ECE385
DIGITAL SYSTEMS LABORATORY

Experiment 8
PS/2 Keyboard and VGA Interface

Deming Chen

Slides mainly adapted from those designed by Prof. Rakesh Kumar and Prof. Janak Patel

Department of Electrical and Computer Engineering

University of Illinois at Urbana-Champaign

Today’s Topics

- **Quick review of Exp. 7**
- **Experiment 8 (due next week)**
  - PS/2 Keyboard Description
  - Hardware interface for inputting from a Keyboard
  - VGA Description
  - Hardware interface for outputting to a VGA Monitor

Review: 2’s complement 8-bit multiplication

```
          00000111                 7 (multiplicand)
        x 11000101          x (-)59 (multiplier)
          00000111             (-)413
        +00000000x
       +00000111xx
      +00000000xxx
     +00000000xxxx
    +00000000xxxxx
   +00000111xxxxxx
  -00000111xxxxxxx    Subtract
  111111100110011
```

May require only 1 16-bit register (or 2 8-bit registers) for 8-bit multiplication

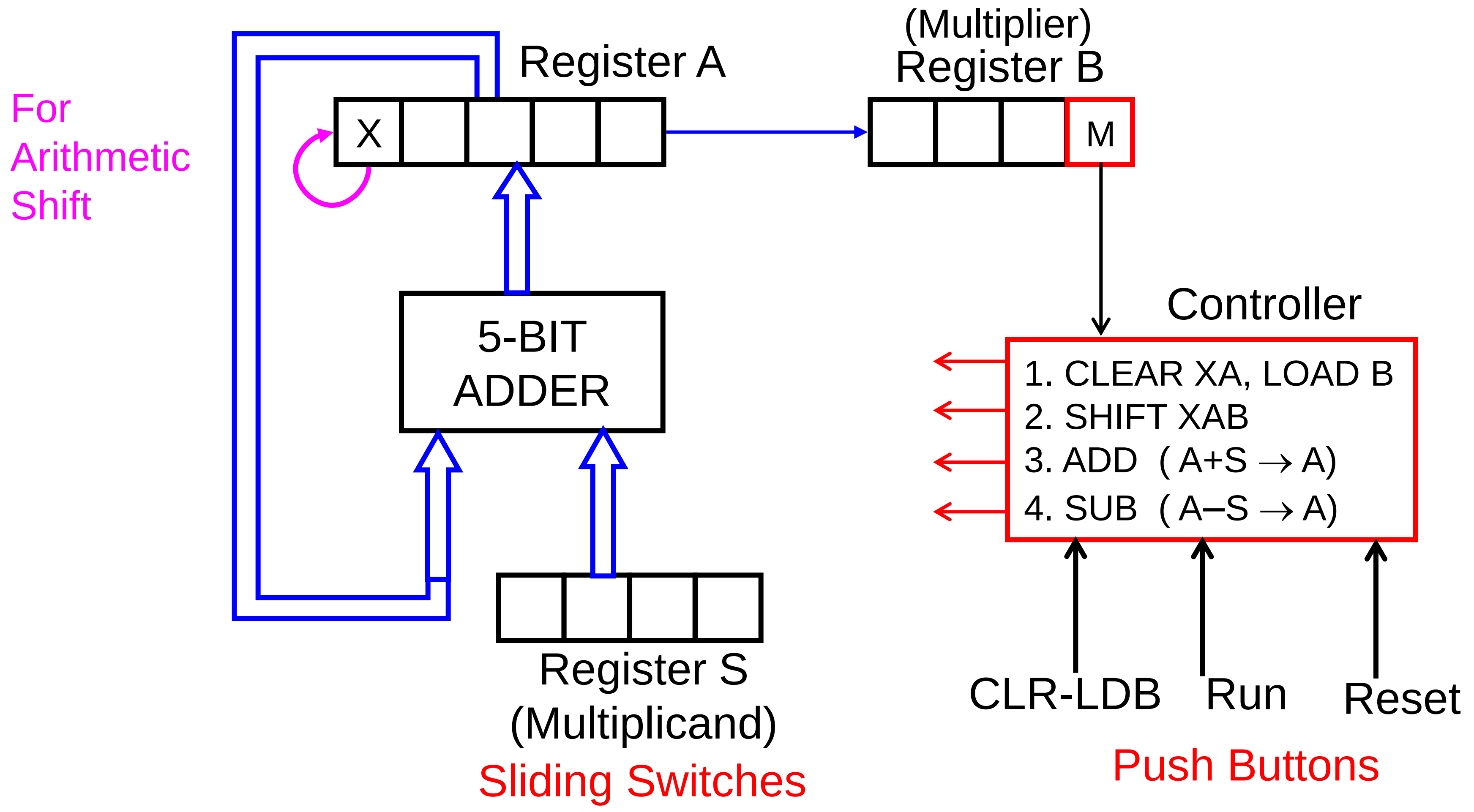
May requires only 8 “time units” for 8-bit multiplication

