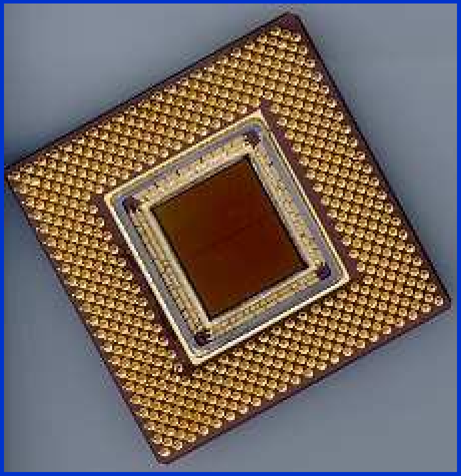


Microcontroller-Based Digital System Design

Module 3
Microcontroller Peripherals

Module 3

- **Learning Outcome:** "An ability to effectively utilize the wide variety of peripherals integrated into a contemporary microcontroller"
  - **Part A:** Analog-to-Digital Converter (ATD)
  - **Part B:** Serial Communications Interface (SCI)
  - **Part C:** Serial Peripheral Interface (SPI)
  - **Part D:** Timer Module (TIM)
  - **Part E:** Pulse Width Modulation (PWM)

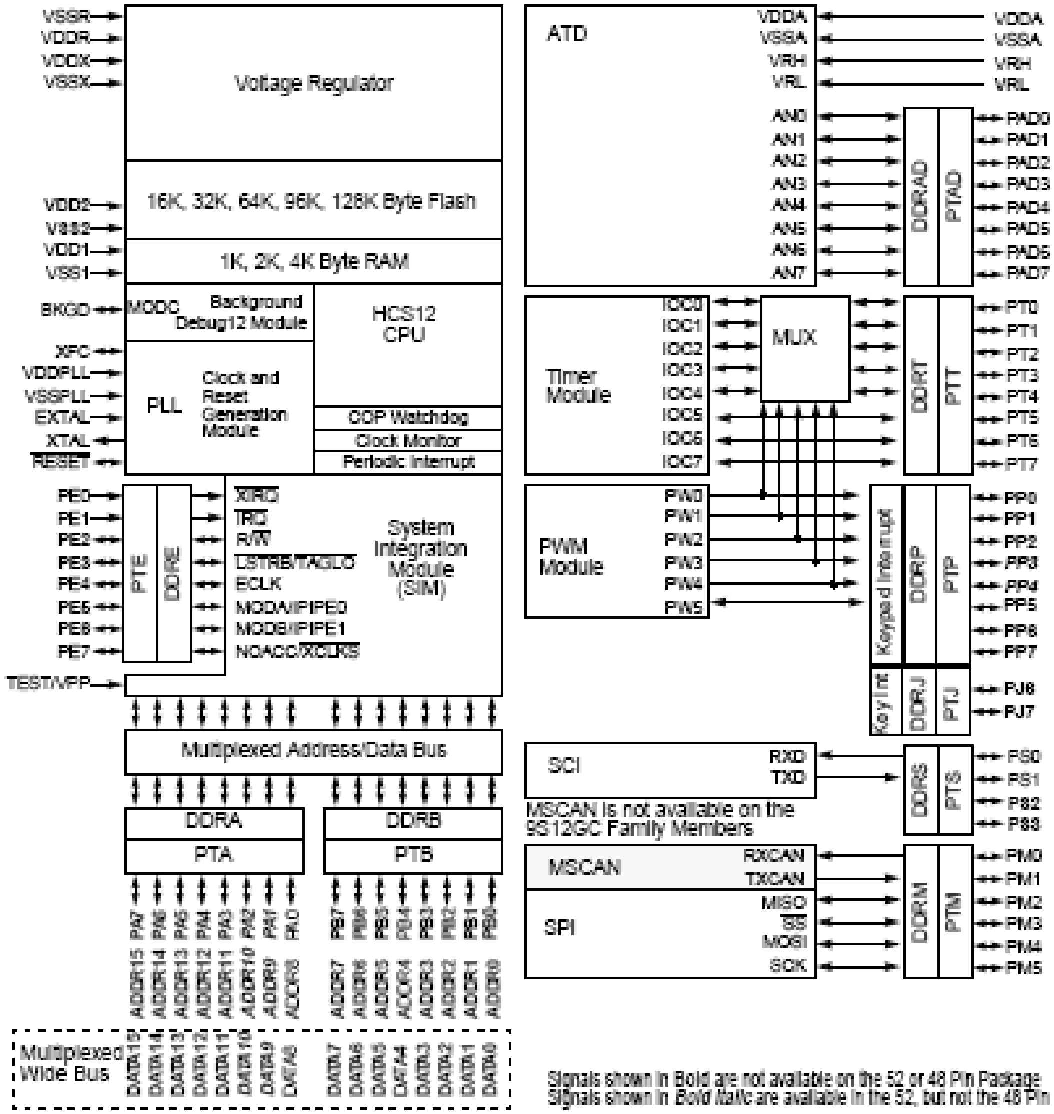
Key Learning Objectives

- **To learn about peripheral devices that are typically integrated into contemporary microcontrollers, and how to use them**
- **To learn how the 9S12C32 integrated peripherals can be incorporated into a significant system design**

Overview of 9S12C32 On-Chip Peripherals

Several things to note…..

Figure 1-1 MC9S12C-Family Block Diagram



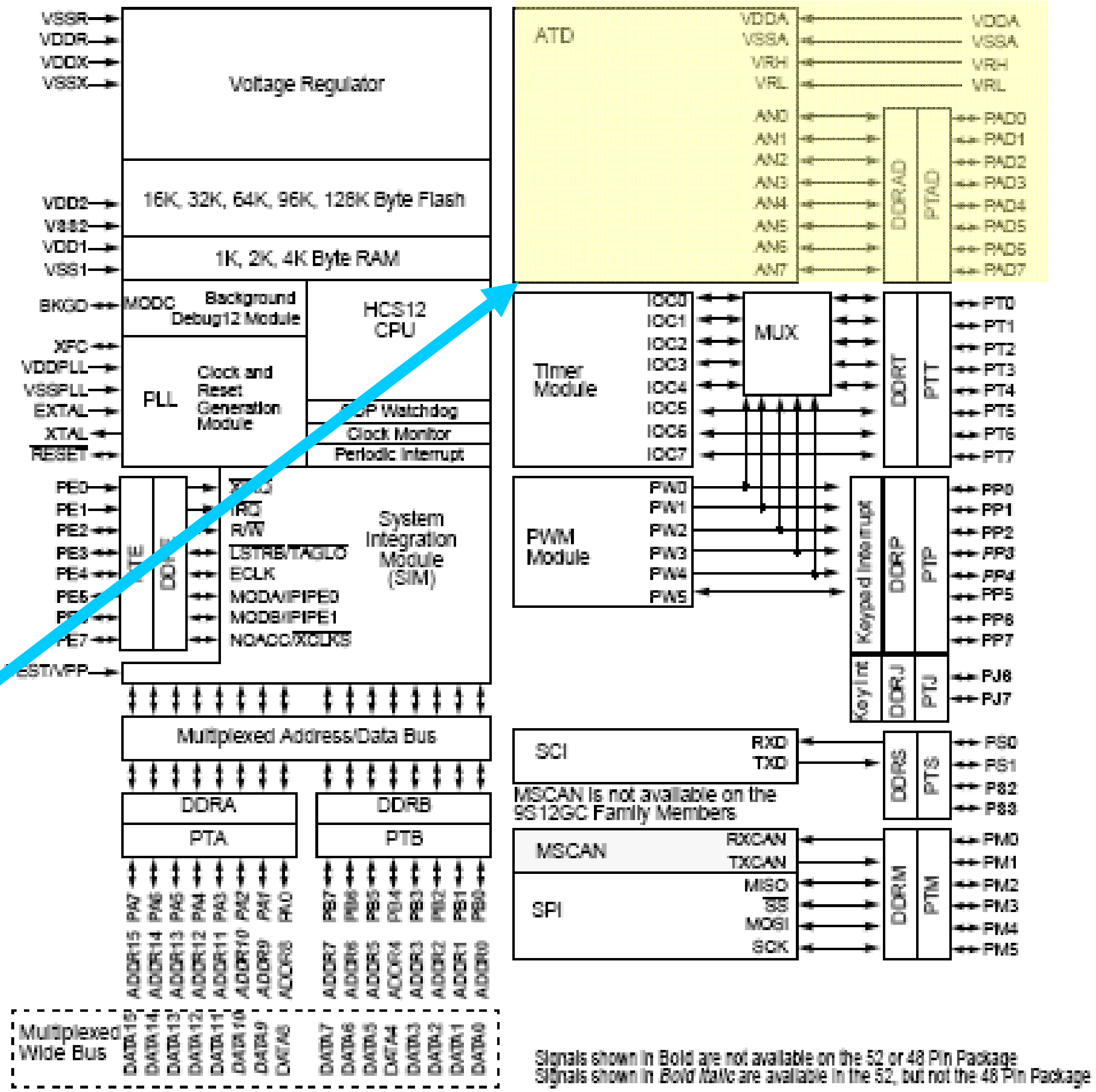
Signals shown in Bold are not available on the 52 or 48 Pin Package
Signals shown in *Bold Italic* are available in the 52, but not the 48 Pin Package

Overview of 9S12C32 On-Chip Peripherals

Several things to note…..

Analog-to-digital (ATD) converter module – inputs are on Port PAD

Figure 1-1 MC9S12C-Family Block Diagram



VSSR
VDDR
VDDX
VSSX
Voltage Regulator
VDD2
VSS2
VDD1
VSS1
16K, 32K, 64K, 96K, 128K Byte Flash
1K, 2K, 4K Byte RAM
BKGD
MODC
Background Debug12 Module
HCS12 CPU
XFC
VDDPLL
VSSPLL
EXTAL
XTAL
RESET
PLL
Clock and Reset Generation Module
COP Watchdog
Clock Monitor
Periodic Interrupt
PE0
PE1
PE2
PE3
PE4
PE5
PE6
PE7
PTE
DDRE
XIRQ
IRQ
R/W
LSTRB/TAGLO
ECLK
MODA/IPIPE0
MODB/IPIPE1
NOACC/XCLKS
System Integration Module (SIM)
TEST/VPP
Multiplexed Address/Data Bus
DDRA
PTA
DDRB
PTB
PA7 PA6 PA5 PA4 PA3 PA2 PA1 PA0
PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0
ADDR15 ADDR14 ADDR13 ADDR12 ADDR11 ADDR10 ADDR9 ADDR8
ADDR7 ADDR6 ADDR5 ADDR4 ADDR3 ADDR2 ADDR1 ADDR0
DATA15 DATA14 DATA13 DATA12 DATA11 DATA10 DATA9 DATA8
DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0
Multiplexed Wide Bus
ATD
VDDA
VSSA
VRH
VRL
AN0 AN1 AN2 AN3 AN4 AN5 AN6 AN7
DDRAD
PTAD
PAD0 PAD1 PAD2 PAD3 PAD4 PAD5 PAD6 PAD7
Timer Module
IOC0 IOC1 IOC2 IOC3 IOC4 IOC5 IOC6 IOC7
MUX
DDRT
PTT
PT0 PT1 PT2 PT3 PT4 PT5 PT6 PT7
PWM Module
PW0 PW1 PW2 PW3 PW4 PW5
Keypad Interrupt
DDRP
PTP
PP0 PP1 PP2 PP3 PP4 PP5 PP6 PP7
Key Int
DDRJ
PTJ
PJ6
PJ7
SCI
RXD
TXD
DDRS
PTS
PS0 PS1 PS2 PS3
MSCAN is not available on the 9S12GC Family Members
MSCAN
RXCAN
TXCAN
SPI
MISO
SS
MOSI
SCK
DDRM
PTM
PM0 PM1 PM2 PM3 PM4 PM5

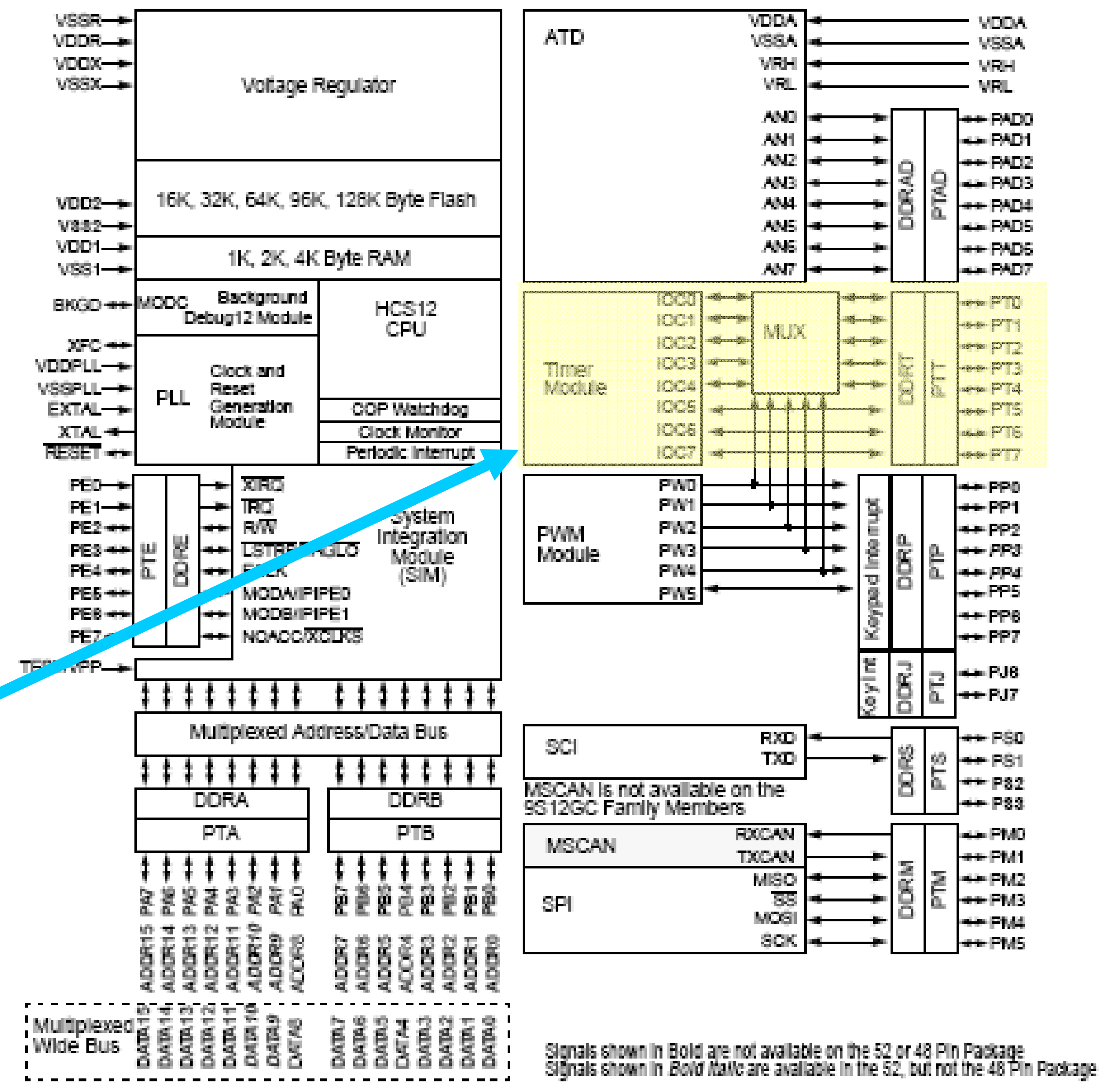
Signals shown in Bold are not available on the 52 or 48 Pin Package
Signals shown in *Bold Italic* are available in the 52, but not the 48 Pin Package

Overview of 9S12C32 On-Chip Peripherals

Several things to note…..

Timer (TIM) module – I/O on Port T

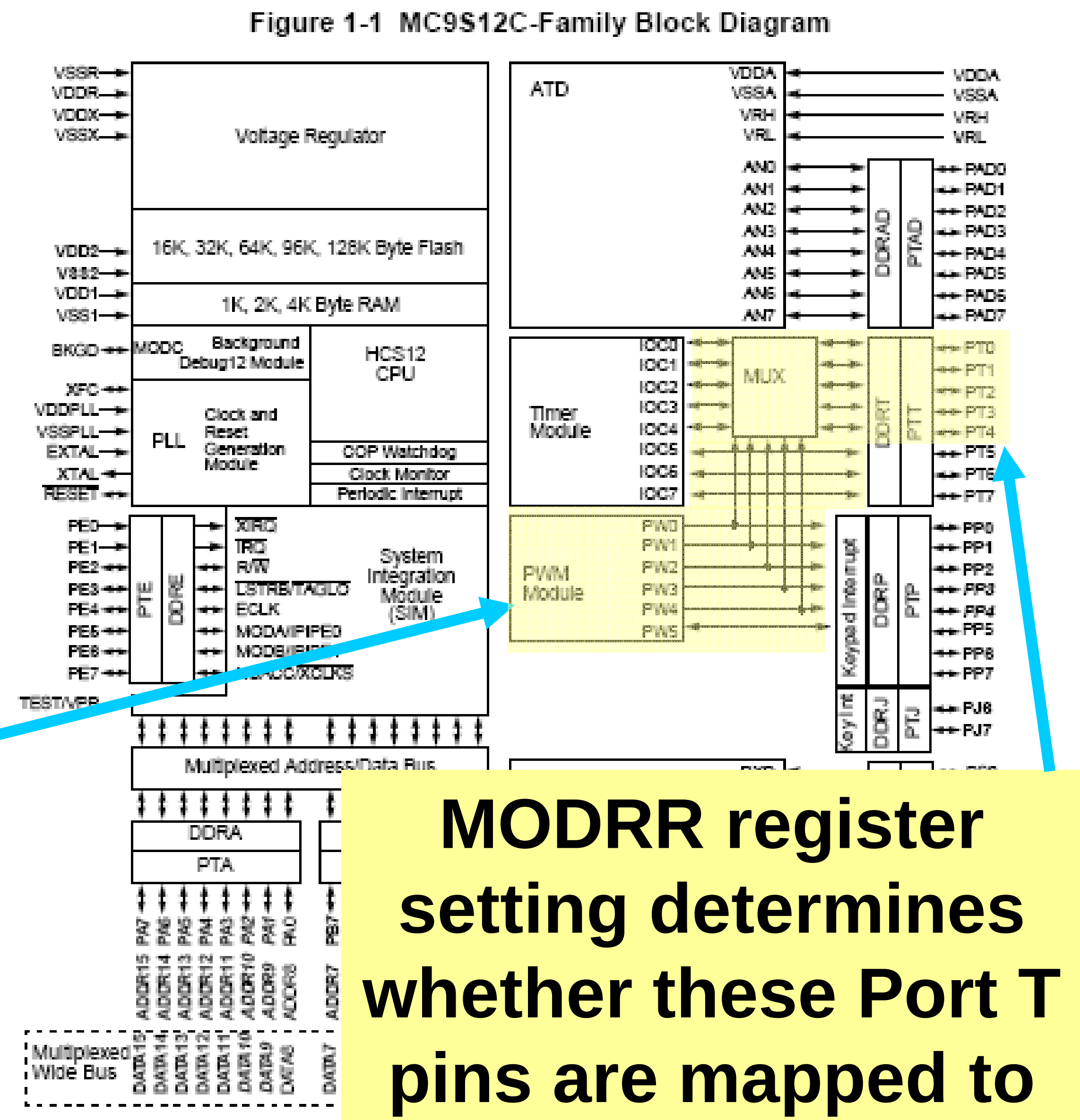
Figure 1-1 MC9S12C-Family Block Diagram



Overview of 9S12C32 On-Chip Peripherals

Several things to note…..

Figure 1-1 MC9S12C-Family Block Diagram



Pulse width modulator (PWM) – here, I/O shared with TIM module on Port T

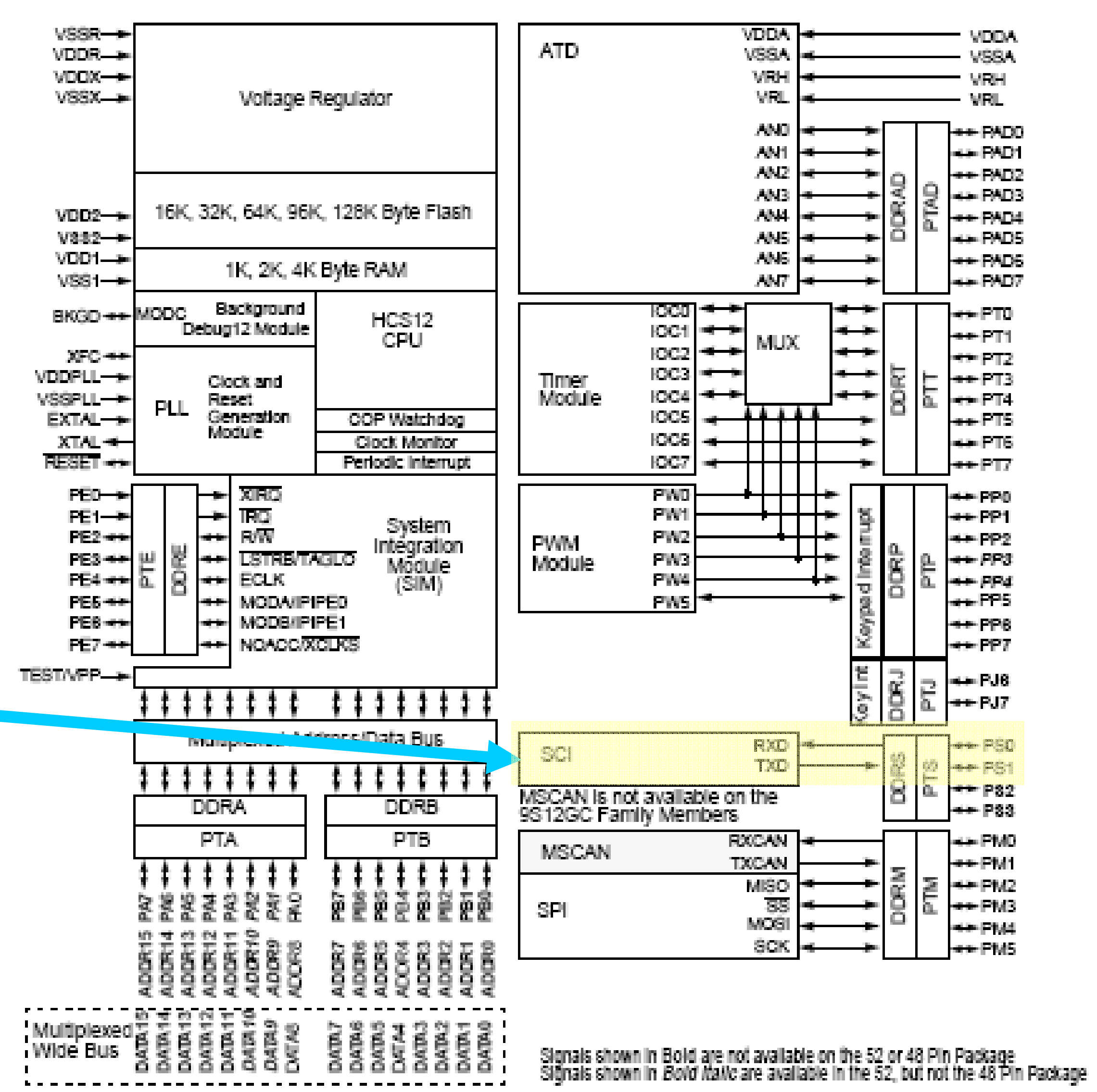
MODRR register setting determines whether these Port T pins are mapped to the TIM or PWM

Overview of 9S12C32 On-Chip Peripherals

Several things to note…..

Asynchronous serial communications interface (SCI) on Port S

Figure 1-1 MC9S12C-Family Block Diagram

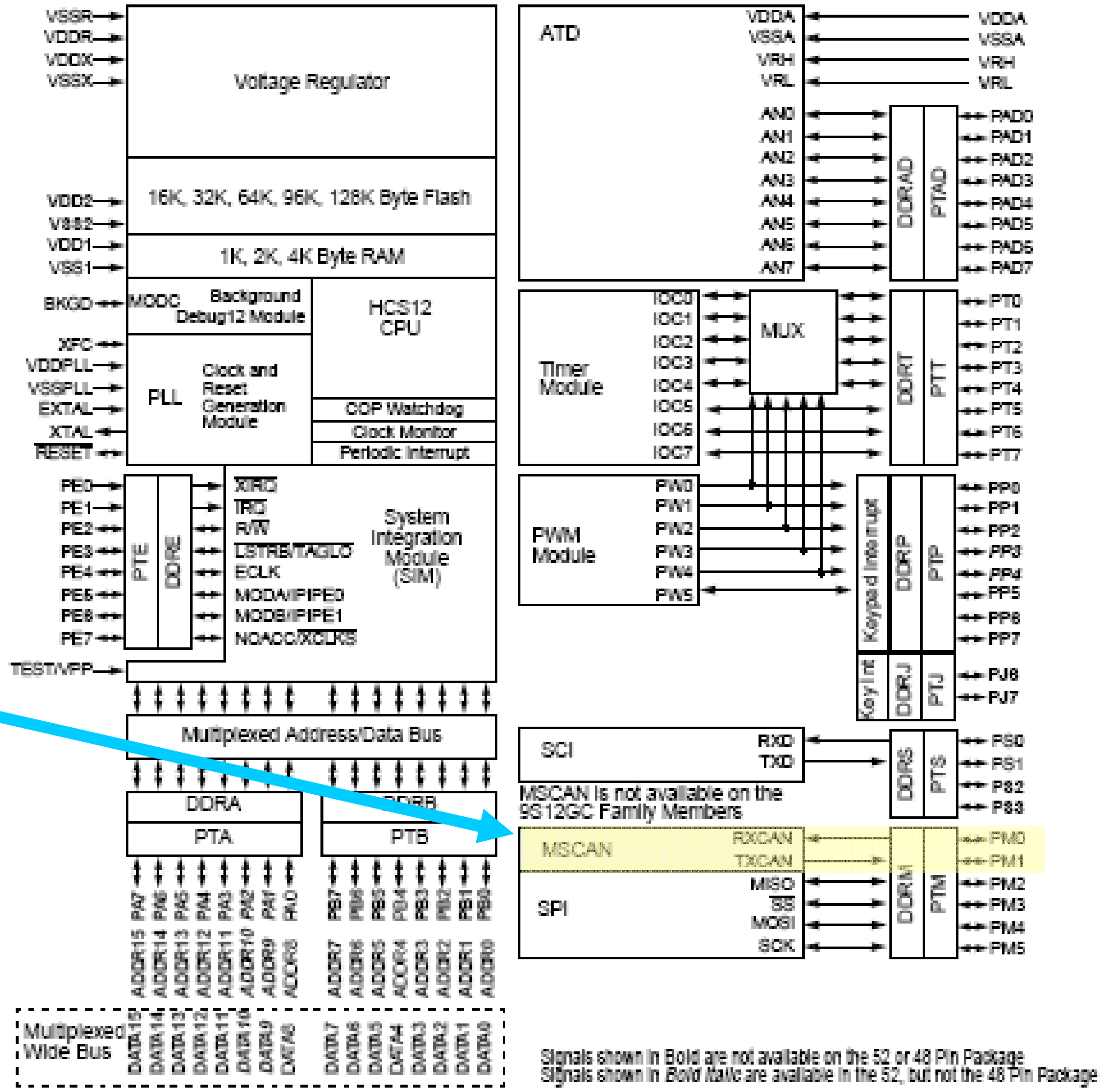


Overview of 9S12C32 On-Chip Peripherals

Several things to note.....

Controller area network (MSCAN) on Port M

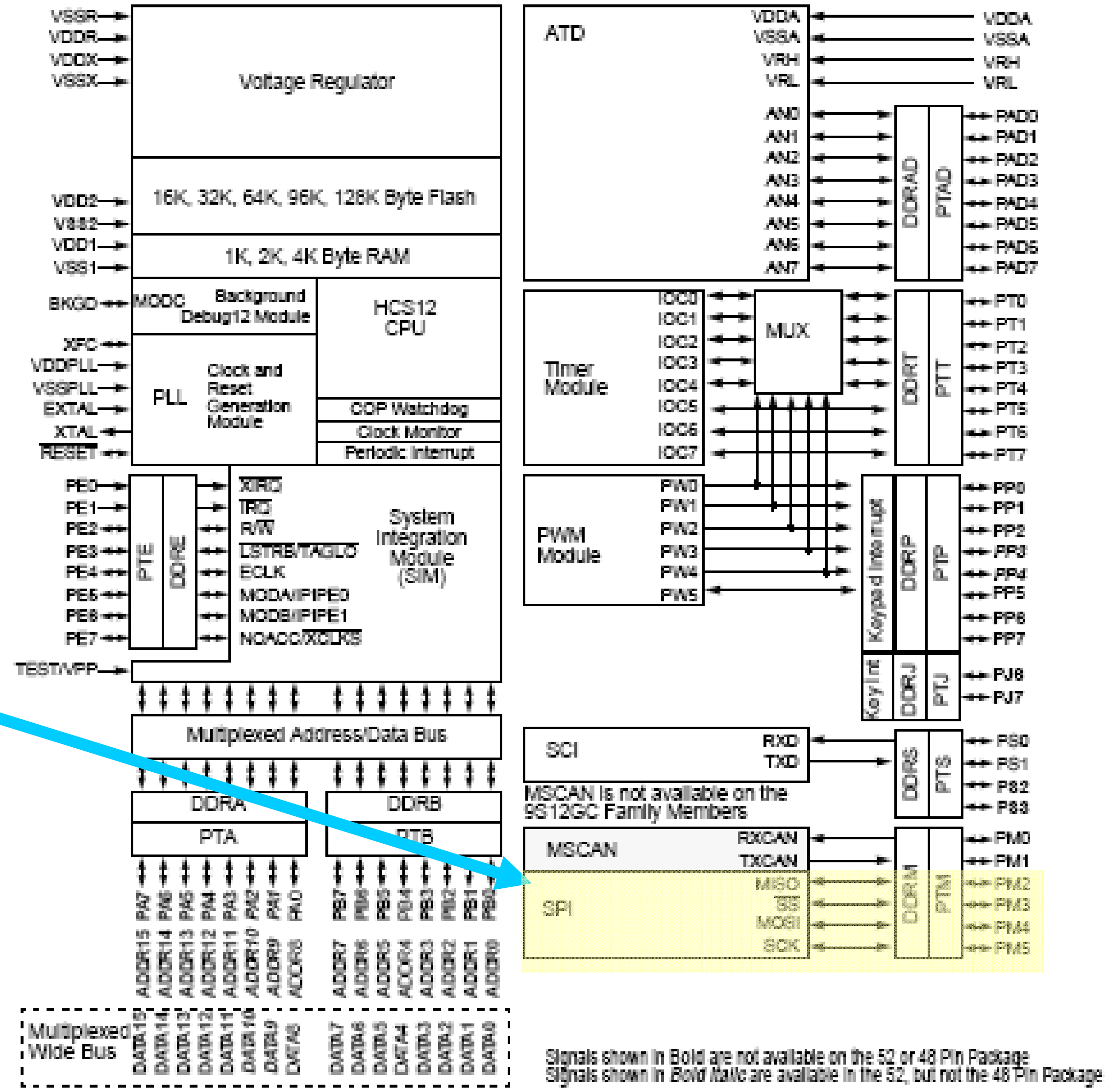
Figure 1-1 MC9S12C-Family Block Diagram



Signals shown in Bold are not available on the 52 or 48 Pin Package
Signals shown in Bold Italic are available in the 52, but not the 48 Pin Package

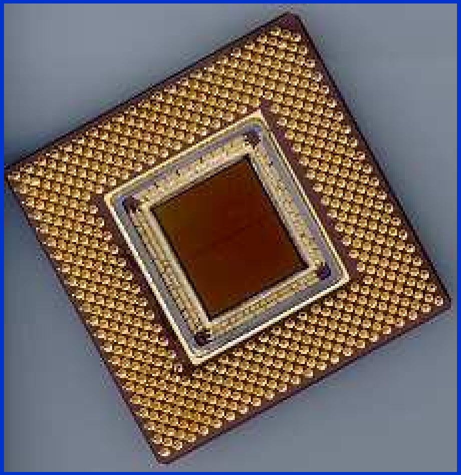
Overview of 9S12C32 On-Chip Peripherals

Figure 1-1 MC9S12C-Family Block Diagram



Several things to note…..

