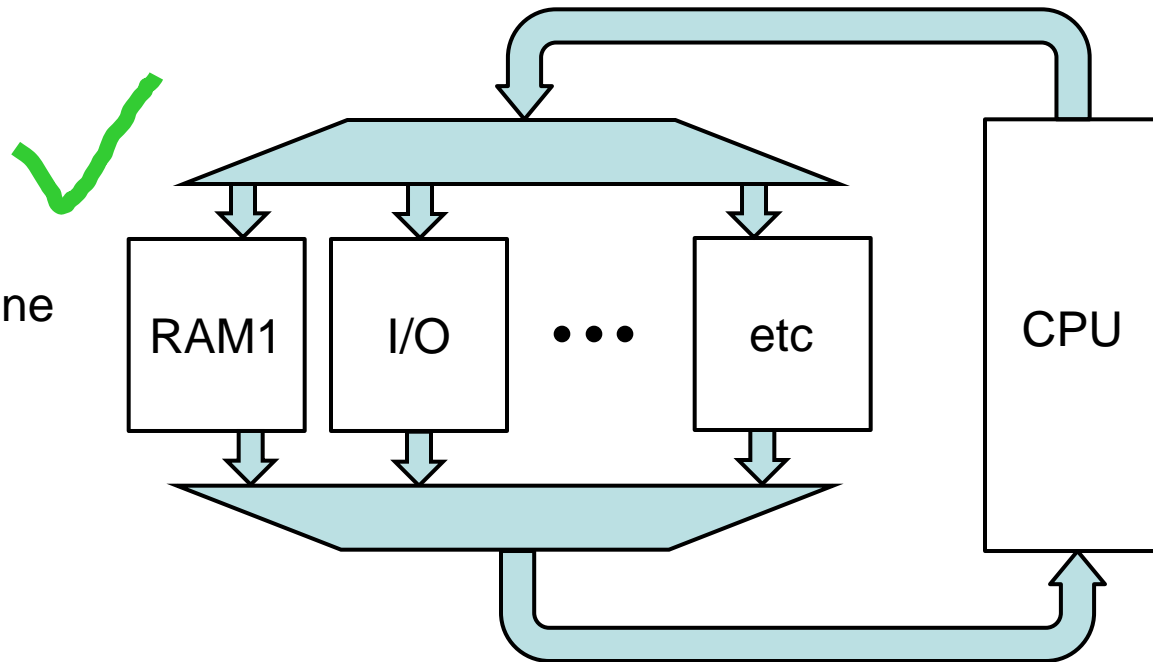
```
LEDES(7 downto 4) <= sel;
```

Since Hi-Z based bus systems are not possible in FPGAs

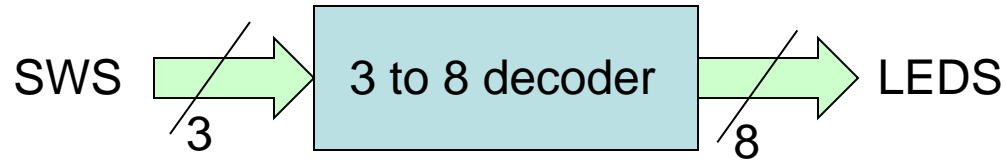
Not possible



What is done



Another Combinatorial Example



```
LEDS <= "00000001" when SWS="000" else  
       "00000010" when SWS="001" else  
       "00000100" when SWS="010" else  
       "00001000" when SWS="011" else  
       "00010000" when SWS="100" else  
       "00100000" when SWS="101" else  
       "01000000" when SWS="110" else  
       "10000000";
```

```
with SWS select  
  LEDS <= "00000001" when "000",  
         "00000010" when "001",  
         "00000100" when "010",  
         "00001000" when "011",  
         "00010000" when "100",  
         "00100000" when "101",  
         "01000000" when "110",  
         "10000000" when others;
```

```
LEDS(0) <= '1' when SWS = "000" else '0';  
LEDS(1) <= '1' when SWS = "001" else '0';  
LEDS(2) <= '1' when SWS = "010" else '0';  
LEDS(3) <= '1' when SWS = "011" else '0';  
LEDS(4) <= '1' when SWS = "100" else '0';  
LEDS(5) <= '1' when SWS = "101" else '0';  
LEDS(6) <= '1' when SWS = "110" else '0';  
LEDS(7) <= '1' when SWS = "111" else '0';
```

do not forget to cover all possibilities
when using `select`

```
if(SWS="000") LEDS<="00000001";  
elsif(SWS="001") LEDS<="00000010";  
...  
else ...  
end if;
```

classic `if-elsif-else-end if;` can be
used in `processes`

3 to 7 Decoder

```
library IEEE;
use IEEE.STD_LOGIC_VECTOR.ALL;

entity decoder3to7 is
    Port ( D : in  STD_LOGIC_VECTOR(2 downto 0);
          Q : out STD_LOGIC_VECTOR(7 downto 0));
end decoder3to7;

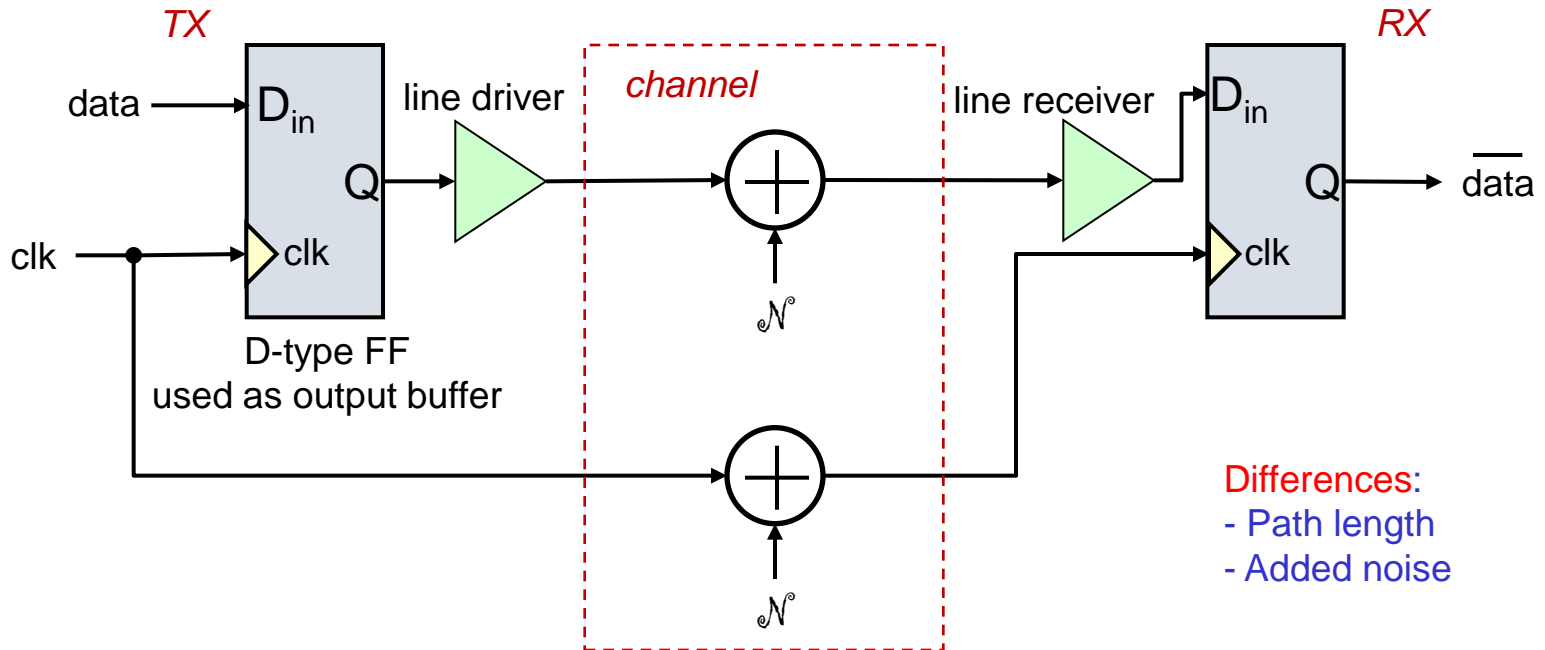
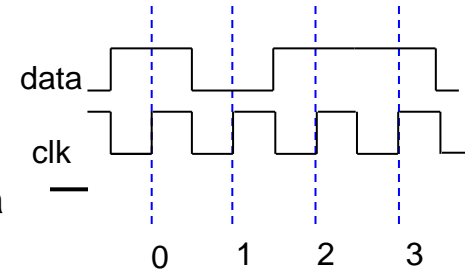
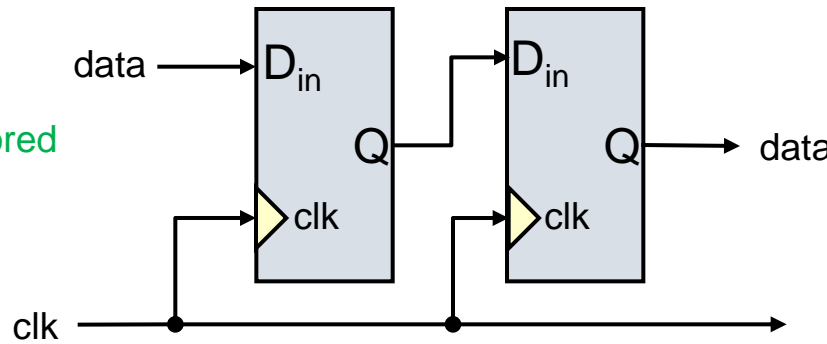
architecture decoder3to7 of decoder3to7 is
begin

    Q <= "00000001" when D="000" else
        "00000010" when D="001" else
        "00000100" when D="010" else
        "00001000" when D="011" else
        "00010000" when D="100" else
        "00100000" when D="101" else
        "01000000" when D="110" else
        "10000000";

end decoder3to7 ;
```


As simple as it gets

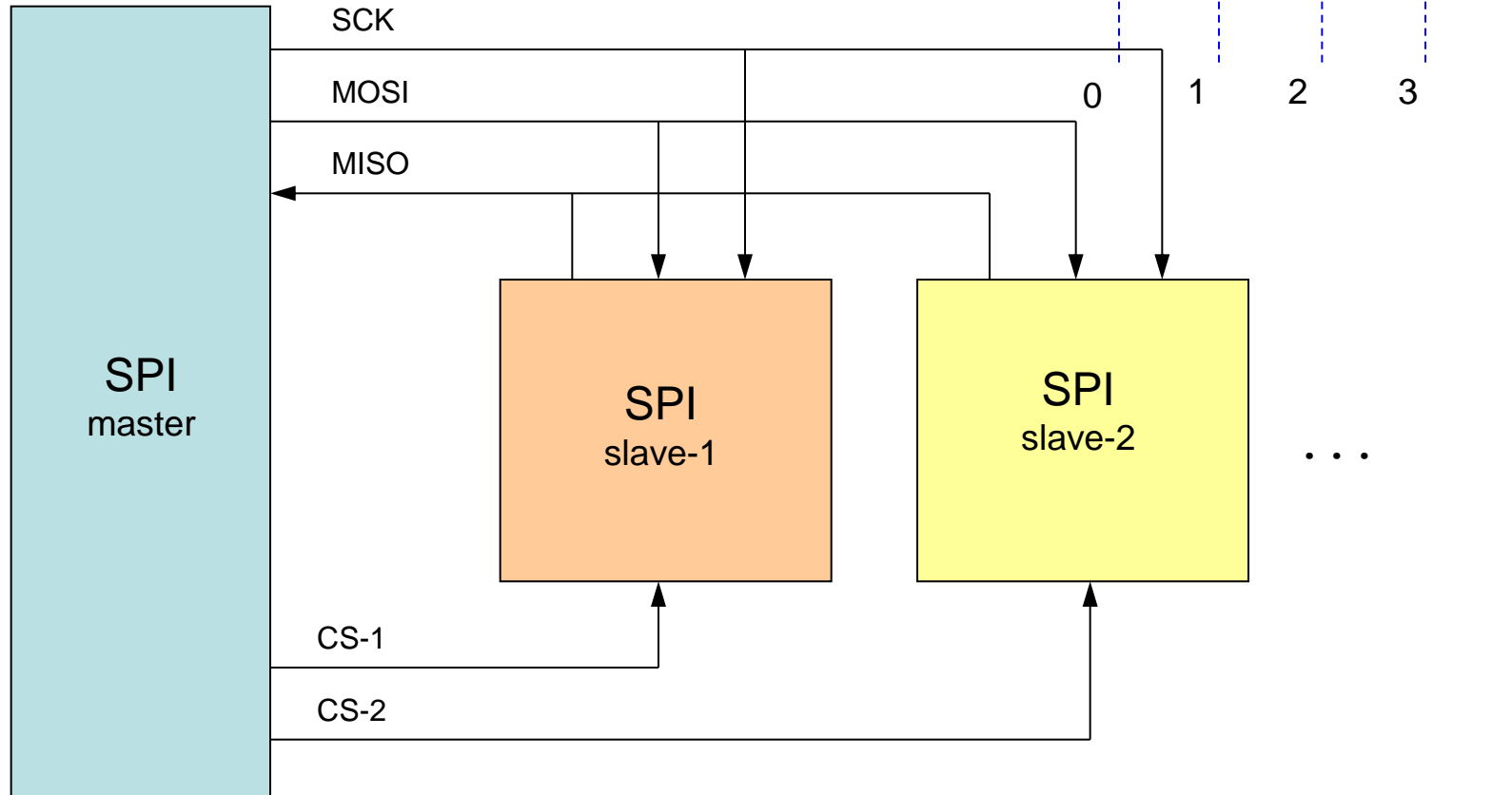
Shift register transfers stored bit values from one FF to another at each clk pulse



Differences:
- Path length
- Added noise

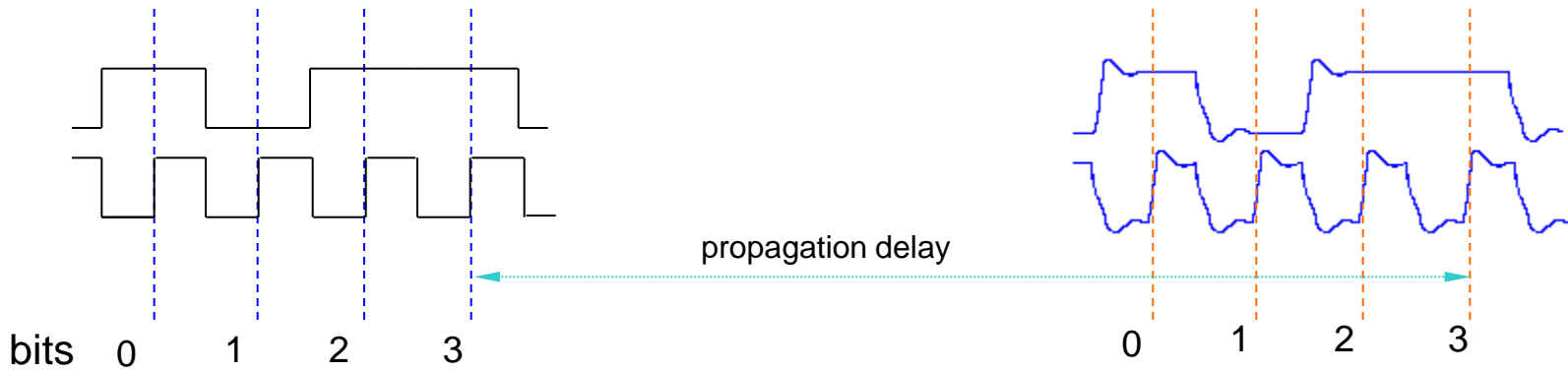
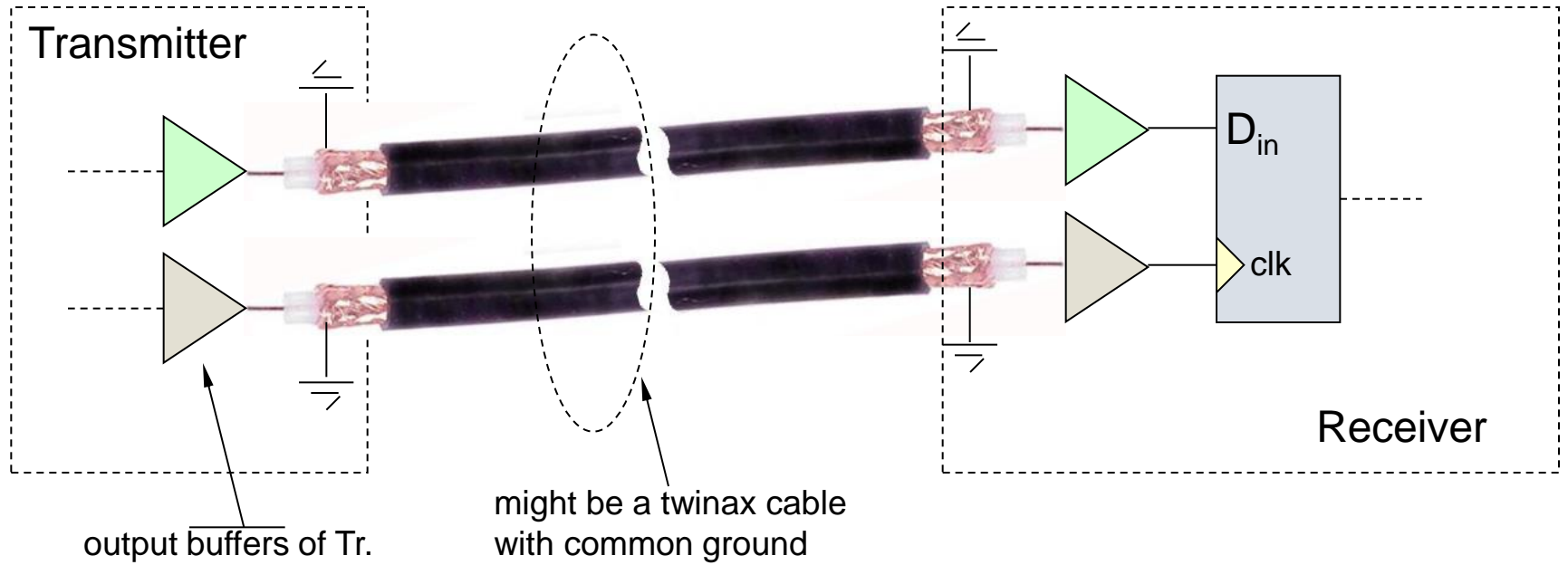
SPI : Serial Peripheral Interface

is a clocked serial communication protocol for short lengths



SPI is a modern way to communicate between master and slave IC's on a single PC-board