Review: Add-Shift Algorithm illustrated

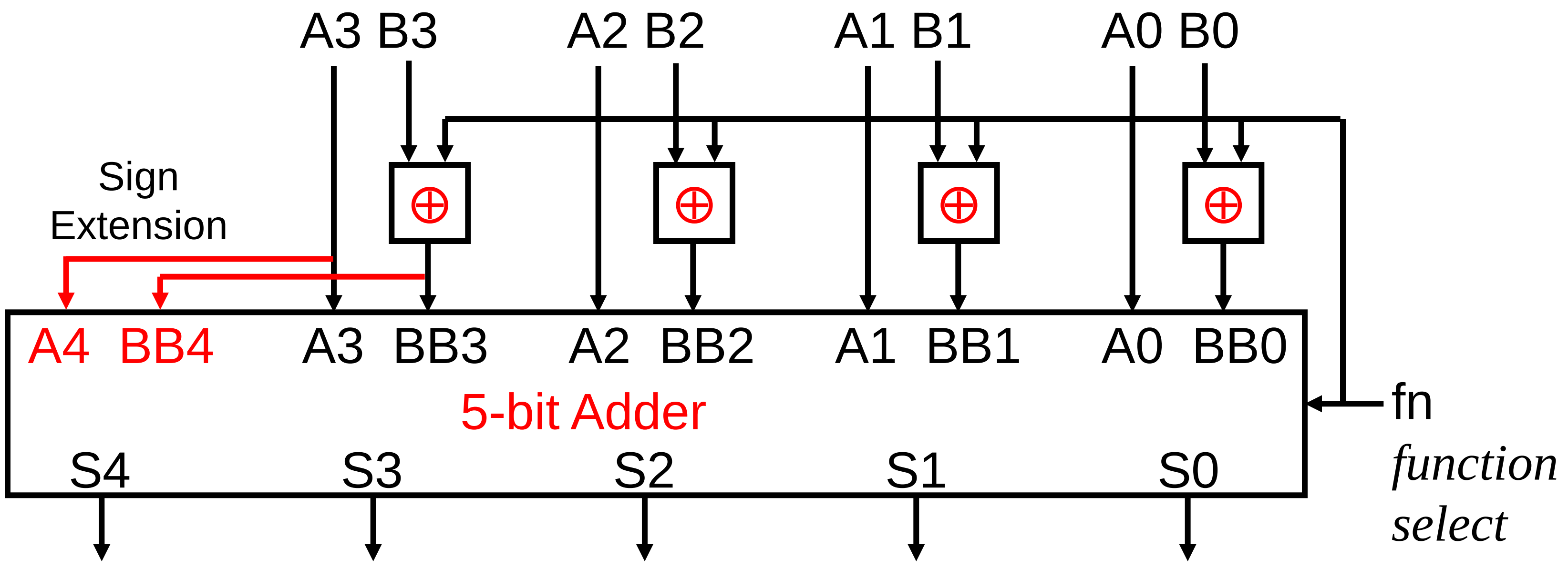
Multiply 0000 0111 × 1100 0101

Function	X	$A_{7\text{-}0}$	$B_{7\text{-}0}$	M	Next Step in words
ClearA, LoadB	0	0000 0000	*11000101*	B_0 1	Since M = 1, multiplicand (available from switches S) will be added to A.
ADD	0	0000 0111	*11000101*	1	Shift XAB by one bit after ADD complete
SHIFT	0	0000 0011	1 *1100010*	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	0	0000 0001	11 *110001*	1	Add S to A since M = 1.
ADD	0	0000 1000	11 *110001*	1	Shift XAB by one bit after ADD complete
SHIFT	0	0000 0100	011 *11000*	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	0	0000 0010	0011 *1100*	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	0	0000 0001	00011 *110*	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	0	0000 0000	100011 *11*	1	Add S to A since M = 1
ADD	0	0000 0111	100011 *11*	1	Shift XAB by one bit after ADD complete
SHIFT	0	0000 0011	1100011 *1*	1	Subtract S from A since 8th bit M = 1.
SUB	1	1111 1100	1100011 *1*	1	Shift XAB after SUB complete
SHIFT	1	1111 1110	01100011	1	8th shift done. Stop. 16-bit Product in AB.

Review: 4-bit Multiplier Block Diagram



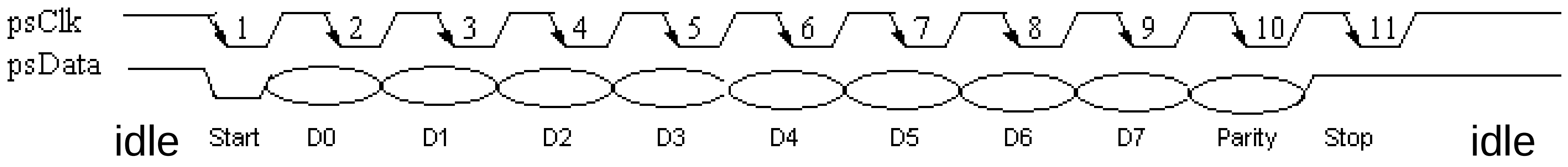
Review: 4-bit Adder-Subtractor



```
entity ADD_SUB5 is
  port (A,B : in std_logic_vector (3 downto 0);
        S : out std_logic_vector (4 downto 0);
        fn : in std_logic);
end entity;
```