Synchronous peripheral interface (SPI) on Port M



Microcontroller-Based Digital System Design

Module 3-A

Integrated Peripherals: Analog-to-Digital Converter

Learning Objectives

- discuss key design considerations of a microcontroller-based data acquisition system: sampling, quantizing, and encoding
- determine the sampling rate required (Nyquist rate) based on input signal spectral characteristics
- describe aliasing and discuss how it can be prevented
- estimate the dynamic range required based on converter resolution and calculate signal-to-quantizing-noise-ratio (SQNR)
- describe the operation of a successive approximation analog-to-digital (ATD) converter
- identify ATD features and operating modes
- configure the ATD module to operate in a prescribed mode
- create an ATD device driver routine for a prescribed application

Outline

- Analog conversion basics
  - Sampling
  - Quantization
  - Converter types
- 9S12C ATD features and modes
- 9S12C ATD registers
- 9S12C ATD initialization
- ATD program-driven device driver

Reference: **Analog-to-Digital (ATD) Converter Block User Guide**

Analog Conversion Basics

- **Introduction**
  - **A typical embedded system may have many input signals, some of which may be "analog" (*continuous time*) in nature**
  - **Process of converting continuous time signals into discrete representations is called *analog-to-digital* conversion**
  - **A-to-D (ATD) conversion process involves three distinct steps**
    - ***sampling***
    - ***quantizing***
    - ***encoding***

Analog Conversion Basics

- Data acquisition system design issues
  - *Sampling rate* required for sufficient bandwidth
  - *Number of bits* required for sufficient dynamic range
  - *Type of converter* most appropriate for a given application

Analog Conversion Basics

- **Sampling**
  - **Ideally would like to determine the digital equivalent of an analog signal at a precise instant of time ("*snapshot*")**
  - **Because ATD process takes a finite amount of time, a rapidly changing analog input signal can present an ATD converter with an ambiguous input ("*blurred image*")**
  - **One solution is to incorporate an "*analog memory*" that can hold a snapshot of the analog input for the converter**
  - **Called a "*sample-and-hold*" (S/H) circuit**

Analog Conversion Basics

- Sampling rate
  - Major design issue in data acquisition systems is the ***sampling rate*** Ts
  - Need to know spectral characteristics of incoming signal (can be determined using a ***spectrum analyzer***)
  - Sampling frequency Fs must be at least ***twice*** that of the highest frequency component present in input signal (called the ***Nyquist rate***)
  - Use low-pass filter (LPF) to ensure no frequency components ***in excess of Fs/2*** are applied to ATD input

Analog Conversion Basics

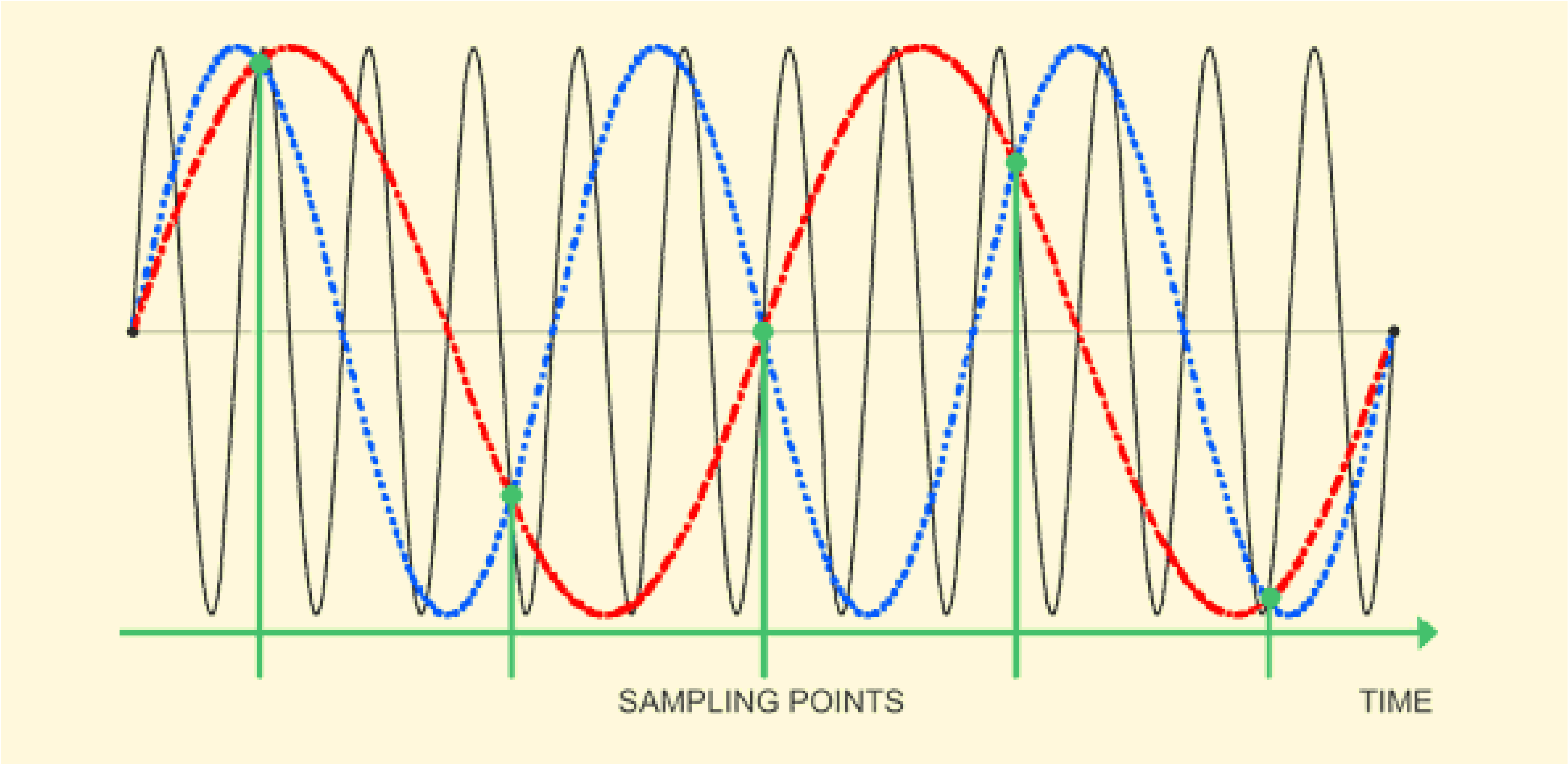
- **Sampling rate, continued**
  - **Input LPF referred to as an "*anti-aliasing*" filter**
  - **Frequency components in excess of Fs/2 are "*folded back*" into the baseband at an "*alias*" frequency (non-linear distortion)**

Aliasing Example:

If Fs = 20 KHz → maximum input frequency component is 10 KHz

If input frequency component is 11 KHz, will be encoded as a 9 KHz component

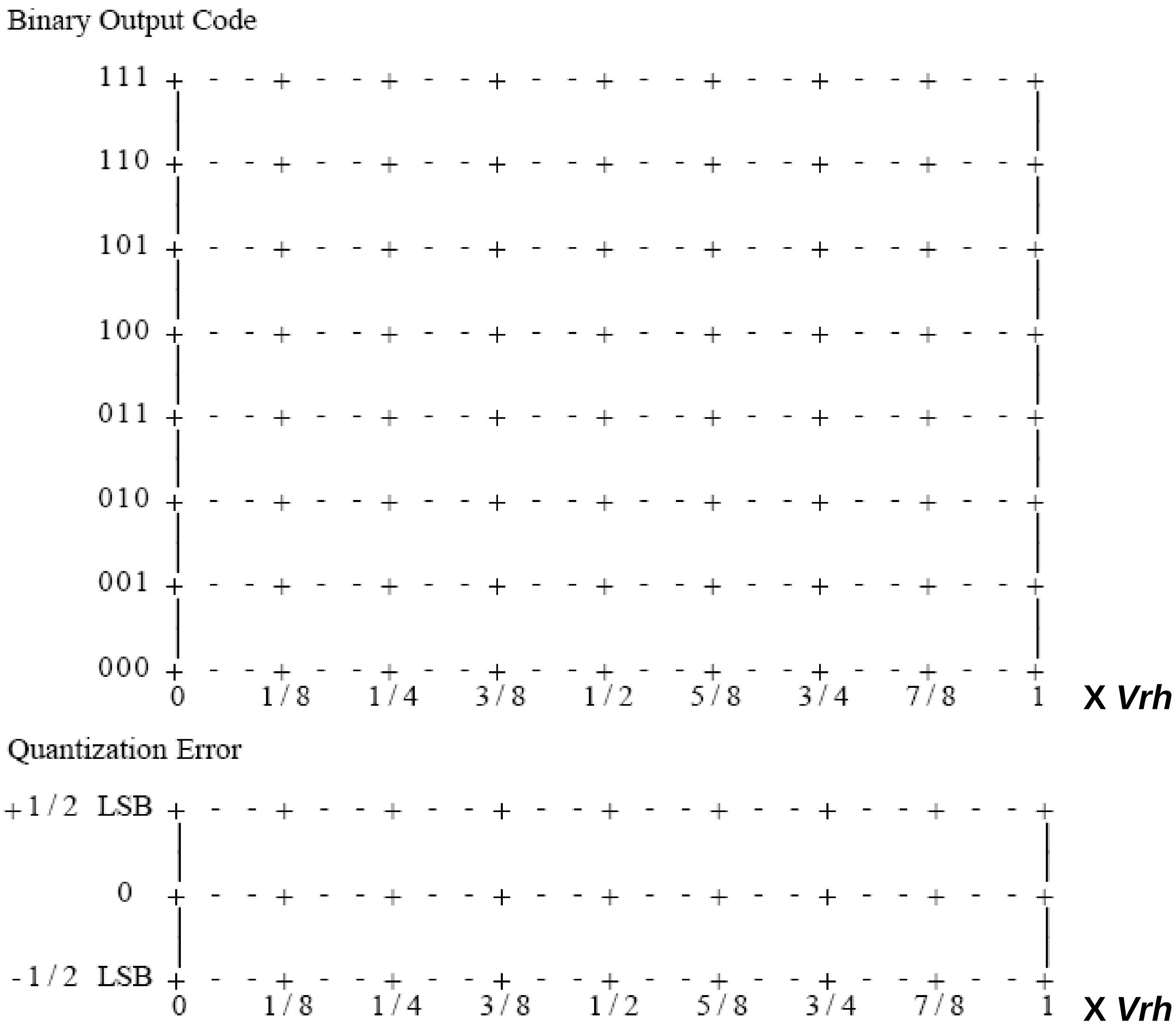
Illustration of Aliasing Frequencies



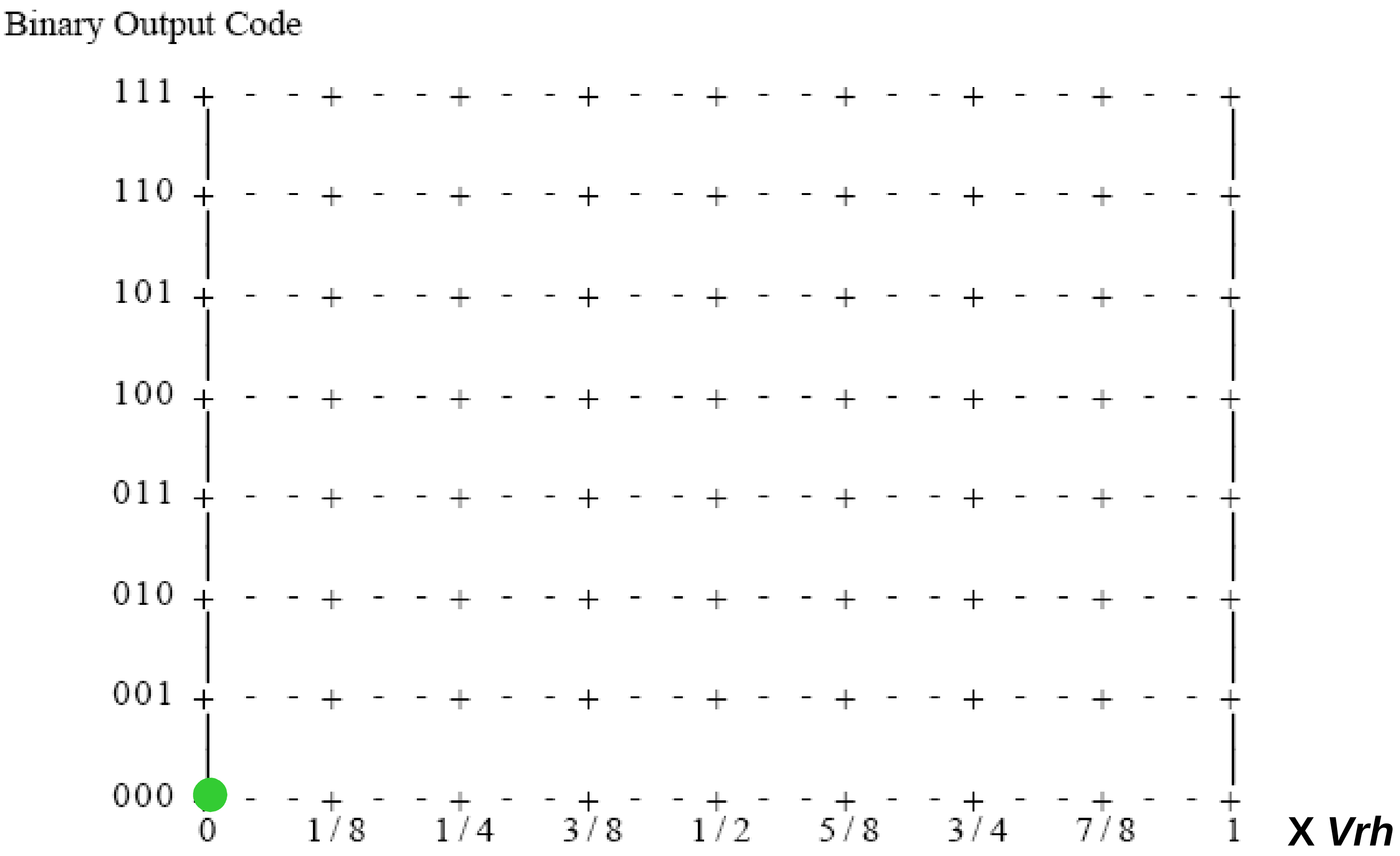
Analog Conversion Basics

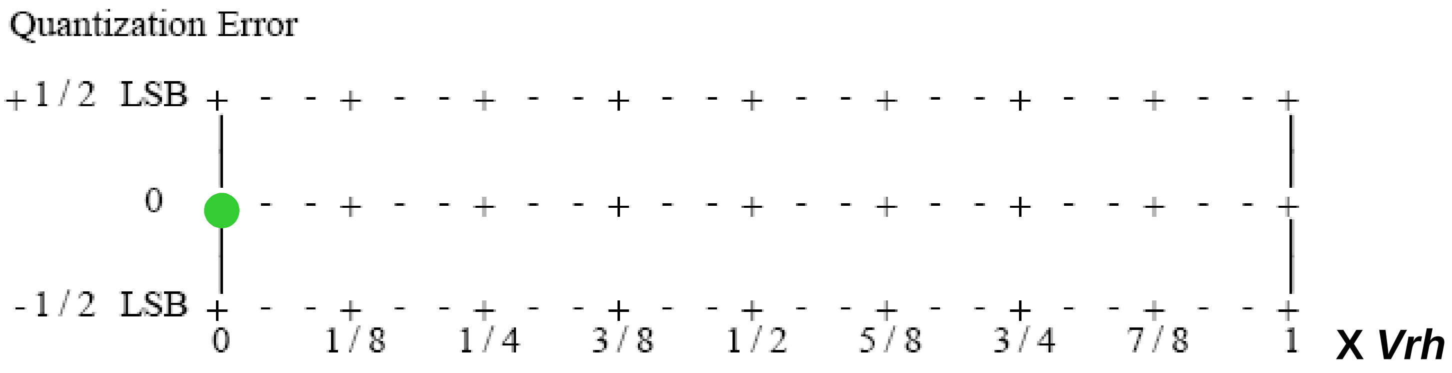
- **Quantization**
  - **Another important design consideration is the number of bits required to obtain adequate *resolution* and/or sufficient *dynamic range***
  - **Quantization is the assignment of a *fixed amplitude level* (corresponding to an available binary code) to the incoming analog signal**
  - **Note that the converted code is relative to the *reference voltage(s)* applied to the converter (V_{RH} = voltage reference high, V_{RL} = voltage reference low)**