Initial Design (Synchronous Serial)



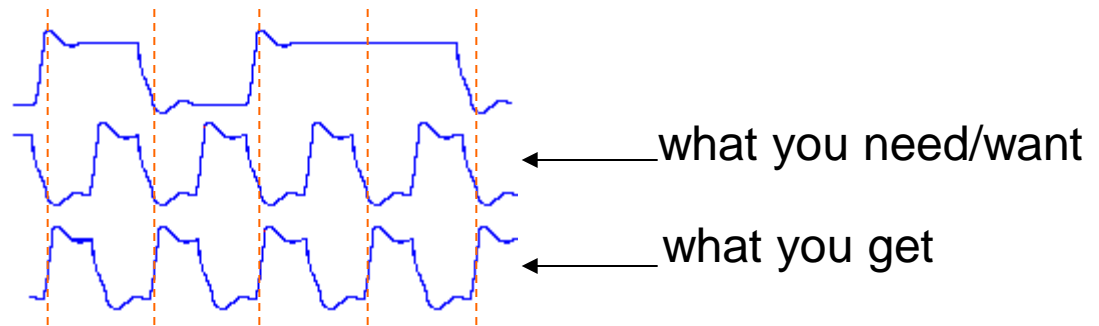
Let us assume that data and clock line lengths differ by 10 cm



one of the signals arrive 0.5 picoseconds late.

(speed of e.m. wave on copper is about 2×10^8 m/s)

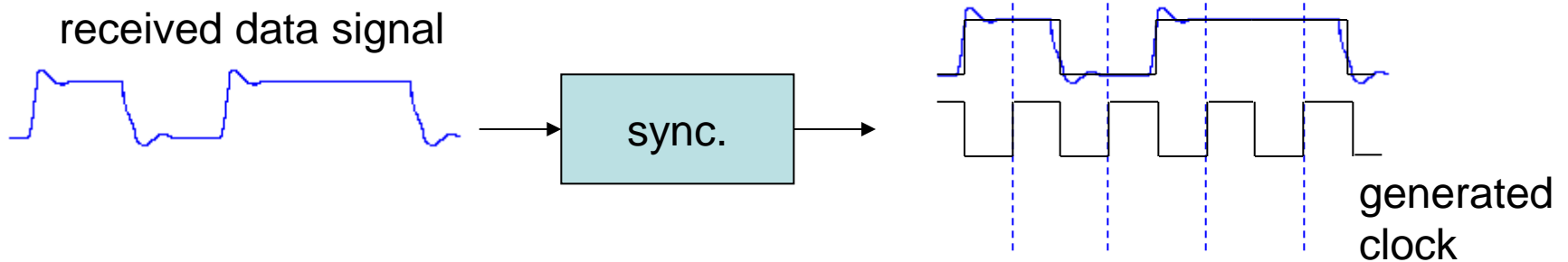
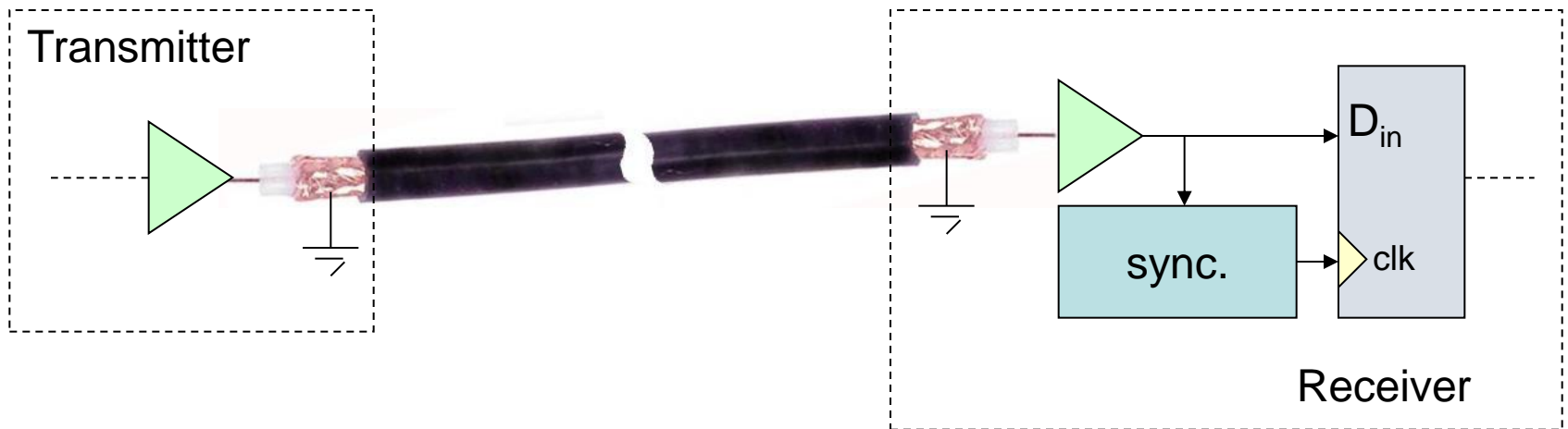
Problem is : for a 1GHz clock, 0.5 ps is about half a clock cycle.



Solution

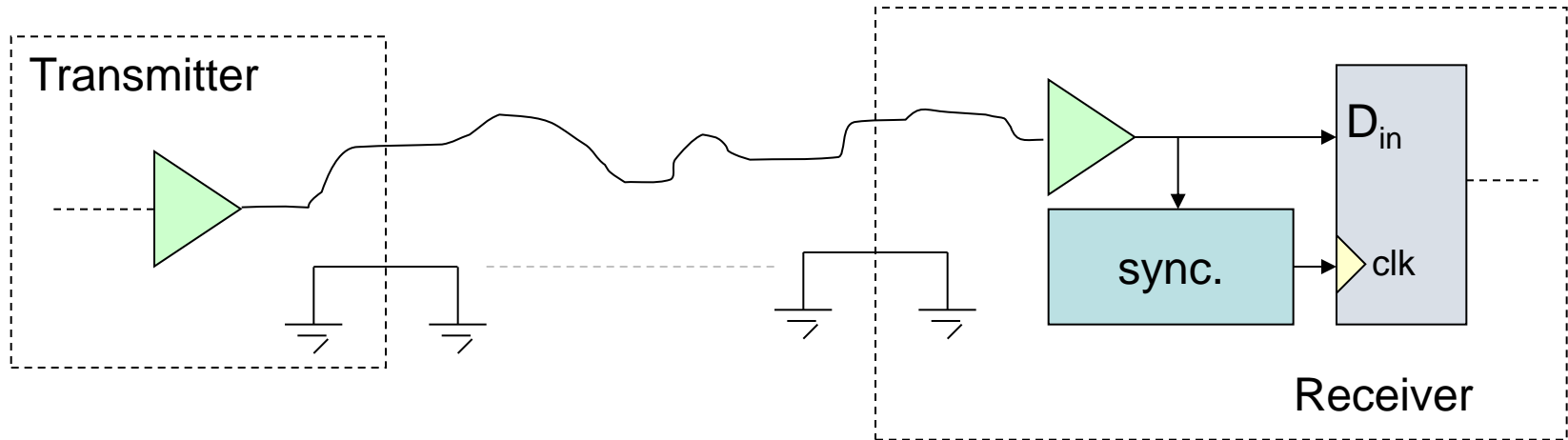
Solution is to generate clock from data at the receiver.

The data signal should necessarily be designed to perform such an operation



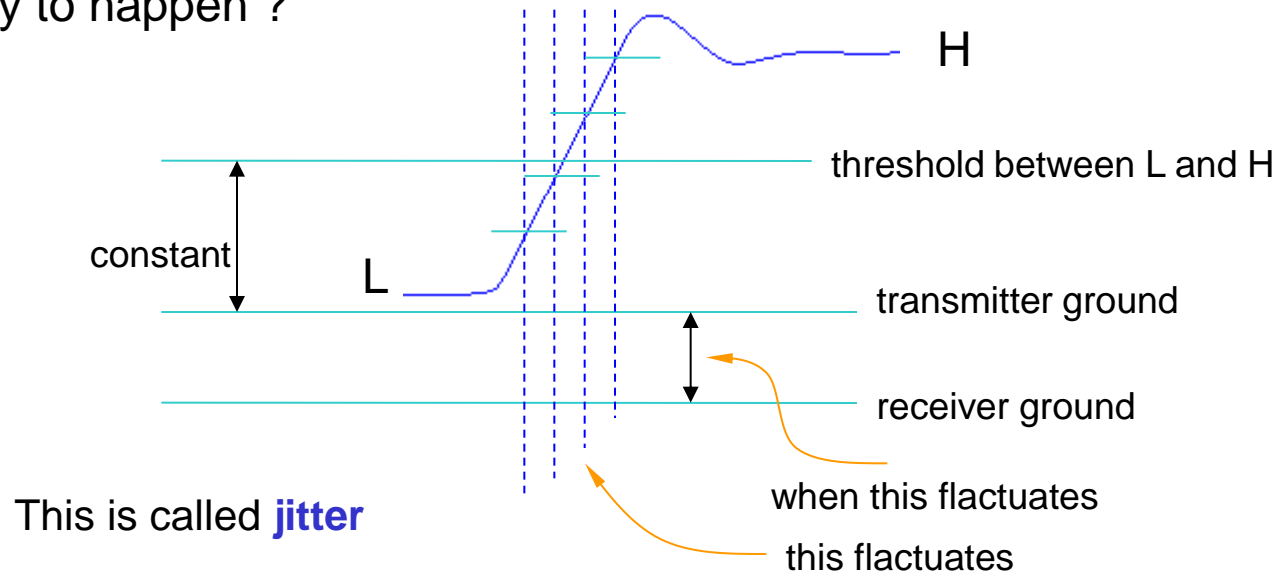
A Phase Locked Loop (PLL) can be used if there are enough transitions in the signal

Use of Ground as signal return

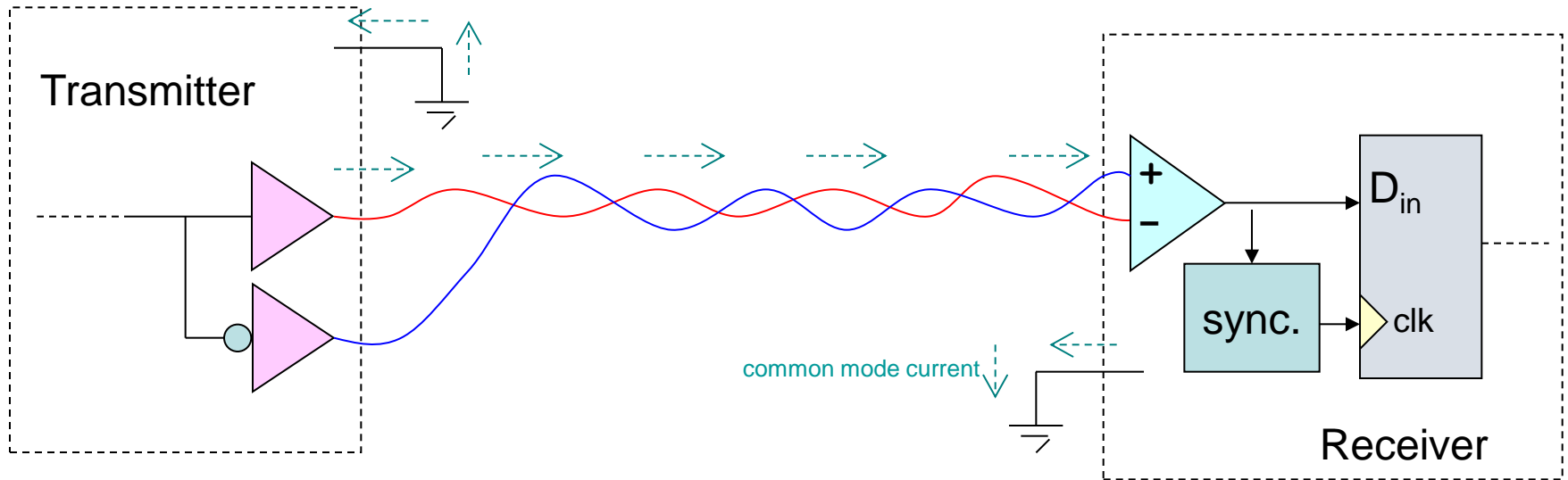


This requires that grounds at both side must have the same potential which we cannot guarantee

What is likely to happen ?



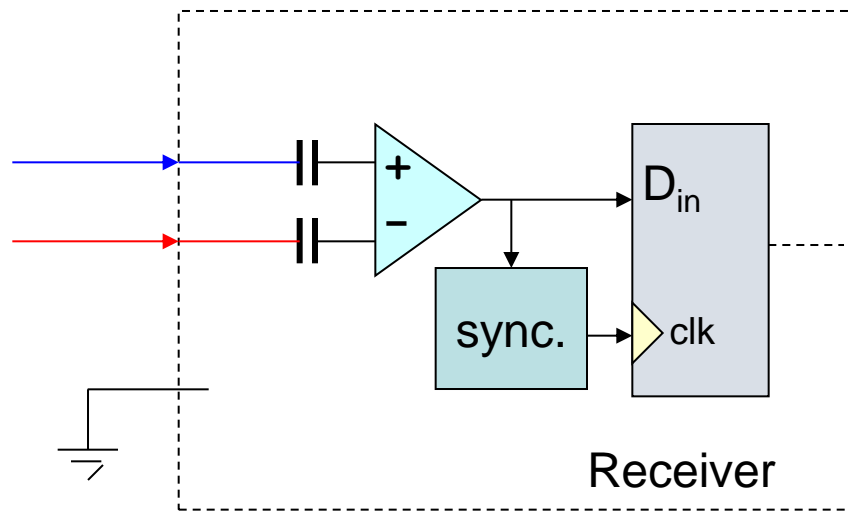
Differential Signaling



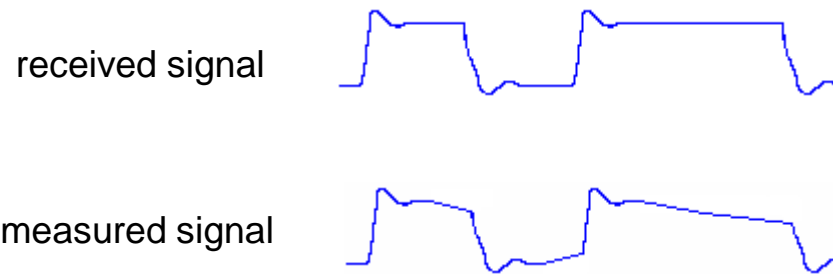
Receiver uses the voltage **difference** between two inputs.
The voltage between a signal line and the ground is not in the formula here, but we have another problem.