Experiment 8: PS/2 Serial Protocol

- **The keyboard/mouse always generate the clock signal.**
  - All data is transmitted one byte at a time and each byte is sent in a **frame** consisting of 11 bits. These bits are:
  - 1 start bit. This is always 0.
  - 8 data bits, least significant bit first
  - 1 parity bit (odd parity) *You can ignore it for Expt. 8.*
  - 1 stop bit. This is always 1.
- **A data frame is exactly 11 trailing edges of the psClk**
  - Between successive frames there can be arbitrary idle period



psClk frequency is between 10KHz to 16.7KHz

Keyboard Scan Codes

- **Most key strokes send three bytes (11 bits each)**
  - A byte of Make-Code (also called Scan-Code)
  - Two bytes of Break-Code (release key)
  - A key code xy is sent as 3 (or more) bytes: xy-F0-xy
    - Key “E” is Hex 24–(F0–24) (makecode-breakcode)
      - – If the key is kept pressed, makecode will be sent continuously
    - Key “7” is Hex 3D–F0–3D
  - “Extend” key code xy is 5 bytes : E0-xy-E0-F0-xy
    - Key “↑” is Hex E0-75-E0-F0-75
    - Key “←” is Hex E0-6B-E0-F0-6B
  - F0 byte is always one before the LAST byte

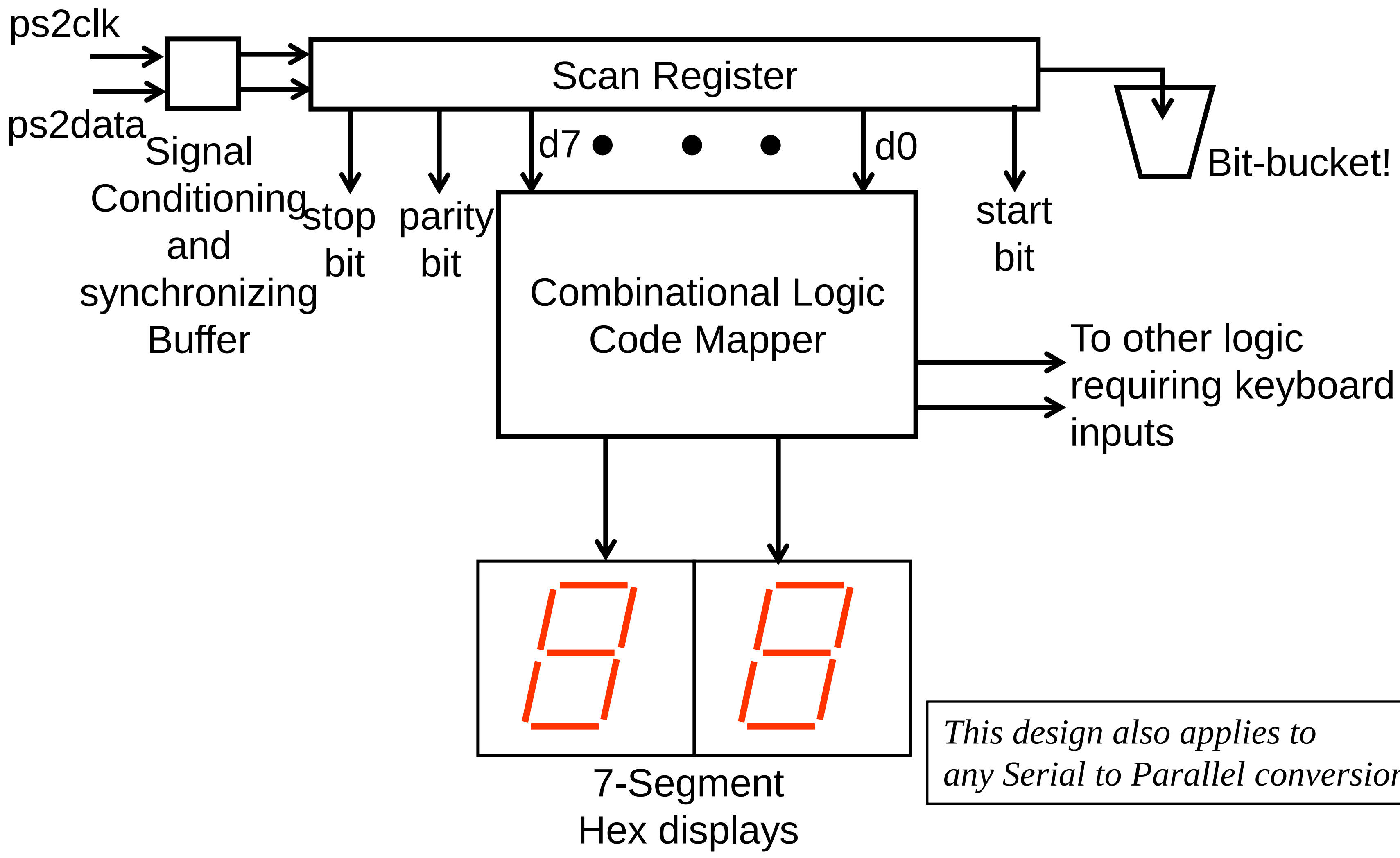
Keyboard Scan Set 2 (part 1)