Quantizing Intervals and Quantization Error

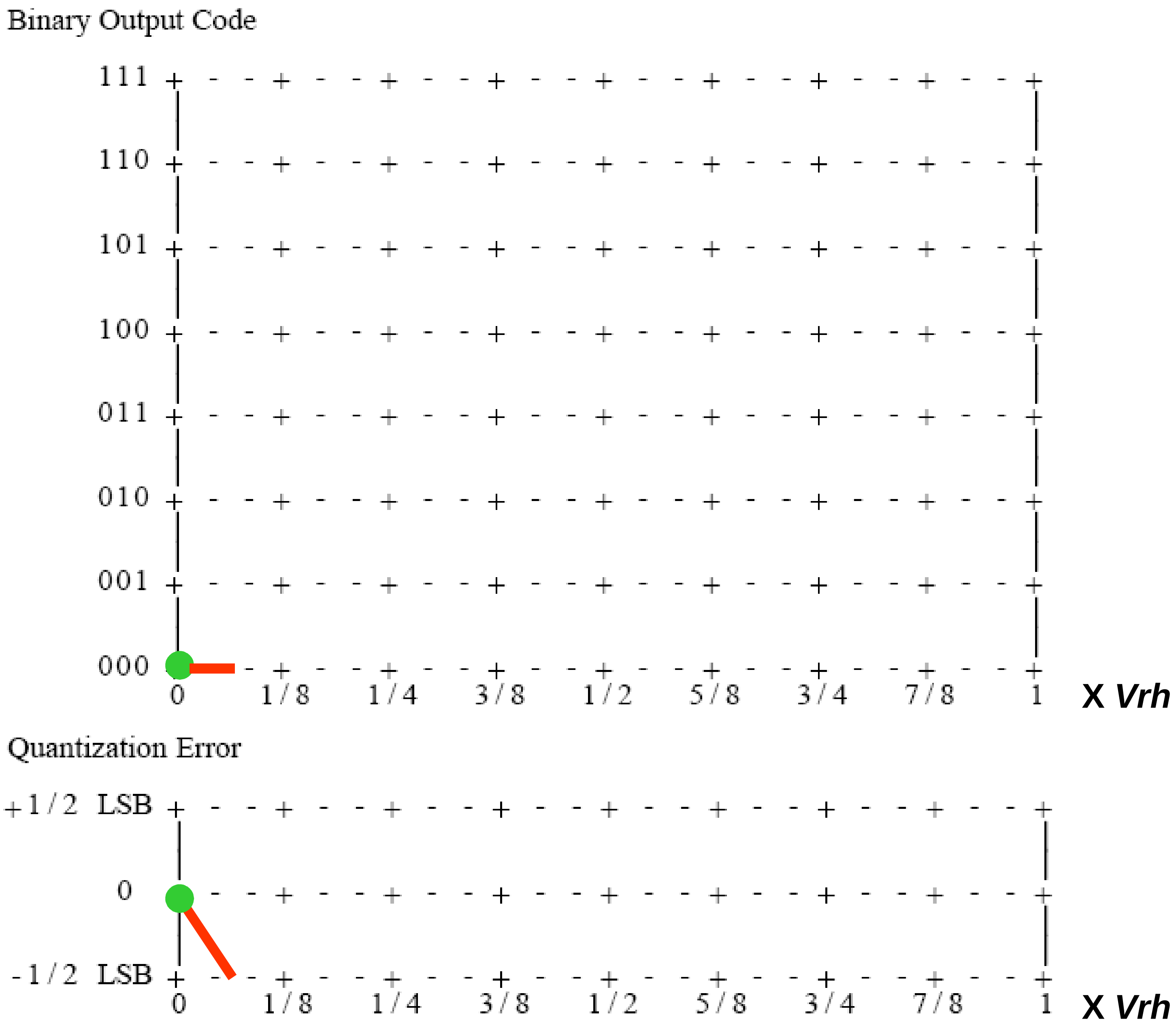


Quantizing Intervals and Quantization Error

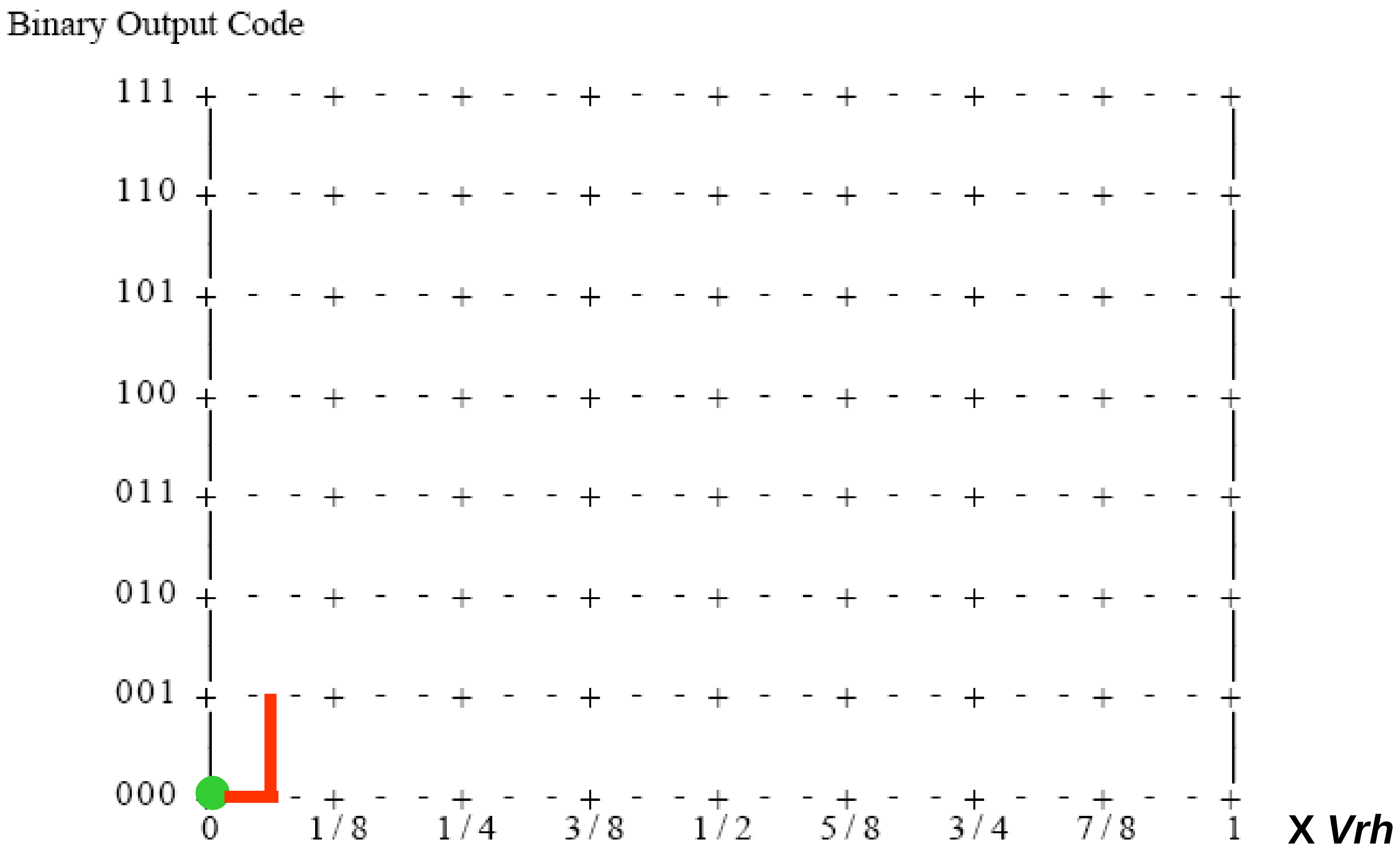


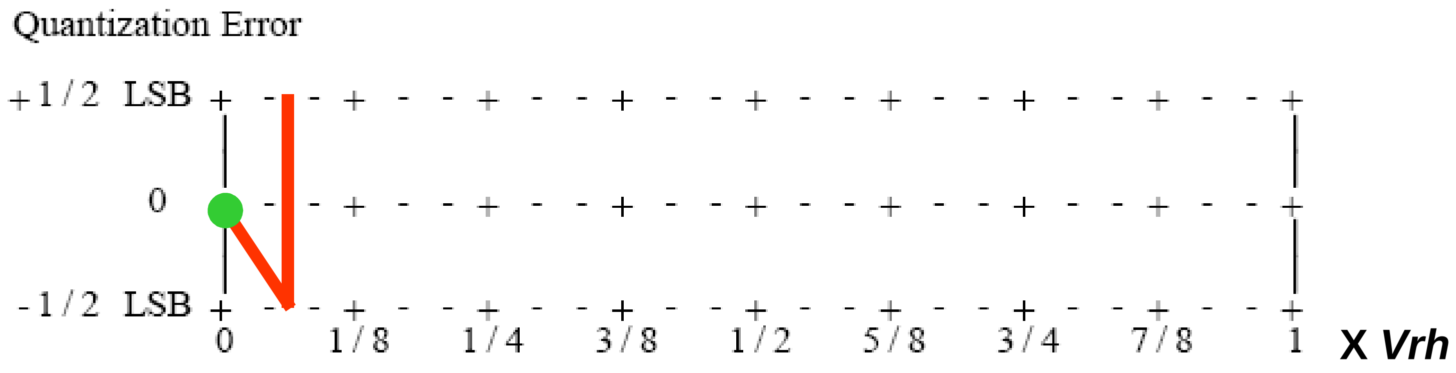


Quantizing Intervals and Quantization Error

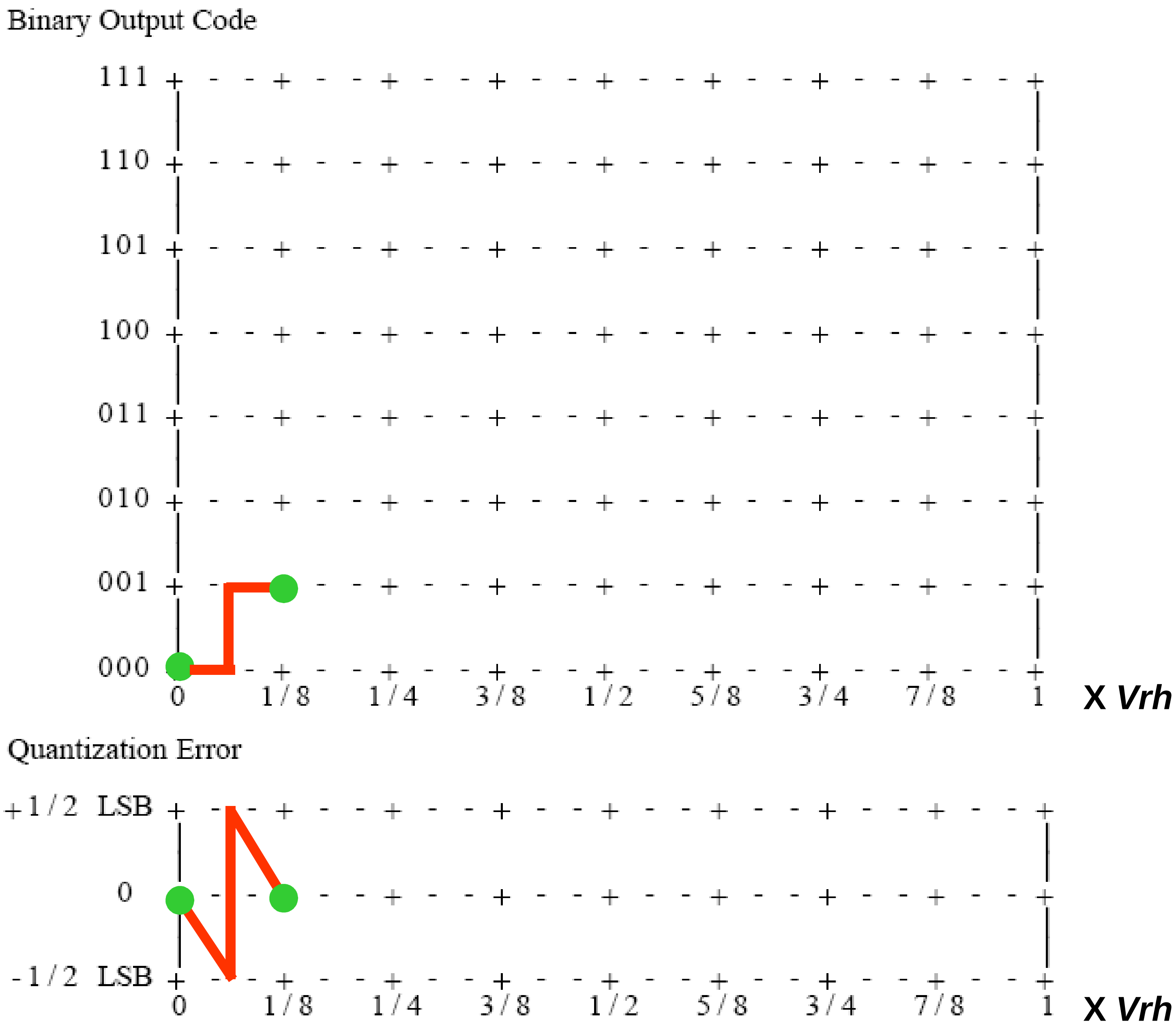


Quantizing Intervals and Quantization Error

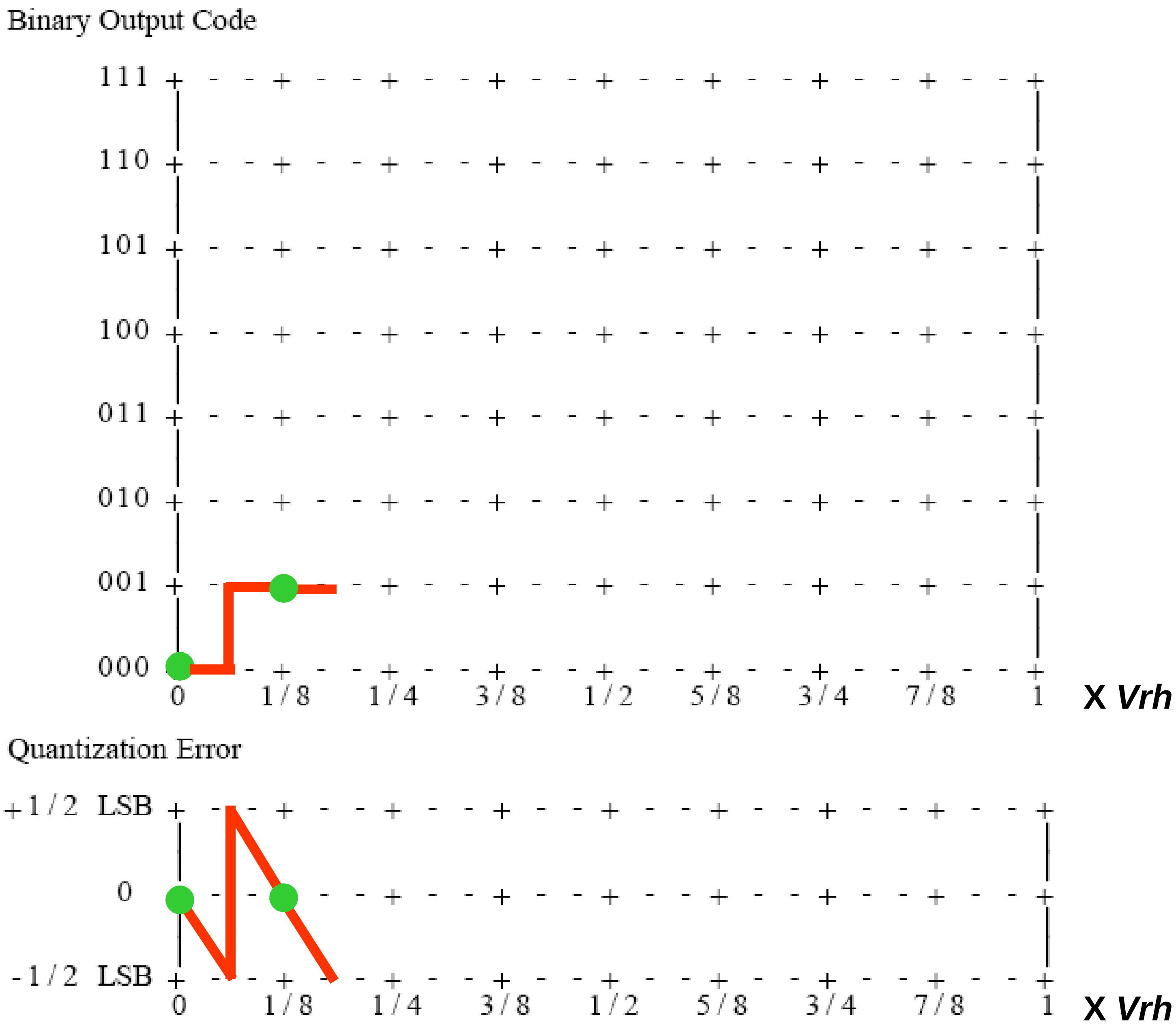




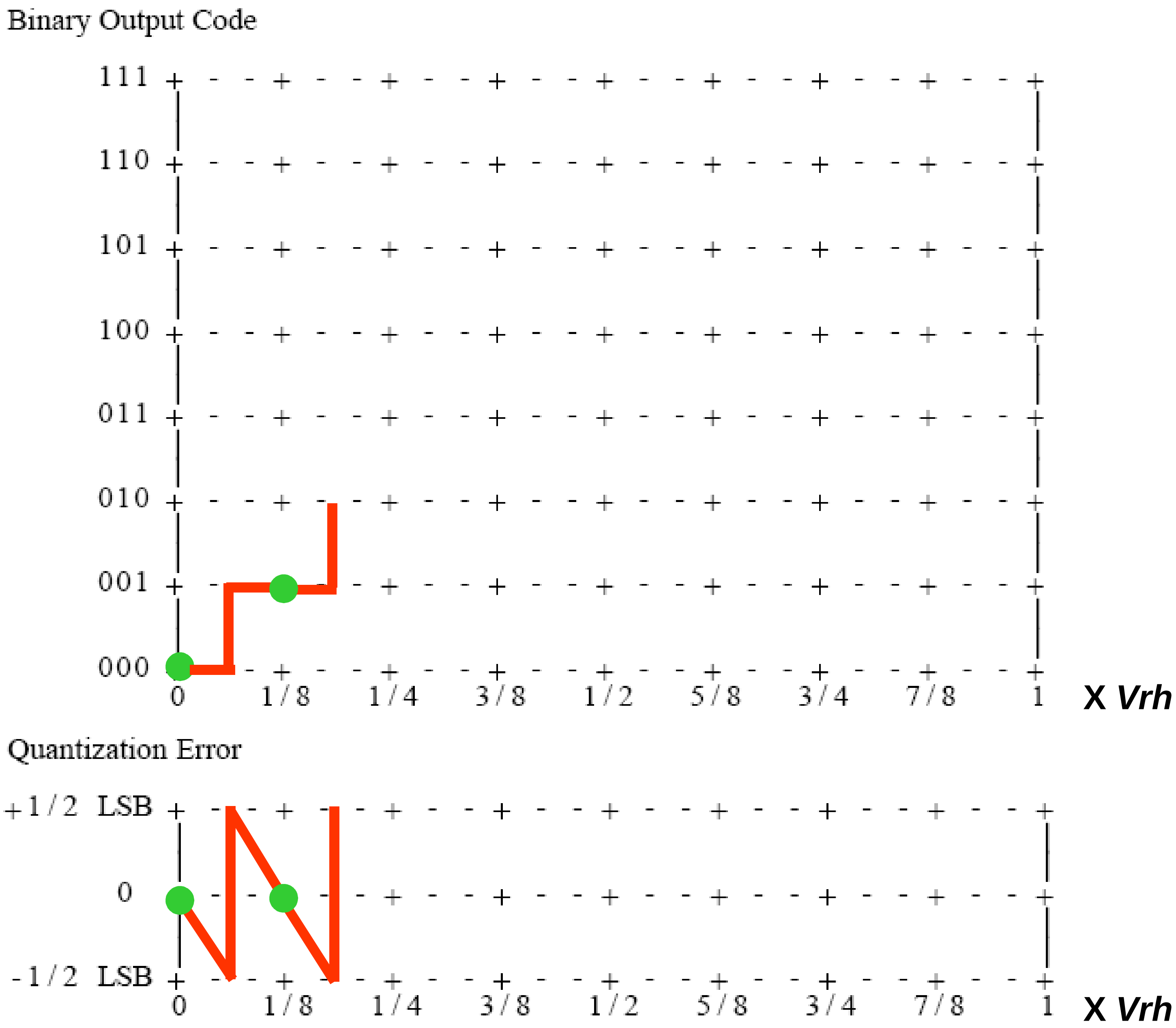
Quantizing Intervals and Quantization Error



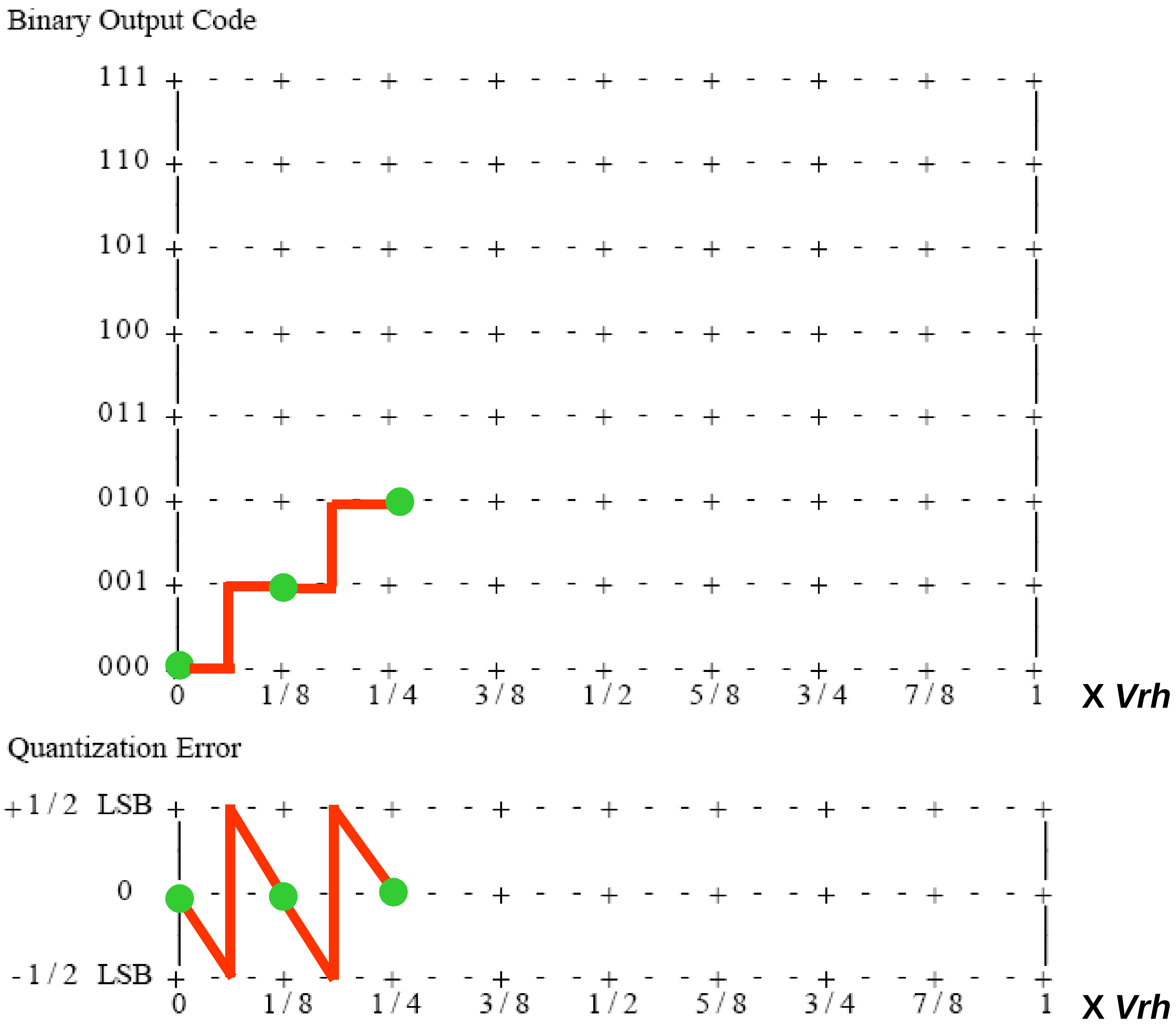
Quantizing Intervals and Quantization Error



Quantizing Intervals and Quantization Error

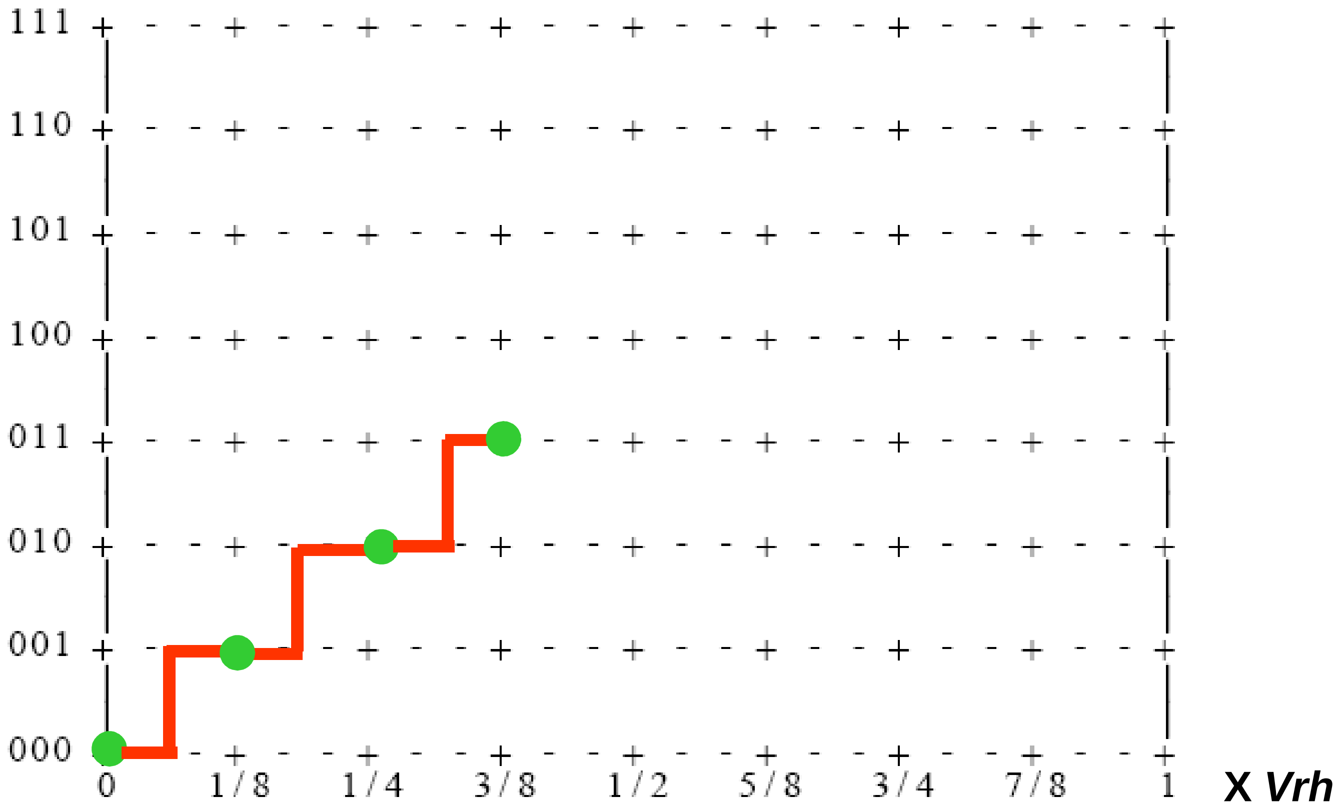


Quantizing Intervals and Quantization Error

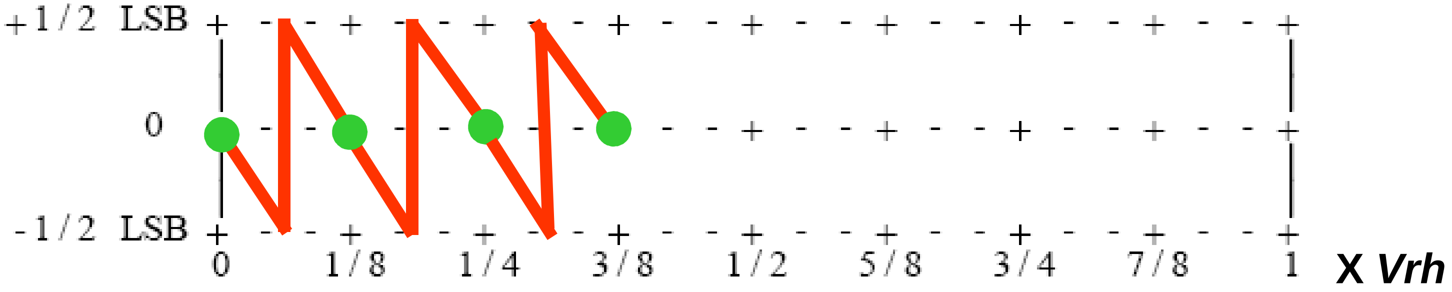


Quantizing Intervals and Quantization Error

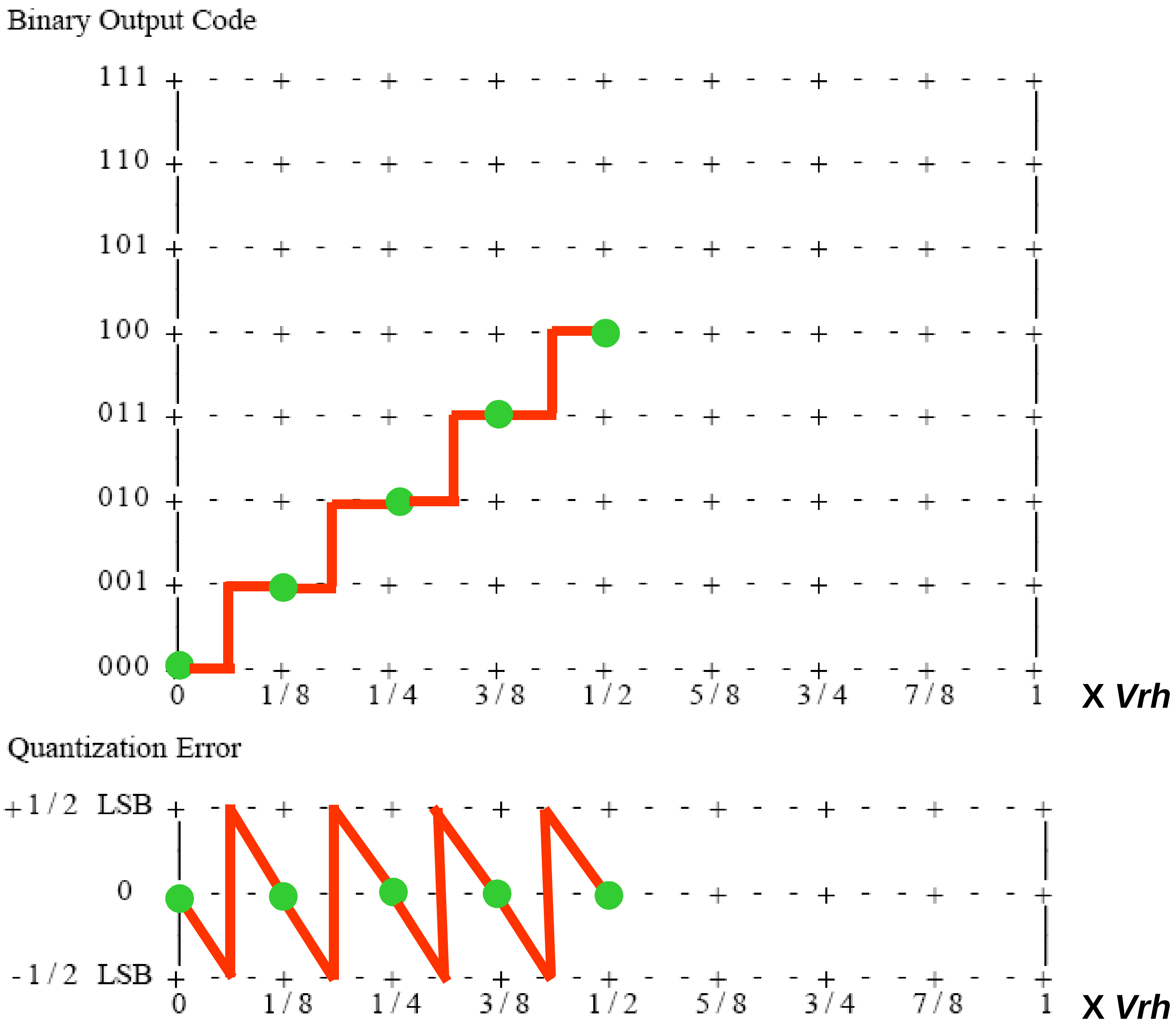
Binary Output Code



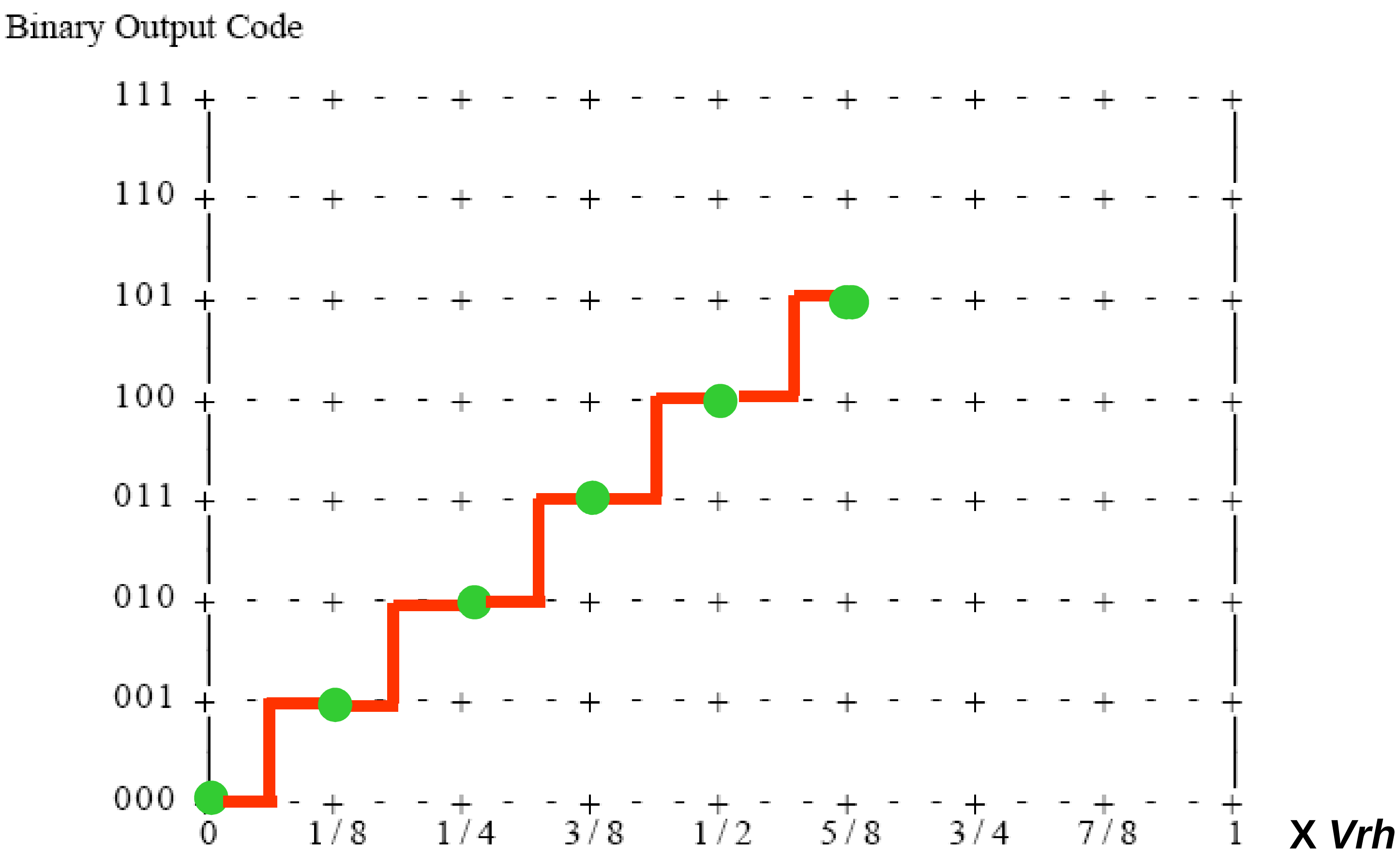
Quantization Error

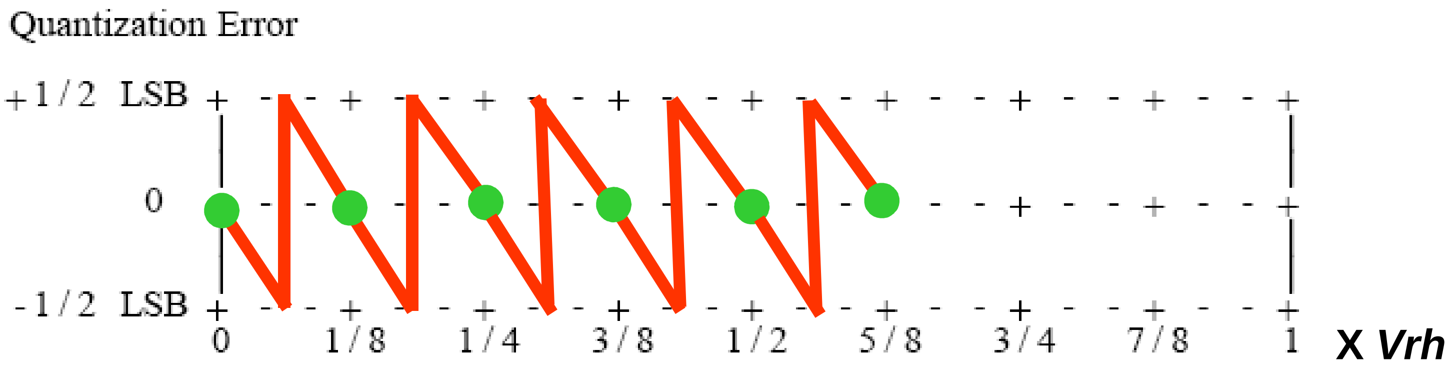


Quantizing Intervals and Quantization Error

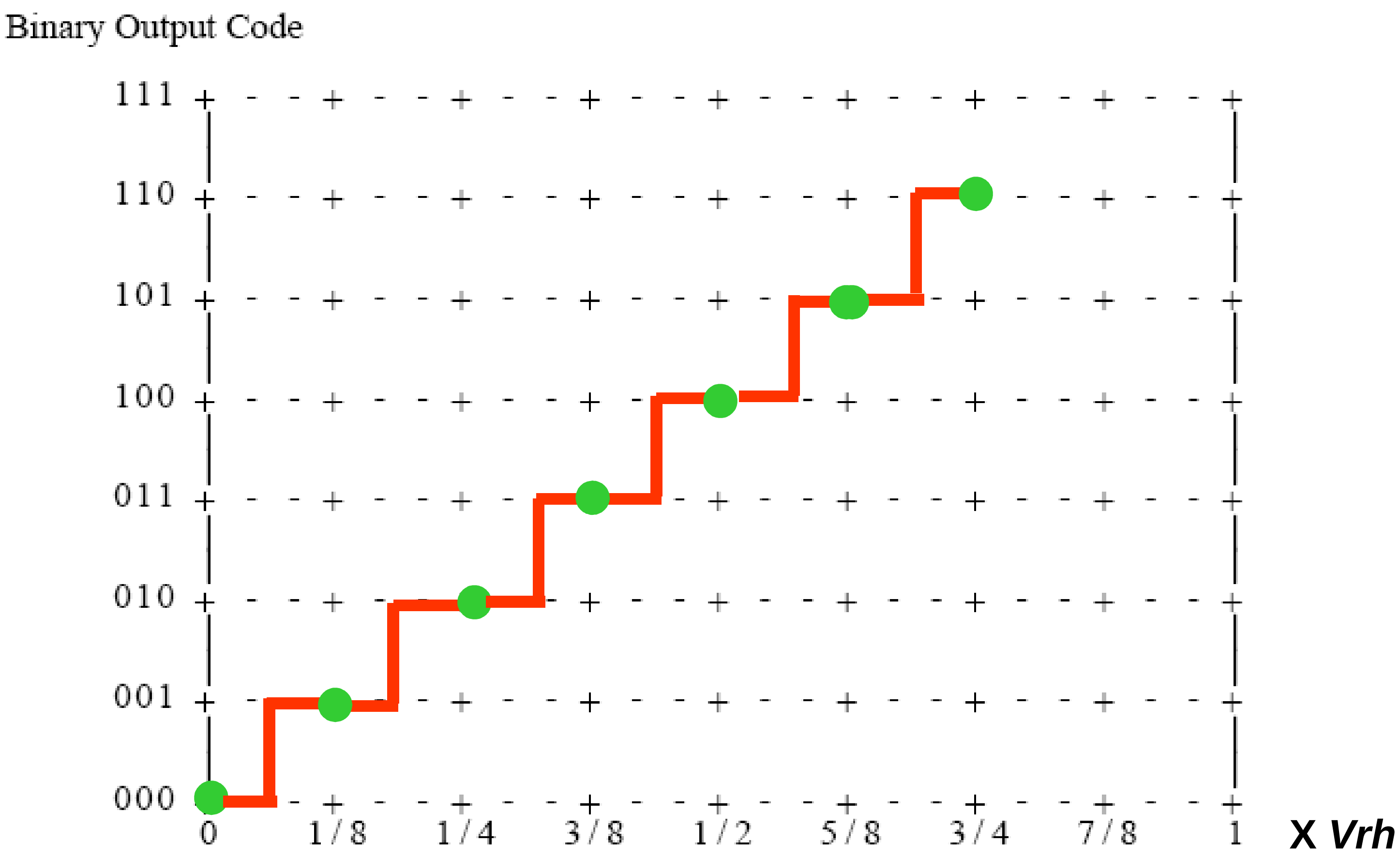


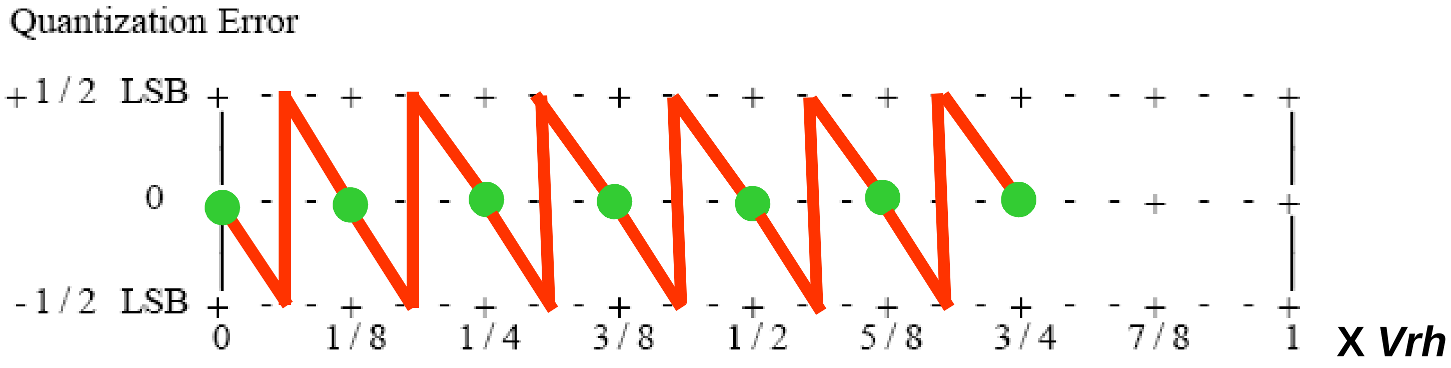
Quantizing Intervals and Quantization Error