If there is a voltage difference between two grounds then there will be a **common mode current** on the signal lines returning from ground.

Capacitors to prevent CMC



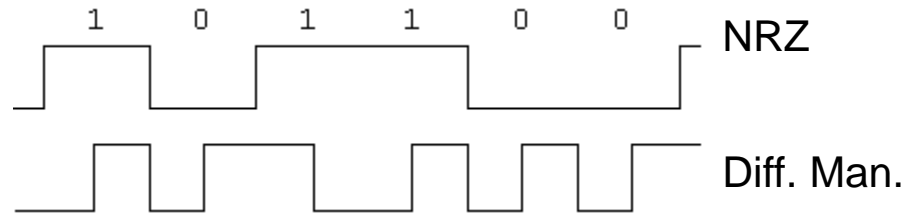
Problem is different this time.



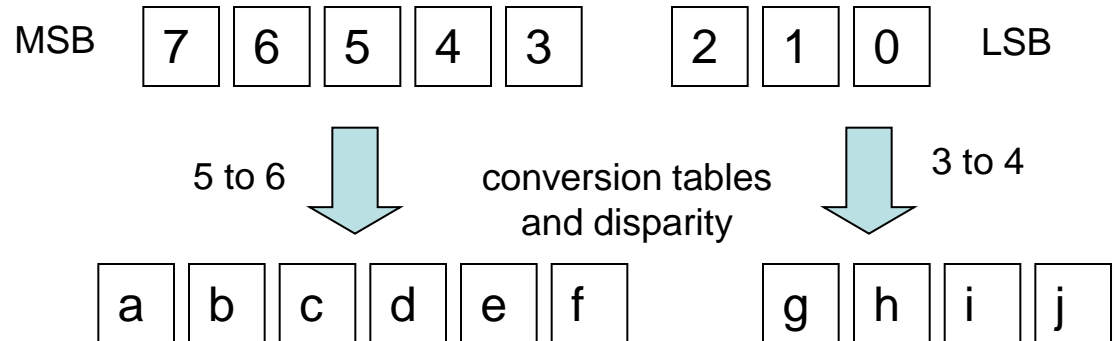
if there are long runs of 0s or 1s in the signal the receiver might lose the synchronization and/or cannot read the data.

Long runs of 0s and 1s is solved by coding

Example : Manchester coding

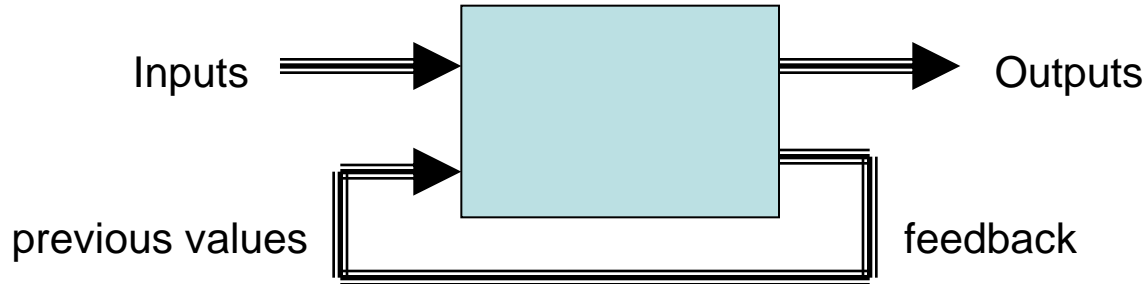


Example : 8B/10B



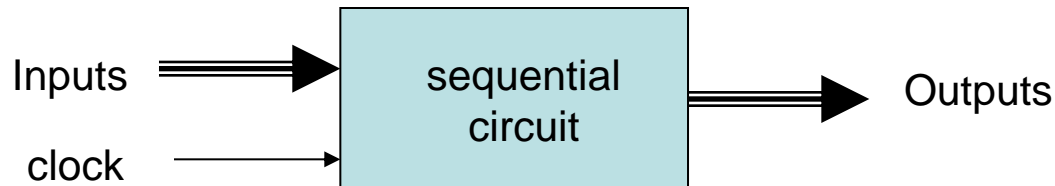
What is a Sequential Circuit?

If a circuit needs some values calculated from previous input values, it has to have some way of remembering these values.

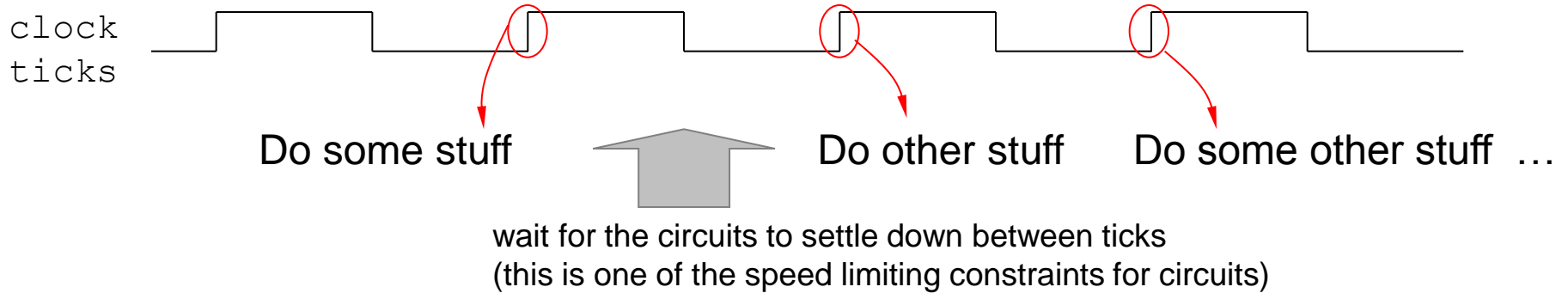


One would design such circuits using two well known models; Mealy (output depends on the inputs and stored values) and Moore (output depends only on the stored values)

Today, to make things simpler, almost all digital circuits are designed as clocked *sequential* machines (Moore). Things are done *synchronously* with the rising or falling (or both) edge of a clock signal.



Clocked & Synchronous



"Sequential", for digital circuits, does not mean that circuit pieces described by each of our VHDL code lines do their stuff one after another. It means that, outputs of designed circuits are somewhat affected from the previous input sequences.

Remember that, VHDL is **not a programming language**. It is a description language. We are describing a digital circuit **★** that does things **concurrently** (keeping in mind the delays caused by the electronics and the finite speed of EM waves on silicon of course).

★ : One may use VHDL for different purposes, but here we use it to describe circuits within FPGAs

Some Attributes

Given

```
signal D : STD_LOGIC_VECTOR(7 downto 0);  
signal X : STD_LOGIC_VECTOR(2 to 5);
```

D'LOW is 0 (lower array index)

X'LOW is 2

D'HIGH is 7 (upper array index)

X'HIGH is 5

D'LEFT is 7 (leftmost array index)

X'LEFT is 2

D'RIGHT is 0 (rightmost array index)

X'RIGHT is 5

D'LENGTH is 8 (size of the array)

X'LENGTH is 4

D'RANGE is (7 downto 0) (range of the array)

X'RANGE is (2 to 5)

D'REVERSE_RANGE is (0 to 7) (range of the array in reverse order)

X'REVERSE_RANGE is (5 downto 2)

```
signal Y: STD_LOGIC_VECTOR(D'RANGE);
```

```
...
```

```
D(D'RIGHT) <= '1'; -- set righthmost bit to 1
```

Some Attributes

Given

```
signal CLK : STD_LOGIC;
```

CLK'EVENT	is TRUE if there is an event on CLK
CLK'STABLE[t]	is TRUE if there is no event on CLK in last t time unit
CLK'ACTIVE	is TRUE when there is a transaction on (assignment) CLK
CLK'QUIET[t]	is TRUE if there is no transaction on CLK during the last t time unit

'EVENT and 'STABLE attributes are synthesizable, others are for simulation only

if keyword is a sequential statement used in **processes**

```
if (CLK'EVENT and CLK='1') then
    ...
end if;
```

|||

```
if (RISING_EDGE(CLK)) then
    ...
end if;
```

processes

synchronous

```
entity Toggle is
  Port ( T : in  STD_LOGIC;
        Q : out STD_LOGIC);
end Toggle;

architecture Toggle of Toggle is


  signal D : STD_LOGIC;

begin

  process(T,D) is begin
    if(RISING_EDGE(T)) then
      D <= not D;
    end if;
  end process;

  Q <= D;

end Toggle;
```



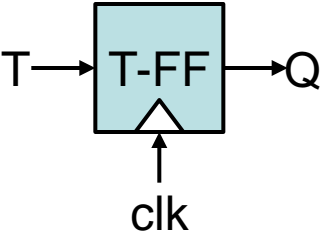
fully synchronous

```
entity Toggle is
  Port ( clk : in  STD_LOGIC;
        T : in  STD_LOGIC;
        Q : inout STD_LOGIC);
end Toggle;

architecture Toggle of Toggle is
  signal D : STD_LOGIC;
begin

  process(clk,T,D) is begin
    if(RISING_EDGE(clk)) then
      if((D~=T) and (T='1')) then
        Q <= not Q;
      end if;
      D <= T;
    end if;
  end process;

end Toggle;
```



Other Synthesizable Pre-Defined Simple Types

STD_ULOGIC : U stands for unresolved

BOOLEAN : True or False

INTEGER : 32 (max) bit integers (-2147483647 to +2147483647)

NATURAL : non-negative integers

...and arrays of these...

```
signal OE : STD_LOGIC;  
signal count : integer;  
signal IsOK, DOIT : BOOLEAN;
```

```
OE <= 'Z';  
count <= count + 1;  
IsOK <= not DOIT ;  
DOIT <= False;
```


we need to use related library for some types

Vectors

Notice that integer and natural are actually collections of bits.
We have other collections too.

BIT_VECTOR : collection of **BITS**
STD_LOGIC_VECTOR : collection of **STD_LOGIC** types
STD_ULOGIC_VECTOR :
SIGNED, UNSIGNED : kinda **integers**

```
signal sel,sel2 : BIT_VECTOR (0 to 3);  
signal LEDS : STD_LOGIC_VECTOR (7 downto 0);  
signal count : integer range 0 to 15;
```

 inherently creates a 4 bit signal

```
sel <= "0110";  
sel2(2 to 3) <= "01";  
LEDS <= (7=>'1', 6=>'0', others=>'Z');  
count <= count +1; -- counts up to 15
```

```
LEDS(7 downto 4) <= sel; -- error, incompatible types
```


Text Book : V.A. Pedroni, Circuit Design with VHDL, MIT Press.

Homework : Read sections 1, 2, 3, 4

Do problems 3.2, 3.4, 4.1, 4.2.

Design a 7-segment decoder using a 10x7 ROM array.

Design a press-on/press-off button controlled LED circuit

Text Book : B. Sklar, Digital Communications (2nd Ed.)
Fundamentals and Applications, MIT Press.

Homework : Read section 2.3, 2.3.1, 2.8.5

Do problems 2.2, 2.3.

An Up-Counter With Asynchronous Reset

inout keyword allows reading back what is written previously

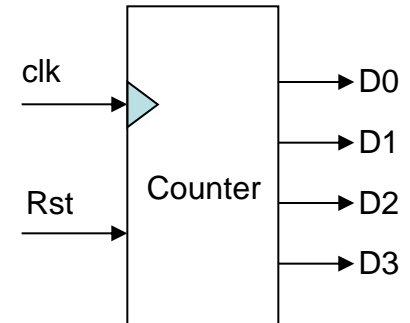
```
entity CntrWRst is
    Port ( clk : in  STD_LOGIC;
          Rst  : in  STD_LOGIC;
          Data : inout STD_LOGIC_VECTOR (3 downto 0));
end CntrWRst;
```

process keyword has *sensitivity list* (think of a C-function call when one or more arguments change)

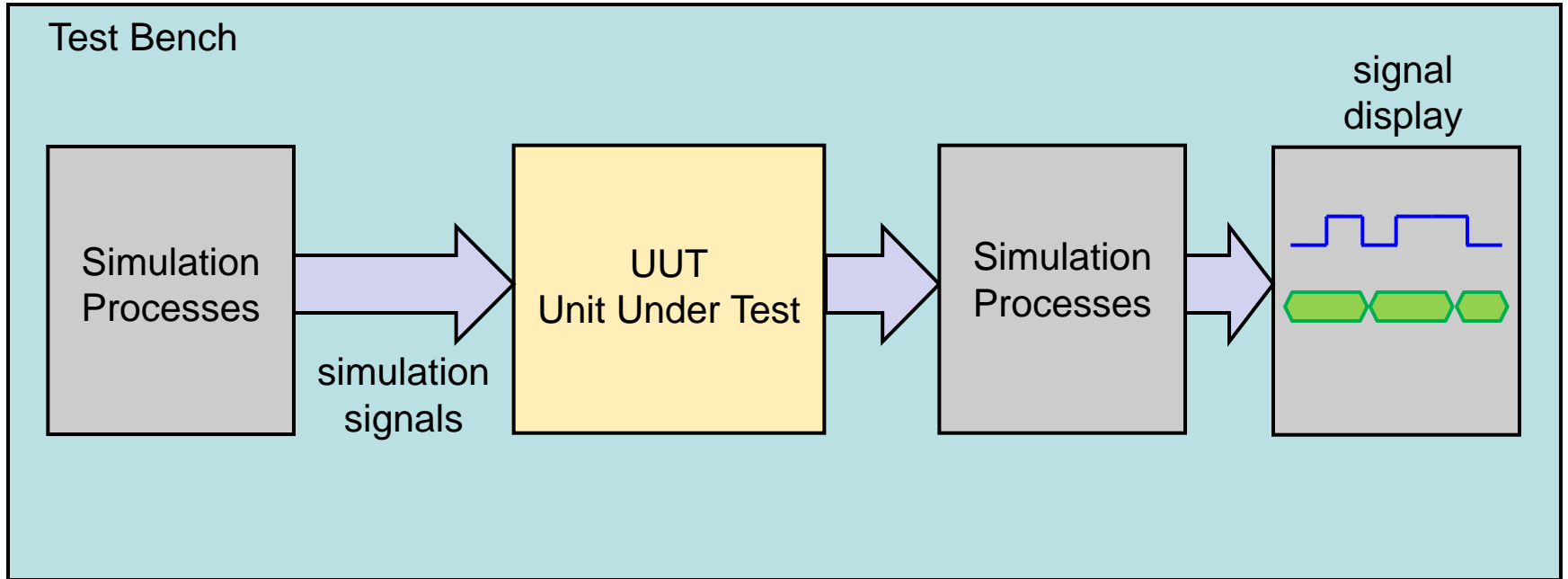
```
architecture CntrWRst of CntrWRst is
begin
    process (clk, Rst) is
    begin
        if (Rst='1') then
            Data <= "0000";
        else
            if (RISING_EDGE(clk)) then
                Data <= Data +1;
            end if;
        end if;
    end process;
end CntrWRst;
```

This is equivalent to `clk'EVENT and clk='1'`

if-elsif-else-end if is a sequential struct to be used in **processes**

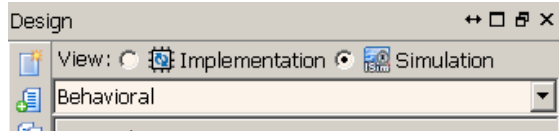


Model of the Simulation

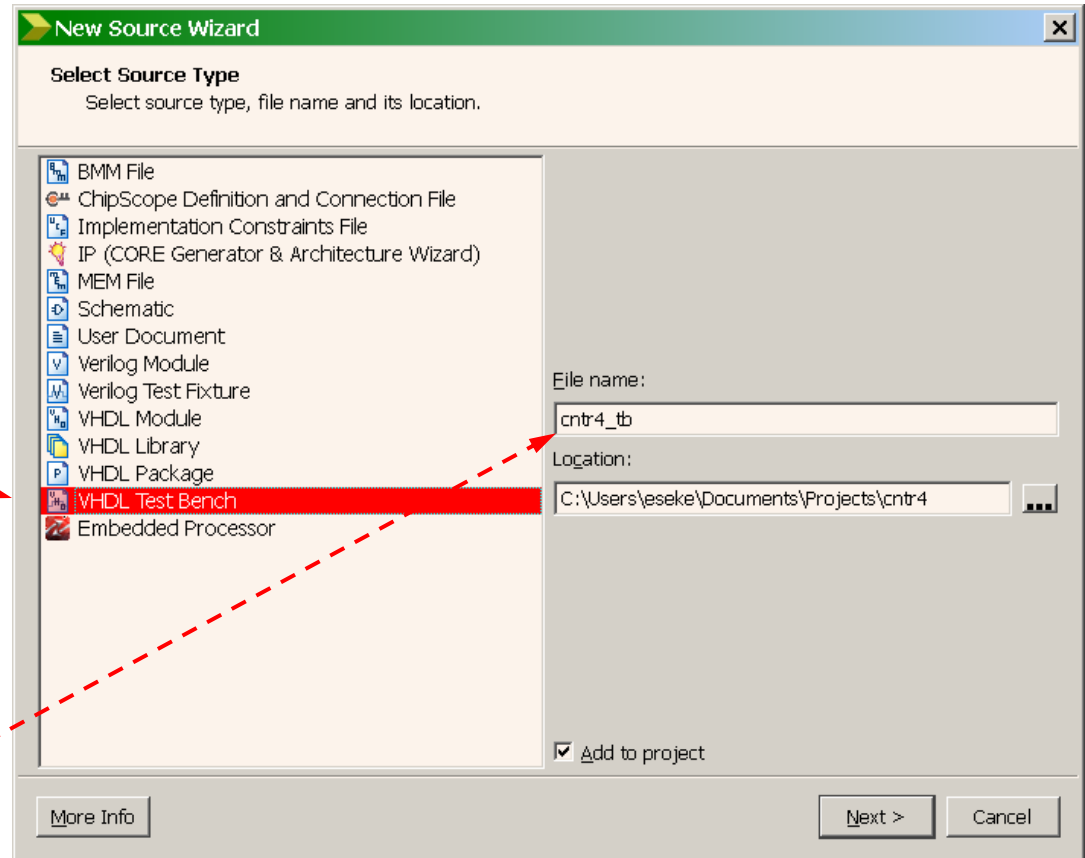


VHDL Test Bench & Simulation

1 Add a New Source File



Select *VHDL Test Bench*



since the extension will be *.VHD,
adding _tb to the name is a good
idea.