KEY	MAKE	BREAK	KEY	MAKE	BREAK	KEY	MAKE	BREAK
A	1C	F0,1C	9	46	F0,46	[	54	FO,54
B	32	F0,32	`	0E	F0,0E	INSERT	E0,70	E0,F0,70
C	21	F0,21	-	4E	F0,4E	HOME	E0,6C	E0,F0,6C
D	23	F0,23	=	55	FO,55	PG UP	E0,7D	E0,F0,7D
E	24	F0,24	\	5D	F0,5D	DELETE	E0,71	E0,F0,71
F	2B	F0,2B	BKSP	66	F0,66	END	E0,69	E0,F0,69
G	34	F0,34	SPACE	29	F0,29	PG DN	E0,7A	E0,F0,7A
H	33	F0,33	TAB	0D	F0,0D	U ARROW	E0,75	E0,F0,75
I	43	F0,43	CAPS	58	F0,58	L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B	L SHFT	12	FO,12	D ARROW	E0,72	E0,F0,72
K	42	F0,42	L CTRL	14	FO,14	R ARROW	E0,74	E0,F0,74
L	4B	F0,4B	L GUI	E0,1F	E0,F0,1F	NUM	77	F0,77
M	3A	F0,3A	L ALT	11	F0,11	KP /	E0,4A	E0,F0,4A
N	31	F0,31	R SHFT	59	F0,59	KP *	7C	F0,7C
O	44	F0,44	R CTRL	E0,14	E0,F0,14	KP -	7B	F0,7B
P	4D	F0,4D	R GUI	E0,27	E0,F0,27	KP +	79	F0,79
Q	15	F0,15	R ALT	E0,11	E0,F0,11	KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D	APPS	E0,2F	E0,F0,2F	KP .	71	F0,71
S	1B	F0,1B	ENTER	5A	F0,5A	KP 0	70	F0,70

Keyboard Scan Set 2 (part 2)

KEY	MAKE	BREAK	KEY	MAKE	BREAK	KEY	MAKE	BREAK
T	2C	F0,2C	ESC	76	F0,76	KP 1	69	F0,69
U	3C	F0,3C	F1	05	F0,05	KP 2	72	F0,72
V	2A	F0,2A	F2	06	F0,06	KP 3	7A	F0,7A
W	1D	F0,1D	F3	04	F0,04	KP 4	6B	F0,6B
X	22	F0,22	F4	0C	F0,0C	KP 5	73	F0,73
Y	35	F0,35	F5	03	F0,03	KP 6	74	F0,74
Z	1A	F0,1A	F6	0B	F0,0B	KP 7	6C	F0,6C
0	45	F0,45	F7	83	F0,83	KP 8	75	F0,75
1	16	F0,16	F8	0A	F0,0A	KP 9	7D	F0,7D
2	1E	F0,1E	F9	01	F0,01	]	5B	F0,5B
3	26	F0,26	F10	09	F0,09	;	4C	F0,4C
4	25	F0,25	F11	78	F0,78	'	52	F0,52
5	2E	F0,2E	F12	07	F0,07	,	41	F0,41
6	36	F0,36	PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12	.	49	F0,49
7	3D	F0,3D	SCROLL	7E	F0,7E	/	4A	F0,4A
8	3E	F0,3E	PAUSE	E1,14,77, E1,F0,14, F0,77	-NONE-			

Hardware Block Diagram


ps2clk
ps2data
Signal Conditioning and synchronizing Buffer
Scan Register
stop bit
parity bit
d7
d0
start bit
Bit-bucket!
Combinational Logic Code Mapper
To other logic requiring keyboard inputs
7-Segment Hex displays

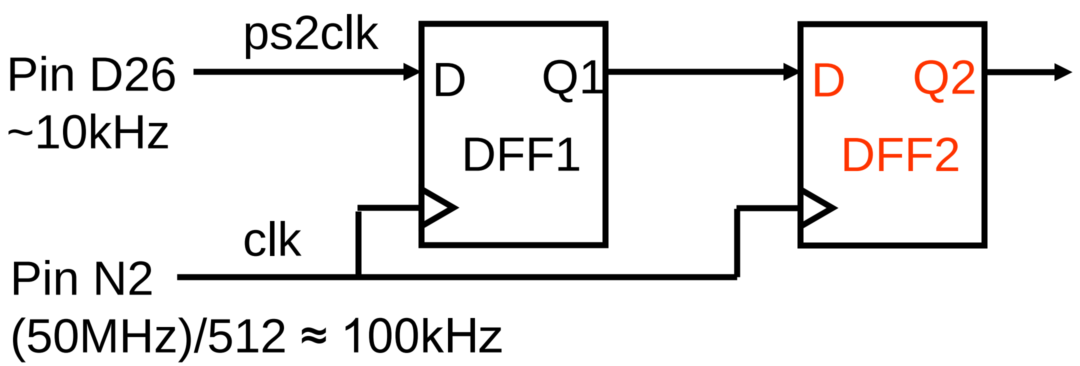
This design also applies to any Serial to Parallel conversion