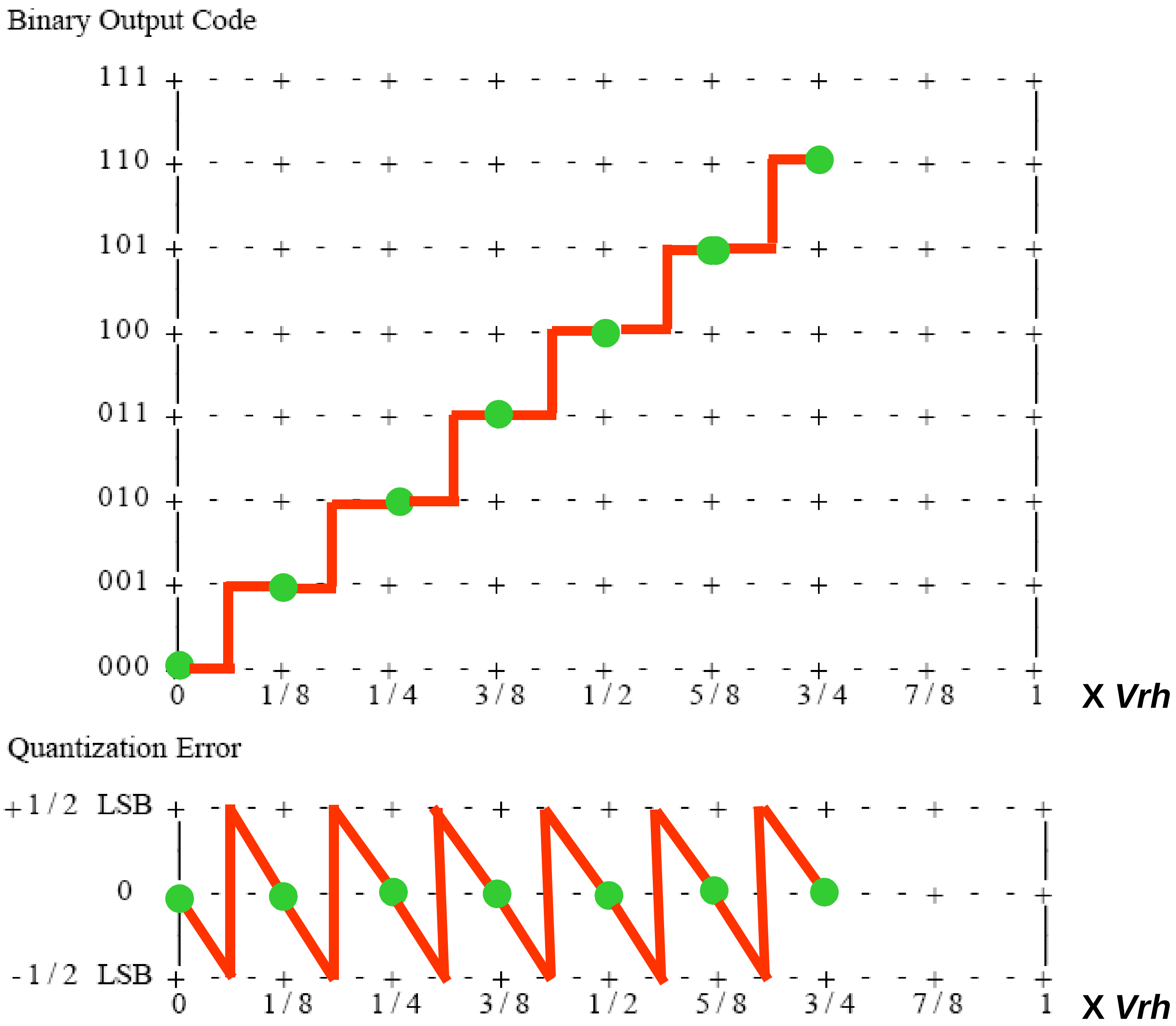


Quantizing Intervals and Quantization Error

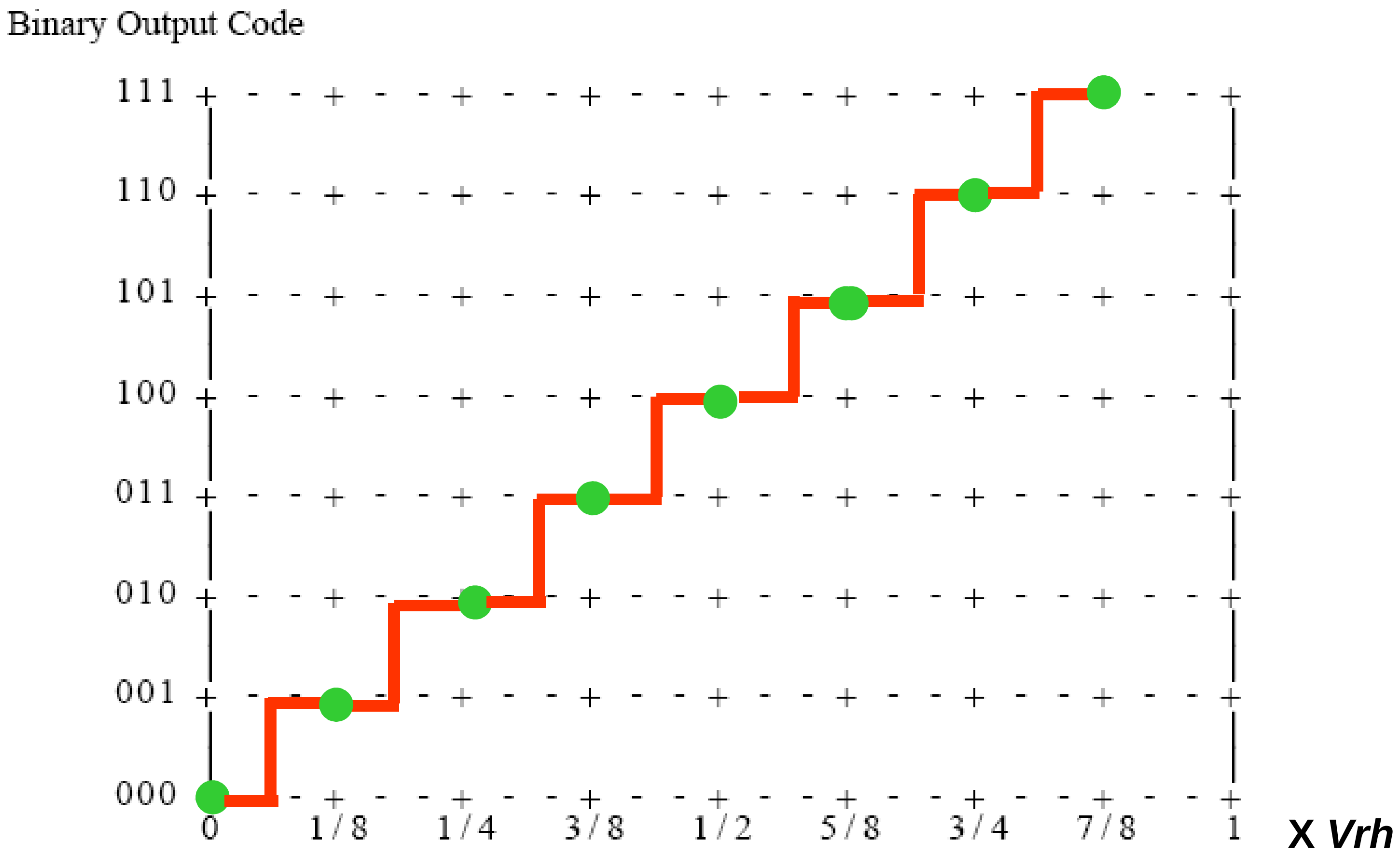


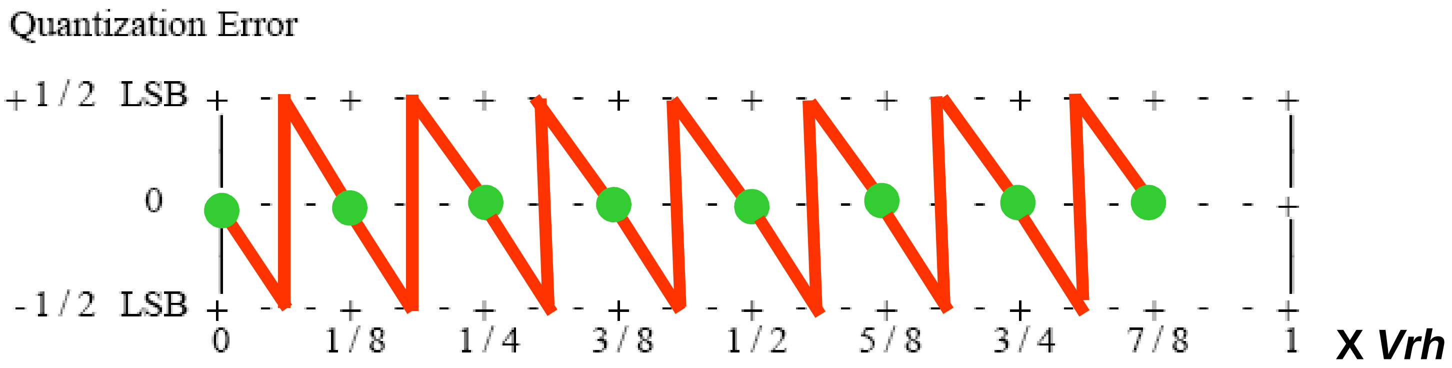


Quantizing Intervals and Quantization Error

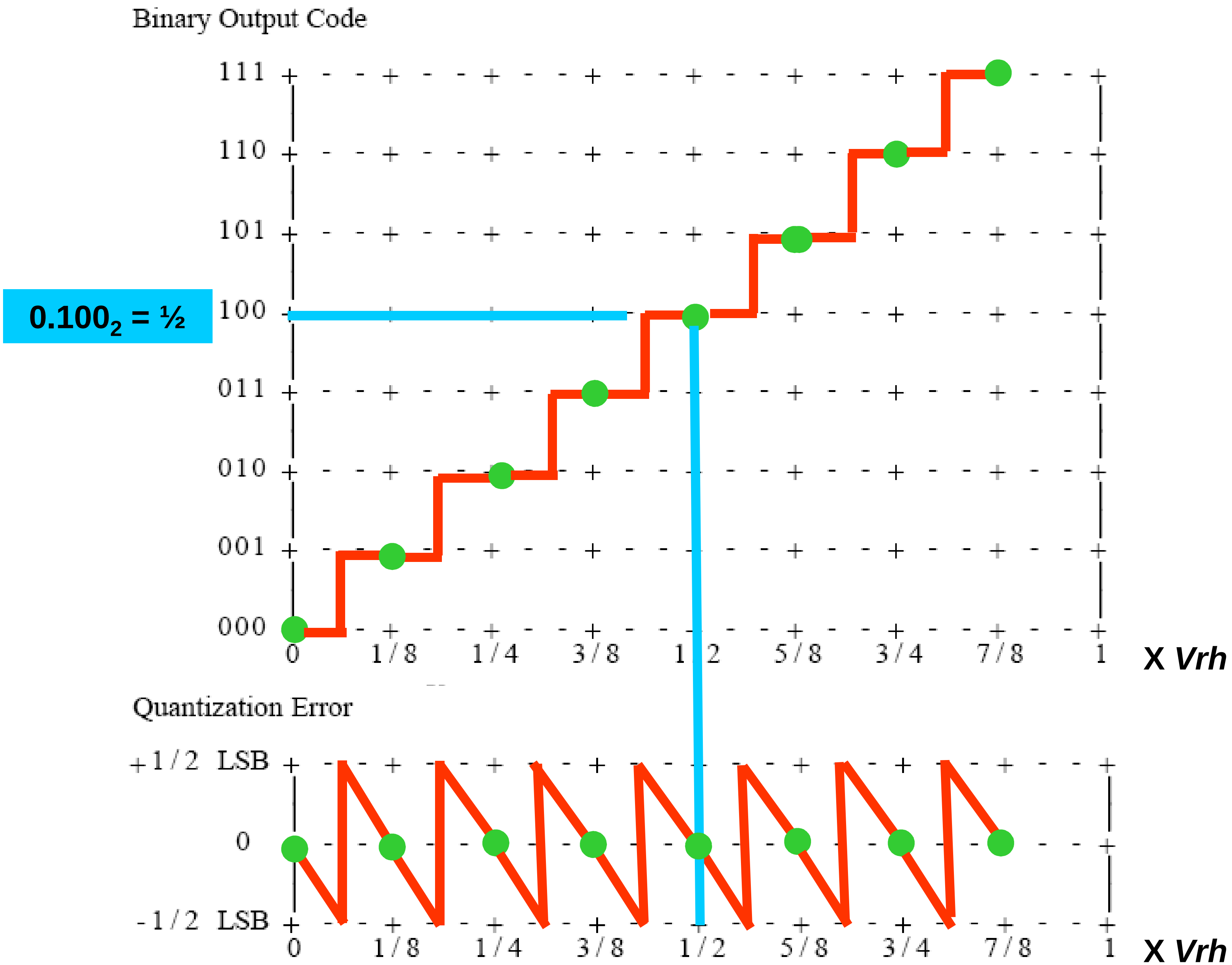


Quantizing Intervals and Quantization Error

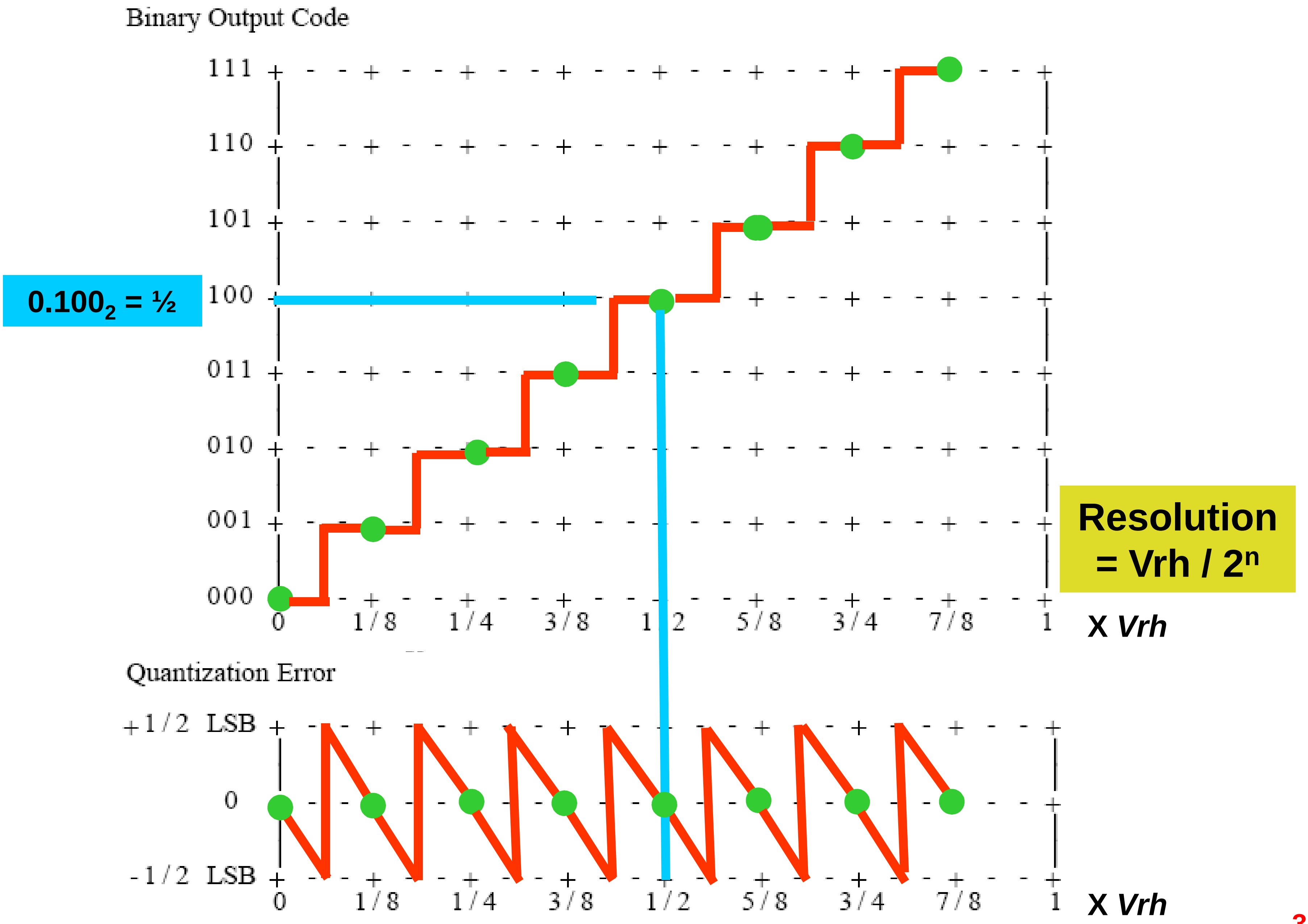




Quantizing Intervals and Quantization Error



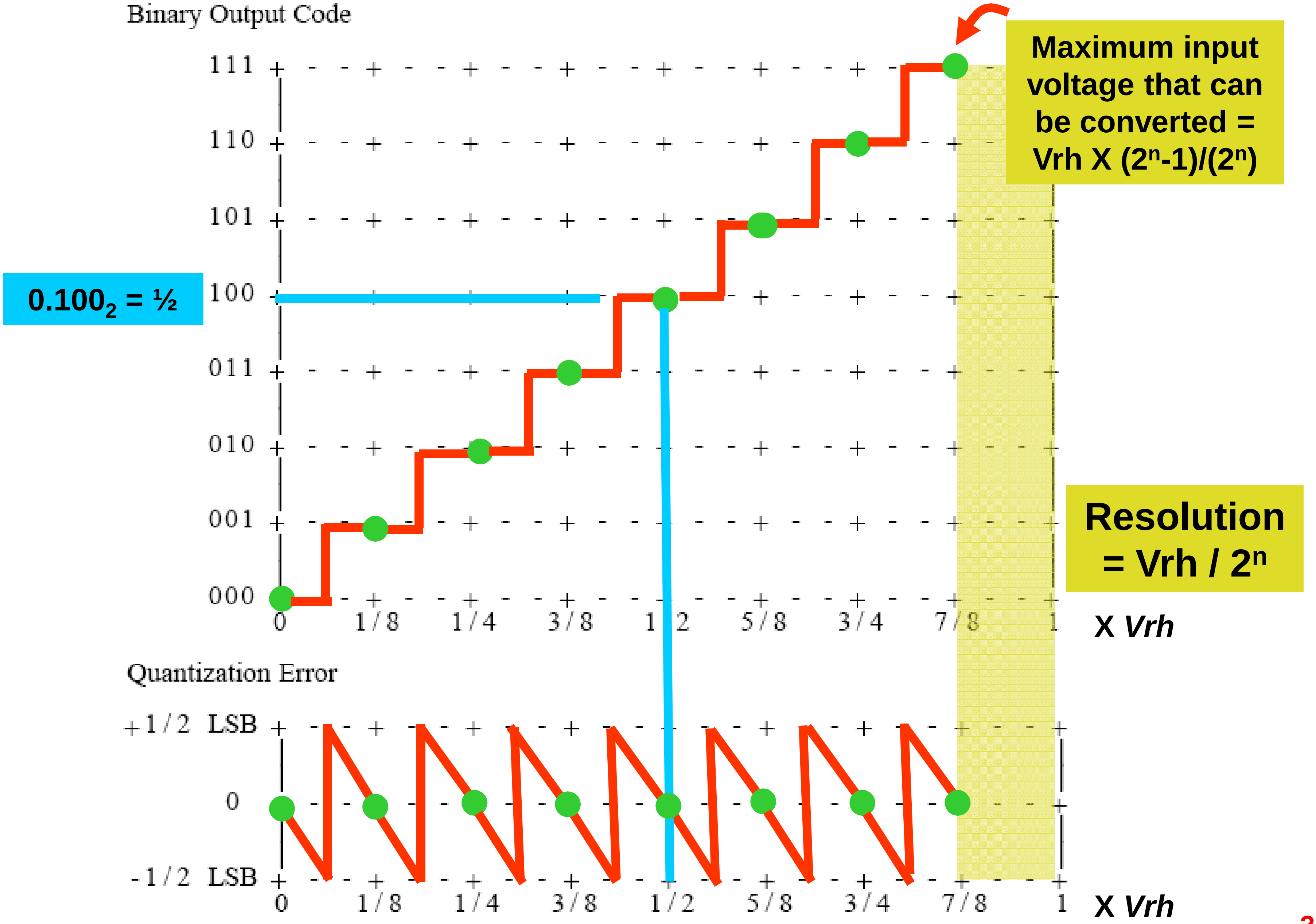
Quantizing Intervals and Quantization Error



0.100_2 = ½

Resolution = $Vrh / 2^n$

Quantizing Intervals and Quantization Error

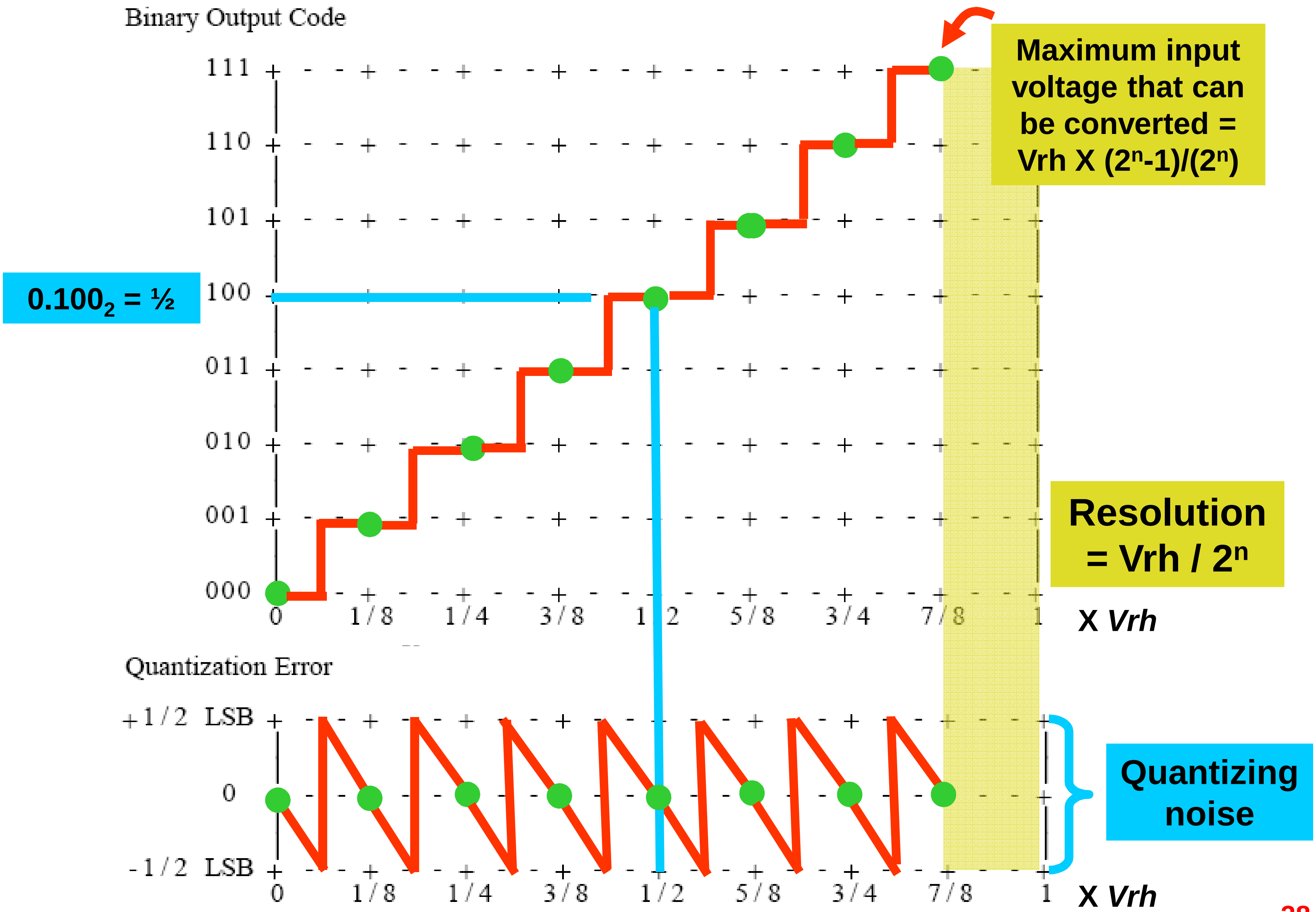


$0.100_2 = ½$

Maximum input voltage that can be converted = Vrh X $(2^n-1)/(2^n)$

Resolution = Vrh / 2^n

Quantizing Intervals and Quantization Error



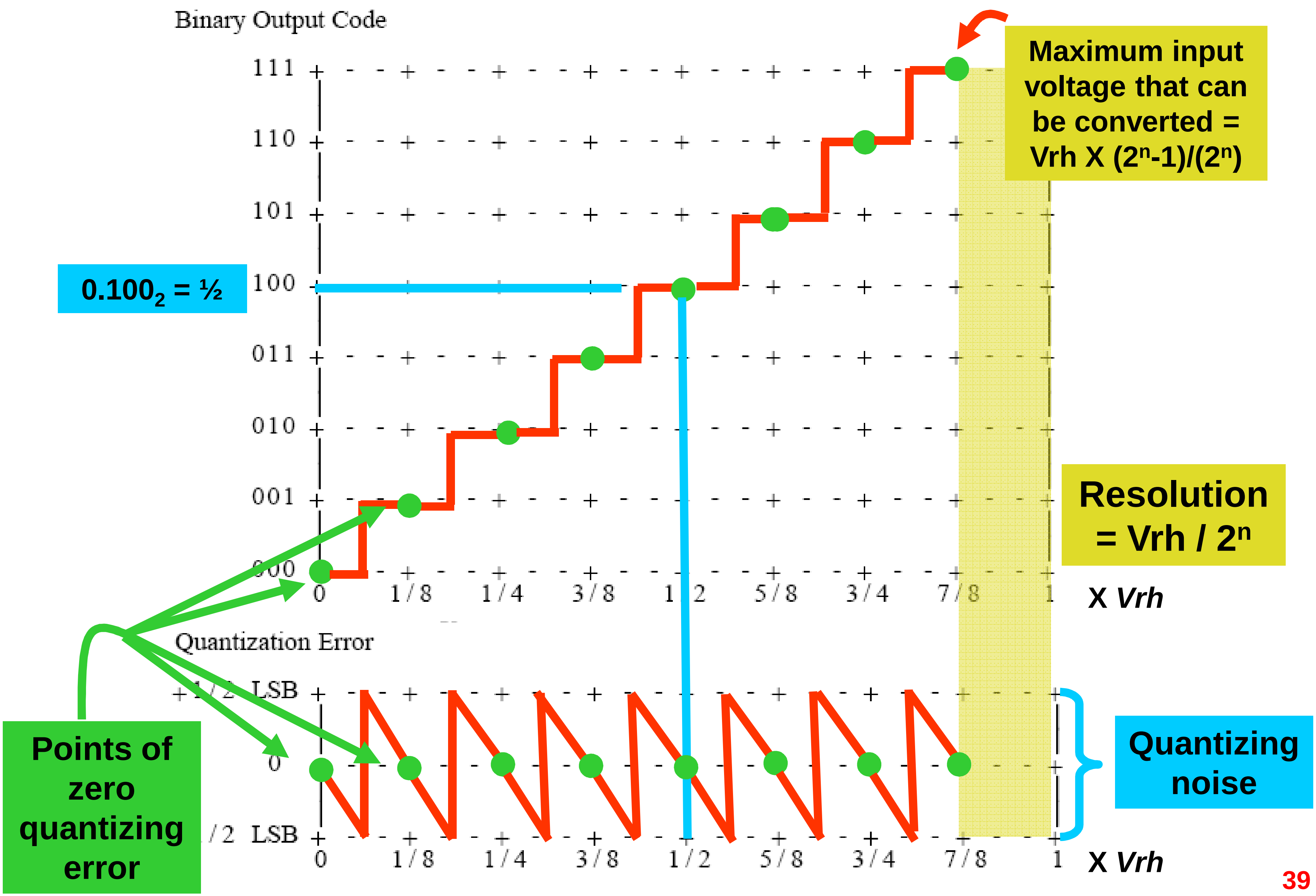
Maximum input voltage that can be converted = Vrh X $(2^n-1)/(2^n)$

0.100_2 = ½

Resolution = Vrh / 2^n

Quantizing noise

Quantizing Intervals and Quantization Error



Maximum input voltage that can be converted = Vrh X $(2^n-1)/(2^n)$

$0.100_2 = ½$

Resolution = Vrh / 2^n

Quantizing noise

Points of zero quantizing error

Analog Conversion Basics

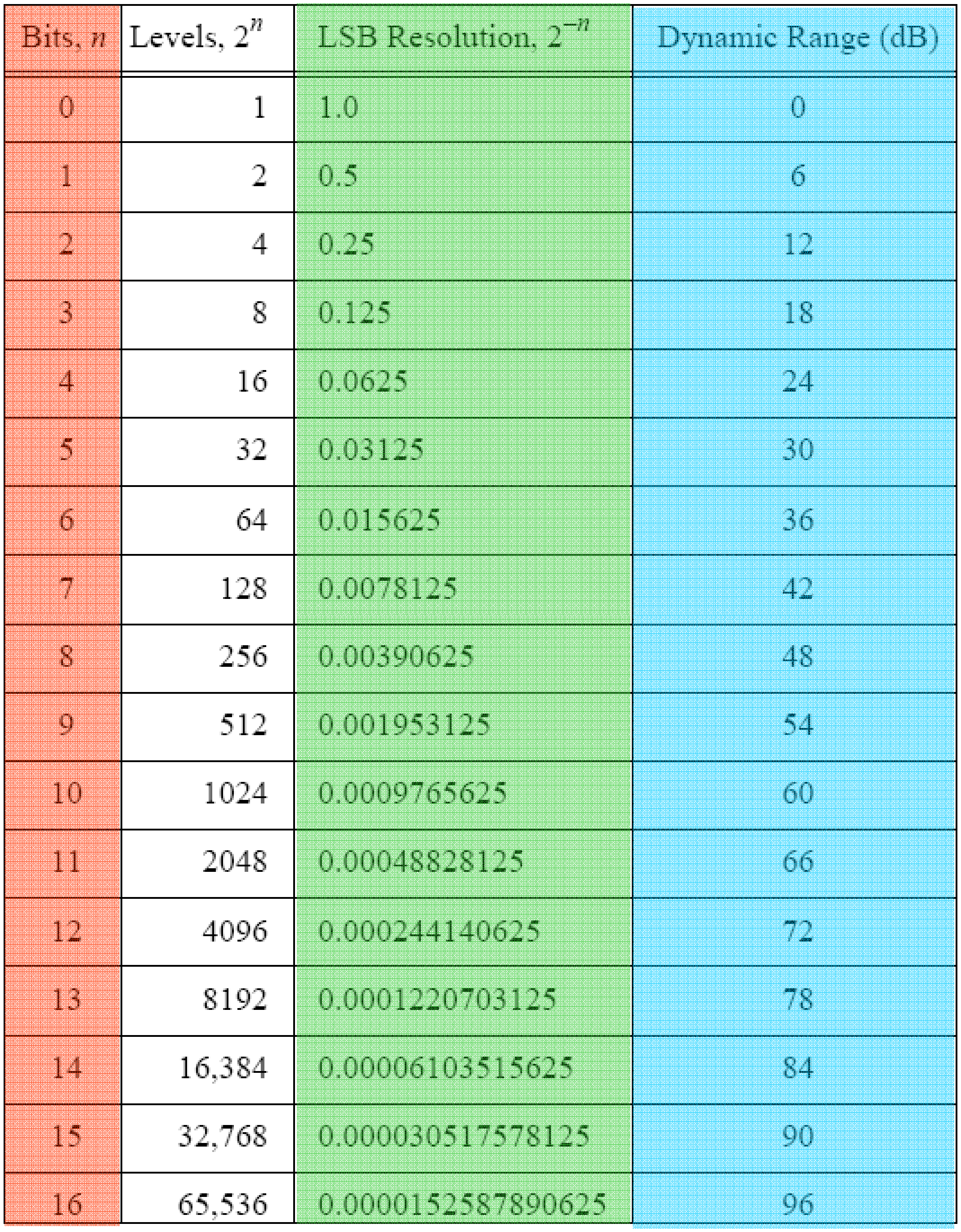
- **Quantization, continued**
  - **The quantization error imposes "noise" on the converted value ("*quantization noise*")**
  - **The *dynamic range* of a converter is its signal to quantizing noise ratio (SQNR), measured in dB: SQNR = 20 $\log_{10}$ (2^n / 1)**

Example: For n = 8,

SQNR = 20 $\log_{10}$ (256 / 1) ≅ 48 dB

Solving for other values of n shows that the dynamic range is approximately 6 dB/bit