Click *Next*

Associate it with the implementation file and close the summary dialog

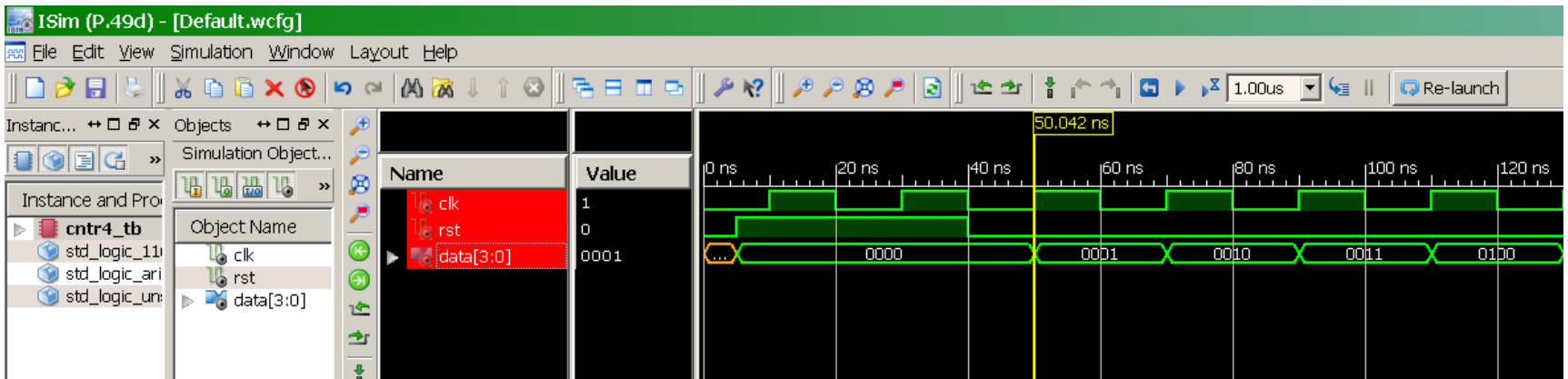
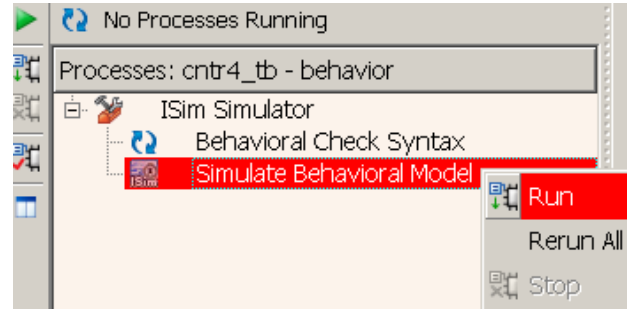
2

Create Generators

```
ARCHITECTURE behavior OF cntr4_tb IS
  COMPONENT CntrWRst
    PORT (
      clk : IN  std_logic;
      Rst : IN  std_logic;
      Data : INOUT std_logic_vector(3 downto 0));
  END COMPONENT;
  signal clk : std_logic := '0';
  signal Rst : std_logic := '0';
  signal Data : std_logic_vector(3 downto 0);
BEGIN
  uut: CntrWRst PORT MAP (
    clk => clk,
    Rst => Rst,
    Data => Data
  );
  clk_process :process begin
    clk <= '0';
    wait for 10ns;
    clk <= '1';
    wait for 10ns;
  end process;

  stim_proc: process
  begin
    wait for 5ns;
    Rst <= '1';
    wait for 35ns; Rst <= '0';
    wait for 400ns; Rst <= '1';
    wait for 60ns; Rst <= '0';
    wait;
  end process;
END;
```

3 Start Simulation



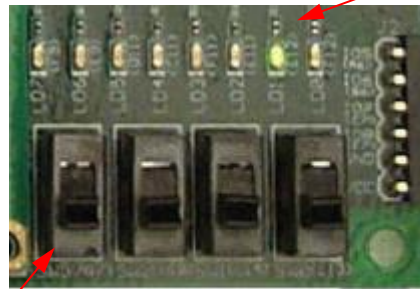
A Simple Serial Communication System

Problem: Design of an asynchronous serial comm. system between two boards with FPGA devices (Xilinx Spartan-3E starter kit)

Simple test setup

4 bit output to LEDs

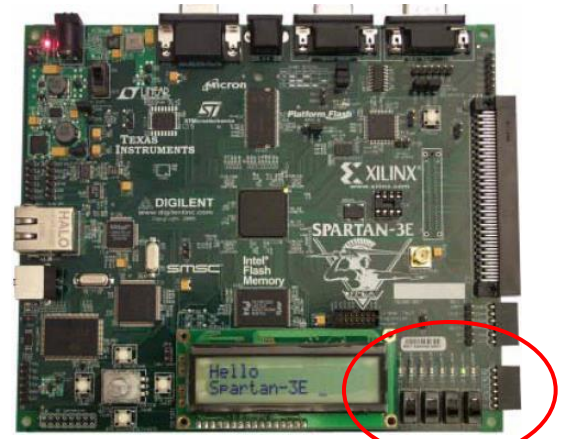
F9 E9 D11 C11 F11 E11 E12 F12



A6 serial output
B6 serial input
GND (signal return)

these two can be connected for a loopback test

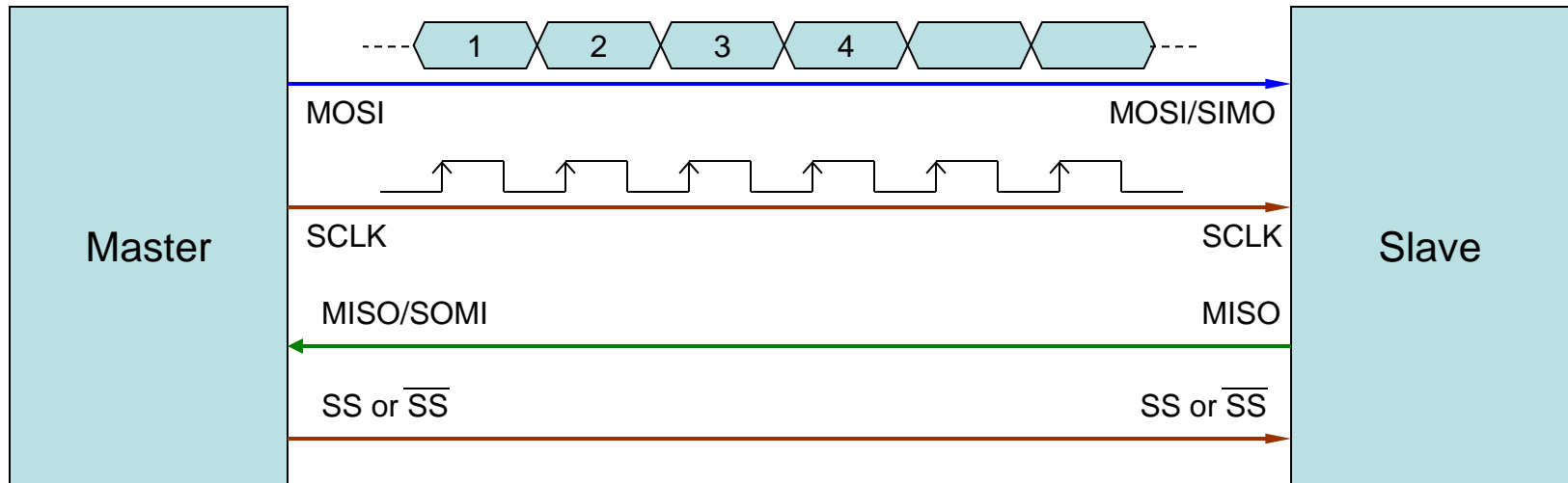
4 bit data input from switches



This example is prepared using very basic VHDL constructs.
No or very little VHDL knowledge is necessary to understand the operations

Serial Peripheral Interface (SPI)

Transmit: Master puts the bit to be transmitted on MOSI and raises the clock (SCLK)
Slave latches the serial input on the rising edge of the incoming clock.



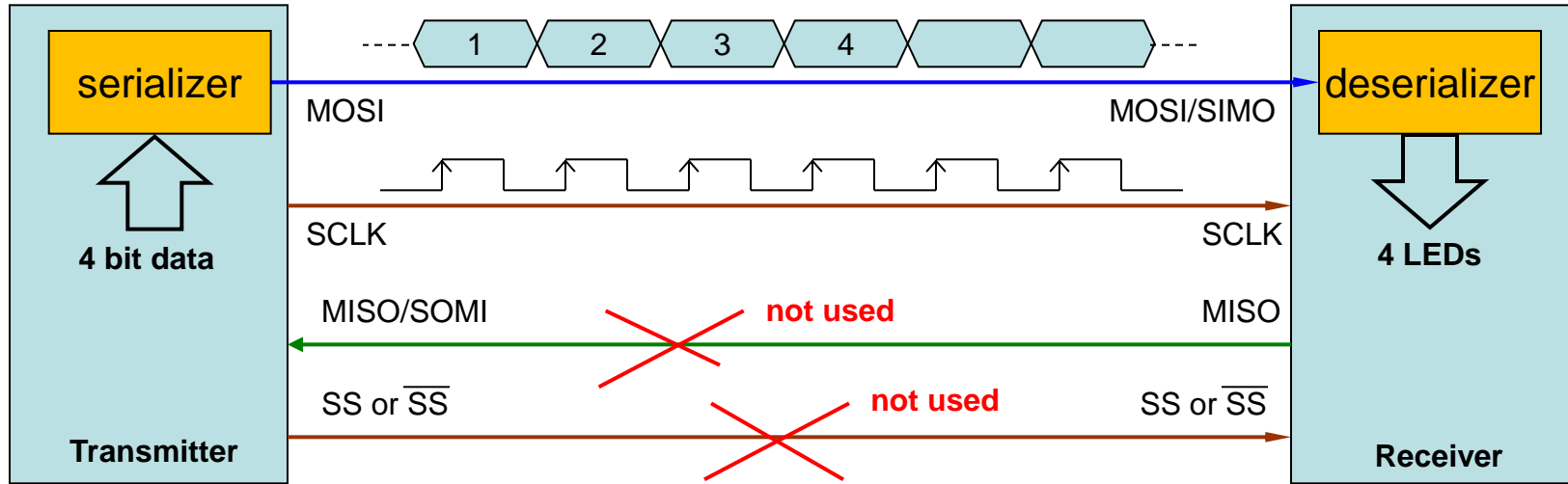
Receive: Master raises the SCLK and reads in MISO on the falling edge of SCLK.

SPI can not be used in long distance (a couple meters) communications, because of the timing problems. The length difference between clock and data lines must be much smaller than the clock wavelength.

SPI is used for board to board or chip to chip data transfers

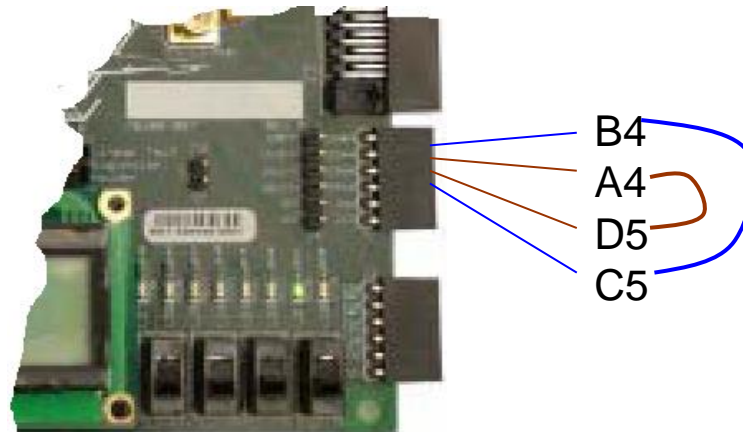
Use of SPI in our Example

Transmitter: Master serializes the 4 bit data and puts each bit on the line.

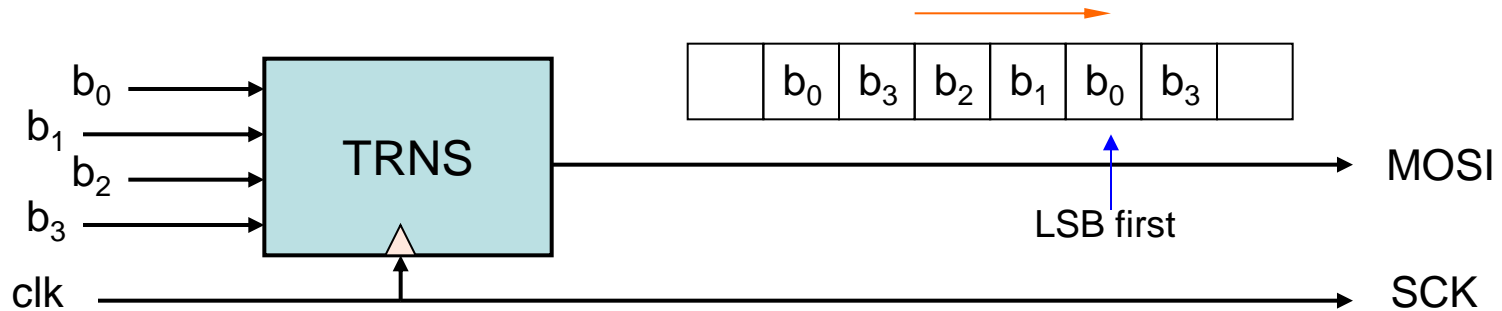


Receiver: On the rising edge of SCLK slave reads in MOSI and deserializes every 4 bits.