A word about Clocks

- **DE2 default System clock is 50MHz (pin N2)**
  - This clock can be divided using counters and taking the output directly from a counter <u>flip-flop</u> as the divided clock signal
  - Any other data signal from combinational logic if used as clock will result in unreliable design and create synchronization problems

External Clocks as Data Input

- **External Clocks such as ps2clk can make ps2data capture simple but at the cost of complications arising from two unrelated clocks (do not use!)**
- **A better design will use only one clock and treat ps2clk as a data waveform that can be decoded for rising and falling edge as below**



Q2	Q1	psClk
1	0	Falling Edge
0	1	Rising Edge
0	0	Level 0
1	1	Level 1

Caution: *As given, DFF1 samples the ps2Clk waveform at 5,000 samples for each ps2Clk period. To reduce possible errors when the falling edge of ps2Clk is noisy, one should reduce the sampling rate. Divide the 50MHz by say 512 and use the divided clock for DFF1 and 2 and Shift operation.*

Hints for VHDL code

- **Scan shift register can be inferred by the synthesizer if you use clock-edge and describe shift operation on a signal bit-vector**
  - In other words, you need not create an entity
- **Map from one code to another can be accomplished by either**
  - “select” – “when” construct outside a process, or in “case” – “when” construct inside a process
- **Make sure to account for F0 bytes when looking for keystrokes**
  - Most common design error is “Framing Error” where incorrect byte is identified as the first byte

Hints for key recognition

- **Make sure to account for F0 bytes when looking for keystrokes**

Scenario 1: A….A…. (1C-1C-…-1C-F0-1C-1C-1C-…)

Scan Register	Scan Register	
1C	1C	=> Same key!

Scenario 2: A….D…. (1C-1C-…-1C-F0-1C-23-23-…)

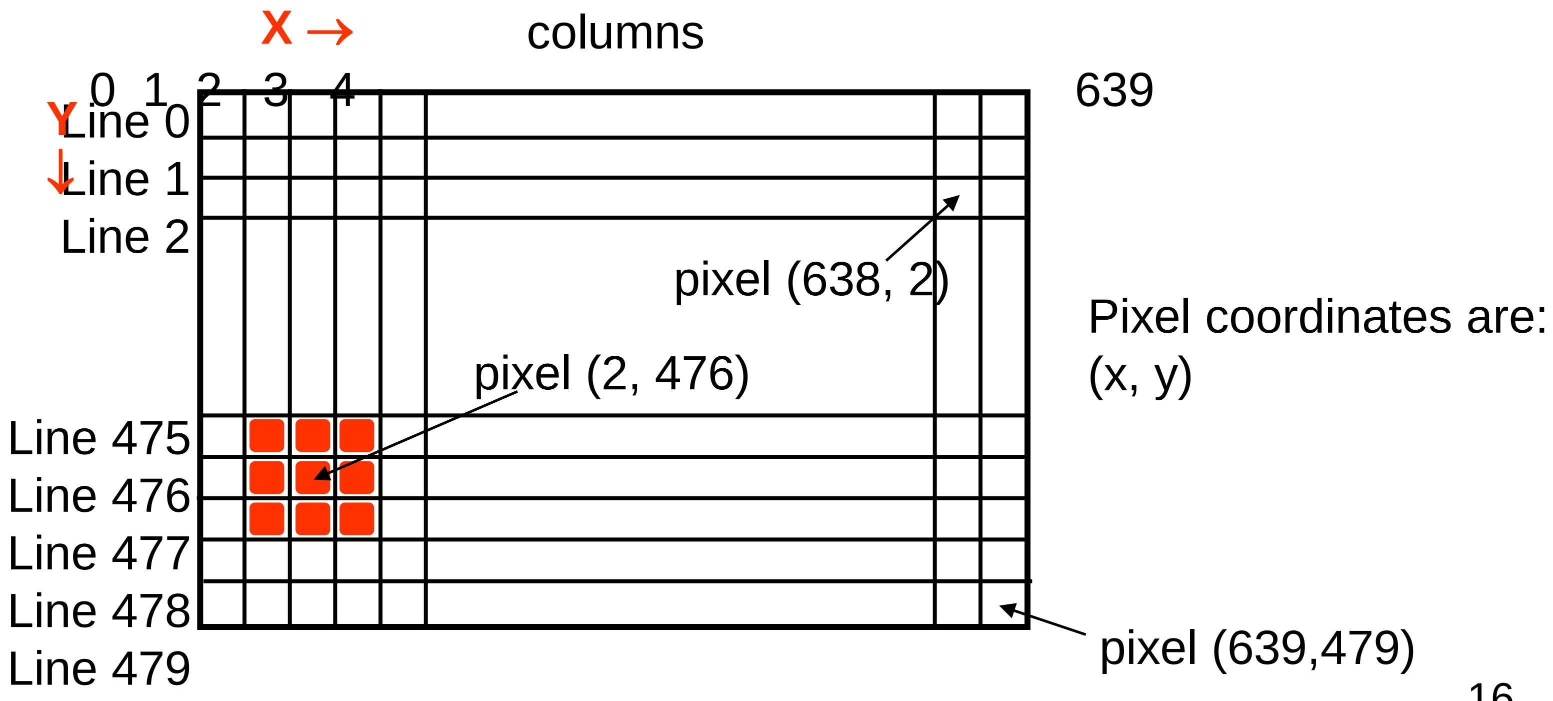
Scan Register	Scan Register	
23	1C	=> Different key!