Relationship of n to Resolution and Dynamic Range



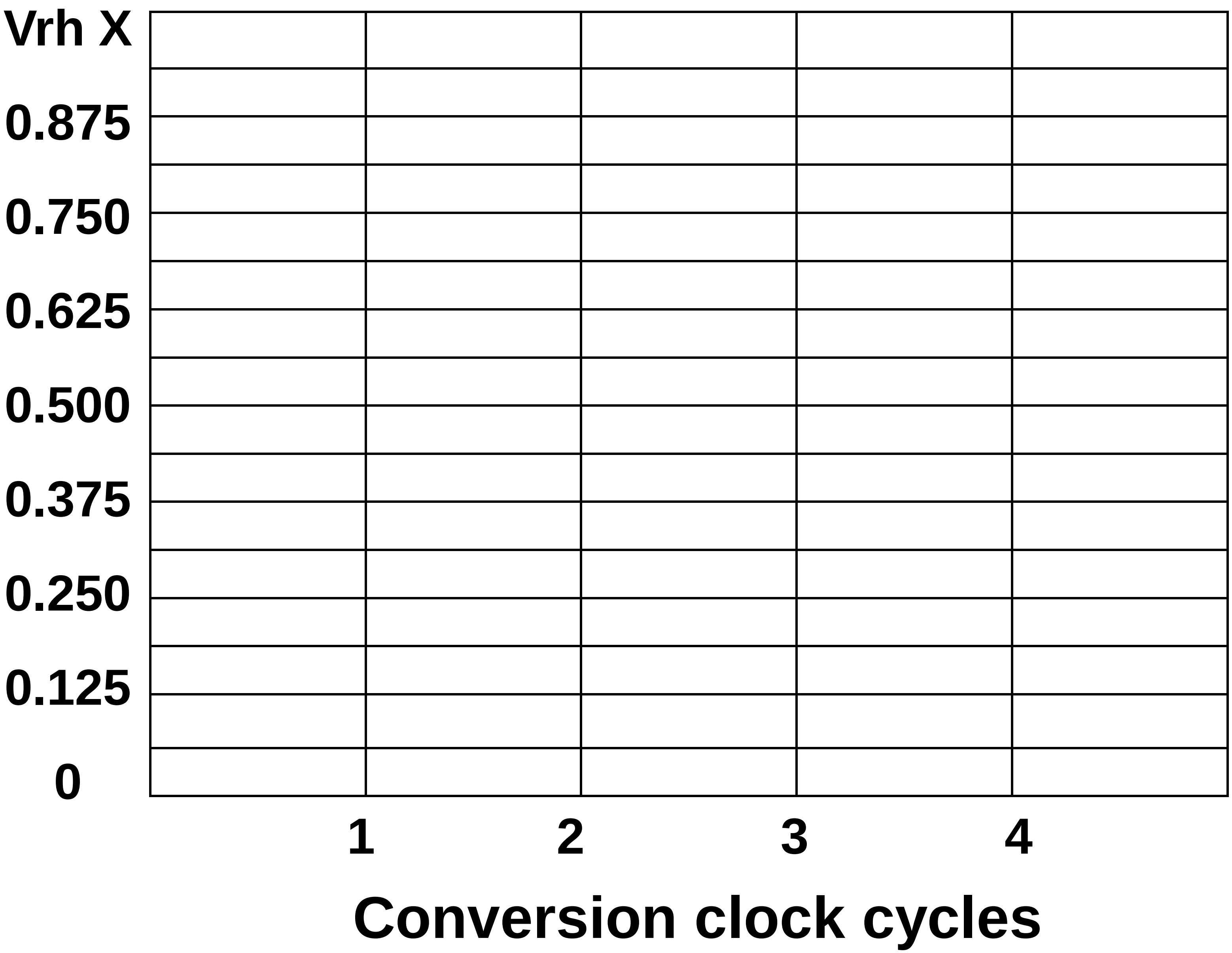
Bits, n	Levels, 2^n	LSB Resolution, 2^{-n}	Dynamic Range (dB)
0	1	1.0	0
1	2	0.5	6
2	4	0.25	12
3	8	0.125	18
4	16	0.0625	24
5	32	0.03125	30
6	64	0.015625	36
7	128	0.0078125	42
8	256	0.00390625	48
9	512	0.001953125	54
10	1024	0.0009765625	60
11	2048	0.00048828125	66
12	4096	0.000244140625	72
13	8192	0.0001220703125	78
14	16,384	0.00006103515625	84
15	32,768	0.000030517578125	90
16	65,536	0.0000152587890625	96

Matlab Demo

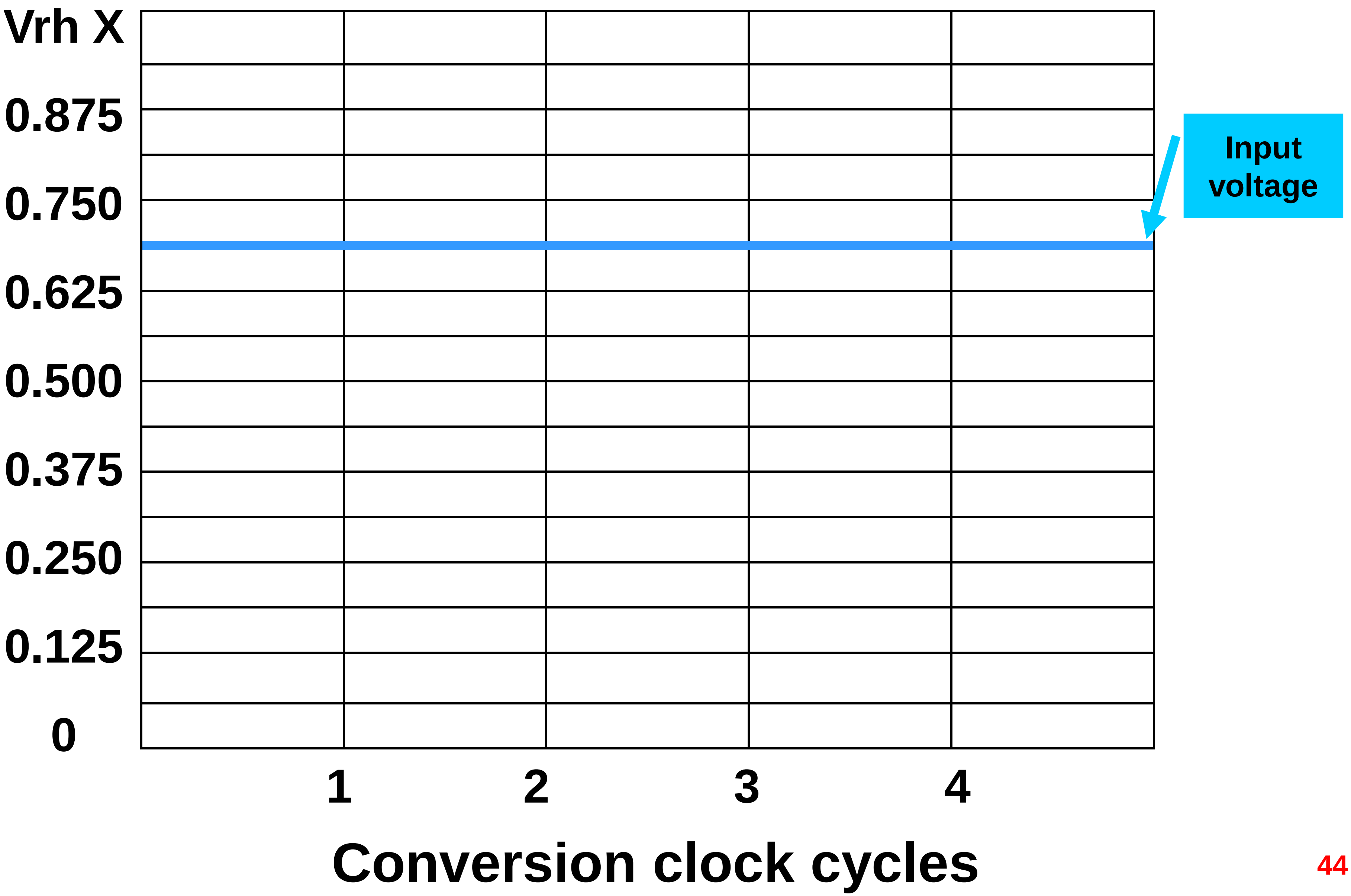
Analog Conversion Basics

- Converter type
  - Wide variety of types, but two basic categories
    - Those requiring a DTA (digital-to-analog converter as an integral component)
    - Those <u>not</u> requiring a DTA
  - *Successive approximation* is one of the most common types of ATD converters integrated into microcontrollers
    - High resolution
    - Conversion time is a *linear function* of n
    - Based on “binary guess a number” game

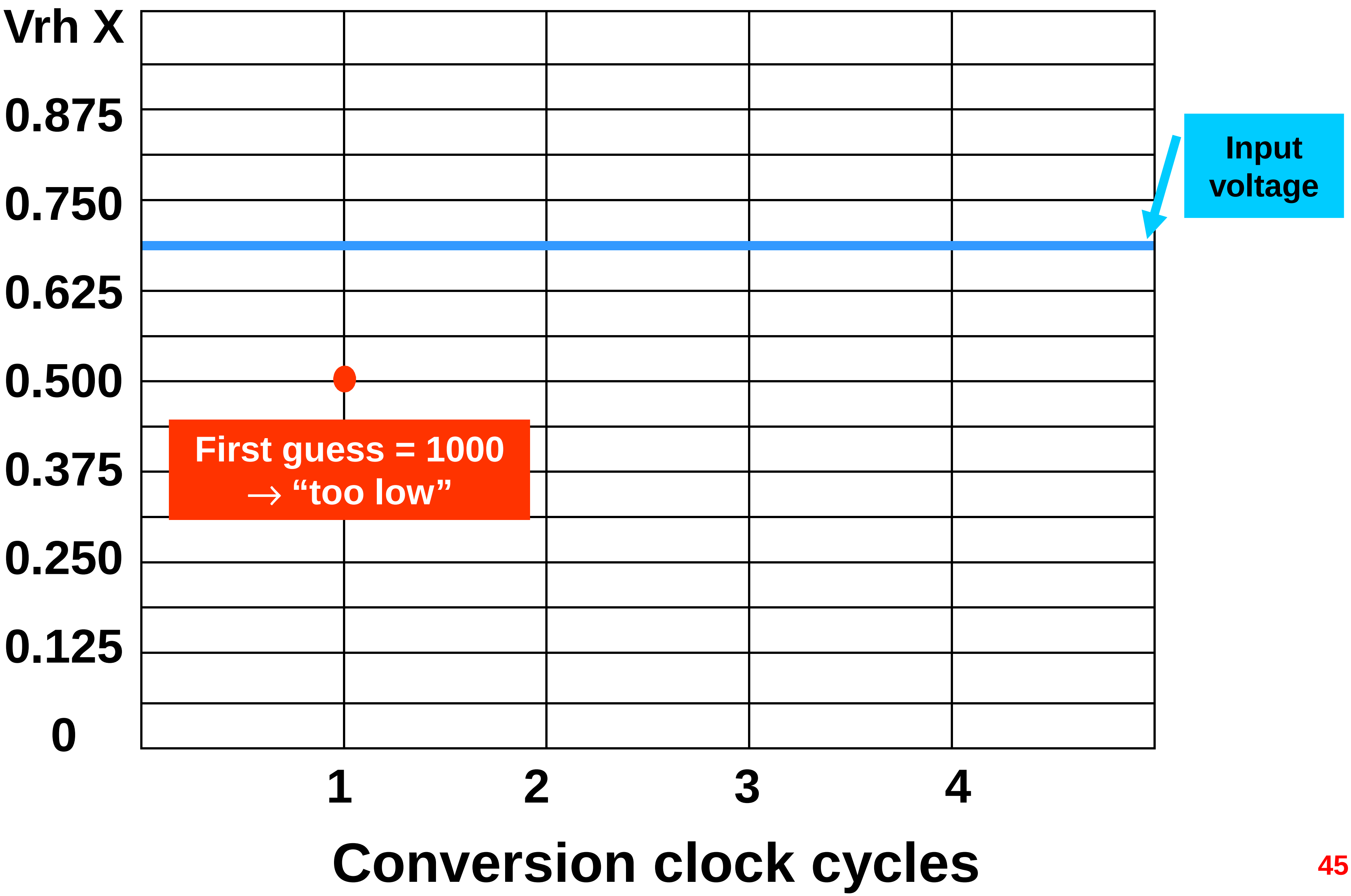
Successive Approximation Conversion Process (n = 4)



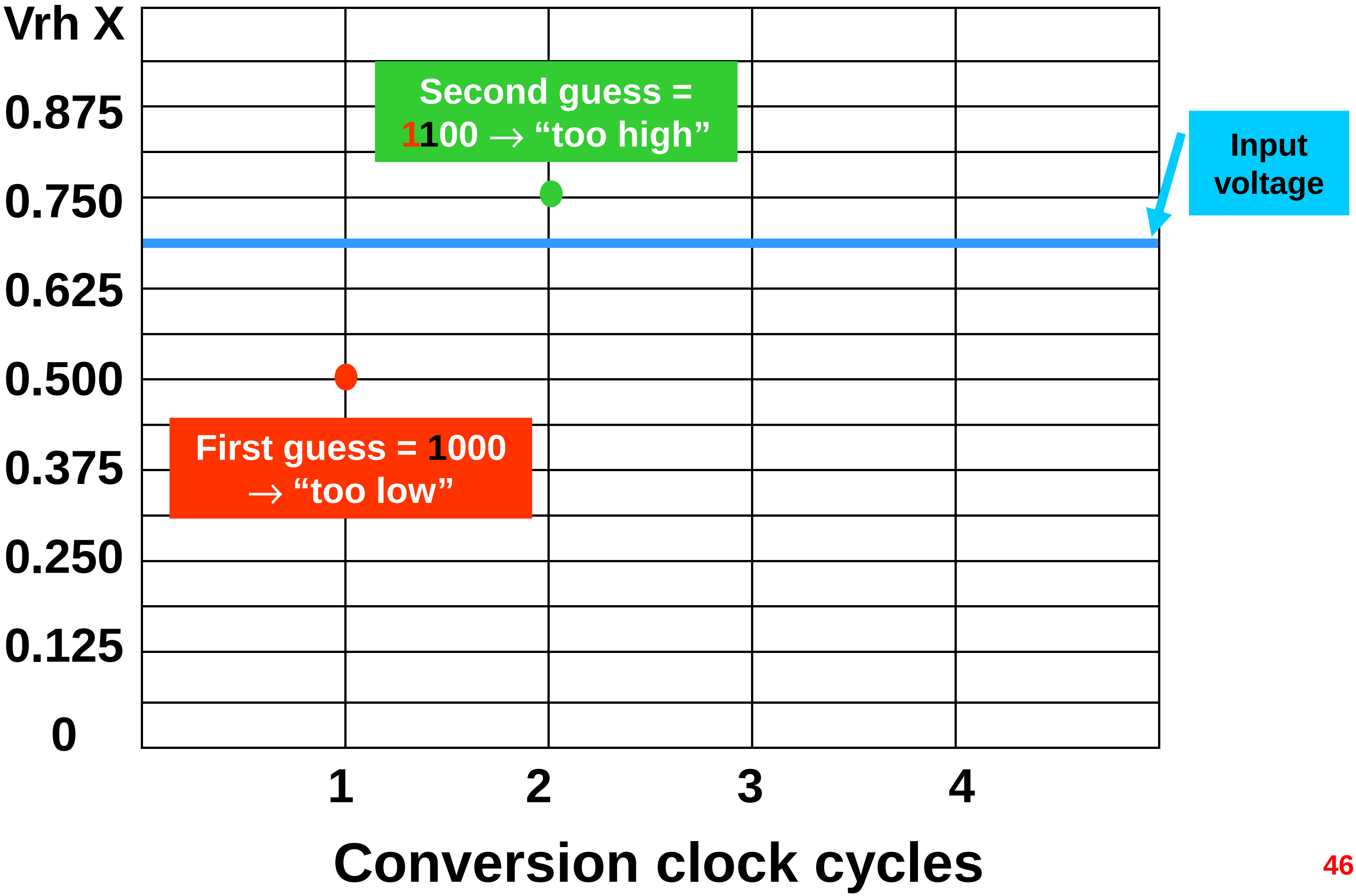
Successive Approximation Conversion Process (n = 4)



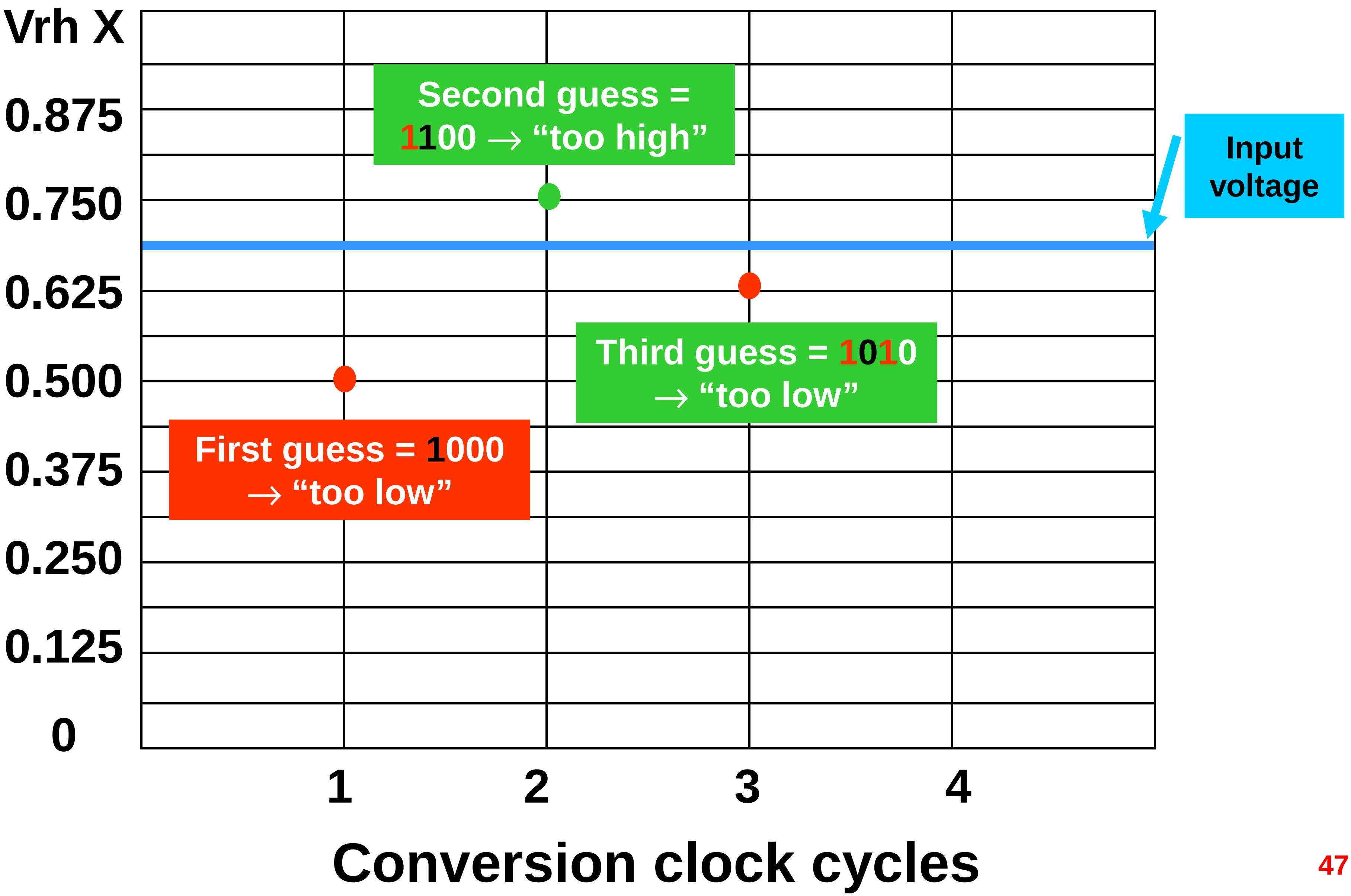
Successive Approximation Conversion Process (n = 4)



Successive Approximation Conversion Process (n = 4)



Successive Approximation Conversion Process (n = 4)



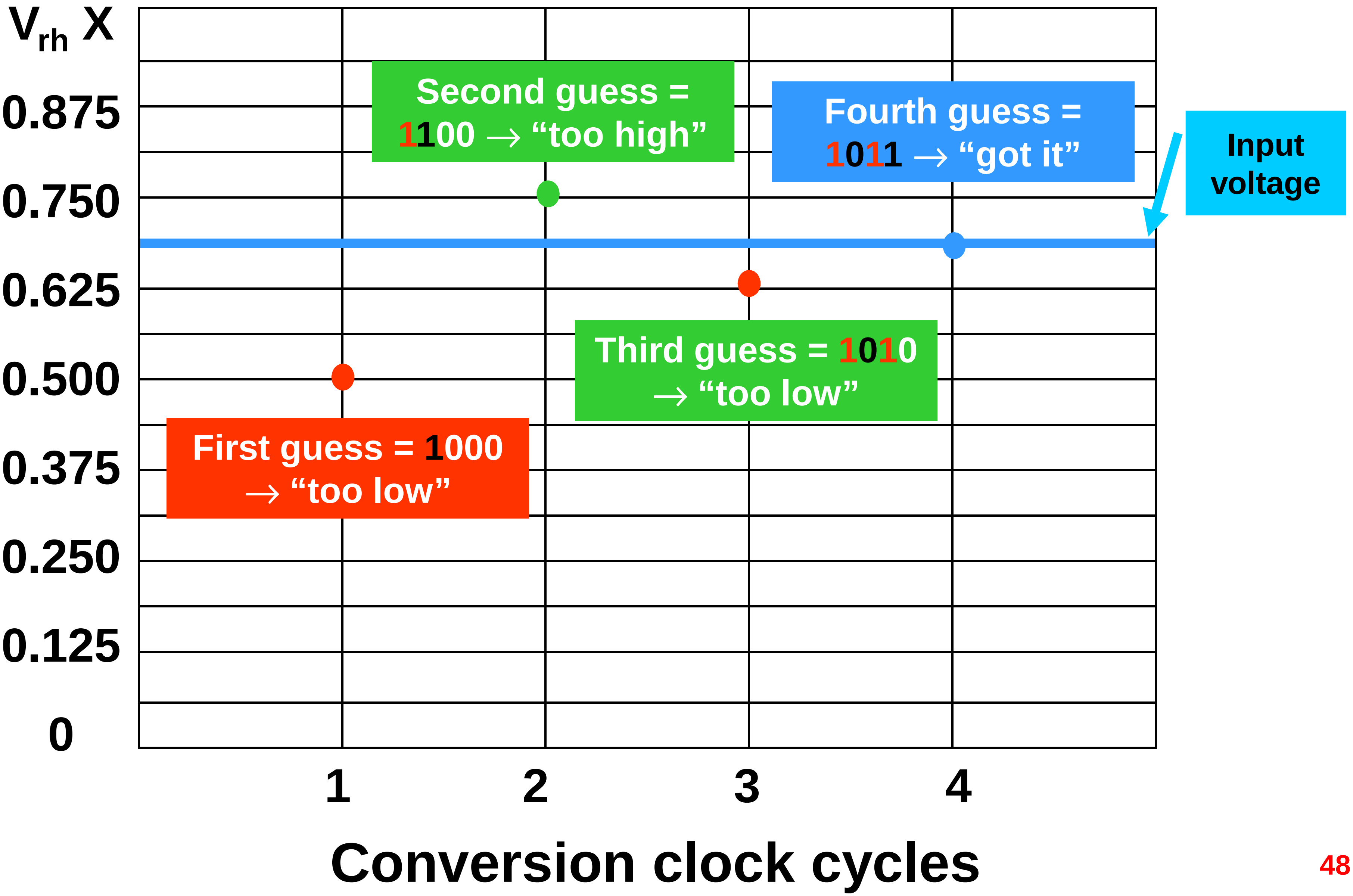
Second guess = 1100 → "too high"

Input voltage

Third guess = 1010 → "too low"

First guess = 1000 → "too low"

Successive Approximation Conversion Process (n = 4)



First guess = 1000 → “too low”

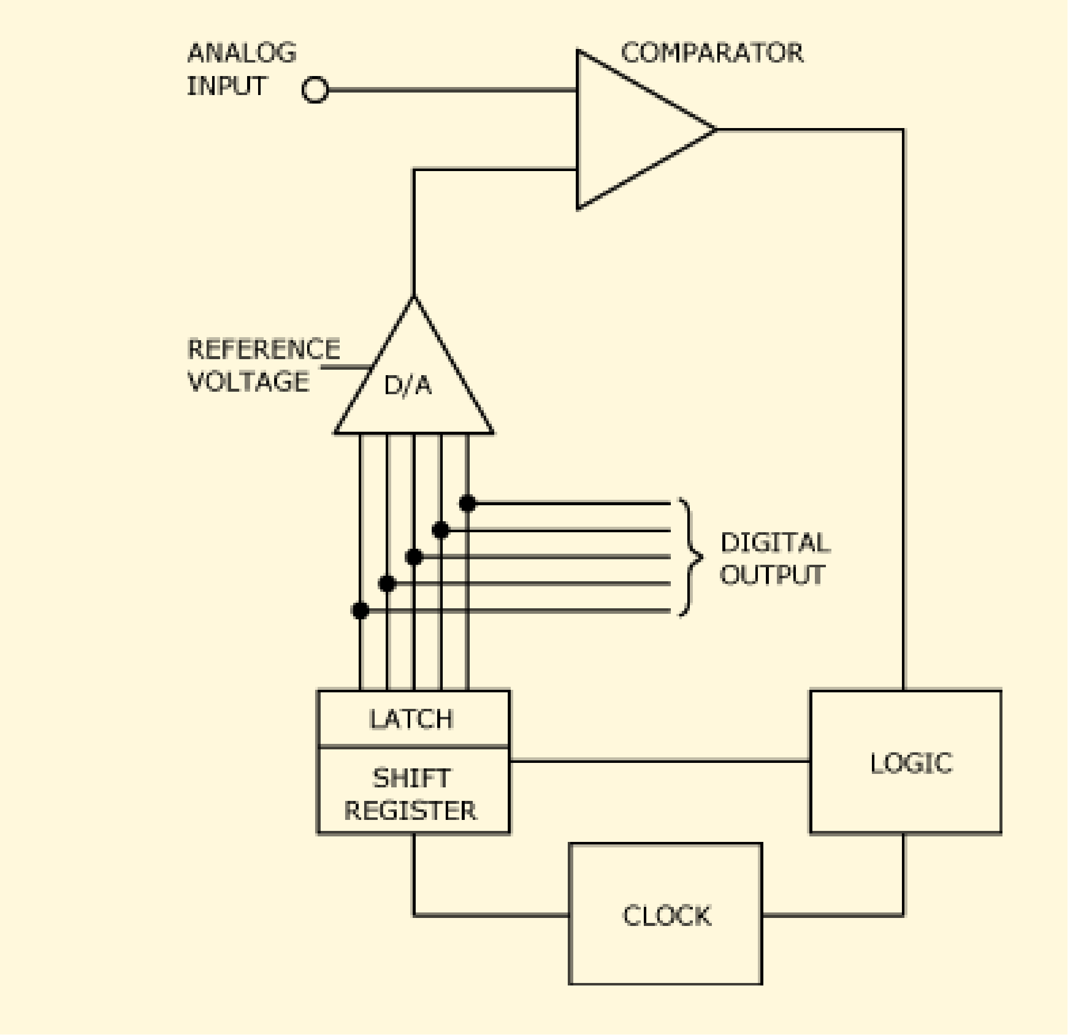
Second guess = 1100 → “too high”

Third guess = 1010 → “too low”

Fourth guess = 1011 → “got it”

Input voltage

Successive Approximation Converter Block Diagram



Clicker Quiz

1. **10-bit PCM audio would have a theoretical SQNR of approximately:**
   A. 10 dB
   B. 30 dB
   C. 60 dB
   D. 100 dB
   E. none of the above

2. **If the desired bandwidth for speech is 5 KHz, the sampling frequency used should be at least:**
   A. 5 KHz
   B. 10 KHz
   C. 20 KHz
   D. 50 KHz
   E. none of the above

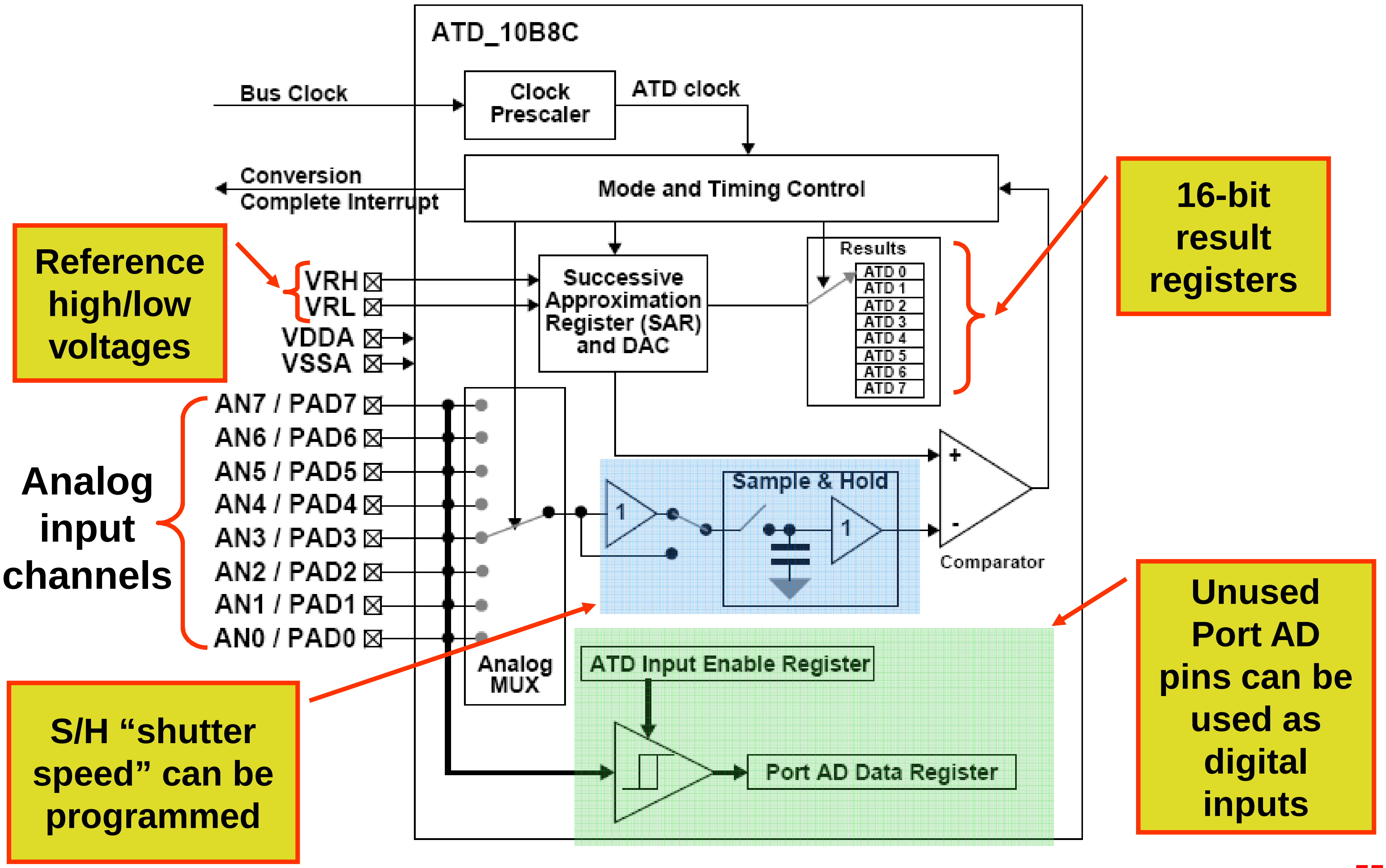
3. If a 7 KHz sine wave is sampled at 10 KHz, the reconstructed waveform will be a sine wave of frequency:

   A. 2 KHz
   B. 3 KHz
   C. 7 KHz
   D. 17 KHz
   E. none of the above

9S12C ATD Features and Modes

- Features
  - 8 input channels
  - 8/10-bit resolution, signed or unsigned
  - 8 separate result registers
  - programmable sample time
  - fast conversion time (8 bits of accuracy in 9 microseconds)
  - may be program- or interrupt-driven
  - port PAD pins may also be used as general-purpose digital inputs (if ATD is not used)
  - Conversion may be externally triggered

ATD Block Diagram



Reference high/low voltages

16-bit result registers

Analog input channels

S/H "shutter speed" can be programmed

Unused Port AD pins can be used as digital inputs

ATD Features and Operating Modes

- Operating modes
  - a single *conversion sequence* consists of from 1 to 8 conversions
  - there are 8 basic *conversion modes*
  - non-scan modes: the sequence complete flag (SCF) is set after the prescribed sequence of conversions has been performed (and the ATD module halts)
  - scan (“continuous conversion”) modes: the sequence complete flag (SCF) is set after the prescribed sequence of conversions has been performed (and the ATD module continues to restart the sequence)

ATD Features and Operating Modes

- Operating modes
  - in all modes, the ***conversion complete flag*** **(CCF)** for a **given channel** is ***set*** when the converted sample is loaded into the corresponding result register
  - the **conversion complete flag (CCF)** for a given channel is automatically ***cleared*** when the corresponding **result register** is ***read***
  - a conversion sequence is **initiated** by ***writing*** to an ATD control register (specifically, **ATDCTL5**)

ATD Features and Operating Modes

- Reference voltages
  - VRH – reference high voltage
  - VRL – reference low voltage
  - minimum (either): -0.3v
  - maximum (either): +6.5v
- Conversion range and resolution

If VRH = 5.0 v and VRL = 0.0 v, then an input voltage of 5.0 v will produce a converted output code of $FF, while an input voltage of 0.0 v will produce a converted output code of $00 (assuming unsigned mode)

Here, the *resolution* (or “step size”) of the converter is (VRH - VRL)/256 = 0.01953125 v

ATD Features and Operating Modes

- **Conversion example:**

If VRH = 5.0 v and VRL = 0.0 v, then an input voltage of 2.90 v will produce what converted output code?