Single board test setup
(Loopback test)



Simple Transmitter



```

entity SPIT is Port (
  clk      : in  STD_LOGIC;  --we need a clock for action
  b        : in  STD_LOGIC_VECTOR(3 downto 0); -- parallel data in
  SCK      : out STD_LOGIC;   -- SPI clock
  MOSI     : out STD_LOGIC); -- serial data out
end SPIT;

```

```

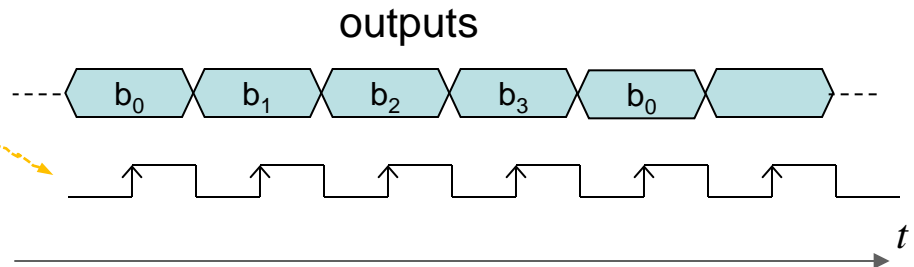
signal cntr : integer range 0 to 3;

```

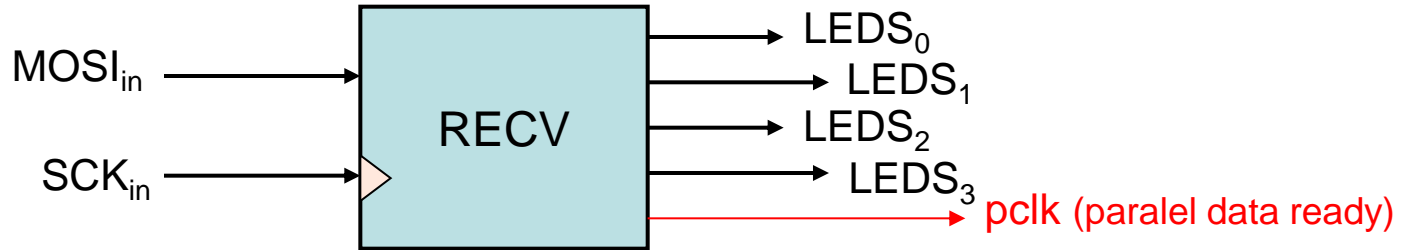
```

SCK <= clk;
TRNS: process(clk) is begin
  if(falling_edge(clk)) then
    MOSI <= b(cntr);
    cntr <= cntr+1;
  end if;
end process;

```



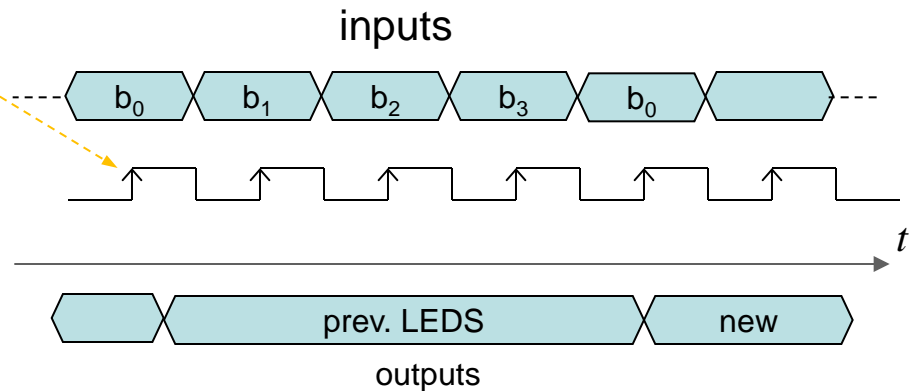
Simple Receiver



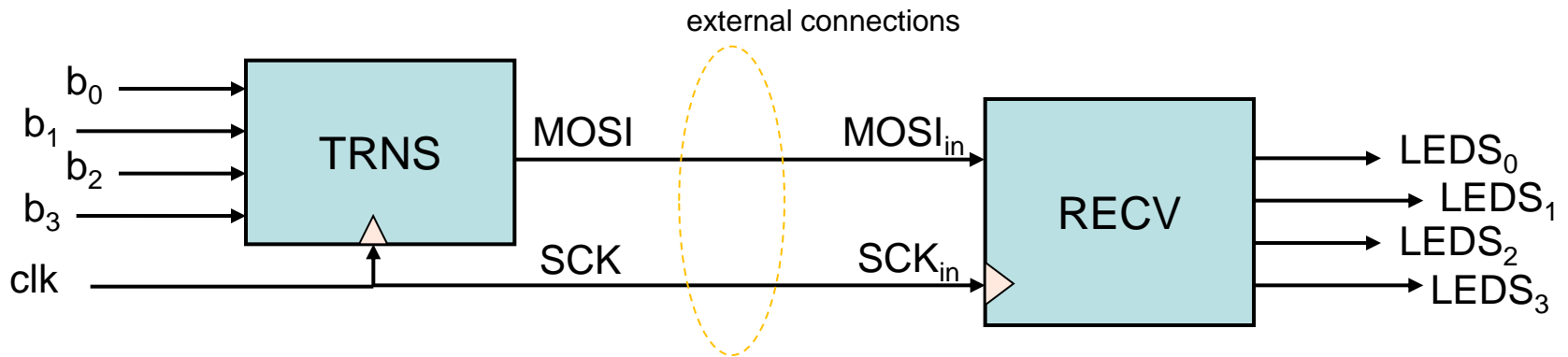
```
entity SPIR is Port (  
  clk      : in  STD_LOGIC;      -- do we need a clock for action?  
  SCKin    : in  STD_LOGIC;      -- SPI clock in  
  MOSIin   : in  STD_LOGIC;      -- serial data in  
  LEADS    : out STD_LOGIC_VECTOR(3 downto 0); -- parallel data out  
end SPIR;
```

```
signal recctr : integer range 0 to 3;  
signal recsig : STD_LOGIC_VECTOR(3 downto 0);
```

```
RECV: process(SCKin) is begin  
  if(rising_edge(SCKin)) then  
    recsig(recctr) <= MOSIin;  
    if(recctr=3) then  
      LEADS <= recsig;  
    end if;  
    recctr <= recctr +1;  
  end if;  
end process;
```



Simple Tranceiver



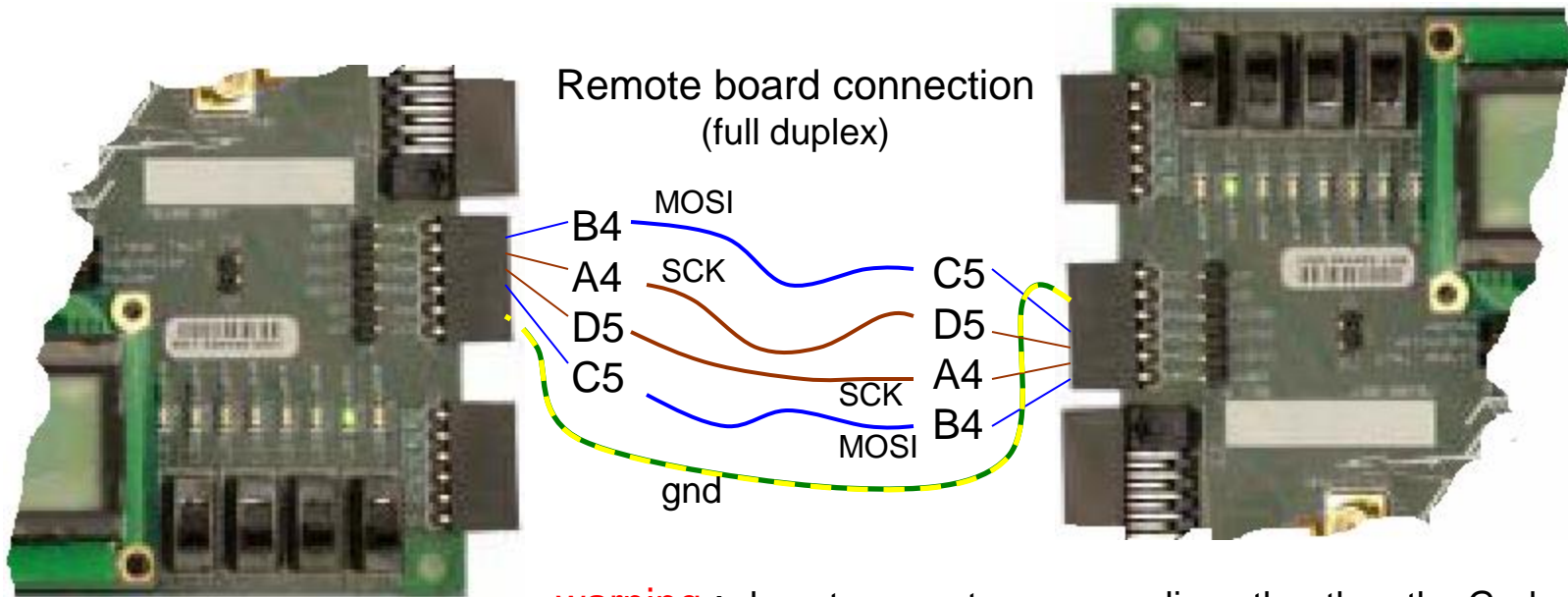
```
entity SPI is Port (  
  clk      : in  STD_LOGIC;  --we need a clock for action  
  ----- transmitter part  
  b        : in  STD_LOGIC_VECTOR(3 downto 0); -- parallel data in  
  SCK      : out STD_LOGIC;    -- SPI clock  
  MOSI     : out STD_LOGIC;    -- serial data out  
  ----- receiver part  
  SCKin    : in  STD_LOGIC;    -- SPI clock in  
  MOSIin   : in  STD_LOGIC;    -- serial data in  
  LE DS    : out STD_LOGIC_VECTOR(3 downto 0); -- parallel data out  
end SPI;
```

F9 E9 D11 C11 F11 E11 E12 F12



SW3 SW2 SW1 SW0
(N17) (H18) (L14) (L13)

Q1 : What if We Use Two Transmitter & Receiver ?

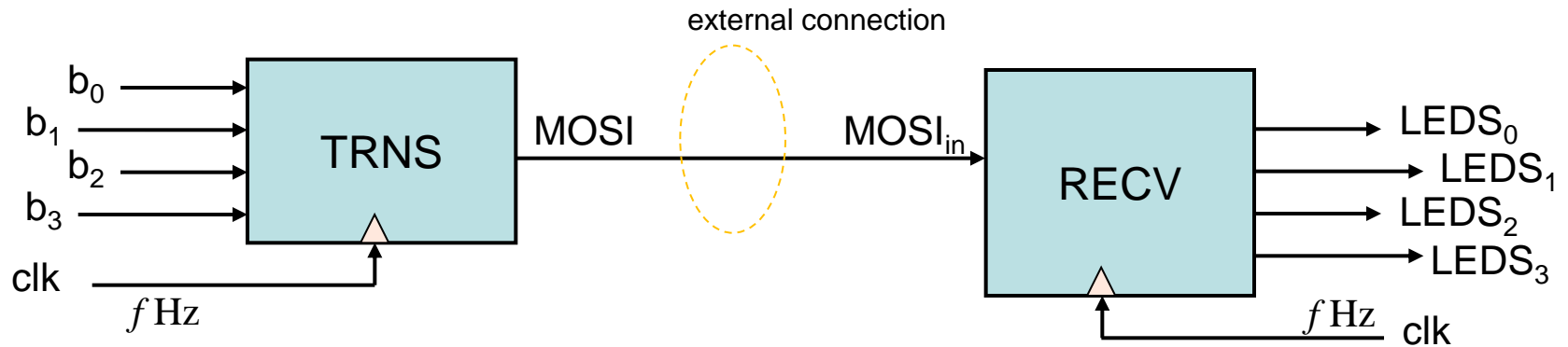


warning : do not connect any power line other than the Gnd

Do we have any problems having **correct** LEDs lid ?

Q2 : We directly connected SCK to internal 50 MHz oscillator.
How about not transmitting SCK and using internal 50 MHz on each board ?

Asynchronous Transmitter & Receiver



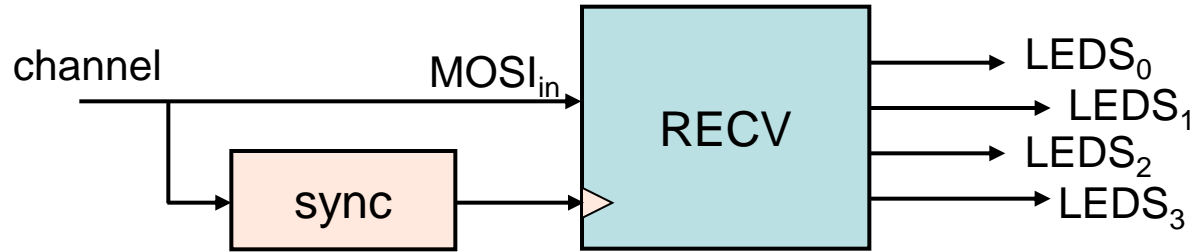
```
entity SPI is Port (  
  clk      : in  STD_LOGIC;  --we need a clock for action  
  ----- transmitter part  
  b        : in  STD_LOGIC_VECTOR(3 downto 0); -- parallel data in  
  MOSI     : out STD_LOGIC;      -- serial data out  
  ----- receiver part  
  MOSIin   : in  STD_LOGIC;      -- serial data in  
  LEDS     : out STD_LOGIC_VECTOR(3 downto 0)); -- parallel data out  
end SPI;
```

Q1: Do we have any problems having correct LEDs lid ?

Q2: Are LEDs stable ?