Scenario 3: A…. (1C-1C-…-1C-F0-1C)

Scan Register	Scan Register	
1C	F0	=> Released key!

VGA Monitor Operation

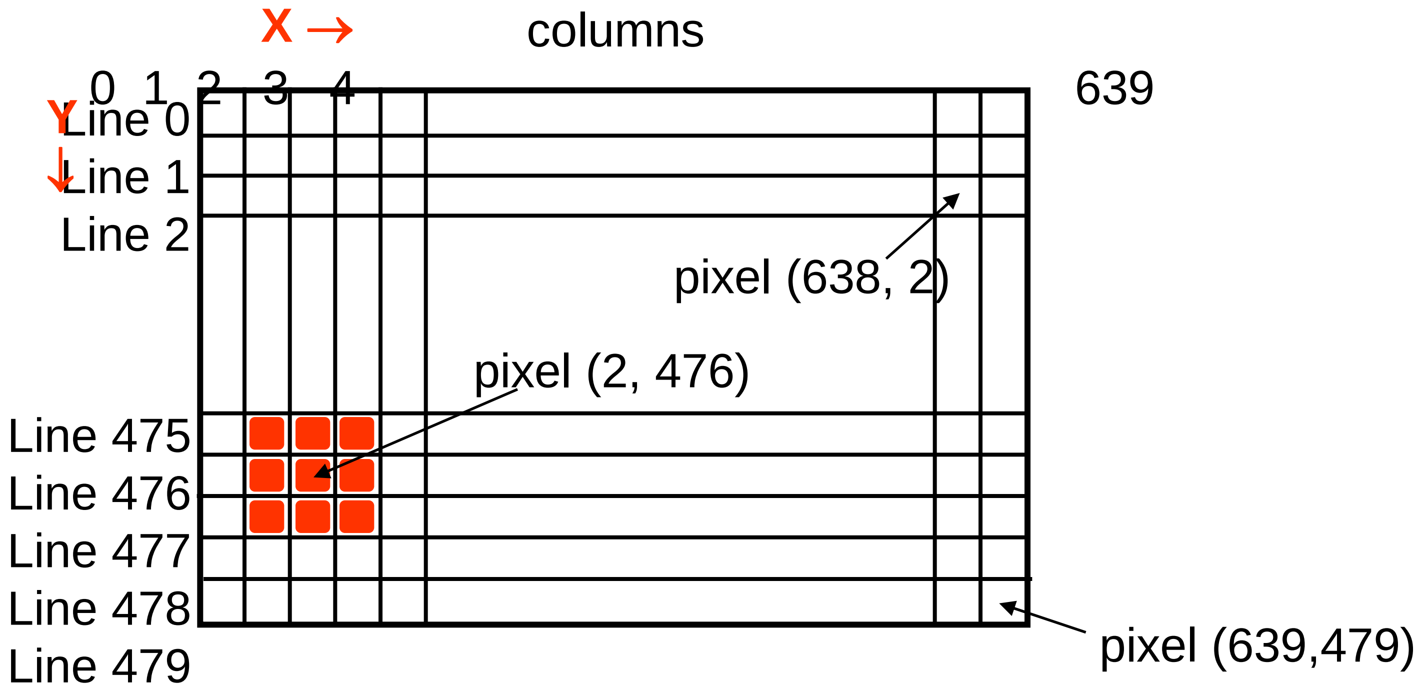
- **VGA (Video Graphics Array) Standard**
  - The screen is organized as a matrix of pixels
    - 640 horizontal pixels x 480 vertical lines
  - An Electron Beam “paints” each pixel from left to right in each row, and each row from top to bottom