Answer: 2.90 / 0.01953125 = 148 = $94

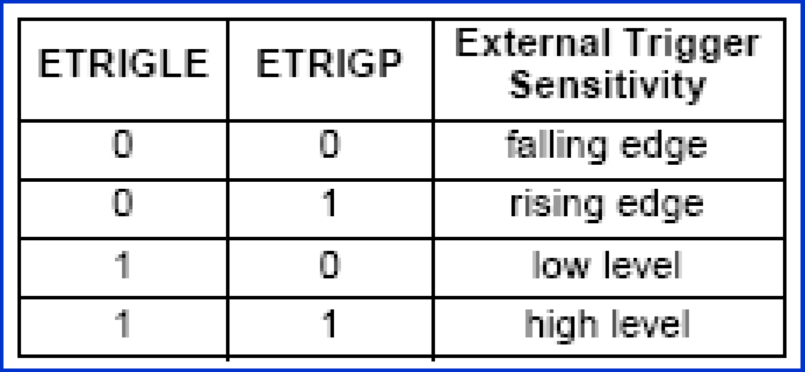
ATD Registers

- **ATDCTL2 (control)**
  - **ADPU (bit 7) – ATD power up**
    - ***“0” – ATD disabled***
    - **“1” – ATD enabled**
  - **AFFC (bit 6) – fast flag clear mode**
    - ***“0” – normal CCF flag clearing mode*** **(must read status register before reading result register to clear individual CCF)**
    - **“1” – fast CCF flag clearing mode (just reading the result register will clear CCF, i.e., do not need to read status register)**

🟩 ***indicates default mode after RESET***

ATD Registers

- ATDCTL2 (control)
  - ETRIGLE and ETRIGP (bits 4-3)



ETRIGLE	ETRIGP	External Trigger Sensitivity
0	0	falling edge
0	1	rising edge
1	0	low level
1	1	high level

Allows use of PAD7 to "trigger" an ATD conversion

  - ETRIGE (bit 2)
    - *"0" – disable external trigger*
    - "1" – enable external trigger

🟩 *indicates default mode after RESET*

ATD Registers

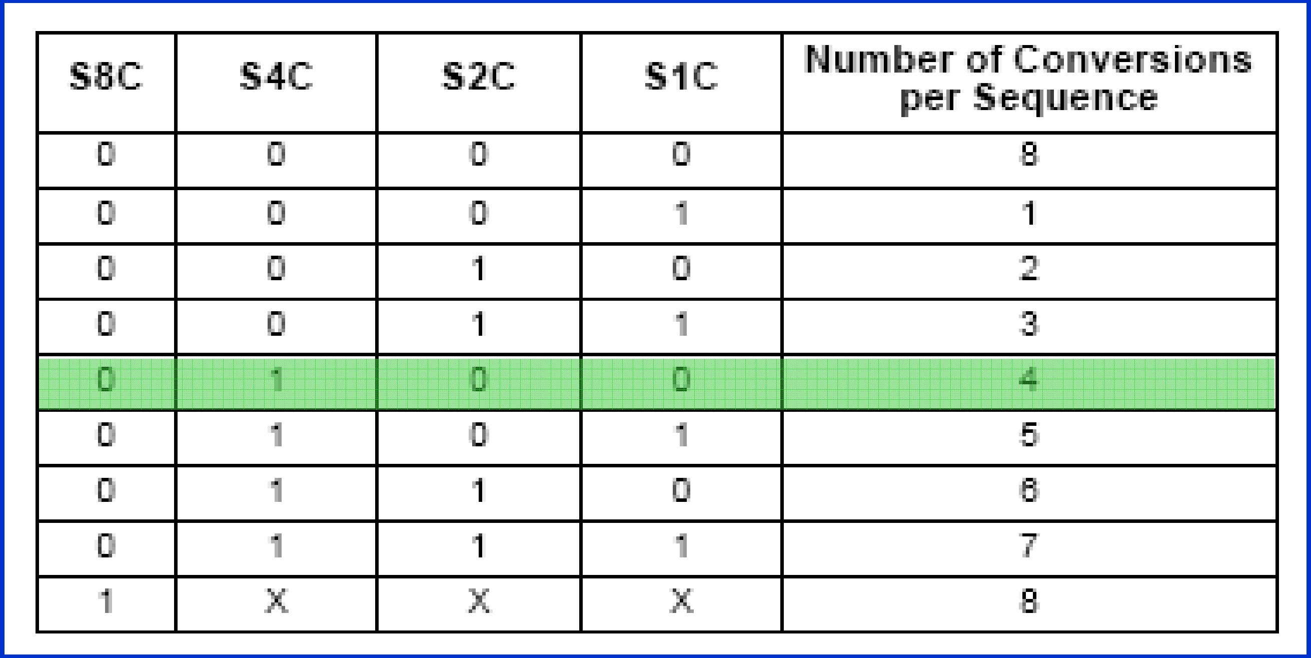
- ATDCTL2 (control)
  - ASCIE (bit 1) – sequence complete interrupt enable
    - “0” – *ATD interrupt disabled*
    - “1” – ATD interrupt enabled
  - ASCIF (bit 0) – interrupt flag
    - “0” – flag not set
    - “1” – flag is set

🟩 *indicates default mode after RESET*

🟦 *indicates read only bit*

ATD Registers

- **ATDCTL3 (control)**
  - **S8C, S4C, S2C, and SC1 (bits 6-3) – conversion sequence length**



S8C	S4C	S2C	S1C	Number of Conversions per Sequence
0	0	0	0	8
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	X	X	X	8

🟩 ***indicates default mode after RESET***

ATD Registers

- **ATDCTL3 (control)**
  - **FIFO (bit 2) – result register FIFO mode**
    - ***"0" – non-FIFO mode*** **(conversion results map into result registers based on the conversion sequence)**
    - **"1" – FIFO mode (conversion results are placed in consecutive result registers between sequences)**

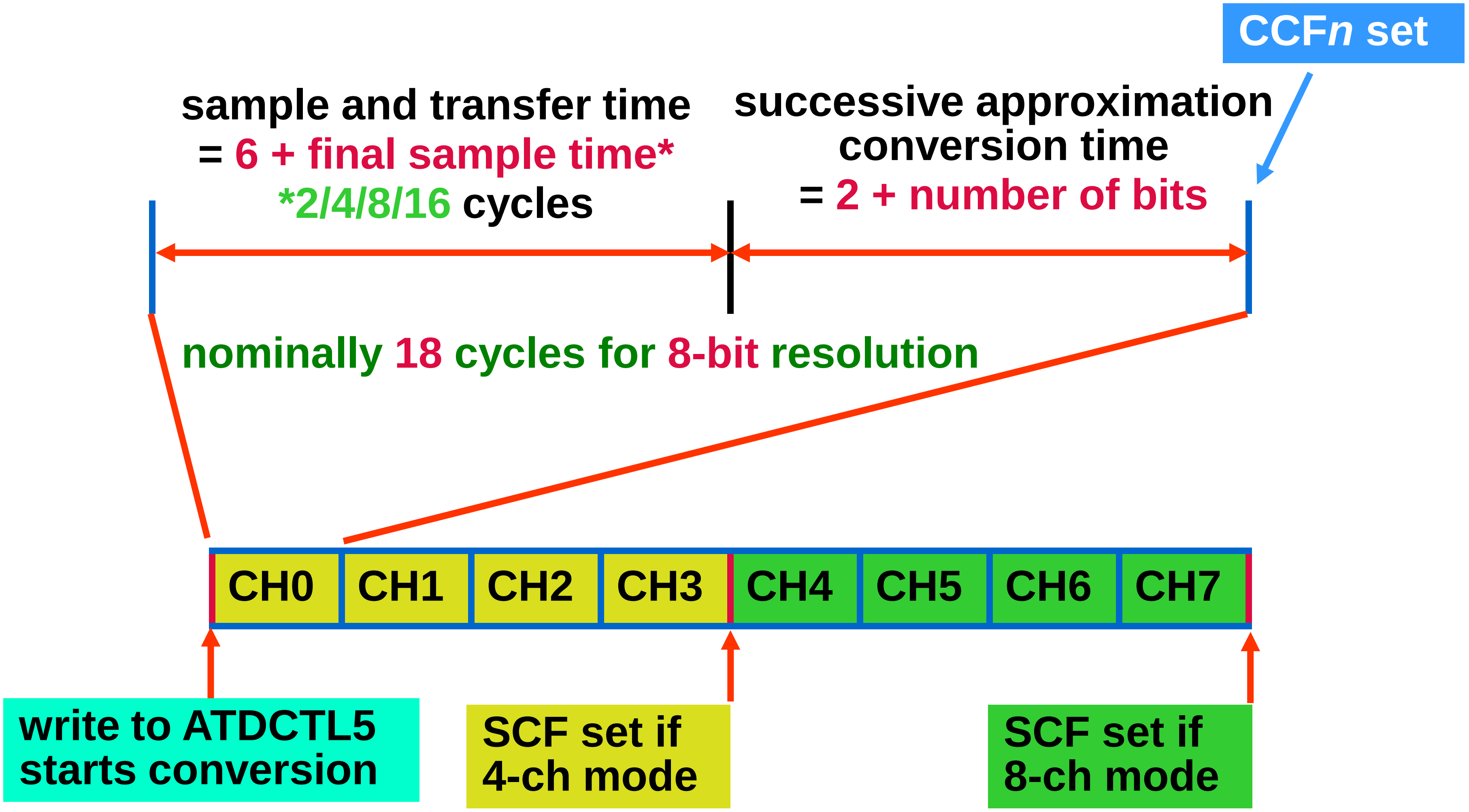
🟩 ***indicates default mode after RESET***

ATD Registers

- ATDCTL4 (control)
  - S10BM (bit 7) 10-bit mode control
    - *“0” – 10-bit resolution*
    - “1” – 8-bit resolution
  - SMP (bits 6 & 5) – sample time select
    - *“00” – 2 ATD clock periods*
    - “01” – 4 ATD clock periods
    - “10” – 8 ATD clock periods
    - “11” – 16 ATD clock periods

🟩 *indicates default mode after RESET*

ATD Conversion Timing



CCF*n* set

sample and transfer time
= 6 + final sample time*
*2/4/8/16 cycles

successive approximation
conversion time
= 2 + number of bits

nominally 18 cycles for 8-bit resolution

CH0	CH1	CH2	CH3	CH4	CH5	CH6	CH7

write to ATDCTL5
starts conversion

SCF set if
4-ch mode

SCF set if
8-ch mode

Note: The SCF bit is *cleared* when a write to ATDCTL5 starts a new conversion (AFFC=0), or the first result register is read (AFFC=1)

ATD Registers

- ATDCTL4 (control)
  - PRS (bits 4-0) – clock pre-scalar
    - “00000” – divide by 2
    - “00001” – divide by 4

      .
      .
      .
    - *“00101” – divide by 12* (24/12 = 2 MHz)

      .
      .
      .
    - “11111” – divide by 64

maximum bus clock frequency

maximum ATD clock frequency

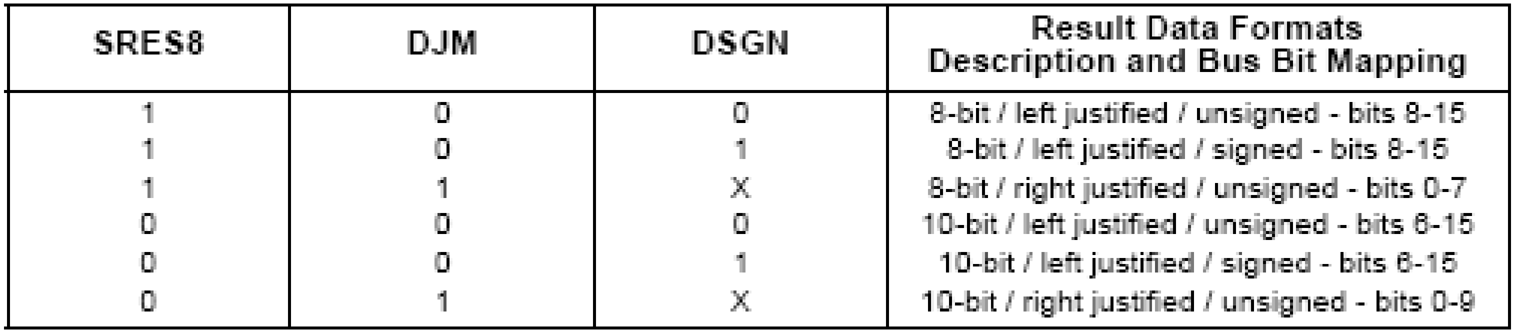
🟩 *indicates default mode after RESET*

ATD Registers

- **ATDCTL5 (control)**
  - DJM (bit 7) – result register data justification
    - ***“0” – left justified data***
    - **“1” – right justified data**
  - DSGN (bit 6) – signed/unsigned result
    - ***“0” – unsigned data***
    - **“1” – two’s complement data**
  - SCAN (bit 5) – enable continuous scan
    - ***“0” – single conversion sequence***
    - **“1” – scan mode (continuous conversion)**

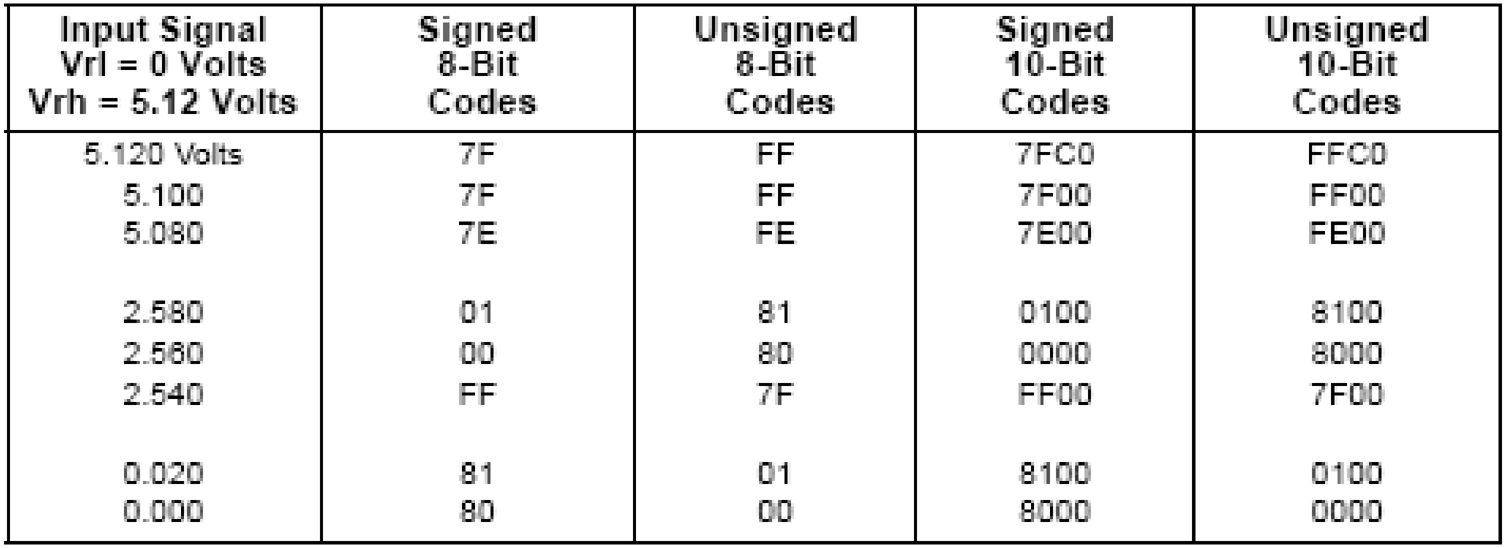
🟩 ***indicates default mode after RESET***

Result Data Formats



SRES8	DJM	DSGN	Result Data Formats Description and Bus Bit Mapping
1	0	0	8-bit / left justified / unsigned - bits 8-15
1	0	1	8-bit / left justified / signed - bits 8-15
1	1	X	8-bit / right justified / unsigned - bits 0-7
0	0	0	10-bit / left justified / unsigned - bits 6-15
0	0	1	10-bit / left justified / signed - bits 6-15
0	1	X	10-bit / right justified / unsigned - bits 0-9

Left Justified Signed/Unsigned Output Codes



Input Signal
Vrl = 0 Volts
Vrh = 5.12 Volts
	Signed
	8-Bit
	Codes
		Unsigned
		8-Bit
		Codes
			Signed
			10-Bit
			Codes
				Unsigned
				10-Bit
				Codes

5.120 Volts	7F	FF	7FC0	FFC0
5.100	7F	FF	7F00	FF00
5.080	7E	FE	7E00	FE00
2.580	01	81	0100	8100
2.560	00	80	0000	8000
2.540	FF	7F	FF00	7F00
0.020	81	01	8100	0100
0.000	80	00	8000	0000

ATD Registers

- **ATDCTL5 (control)**
  - **MULT (bit 4) – multi-channel mode**
    - ***“0” – sample only one channel***
    - **“1” – sample across *multiple* (successive) channels**
  - **CC-CB-CA (bits 2-0) – input channel select *(000 by default)***

🟩 ***indicates default mode after RESET***

ATD Registers

- **ATDSTAT (status, high byte)**
  - **SCF (bit 7) – sequence complete flag**
    - **“0” – *conversion sequence not complete***
    - **“1” – conversion sequence is complete**
    - **Cleared by either writing a “1” to this bit -or- starting a new conversion by writing to ATDCTL5 (note that the SCF flag *automatically clears* if AFFC=1 and a result register is read)**

🟩 ***indicates default mode after RESET***

ATD Registers

- **ATDSTAT (status, high byte)**
  - **– ETORF (bit 5) – external trigger overrun flag**
    - **“0” – *no external trigger overrun occurred***
    - **“1” – edge detected while conversion in process → overrun occurred**
    - **Cleared by writing a “1” to the ETORF bit or by starting a new conversion**

🟩 ***indicates default mode after RESET***

ATD Registers

- **ATDSTAT (status, high byte)**
  - **FIFOR (bit 4) – FIFO overrun flag**
    - ***“0” – no FIFO overrun occurred***
    - **“1” – result register has been written before its associated conversion complete flag (CCF) has been cleared**
    - **Cleared by writing a “1” to the FIFOR bit or by starting a new conversion**
  - **CC (bits 2-0) – conversion counter for current sequence of conversions (read only)**

🟩 ***indicates default mode after RESET***

ATD Registers

- ATDSTAT (status, low byte)
  - CCF (bits 7-0) – conversion complete flags for each ATD result register
    - *“0” – conversion not complete*
    - “1” – conversion is complete

🟩 *indicates default mode after RESET*

ATD Registers

- ATDDIEN (digital input enable)
  - IEN (bits 7-0) – digital input enable on channel on Port AD pin
    - *“0” – Port AD bit used as analog input*
    - “1” – Port AD bit used as digital input

🟩 *indicates default mode after RESET*

ATD Registers

- PORTAD (data input) register
  - Returns digital value on Port AD pin if the corresponding ATDDIEN register bit is set to “1”
- ADRxH (result, 16-bit) registers
  - ADR0H
  - ADR1H
  - ADR2H
  - ADR3H
  - ADR4H
  - ADR5H
  - ADR6H
  - ADR7H

Clicker Quiz

1. A conversion sequence can consist of up to __ conversions:
   A. 1
   B. 2
   C. 4
   D. 8
   E. none of the above

2. A conversion sequence is initiated by writing the (starting) channel number to this register:

   A. ATDCTL2
   B. ATDCTL3
   C. ATDCTL4
   D. ATDCTL5
   E. none of the above

3. In non-FIFO mode, the first converted result will always be written to:

   A. ADR0H
   B. ADR1H
   C. the result register corresponding to the converted channel
   D. the next available result register
   E. none of the above

4. **The results from a conversion sequence are valid when this flag is set:**

   A. SCF

   B. CCF

   C. ETORF

   D. FIFOR

   E. none of the above

ATD Initialization – Routine

```
; Initializes ATD for program-driven operation
; STEP (1) Enable ATD for normal flag clear
;               mode with interrupts disabled

atd_ini   movb    #____,atdctl2

; STEP (2) Set the conversion sequence length to one

          movb    #____,atdctl3

; STEP (3) Select 8-bit resolution and nominal sample time

          movb    #____,atdctl4
          rts
```

ATD Registers

- ATDCTL2 (control)
  - ADPU (bit 7) – ATD power up
    - *“0” – ATD disabled*
    - “1” – ATD enabled
  - AFFC (bit 6) – fast flag clear mode
    - *“0” – normal CCF flag clearing mode* (must read status register before reading result register to clear individual CCF)
    - “1” – fast CCF flag clearing mode (just reading the result register will clear CCF, i.e., do not need to read status register)

🟩 *indicates default mode after RESET*

ATD Registers

- **ATDCTL2 (control)**
  - **ASCIE (bit 1) – sequence complete interrupt enable**
    - ★ **“0” – *ATD interrupt disabled***
    - **“1” – ATD interrupt enabled**
  - **ASCIF (bit 0) – interrupt flag**
    - **“0” – flag not set**
    - **“1” – flag is set**

🟩 ***indicates default mode after RESET***

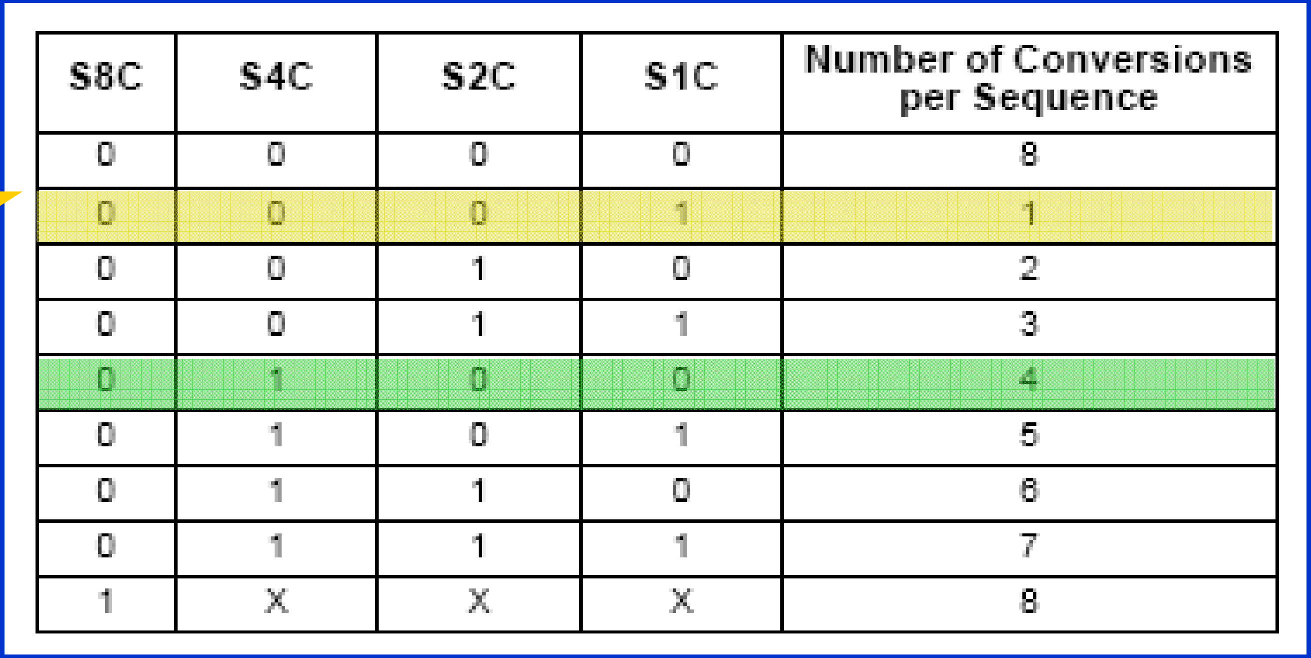
🟦 ***indicates read only bit***

ATD Initialization – Routine

```
; Initializes ATD for program-driven operation
; STEP (1) Enable ATD for normal flag clear
;               mode with interrupts disabled

atd_ini   movb    #$80,atdctl2

; STEP (2) Set the conversion sequence length to one

          movb    #____,atdctl3

; STEP (3) Select 8-bit resolution and nominal sample time

          movb    #____,atdctl4
          rts
```

ATD Registers

- **ATDCTL3 (control)**
  - **S8C, S4C, S2C, and SC1 (bits 6-3) – conversion sequence length**



S8C	S4C	S2C	S1C	Number of Conversions per Sequence
0	0	0	0	8
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	X	X	X	8

🟩 *indicates default mode after RESET*

ATD Initialization – Routine

```
; Initializes ATD for program-driven operation
; STEP (1) Enable ATD for normal flag clear
;                mode with interrupts disabled

atd_ini   movb    #$80,atdctl2

; STEP (2) Set the conversion sequence length to one

          movb    #$08,atdctl3

; STEP (3) Select 8-bit resolution and nominal sample time

          movb    #____,atdctl4
          rts
```

ATD Registers

- ATDCTL4 (control) – $0064
  - S10BM (bit 7) 10-bit mode control
    - *“0” – 10-bit operation*
    - ★ “1” – 8-bit operation
  - SMP (bits 6 & 5) – sample time select
    - ★ *“00” – 2 ATD clock periods*
    - “01” – 4 ATD clock periods
    - “10” – 8 ATD clock periods
    - “11” – 16 ATD clock periods

🟩 *indicates default mode after RESET*

ATD Registers

- ATDCTL4 (control)
  - PRS (bits 4-0) – clock pre-scalar
    - “00000” – divide by 2
    - “00001” – divide by 4

      .
      .
      .
    - ★ *“00101” – divide by 12* (24/12 = 2 MHz)

      .
      .
      .
    - “11111” – divide by 64

maximum core clock frequency

maximum ATD clock frequency

🟩 *indicates default mode after RESET*

ATD Initialization – Routine

```
; Initializes ATD for program-driven operation
; STEP (1) Enable ATD for normal flag clear
;                mode with interrupts disabled

atd_ini  movb    #$80,atdctl2

; STEP (2) Set the conversion sequence length to one

         movb    #$08,atdctl3

; STEP (3) Select 8-bit resolution and nominal sample time

         movb    #$85,atdctl4
         rts
```

Program-Driven ATD Device Driver

- At entry, (B) contains channel number; at exit, (A) contains converted sample
  1. Write 3-bit channel number to lower 3 bits of ATDCTL5 register
  2. Wait for conversion sequence to complete
  3. Read sample from result register and return the result in (A)

Program-Driven ATD Device Driver

```
; Channel number passed in B register
; Converted sample returned in A register

; STEP (1) Start conversion on desired channel
inatd  andb #$07      ;mask unwanted bits
       stab atdctl5   ;start conversion on
                      ;selected channel in
                      ;“default” mode
; STEP (2) Wait for conversion sequence to finish
await  brclr     ______,_______,_______

; STEP (3) Read result register and exit
      ldaa ______  ;(A)=(selected result)
      rts
```

ATD Registers

- **ATDCTL5 (control)**
  - DJM (bit 7) – result register data justification
    - ***“0” – left justified data***
    - “1” – right justified data
  - DSGN (bit 6) – signed/unsigned result
    - ***“0” – unsigned data***
    - “1” – two’s complement data
  - SCAN (bit 5) – enable continuous scan
    - ***“0” – single conversion sequence***
    - “1” – scan mode (continuous conversion)

🟩 ***indicates default mode after RESET***

ATD Registers

- **ATDCTL5 (control)**
  - **MULT (bit 4) – multi-channel mode**
    - **"0" – *sample only one channel***
    - **"1" – sample across *multiple* (successive) channels**
  - **CC-CB-CA (bits 2-0) – input channel select *(000 by default)***

🟩 ***indicates default mode after RESET***

Program-Driven ATD Device Driver

; Channel number passed in B register

; Converted sample returned in A register

; STEP (1) ***Start conversion on desired channel***

```
inatd  andb #$07     ;mask unwanted bits
       stab atdctl5 ;start conversion on
                    ;selected channel in
                    ;“default” mode
```

; STEP (2) ***Wait for conversion sequence to finish***

```
await  brclr  atdstat,$80,await
```

; STEP (3) ***Read result register and exit***

```
      ldaa ______ ;(A)=(selected result)
      rts
```

ATD Registers

- **ATDSTAT (status, high byte)**
  - **SCF (bit 7) – sequence complete flag**
    - ***“0” – conversion sequence not complete***
    - **“1” – conversion sequence is complete**
    - **Cleared by either writing a “1” to this bit -or- starting a new conversion by writing to ATDCTL5 (note that the SCF flag *automatically clears* if AFFC=1 and a result register is read)**

🟩 ***indicates default mode after RESET***

Program-Driven ATD Device Driver

```
; Channel number passed in B register
; Converted sample returned in A register

; STEP (1) Start conversion on desired channel
inatd  andb #$07    ;mask unwanted bits
       stab atdctl5 ;start conversion on
                    ;selected channel in
                    ;“default” mode
; STEP (2) Wait for conversion sequence to finish
await  brclr  atdstat,$80,await

; STEP (3) Read result register and exit
      ldaa  adr0h ;(A)=(selected result)
      rts
```

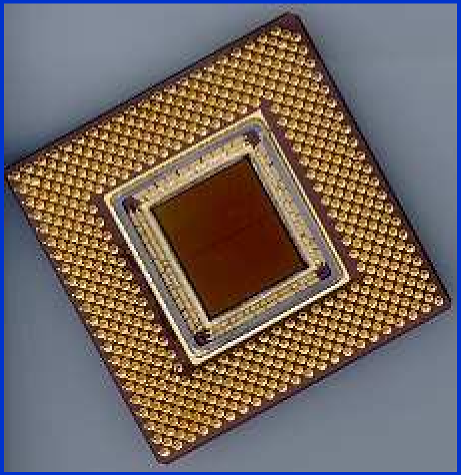
ATD Registers

- ATDCTL3 (control)
  - FIFO (bit 2) – result register FIFO mode
    - ★ “0” – *non-FIFO mode* (conversion results map into result registers based on the conversion sequence)
    - “1” – FIFO mode (conversion results are placed in consecutive result registers between sequences)

🟩 ***indicates default mode after RESET***

ATD Registers

- PORTAD (data input) register
  - Returns digital value on Port AD pin if the corresponding ATDDIEN register bit is set to “1”
- ADRxH (result, 16-bit) registers
  - ADR0H
  - ADR1H
  - ADR2H
  - ADR3H
  - ADR4H
  - ADR5H
  - ADR6H
  - ADR7H



Microcontroller-Based Digital System Design

Module 3-B Serial Communications Interface (SCI)

Learning Objectives

- describe asynchronous serial communication character frame format and justify the need for start/stop bits
- compare non-return-to-zero (NRZ) serial data encoding with Manchester encoding
- describe the difference between half-duplex with full-duplex communication
- define baud rate and describe its relation to the local clock frequency
- diagnose potential errors that can occur in asynchronous serial communications (parity, framing, overrun) and describe how they can be corrected
- distinguish between single-ended and differential serial interfaces, and cite examples of each
- describe the purpose and operation of a level translator (line driver) chip
- identify SCI features and operating modes
- configure the SCI to operate in a prescribed mode
- create an SCI device driver routine for a prescribed application

Outline

- Background and motivation
- Asynchronous serial transmission
- SCI features
- SCI registers
- Program-driven SCI operation
- Interrupt-driven SCI operation