Solutions

solution to clock mismatch

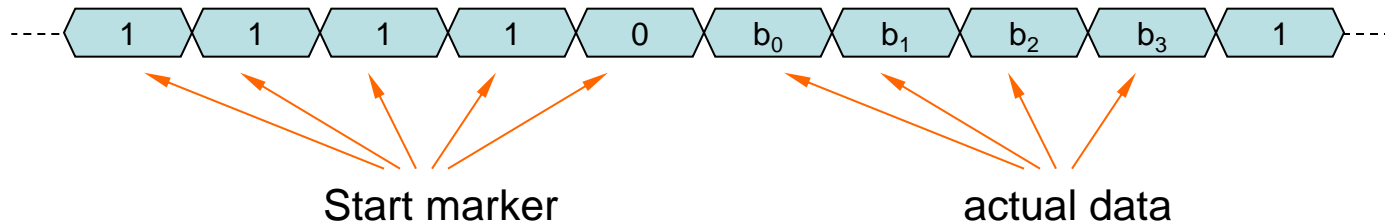


solution to bit number ambiguity

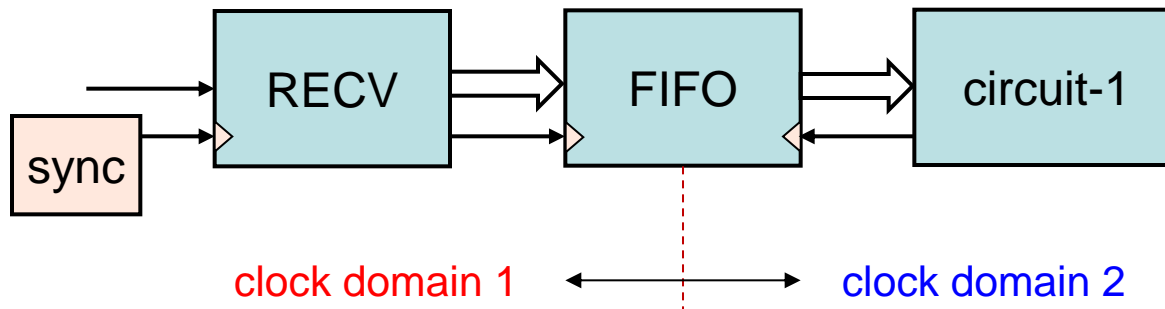
use frame/word synchronization markers



example



When the rest of the receiver works with another clock



When the frame/word marker is actually a data word

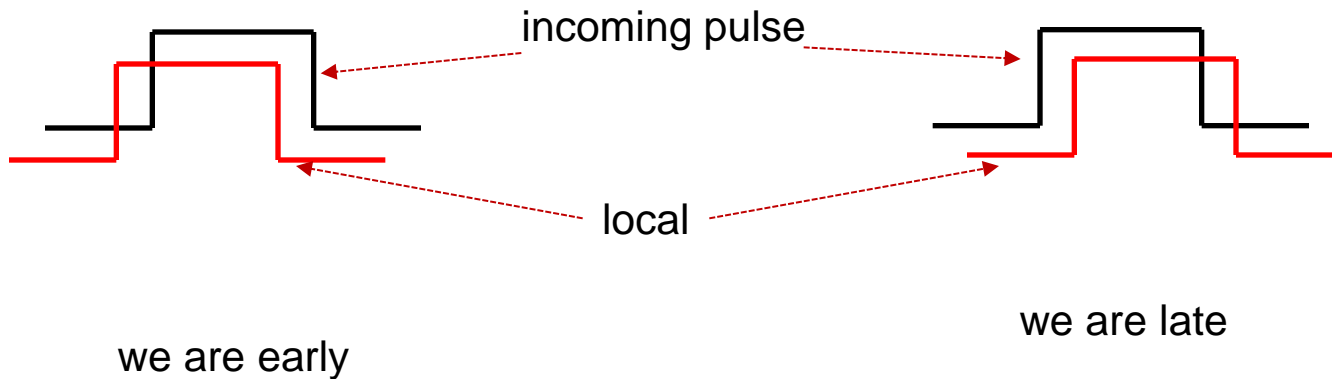
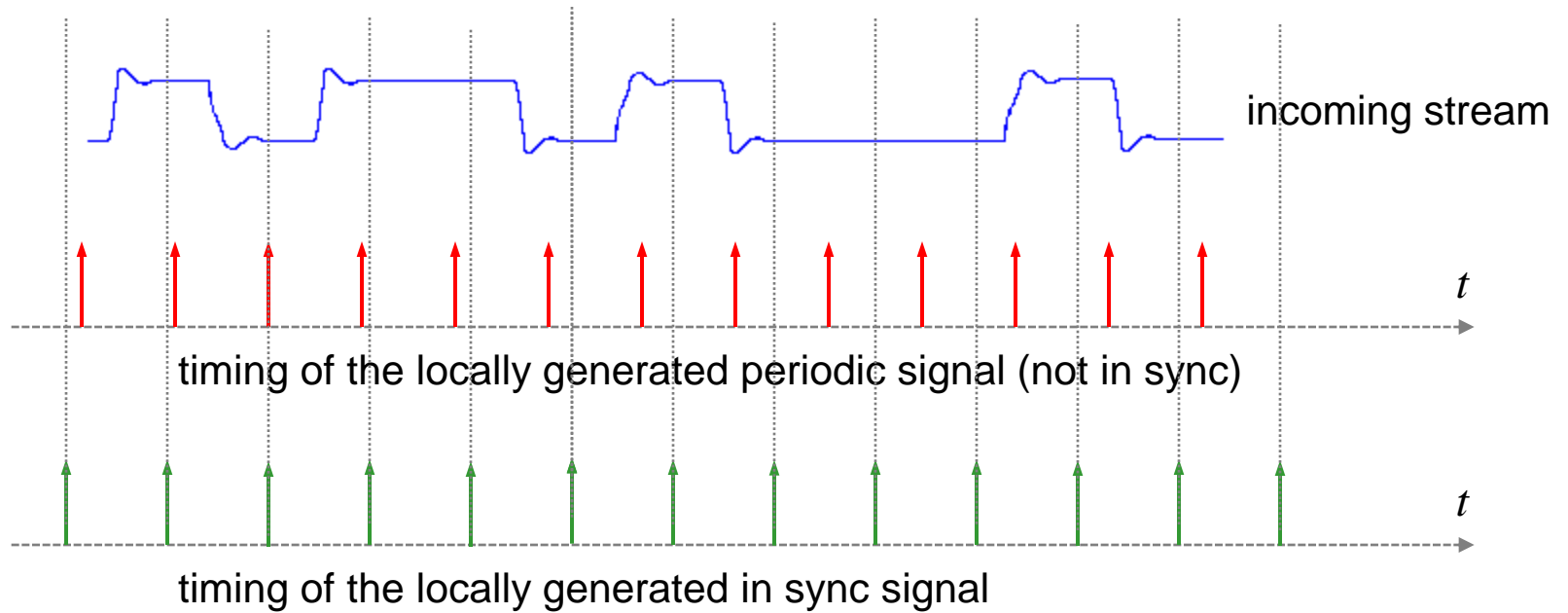
That happens.!

A careful frame design and transmit/receive protocol are required.
Sometimes upper level sync words are inserted to the stream.

Q : Did you notice that problems are almost always on the receiver side?

Symbol Synchronization

The aim is to locally generate a signal that is synchronous to the incoming stream



Correlation

is a measure of similarity of two signals (in signal processing)

$\psi(t)$ and $r(t)$

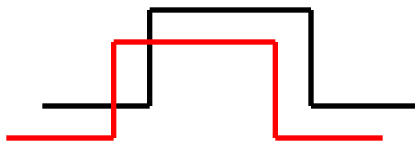
received signal

local signal

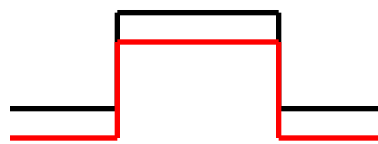
$$R_{\psi} = \int r(t)\psi(t)dt$$

is also named as inner product

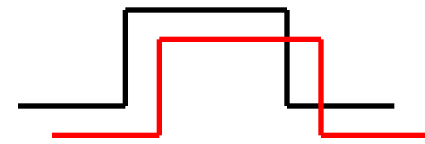
$$\langle r(t), \psi(t) \rangle$$



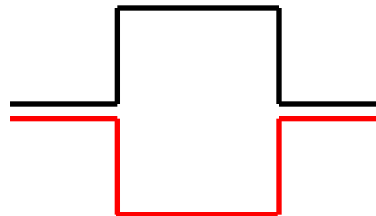
low correlation



high correlation



low correlation

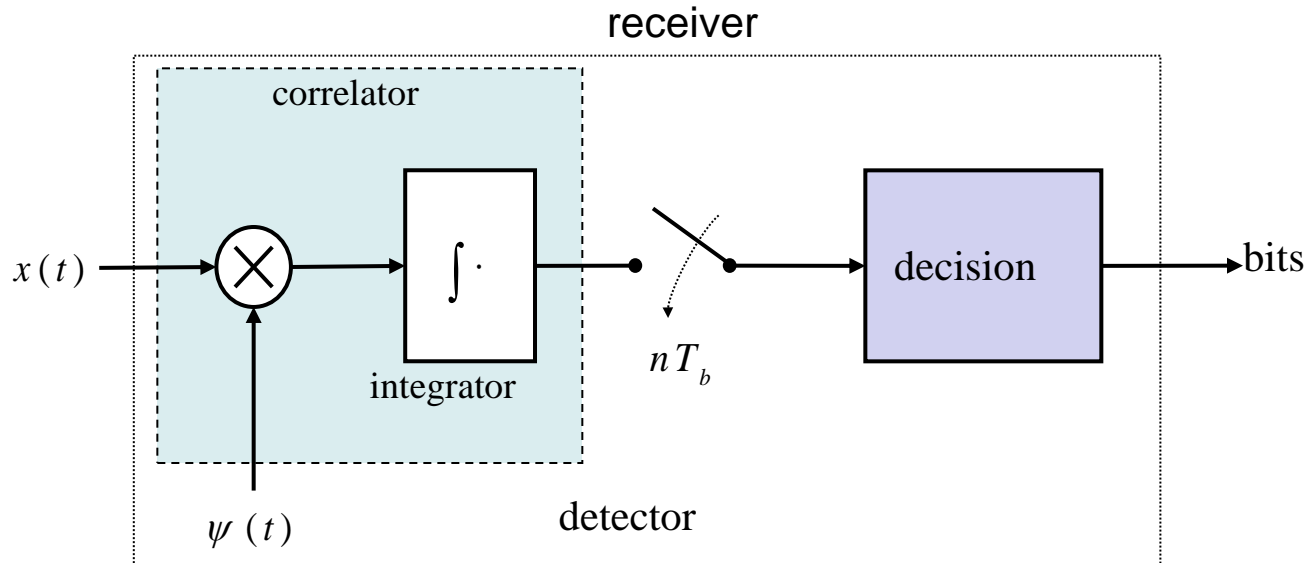


negative high correlation

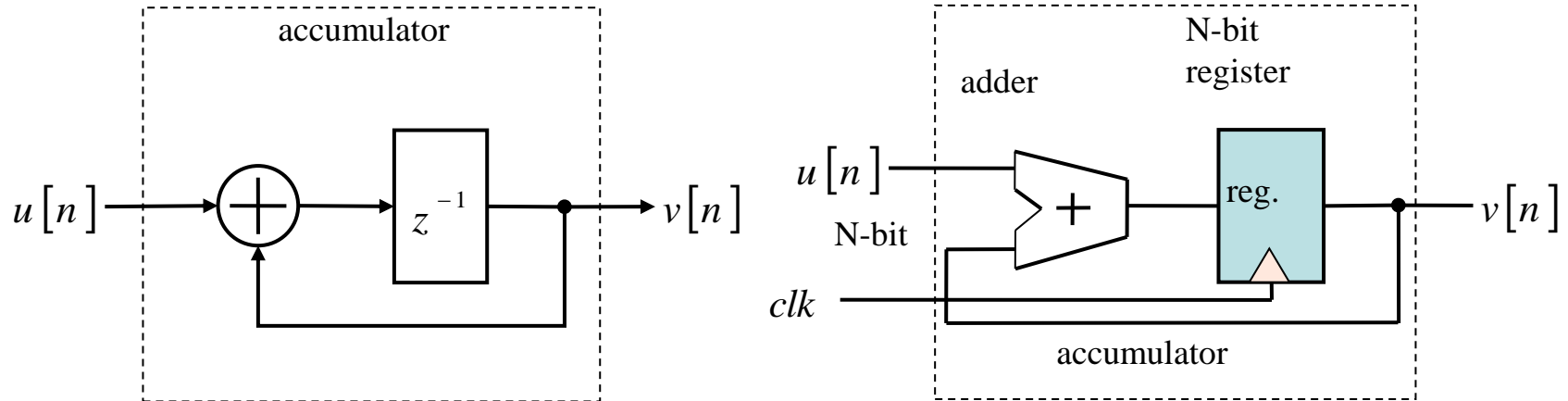
correlation can be used to adjust local oscillator

we are to measure similarity for the symbol period (so that we can change it)

$$R_{\psi} = \int_{\alpha}^{\alpha + T_b} r(t)\psi(t) dt$$



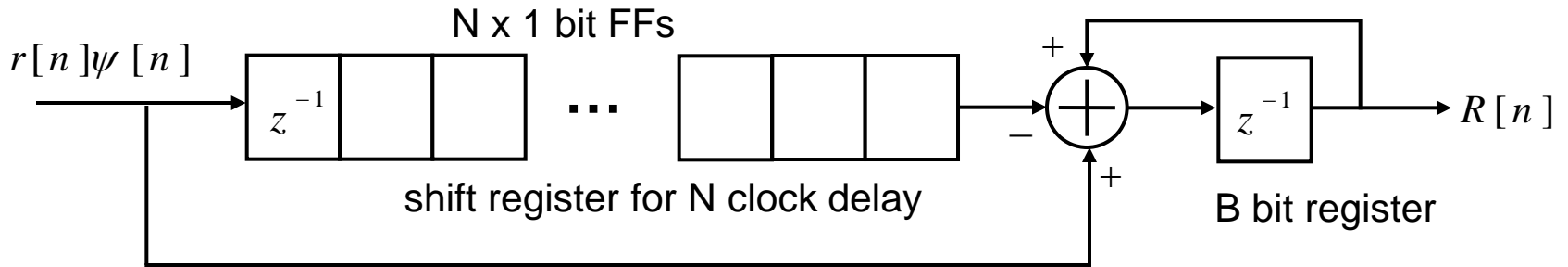
Implementation with Digital Circuits



accumulator (discrete integrator) is as simple as a summation in VHDL

```
ACC: process (clkH) is begin
    if (rising_edge (clkH)) then
        R <= R + u;
    end if;
end process;
```

Integration for a Symbol Period



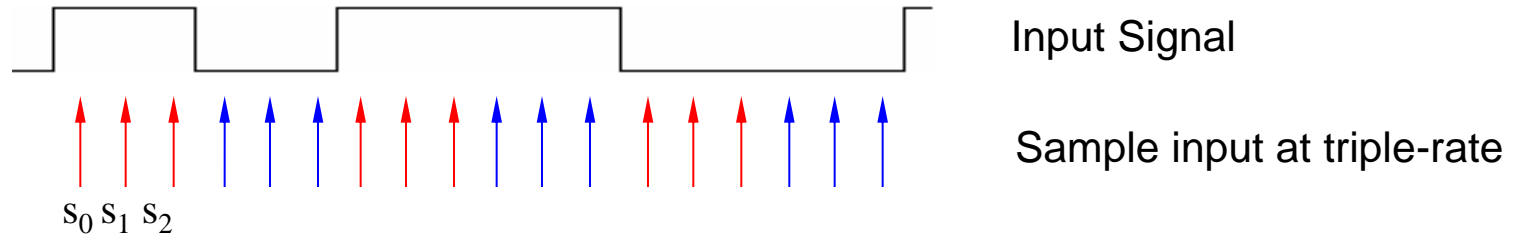
$$R_{\psi} = \int_{\alpha}^{\alpha + T_b} r(t)\psi(t)dt \quad \Longrightarrow \quad R[n] = \sum_{i=n}^{N+n} r[i]\psi[i]$$

```

INTEGRATOR: process(clkH) is begin
  if(rising_edge(clkH)) then
    R <= R + ux - D[N-1];
    D[N-1] <= D[N-2];
    ...
    D[1] <= D[0];
    D[0] <= ux;
  end if;
end process;

```

An Ad-Hoc Method for Synchronization



Cases for s_i 's:

1. All s_i are the same : No problem, continue at the same rate.
2. s_0 is different : Clock is fast, so slow down (ignore next clock pulse)
3. s_2 is different : Clock is slow, so move faster (use s_2 as s_0 for the next bit period)

Take s_1 as the bit value for all three cases.

Notes : Technique is simple and easy to design for digital systems like FPGAs.

It requires that the input to be very clean. With a little more effort a two sample per bit method can be developed.

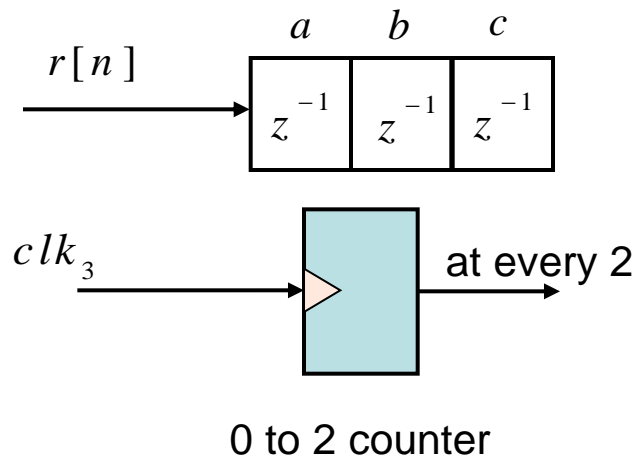
If the signal is noisy, multisample digital or analog integrator may be required.

An Ad-Hoc Method for Synchronization

```
architecture SerialRec of SerialRec is
    signal s0 : STD_LOGIC := '0';
    signal cntr : integer range 0 to 2 := 0;
    signal dummpulse : STD_LOGIC := '0';
begin
    process(clk) is begin
        if(RISING_EDGE(clk)) then
            if(dummpulse='1') then
                dummpulse <= '0'; cntr <= 0;
            else
                if(cntr=0) then
                    s0 <= SDin; oclk <= '0'; cntr <= 1;
                elsif(cntr=1) then
                    SDout <= SDin; oclk <= '0'; cntr <= 2;
                else
                    if(s0/=SDout) then -- Data is slow (clock is fast)
                        dummpulse <= '1'; -- ignore next clock pulse
                    elsif(SDin/=SDout) then -- Data is fast (clock is slow)
                        s0 <= SDin; cntr <= 1;
                    else
                        cntr <= 0;
                    end if;
                    oclk <= '1';
                end if;
            end if; --dummpulse
        end if; -- clk
    end process;
end SerialRec;
```

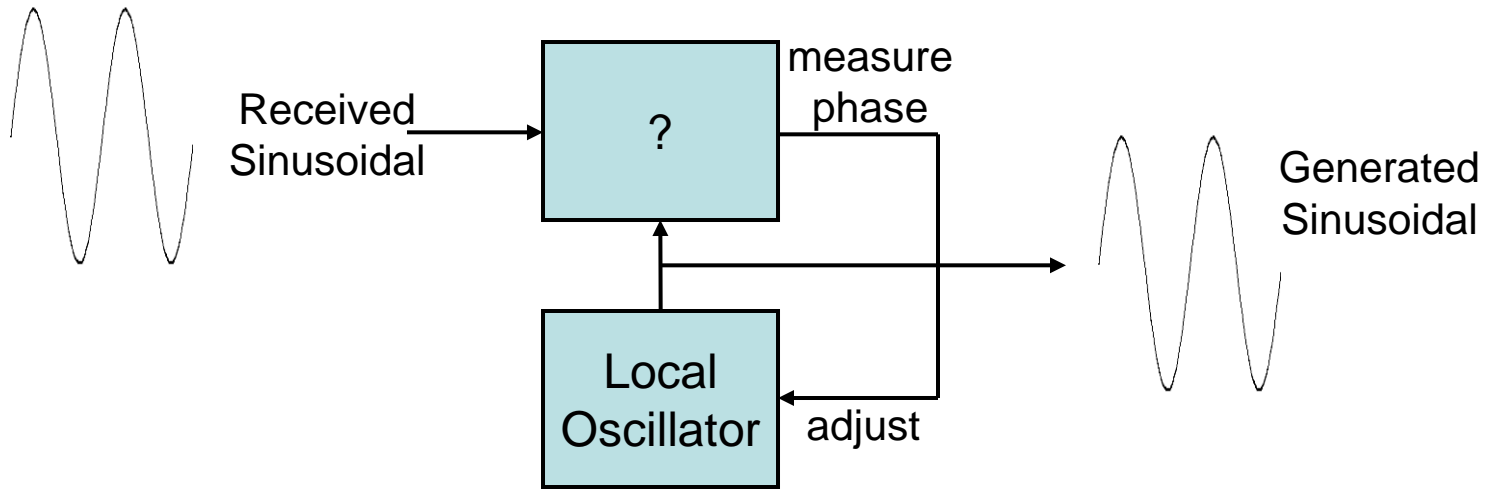
```
entity SerialRec is Port (
    -- three clock per bit
    clk : in STD_LOGIC;
    SDin : in STD_LOGIC;
    SDout : inout STD_LOGIC;
    -- one pulse per output bit
    oclk : out STD_LOGIC );
end SerialRec;
```