Drawing a Shape

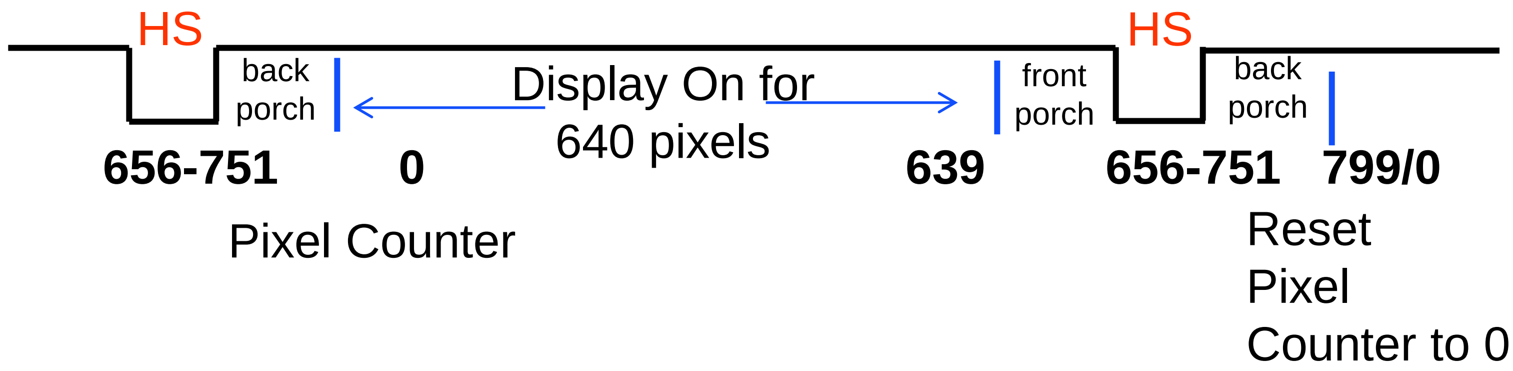
A Shape can be defined by specifying a boundary around a center. In the following example, the center is (2, 476) and the box is defined *Center ± Size*. For Size=1 all pixels in the box satisfy:

$(X \geq 2-1) \text{AND} (X \leq 2+1) \text{AND} (Y \geq 476-1) \text{AND} (Y \leq 476+1)$



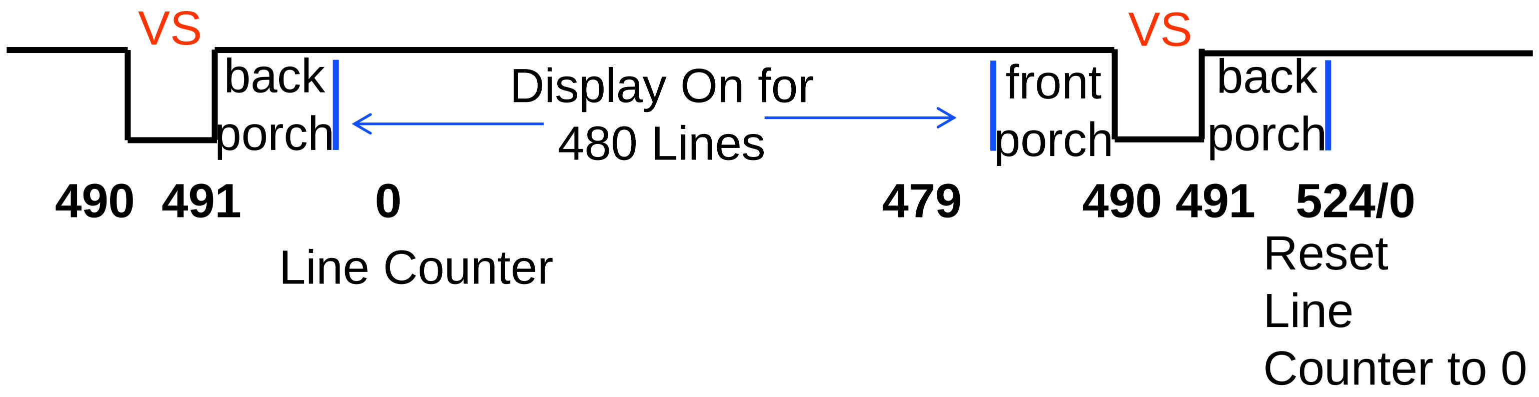
VGA Horizontal Timing

- **To generate a Horizontal Sync Pulse, use a 10-bit pixel counter modulo-800**
  - Counter increments with a 25MHz clock (pixel clock)
  - Pixel Counts <0 thru 639>: Display On
  - Pixel Counts <640 thru 799>: Display Off
  - Pixel Count <656 thru 751>: HS Pulse Active for 96 pixels
  - Pixel Count 799: Reset pixel counter
  - HS Pulse is <u>Active Low</u> for most monitors