Reference: **Serial Communications Interface (SCI) Block User Guide**

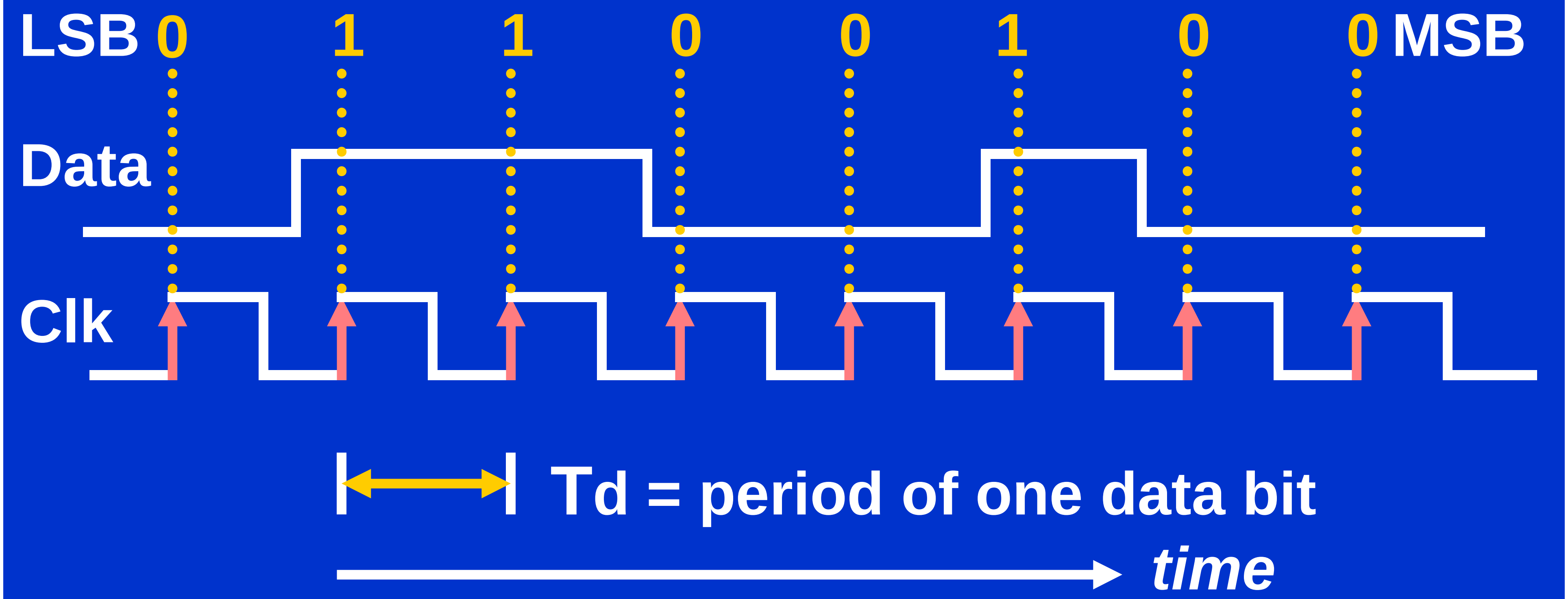
Background and Motivation

- Motivation
  - transmitting “parallel” data over a long distance using multi-conductor cable is expensive
    - each signal requires a “twisted pair”
    - each signal requires a line driver and receiver

Solution: Transmit data serially (potentially using a *single* twisted pair)

Background and Motivation

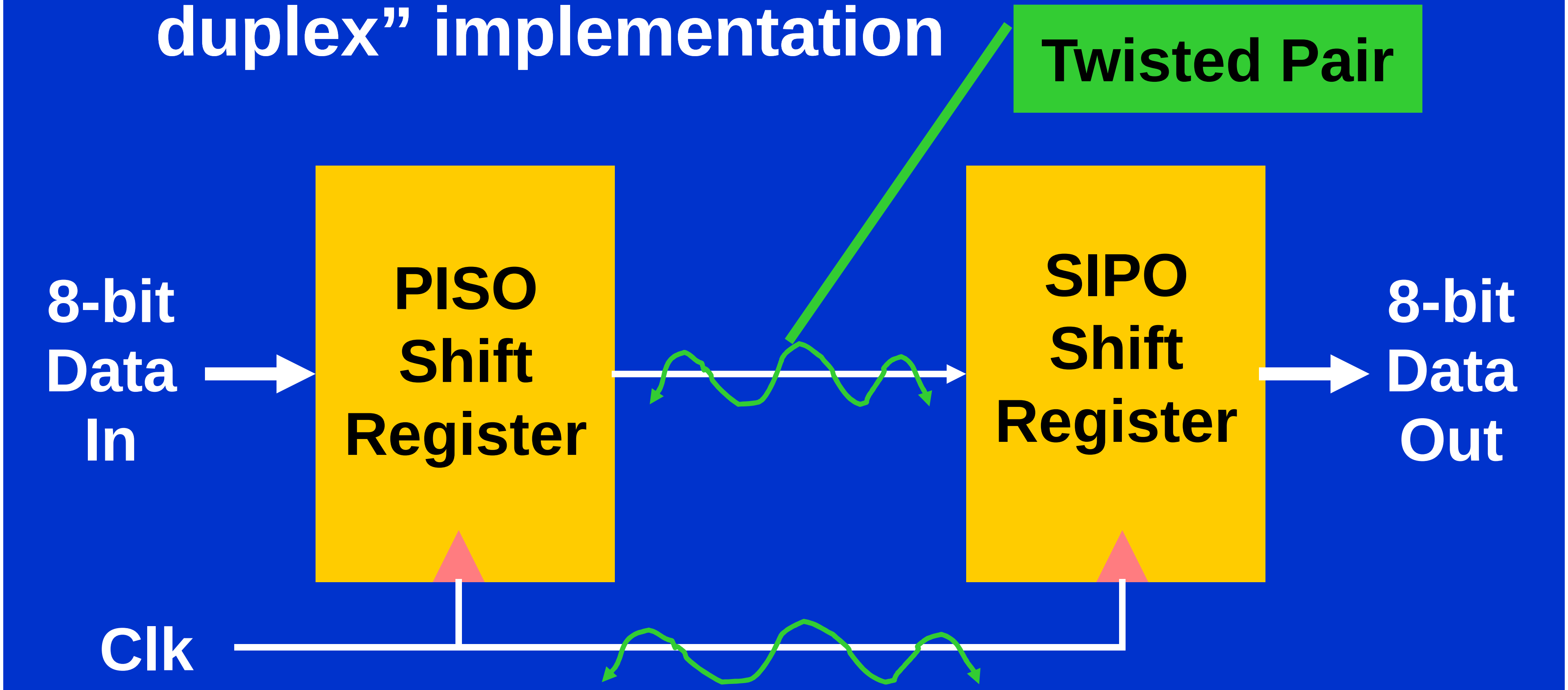
- Illustration of serial bit stream ("NRZ")



Data Transmitted = 0010 0110 = $26

Background and Motivation

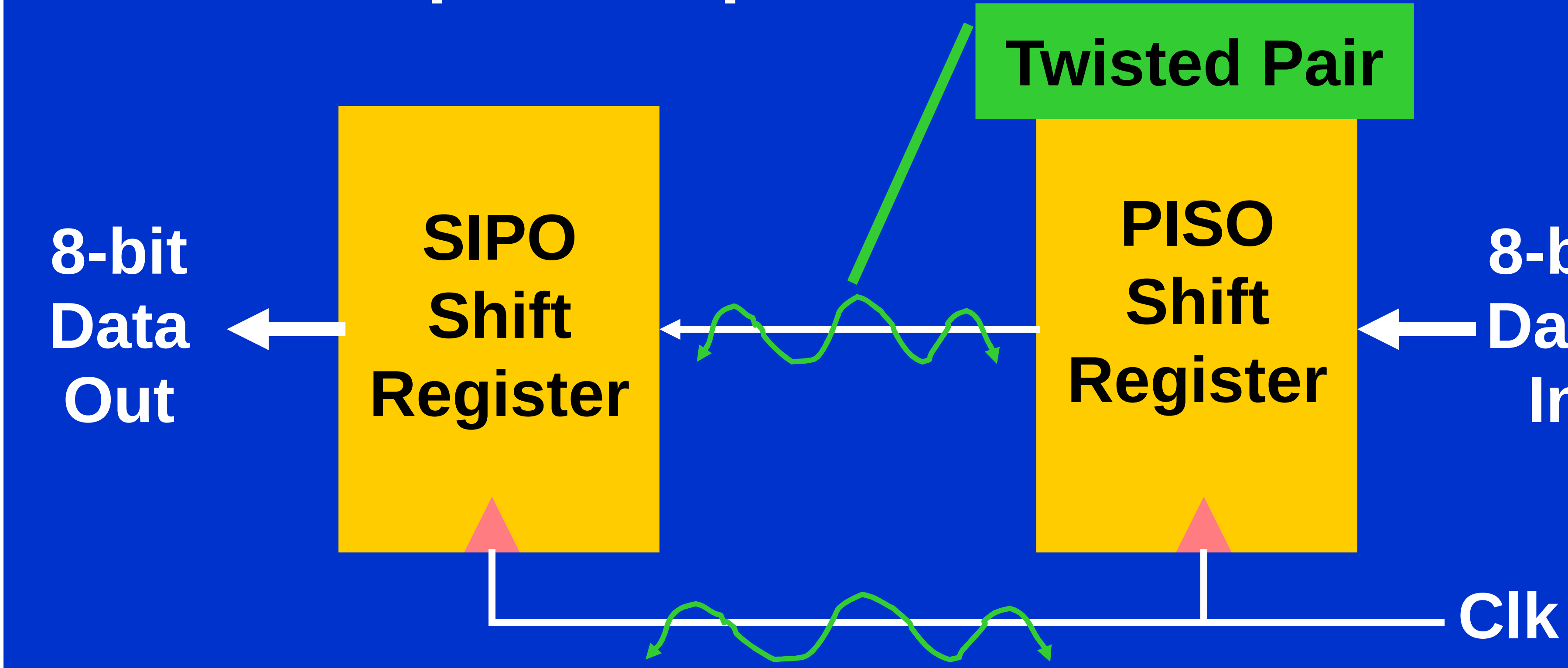
- Basic components required for “half duplex” implementation



<u>Question</u>: What components would be required for a “full duplex” implementation?

Background and Motivation

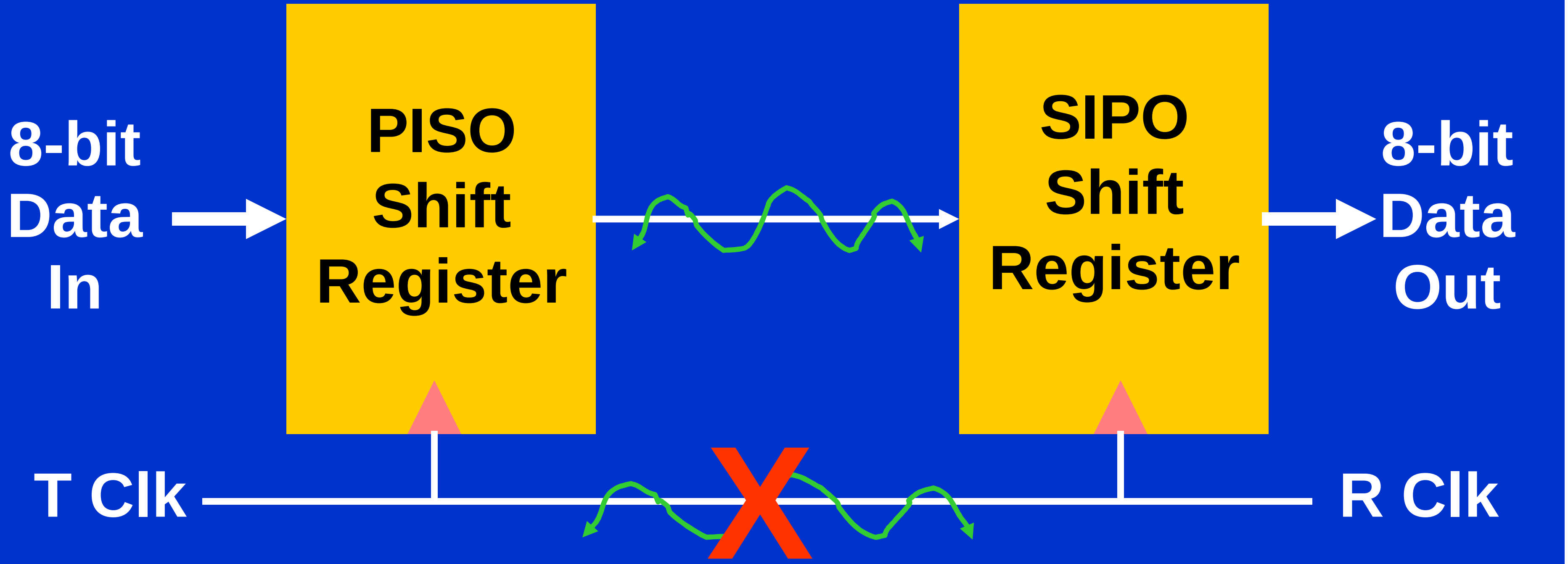
- **Additional components required for "full duplex" implementation**



<u>Answer</u>: ***<u>A duplicate, "mirror image" version of the circuit on the previous slide</u>***

Background and Motivation

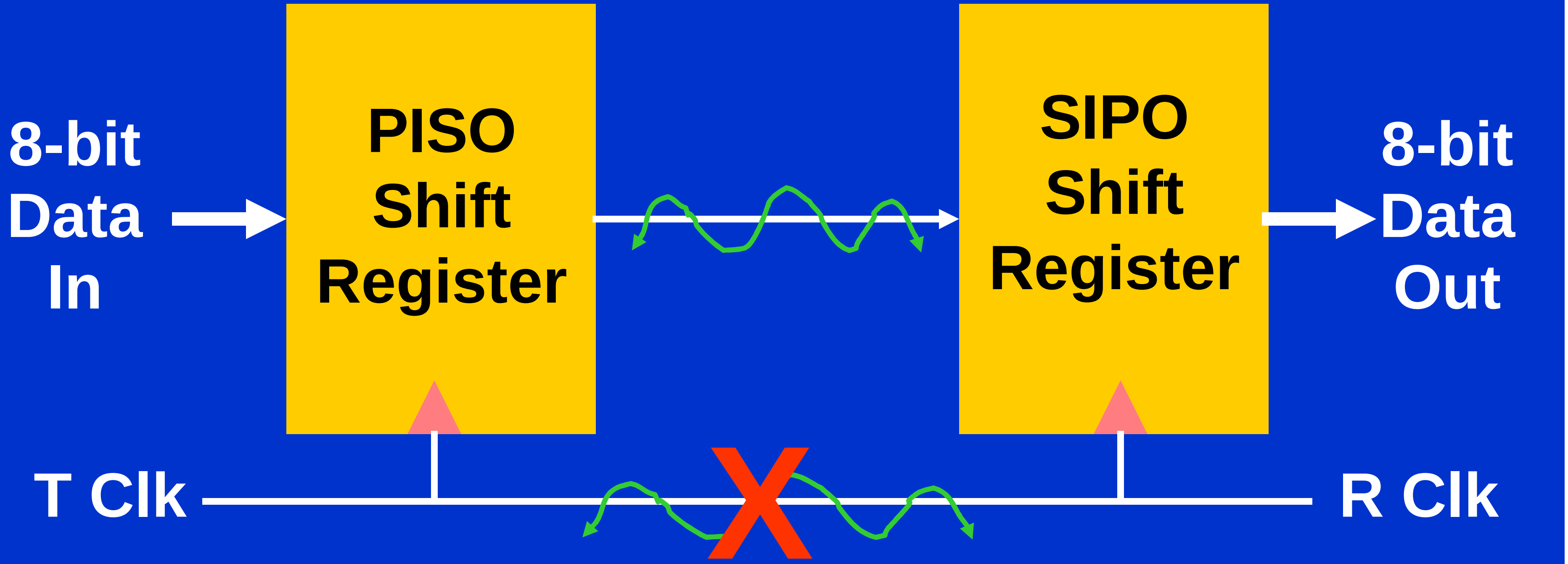
- **Note: Would like to avoid sending clock in order to eliminate a twisted pair**



Question: What difficulties are associated with generating the “transmit” and “receive” clocks *locally*?

Background and Motivation

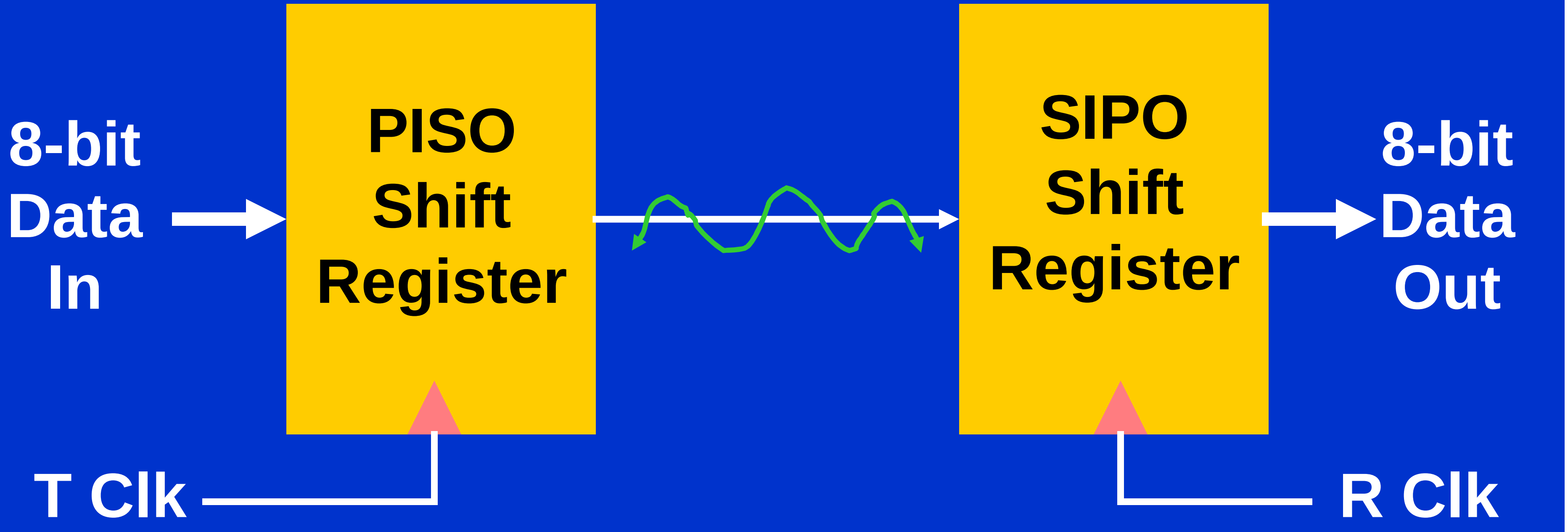
- **Note: Would like to avoid sending clock in order to eliminate a twisted pair**



Question: What difficulties are associated with generating the "transmit" and "receive" clocks *locally*? *Local clock synchronization*

Asynchronous Serial Transmission

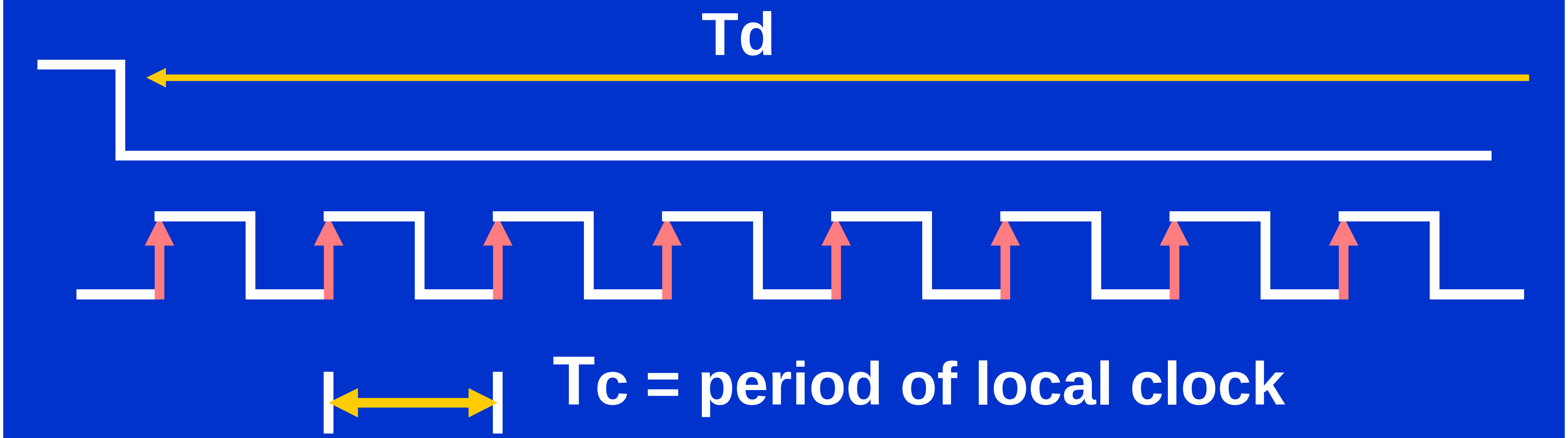
- **Solution**: **Sample serial data stream at a *higher rate than the data rate* (1/Td)**



Typically a local clock period (Tc) that is 1/16 of the data period (Td) is used – i.e., the sampling rate (1/Tc) is 16 times the data rate (1/Td)

Asynchronous Serial Transmission

- **Illustration of asynchronous serial data stream sampling**



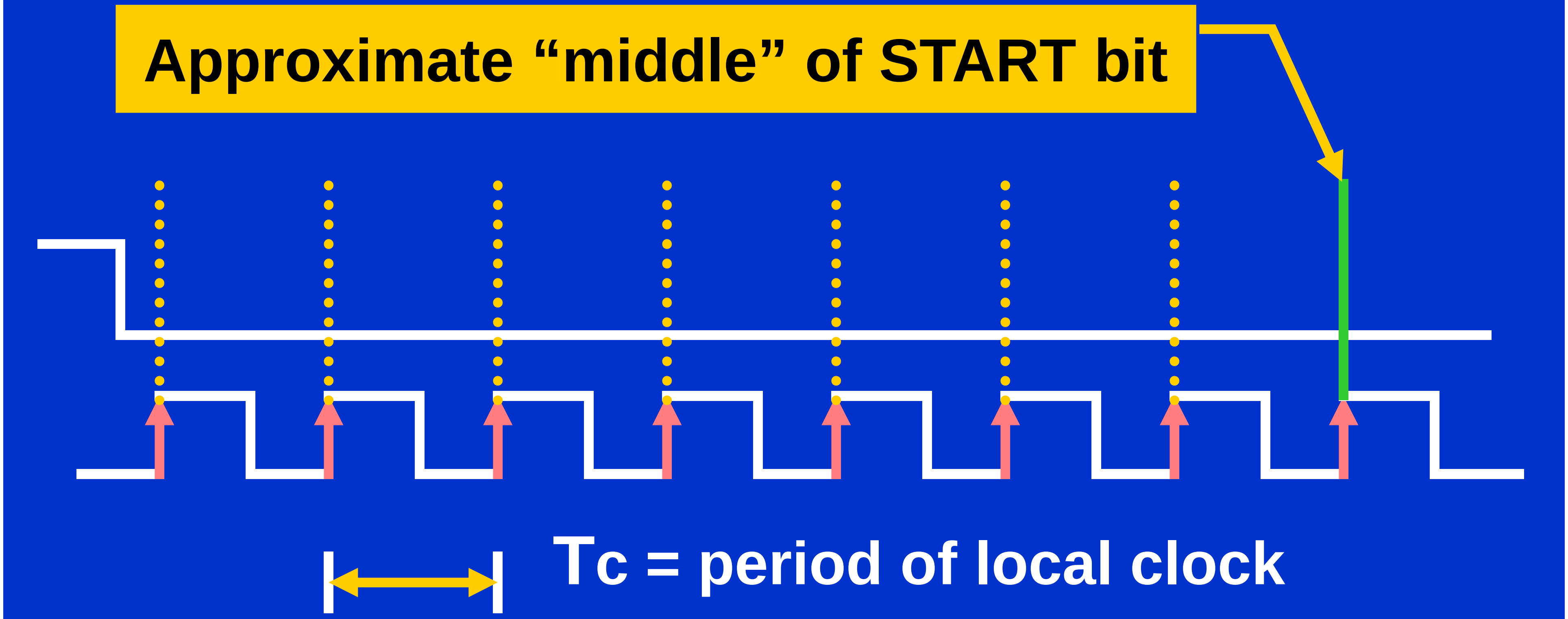
Tc = period of local clock

= 1/16 Td

<u>Note</u>: Receiving end needs to know when a given code transmitted (ASCII character) *<u>starts</u>* and *<u>stops</u>*

Asynchronous Serial Transmission

- **Solution: Append START bit (always "0") to the beginning of each character sent**



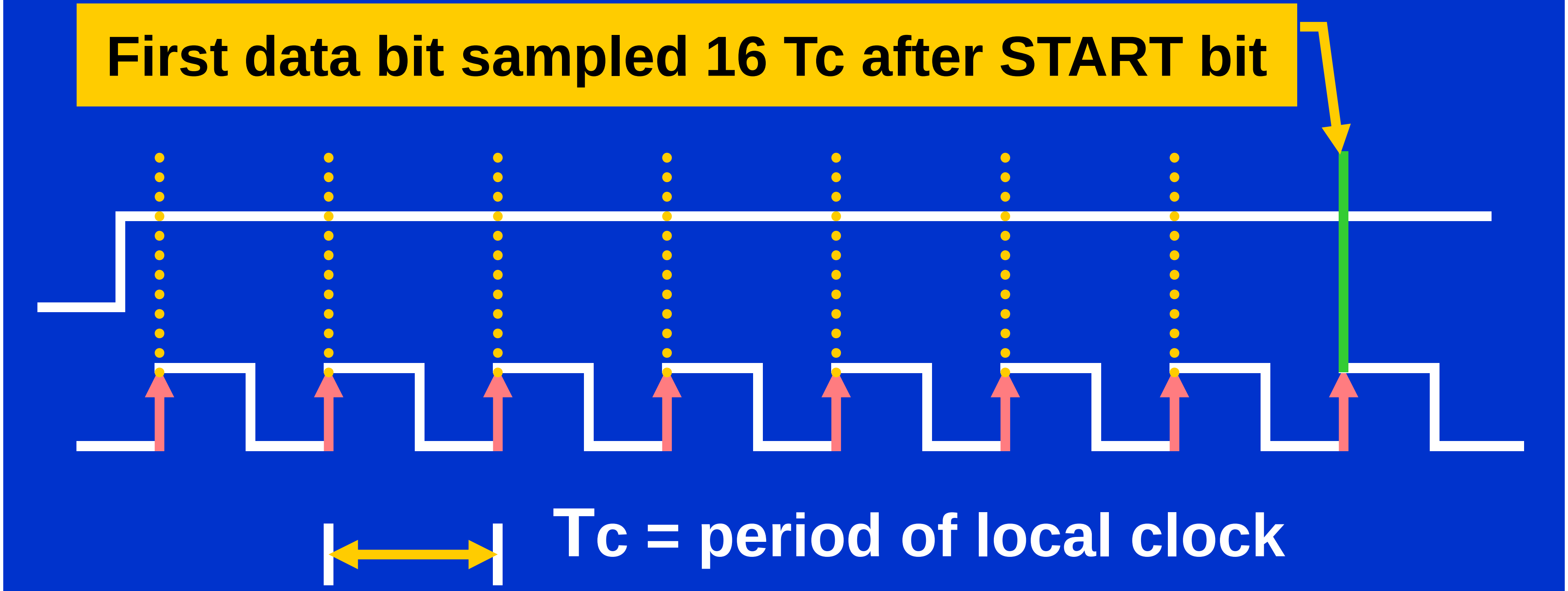
Approximate "middle" of START bit

Tc = period of local clock

= 1/16 Td

Asynchronous Serial Transmission

- **Solution: Sample successive data bits at intervals of 16 Tc referenced to START bit**



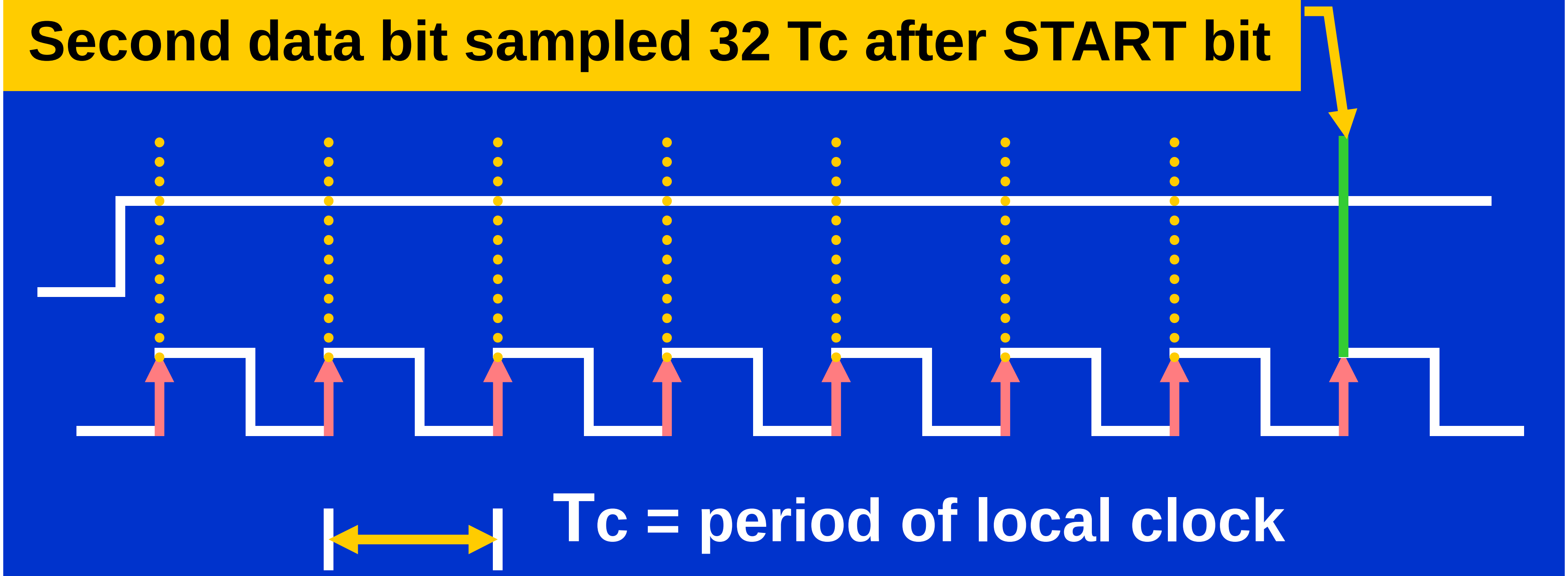
First data bit sampled 16 Tc after START bit

Tc = period of local clock

= 1/16 Td

Asynchronous Serial Transmission

- **<u>Solution</u>: Sample successive data bits at intervals of 16 Tc referenced to START bit**