Another Ad-Hoc Method

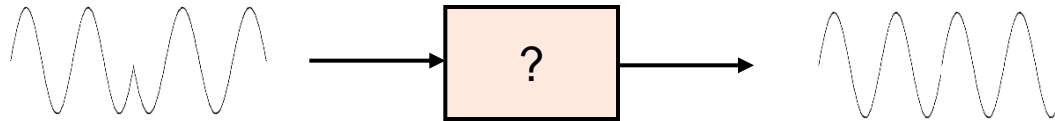


a	b	c	decision	cntr correction
0	0	0	0	-
0	0	1	0	increment cntr
0	1	0	0	x
0	1	1	1	decrement cntr
1	0	0	0	decrement cntr
1	0	1	1	x
1	1	0	1	increment cntr
1	1	1	1	-

Simple Problem

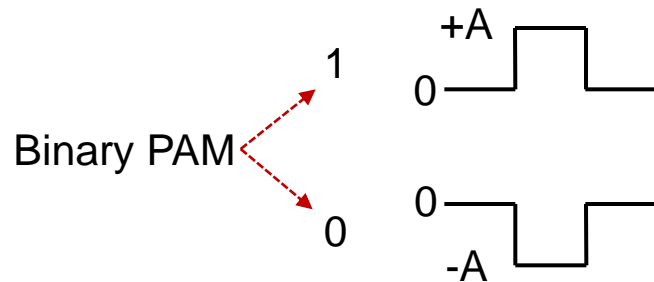


better yet

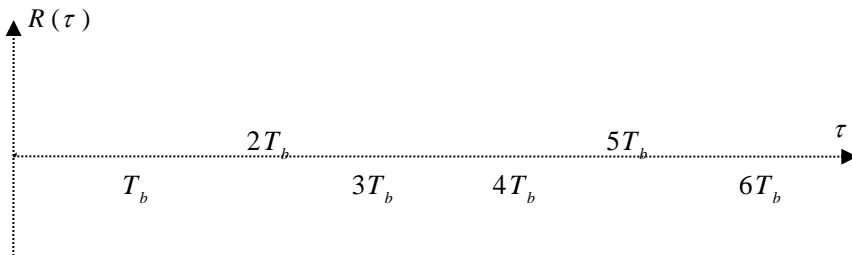
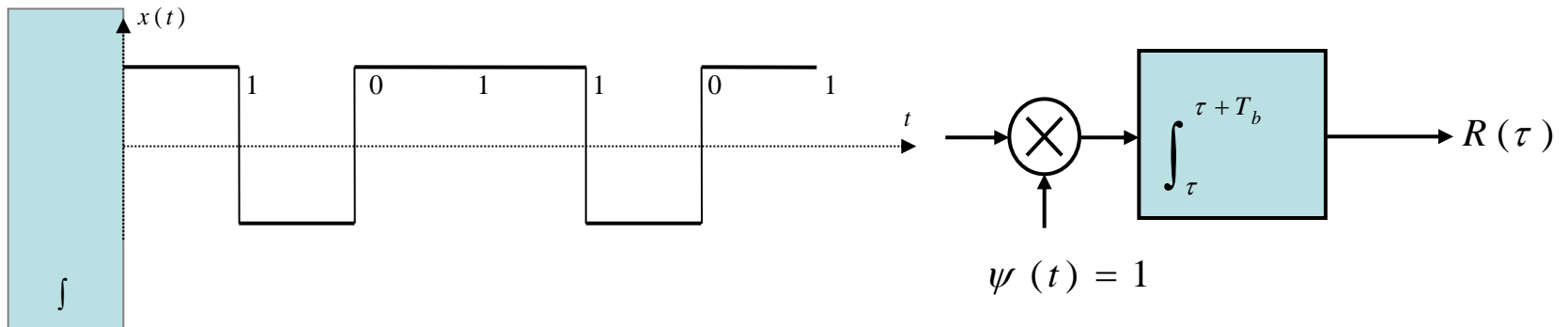


We need to generate required waveforms locally

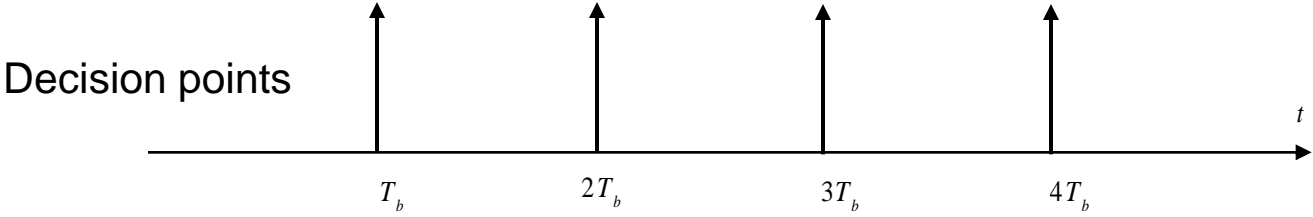
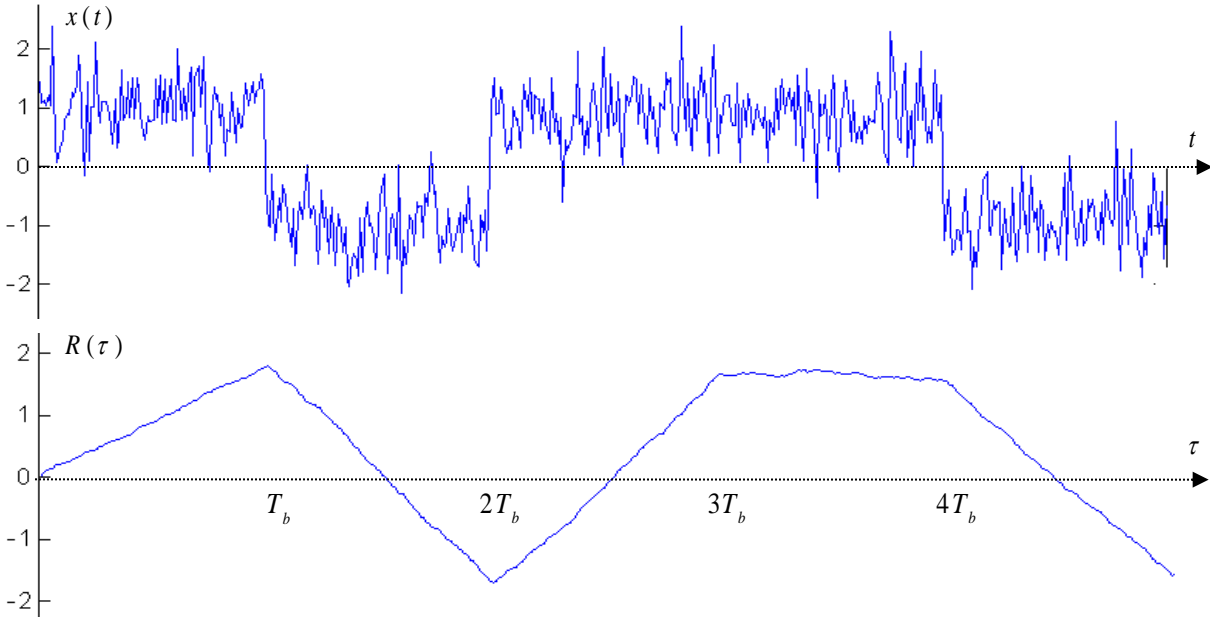
Start with Binary PAM (no waveform generation)



Channel signal is composed of these

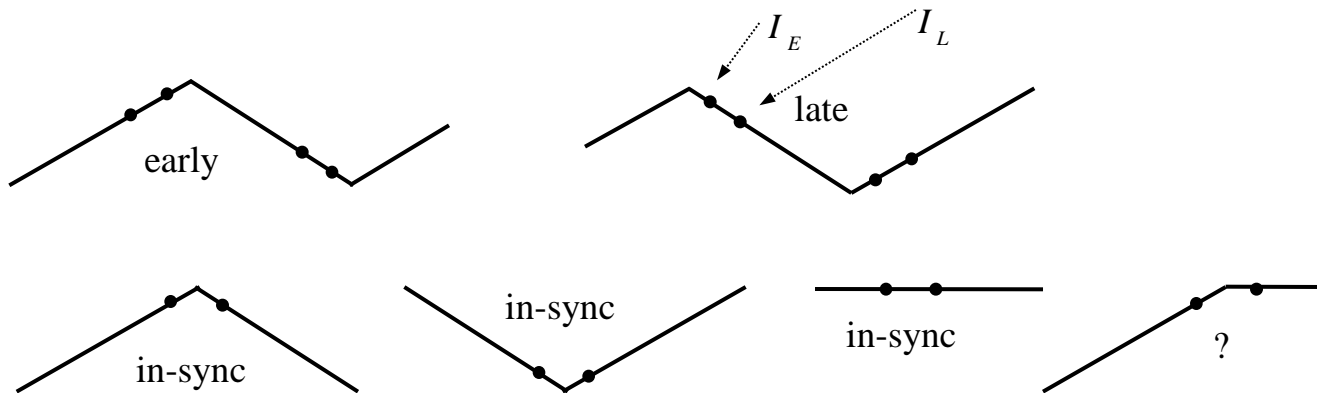
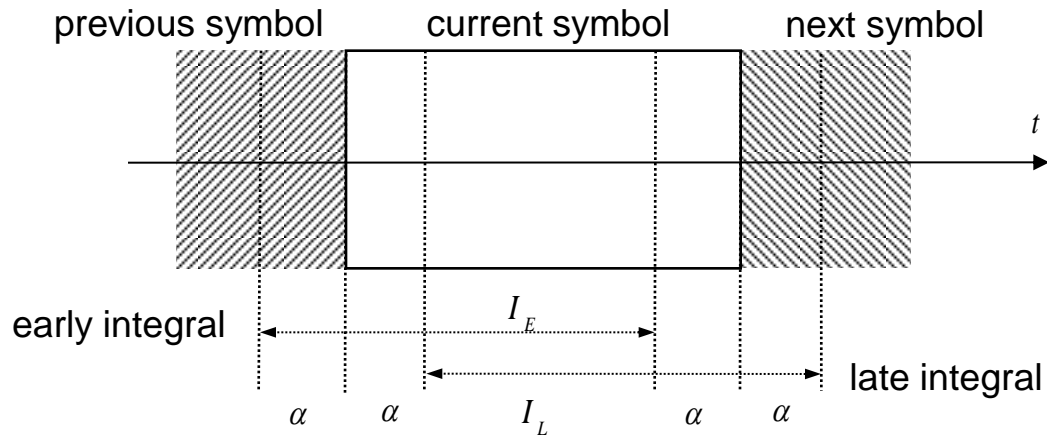


Noisy Signal Case



Having synchronization is equivalent to correct detection of symbols

Early-Late Gating

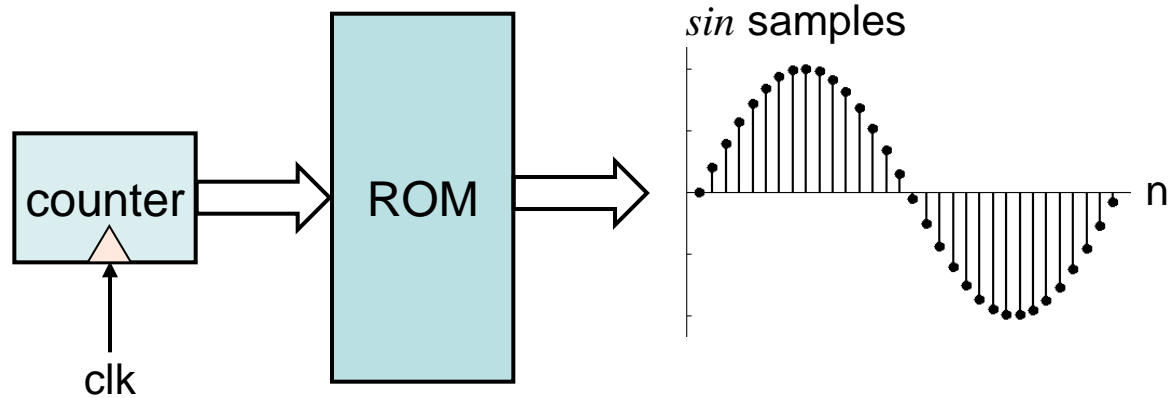


$|I_L| > |I_E|$: we are early

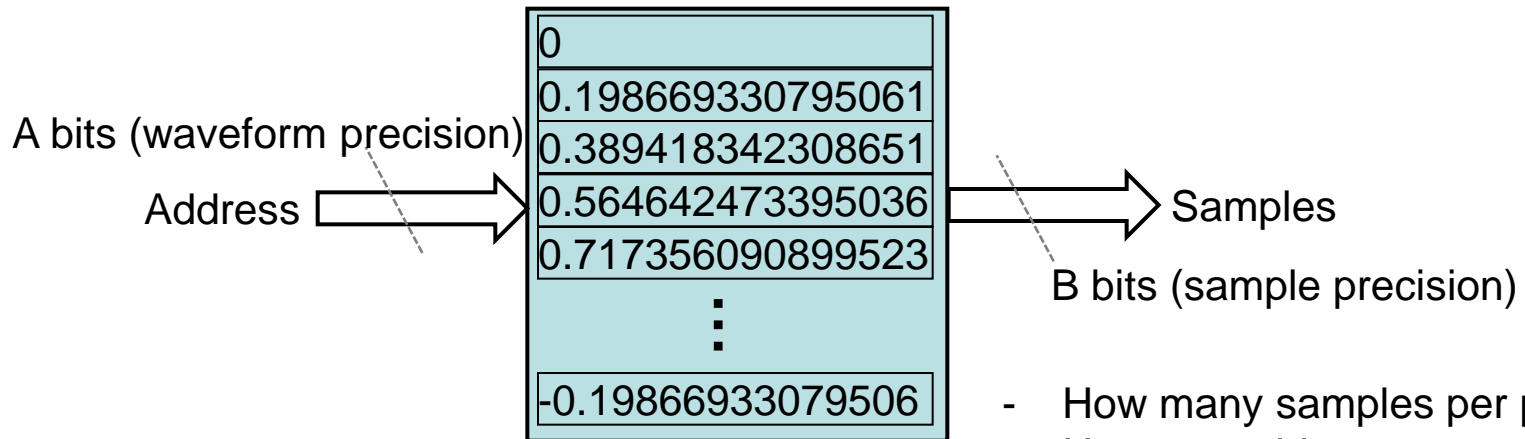
$|I_L| < |I_E|$: we are late

$d = |I_e| - |I_g|$ can be used to adjust local clock (oscillator)

Waveform generation



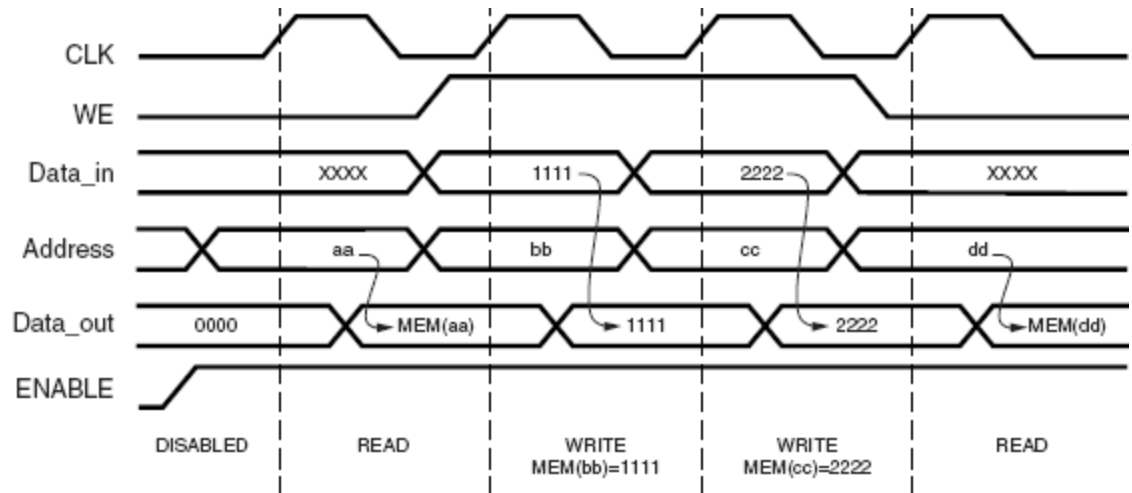
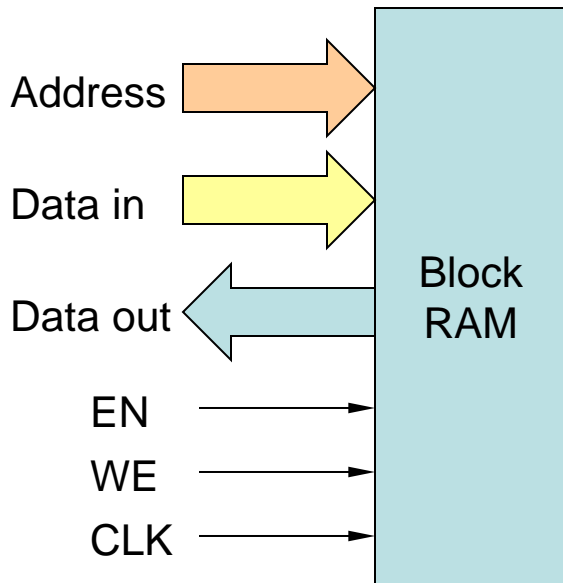
LUT filled with *sin* values



- How many samples per period?
- How many bits per sample?
- How Frequency/Phase controlled?