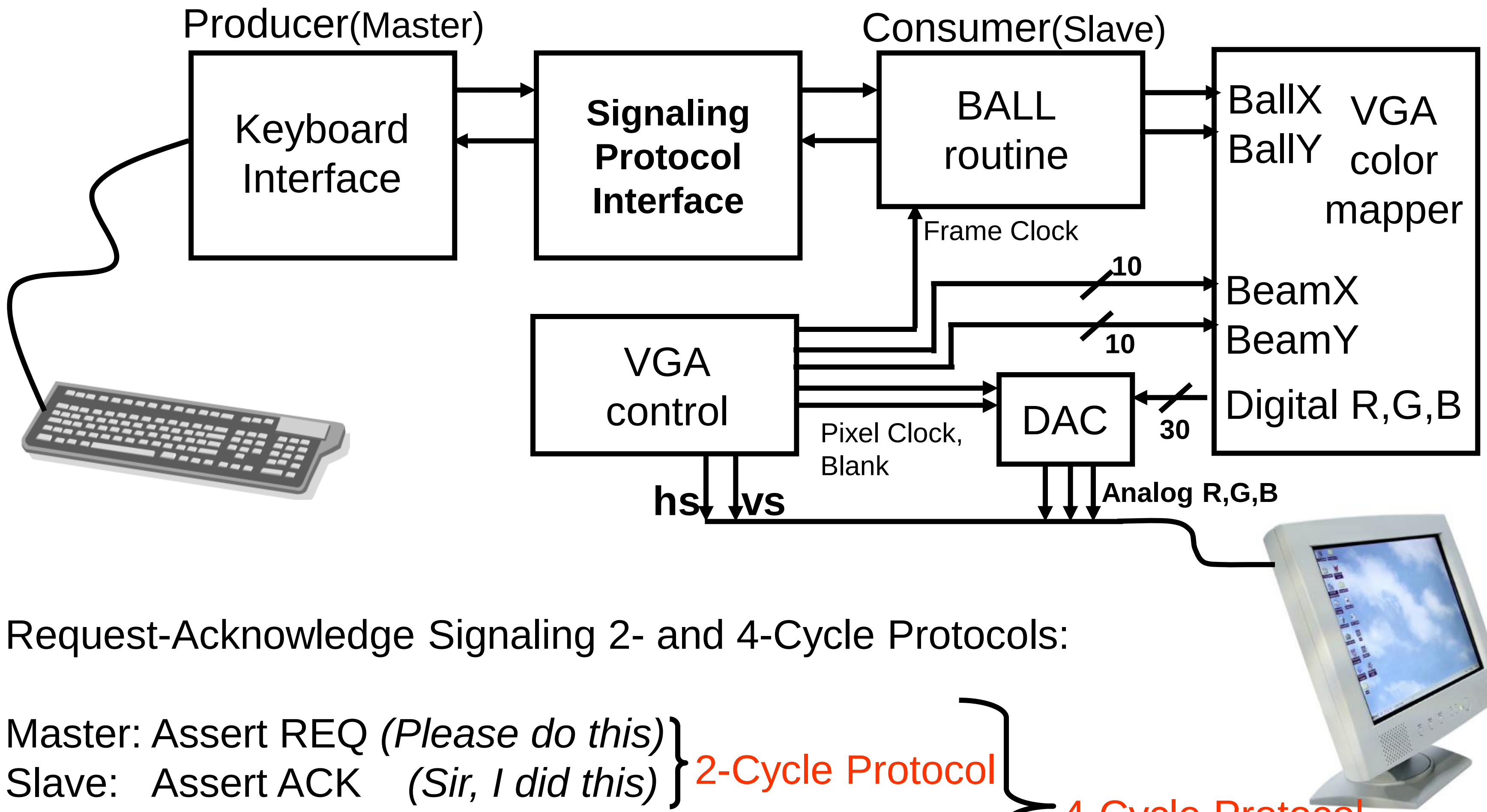


VGA Vertical Timing

- **To generate Vertical Sync Pulse (VS), start a 10-bit Line Counter modulo-525**
  - Counter increments every 800 pixels
  - Line Counts <0 thru 479> : Display On
  - Line Counts <480 thru 524>: Display Off
  - Line Counts 490 and 491: VS Pulse Active
  - Line Count 524: Reset Line Counter
  - VS Pulse is Active Low for most monitors



Producer-Consumer Signaling Protocol



Request-Acknowledge Signaling 2- and 4-Cycle Protocols:

$$\left.\begin{array}{l}\left.\begin{array}{l}\text{Master: Assert REQ } \textit{(Please do this)} \\ \text{Slave: Assert ACK } \textit{(Sir, I did this)}\end{array}\right\} \text{2-Cycle Protocol} \\ \text{Master: De-assert REQ } \textit{(Thank you)} \\ \text{Slave: De-assert ACK } \textit{(You are welcome)}\end{array}\right\} \text{4-Cycle Protocol}$$

↑

Use this if you want to do handshaking.

Backup: Keyboard-Ball-VGA handshaking interface

Assume the Keyboard code uses a flip-flop NewKey to indicate that a new key was pressed and the key code has been obtained.

For the protocol interface Assume a DataReady flip-flop and a DataBuffer

```
Initialize: DataReady <= '0';

Interface: If ((rising_edge(clk) and (DataReady='0') and (NewKey='1'))
        then
            DataBuffer <= NewKeyCode;
            DataReady <= '1';  -- signal Ball that a new command is issued
            NewKeyAck <= '0'; -- tell KeyBoard code that key has been consumed,
        endif;                 --    ok to de-assert NewKey

BALL: if ((rising_edge(clk)) and (DataReady='1'))
        then
            <take appropriate action based on which key was pressed>
            DataReadyAck <='0'; -- signal Interface that command has been
        endif;                  --    consumed
```

Next Lecture

Lab 9: SLC 3.2 -- A simple computer

Quiz 2.