BRAM Primitives in FPGAs

There are several RAM blocks in Xilinx FPGAs including Spartan-3E (and in other vendors' too)

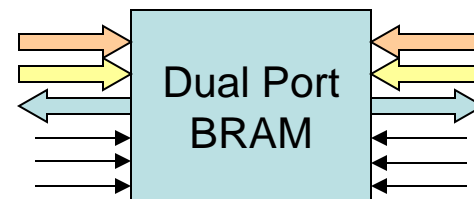
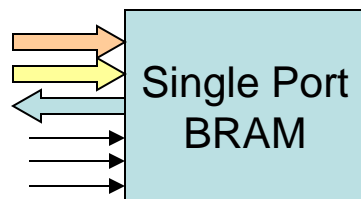
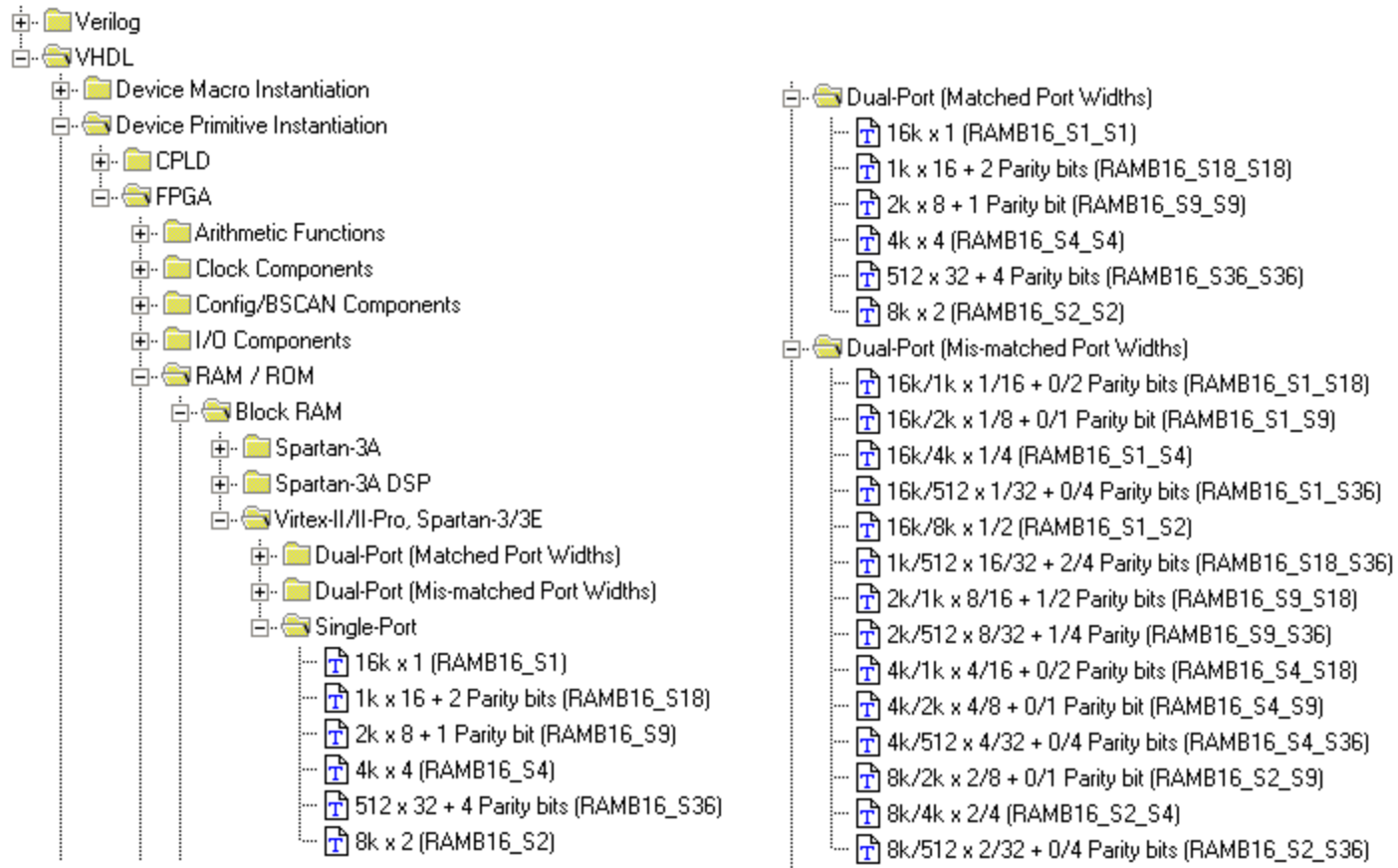


X463_12_020503

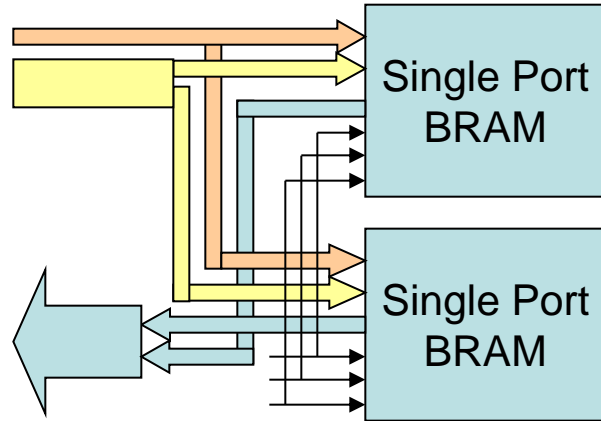
Figure 4-12: WRITE_FIRST Mode Waveforms

Available Configurations in Spartan-3E

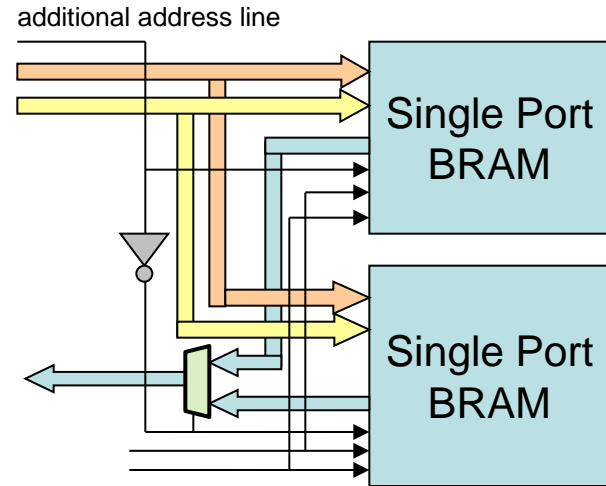
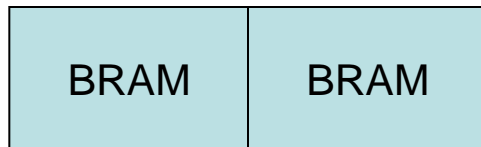
Block RAM instantiation templates can be seen using *Language Templates*



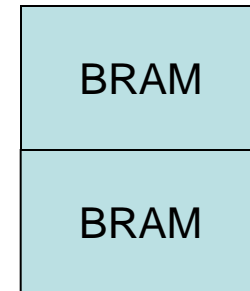
Combining / Extending BRAMs



Data Bus (Horizontal) Extension



Vertical Extension (more addresses)



Defining RAMs with Inference

```
entity DualPortRAM is
    Port ( ADDR_A   : in  STD_LOGIC_VECTOR (6 downto 0);
          DATAIN  : in  STD_LOGIC_VECTOR (7 downto 0);
          WE       : in  STD_LOGIC;
          CLK      : in  STD_LOGIC;
          DATAO_A : out STD_LOGIC_VECTOR (7 downto 0);
          ADDR_B   : in  STD_LOGIC_VECTOR (6 downto 0);
          DATAO_B : out STD_LOGIC_VECTOR (7 downto 0));
end DualPortRAM;
architecture Behavioral of DualPortRAM is
    type TDPRAM is array (0 to 127) of std_logic_vector (7 downto 0);
    signal MRAM : TDPRAM;
    ...
end architecture;
```

```
process(CLK) begin
    if (rising_edge(CLK)) then
        if (WE = '1') then
            MRAM(conv_integer(ADDR_A)) <= DATAIN ;
        end if;
        DATAO_A <= MRAM(conv_integer(ADDR_A));
        DATAO_B <= MRAM(conv_integer(ADDR_B));
    end if;
end process;
```

Synthesizer uses appropriate BRAMS if it can

Sample ROM

```
entity SinSamples is
    Port ( CLK : in  STD_LOGIC;
          DATA : out STD_LOGIC_VECTOR (7 downto 0));
end SinSamples ;

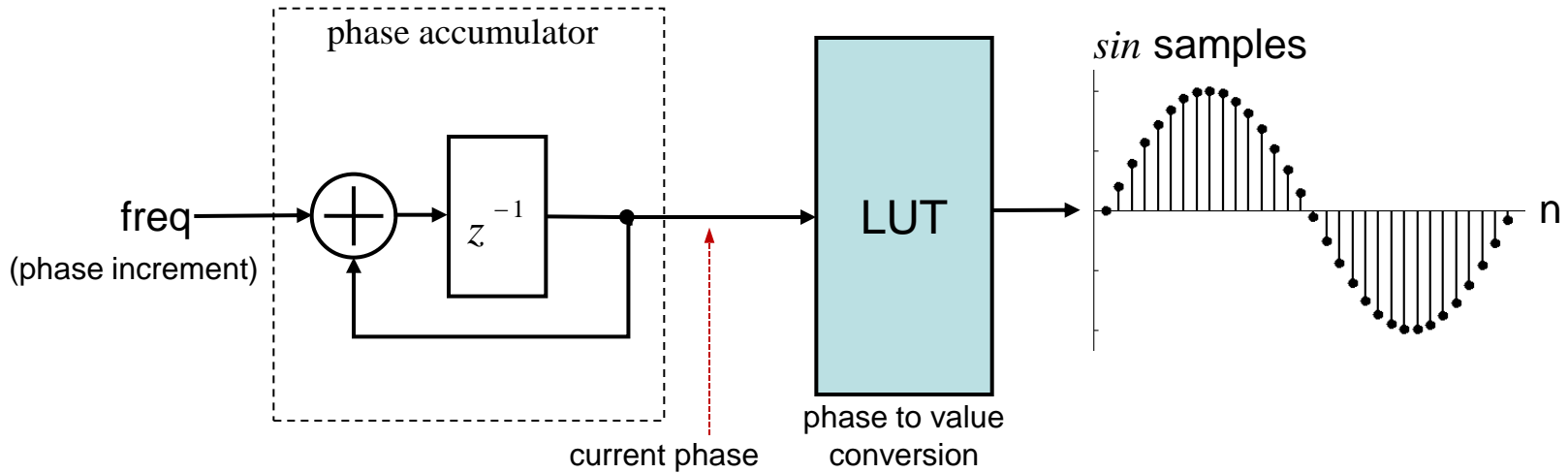
architecture Behavioral of SinSamples is
    type TDPRAM is array (0 to 31) of integer range -127 to 127;
    constant LUT: TDPRAM := {0,25,49,72,91,107,118,125,127,124,115,
                             103,86,65,43,18,-7,-32,-56,-78,-96,-111,-121,-126,
                             -127,-122,-112,-98,-80,-59,-35,-11};
    signal ADDR: integer range 0 to 31;
begin
    process(CLK) begin
        if (rising_edge(CLK)) then
            DATA <= conv_std_logic_vector(LUT(ADDR));
            ADDR <= ADDR+1;
        end if;
    end process;
end;
```

Variable Frequency

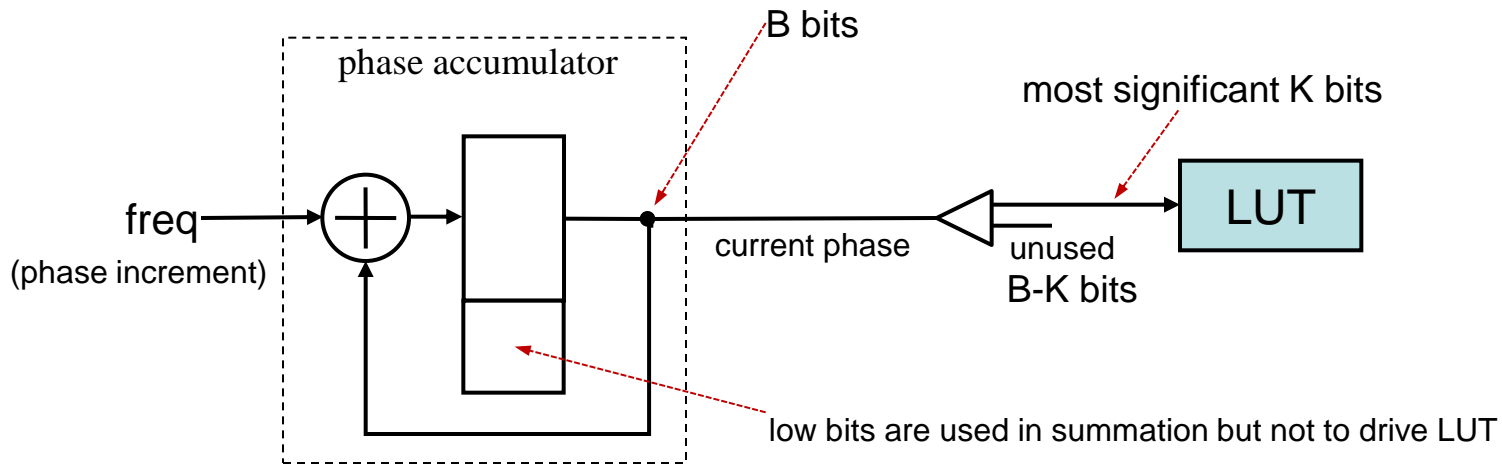
```
entity SinSamples is
    Port ( Incr : in  STD_LOGIC_VECTOR (4 downto 0);
          CLK  : in  STD_LOGIC;
          DATA : out STD_LOGIC_VECTOR (7 downto 0));
end SinSamples ;

architecture Behavioral of SinSamples is
    type TDPRAM is array (0 to 31) of integer range -127 to 127;
    constant LUT: TDPRAM := {0,25,49,72,91,107,118,125,127,124,115,
                             103,86,65,43,18,-7,-32,-56,-78,-96,-111,-121,-126,
                             -127,-122,-112,-98,-80,-59,-35,-11};
    signal ADDR: integer range 0 to 31;
begin
    process(CLK) begin
        if (rising_edge(CLK)) then
            DATA <= conv_std_logic_vector(LUT(ADDR));
            ADDR <= ADDR + conv_integer(Incr);
        end if;
    end process;
end;
```

LUT Based Controlled Frequency Waveform Generator



Maintain phase accuracy



END