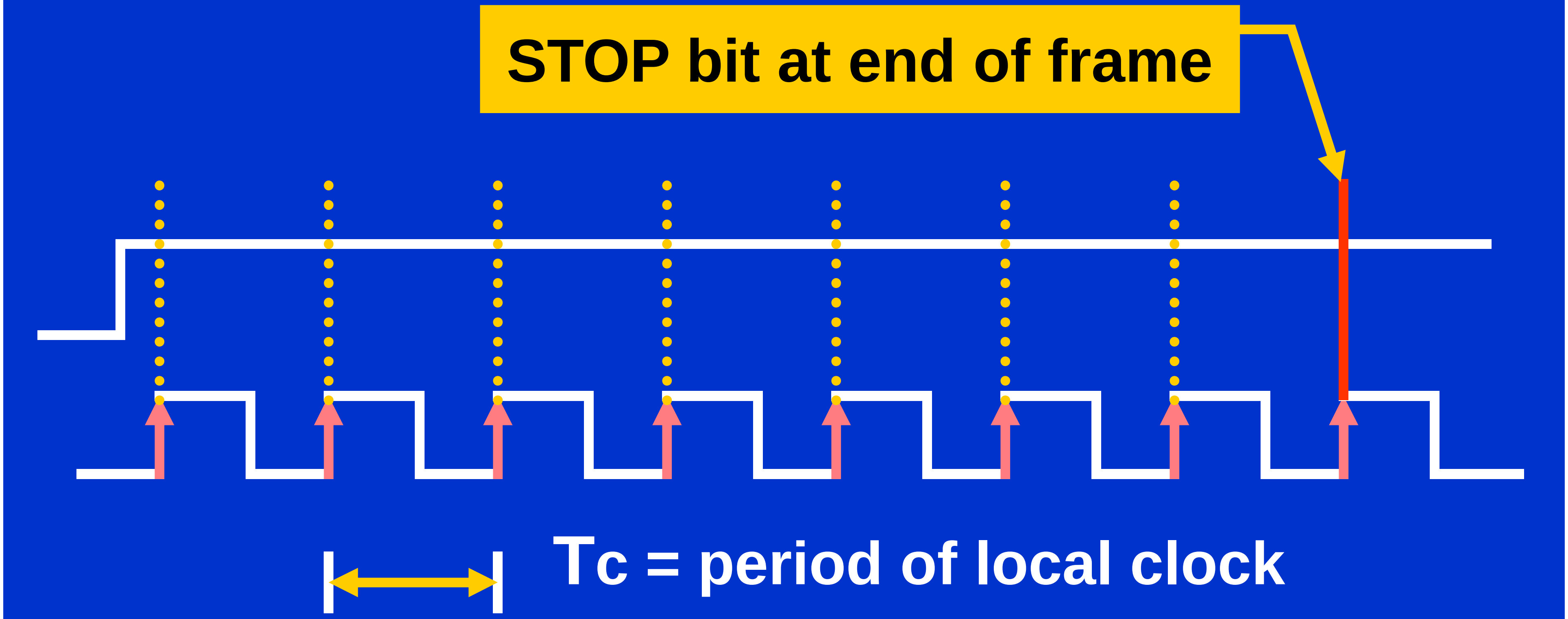
Second data bit sampled 32 Tc after START bit

Tc = period of local clock

= 1/16 Td

Asynchronous Serial Transmission

- **<u>Solution</u>: Append STOP bit (always "<u>1</u>") to the end of each character sent**

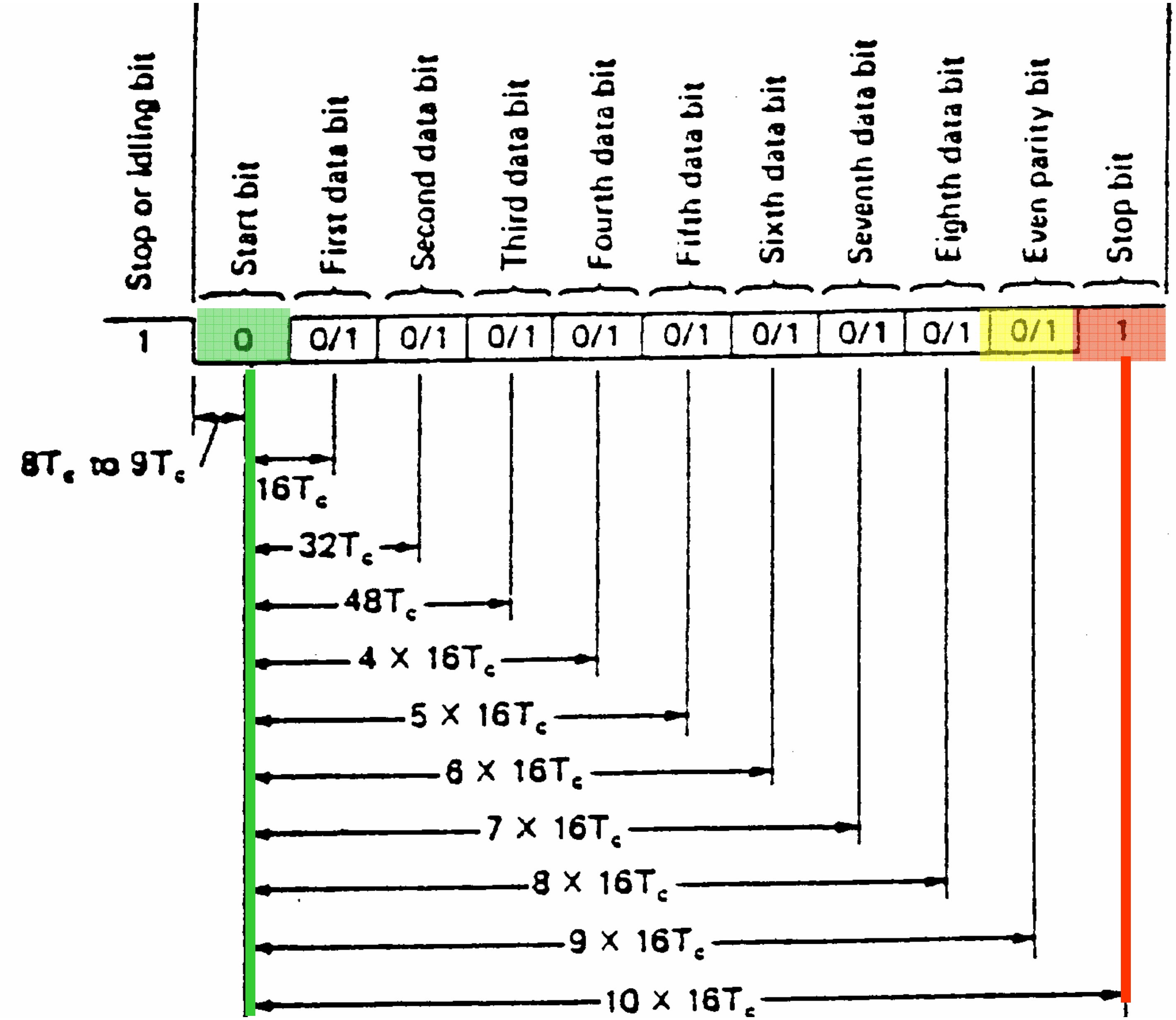


STOP bit at end of frame

Tc = period of local clock

= 1/16 Td

Character Format and Timing Reference



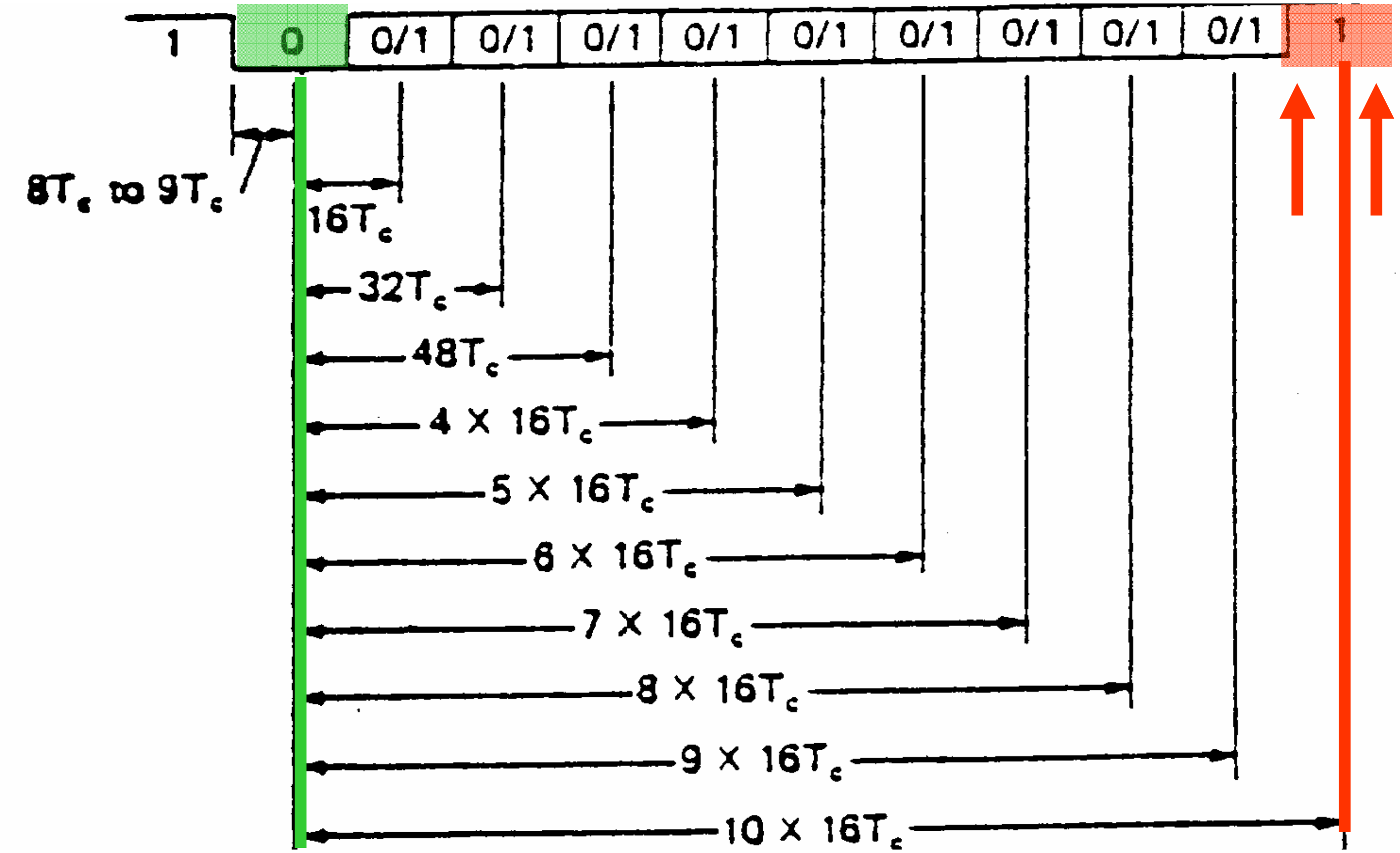
Asynchronous Serial Transmission

- **Definition: BAUD rate is the number of *bits per second* transmitted over the serial link**
- **Example: If 10 bit character frames are used, then if 960 characters/second are transmitted, the BAUD rate is 9600**
- **Question: What is the local clock frequency (assuming Td = 16 x Tc) required to support a 9600 BAUD data transmission rate?**

Answer: 9600 x 16 = 153,600 Hz

Asynchronous Serial Transmission

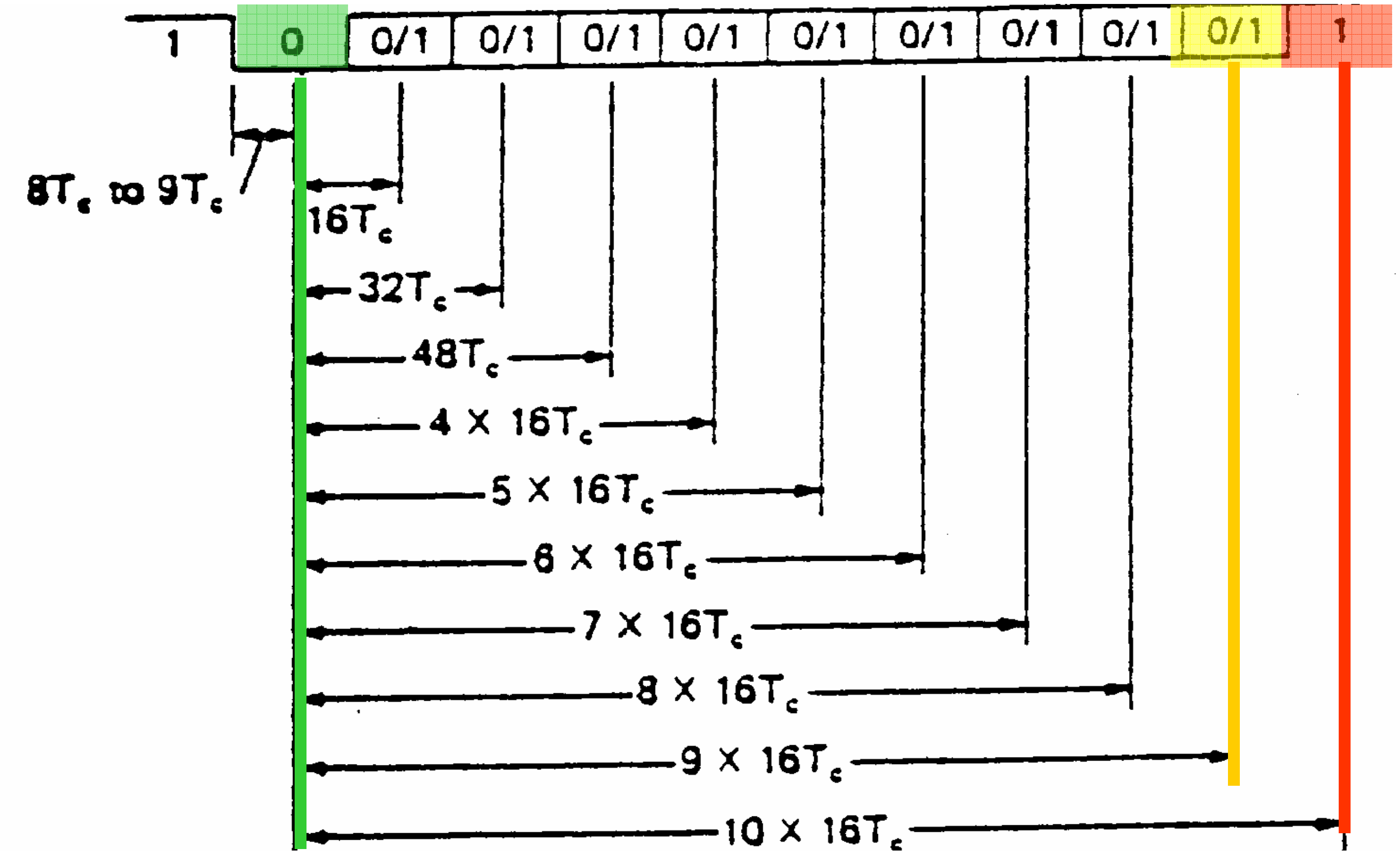
- **Question: What is the *tolerance* required between the transmitting clock and the receiving clock?**



± 1/2 bit at end of frame

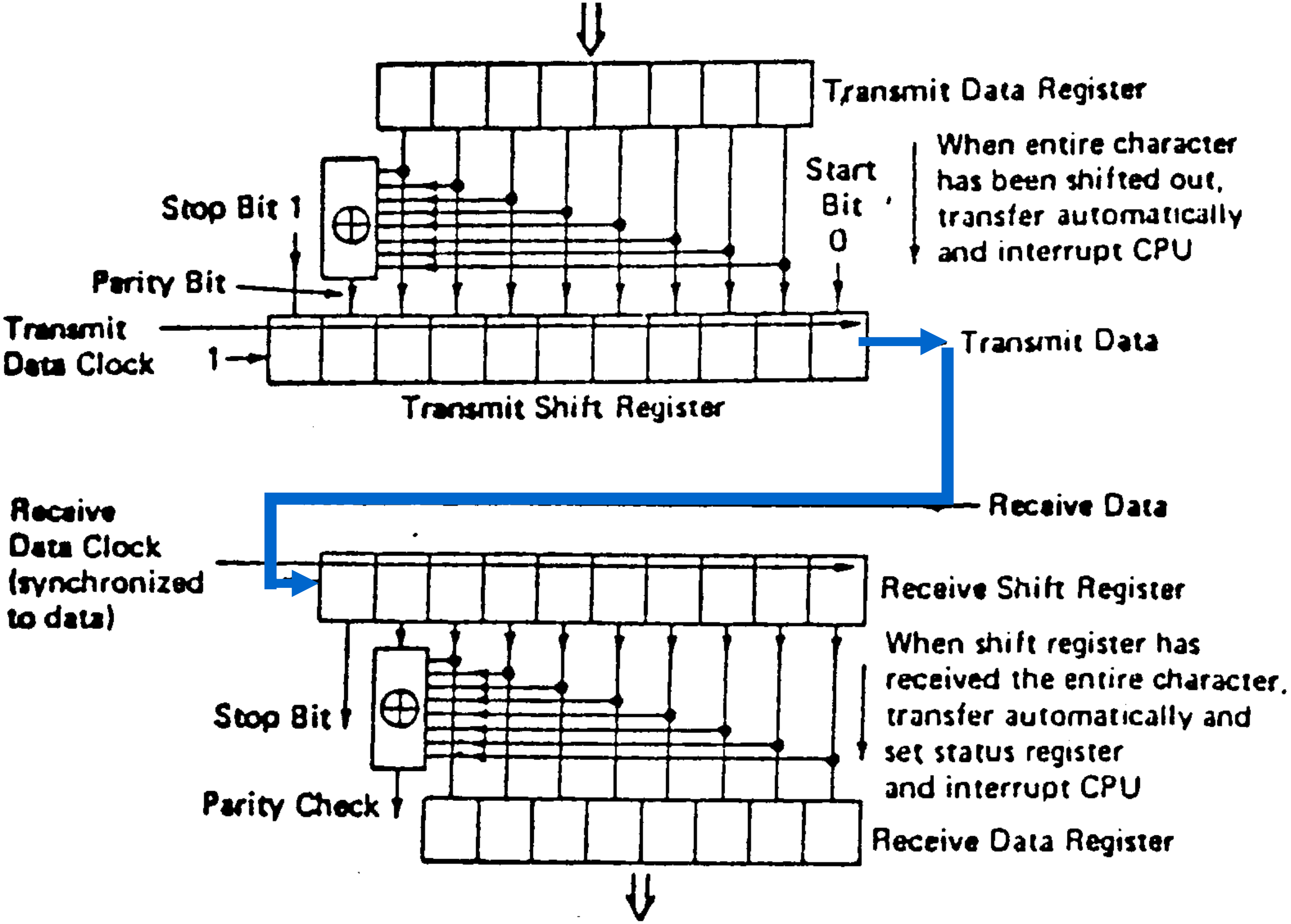
Asynchronous Serial Transmission

- **Question: How can the *integrity* of the data received be tested?**



Solution: Add a *parity bit* to the character frame

Parity Generation/Detection

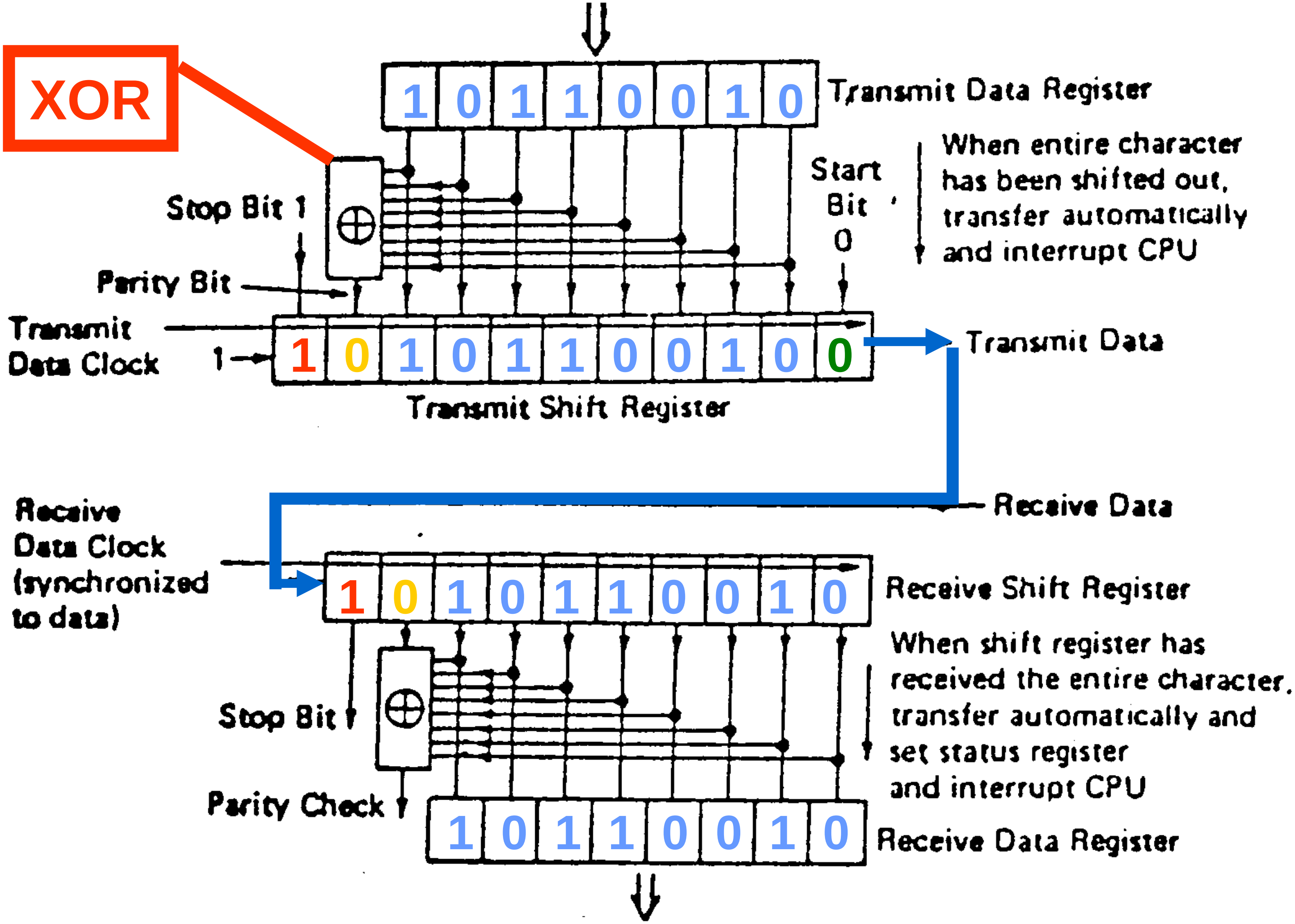


Transmit Data Register
Stop Bit 1
Start Bit 0
When entire character has been shifted out, transfer automatically and interrupt CPU
Parity Bit
Transmit Data Clock
1
Transmit Data
Transmit Shift Register
Receive Data Clock (synchronized to data)
Receive Data
Receive Shift Register
When shift register has received the entire character, transfer automatically and set status register and interrupt CPU
Stop Bit
Parity Check
Receive Data Register

Asynchronous Serial Transmission

- **Parity possibilities**
  - ***even parity*** **– here the parity bit is set so that there are an *even number* of “ones” transmitted in the data bits *plus* parity bit**
  - ***odd parity*** **– here the parity is set so that there are an *odd number* of “ones” transmitted in the data bits *plus* parity bit**
  - ***marking parity*** **– the parity bit is always the same value**

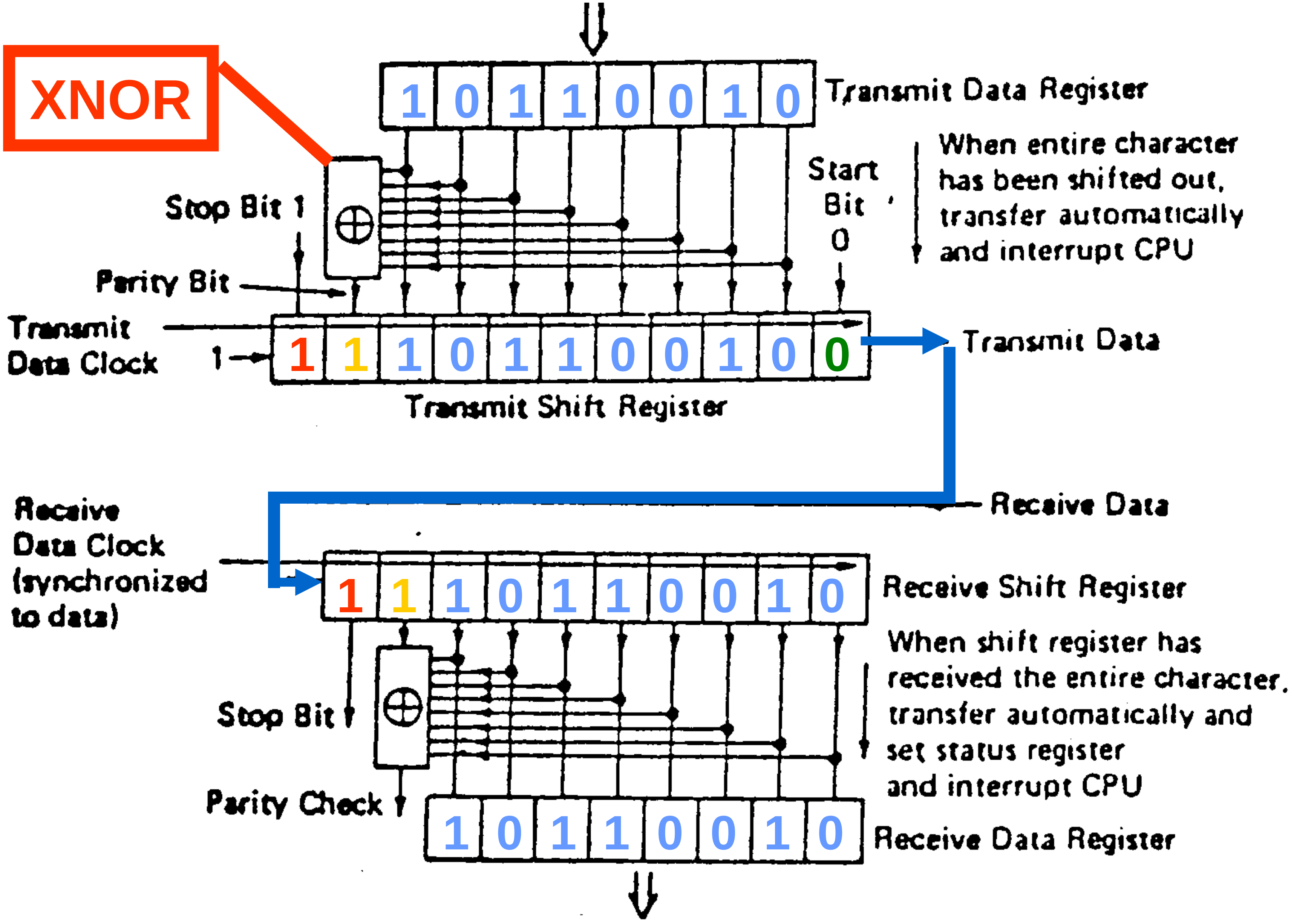
Even Parity



When entire character has been shifted out, transfer automatically and interrupt CPU

When shift register has received the entire character, transfer automatically and set status register and interrupt CPU

Odd Parity


XNOR
1 0 1 1 0 0 1 0
Transmit Data Register
Start Bit 0
When entire character has been shifted out, transfer automatically and interrupt CPU
Stop Bit 1
Parity Bit
Transmit Data Clock
1
1 1 1 0 1 1 0 0 1 0 0
Transmit Data
Transmit Shift Register
Receive Data
Receive Data Clock (synchronized to data)
1 1 1 0 1 1 0 0 1 0
Receive Shift Register
When shift register has received the entire character, transfer automatically and set status register and interrupt CPU
Stop Bit
Parity Check
1 0 1 1 0 0 1 0
Receive Data Register

Asynchronous Serial Transmission

- **Question: What kinds of *errors* can occur in asynchronous serial transmission?**
  - **don’t get STOP bit at end of frame – *framing error***
  - **receiver doesn’t read data from shift register before next character starts shifting in – *receiver overrun***
  - **noise on the transmission line corrupts one or more bits in the data stream – *parity error***

Asynchronous Serial Transmission

- Question: What are the ***most likely causes*** of serial data transmission errors, and ***how can these errors be corrected***?
  - framing error
    - cause: *local clocks too far out of tolerance*
    - correction: *bring clocks within tolerance*
  - overrun error
    - cause: *data not read "fast enough"*
    - correction: *lower baud rate or fix software*
  - parity error
    - cause: *problem with cable/connector*
    - correction: *repair/replace cable*

Clicker Quiz

1. The *baud rate* of a serial transmission link refers to:
   A. the number of bits per second transferred
   B. the number of bytes per second transferred
   C. the local clock frequency
   D. the data error rate
   E. none of the above

2. Local clock error does not accumulate on an asynchronous communication link because:

   A. the sampling clock is 16 times faster than the data rate
   B. the receiver re-synchronizes every time a start bit is detected
   C. the transmitter periodically sends “sync” pulses
   D. the receiver re-synchronizes every time a stop bit is detected
   E. none of the above

3. **The *length of time* required to transmit a 10-bit character frame at 19.2 Kbaud is approximately:**

   A. **1.042 milliseconds**

   B. **2.084 milliseconds**

   C. **5.208 milliseconds**

   D. **0.521 milliseconds**

   E. **none of the above**

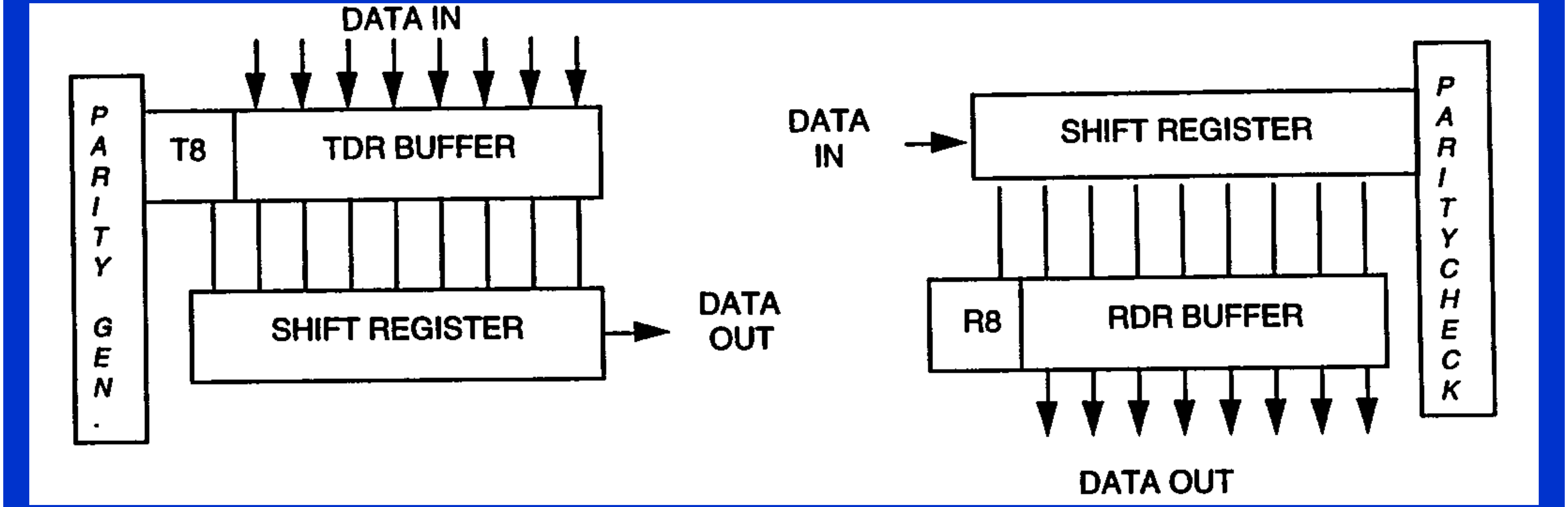
4. A possible cause of a *framing error* on a serial transmission link is:
   A. a bad cable or connector
   B. mismatch in baud rates
   C. error in the transmit side software
   D. error in the receive side software
   E. none of the above

SCI Features

- The 9S12C32 Serial Communications Interface (SCI) is a specific instance of a “UART” (universal asynchronous receiver/transmitter)
- Has software-selectable baud rates (derived from system clock)
- Uses advanced data sampling technique
- Allows special single-wire and loop modes
- Uses standard NRZ format
- Provides full duplex operation
- Has programmable word length
- Allows parity generation and checking
- May be program- or interrupt-driven

SCI Features

- Illustration of ***double buffering***

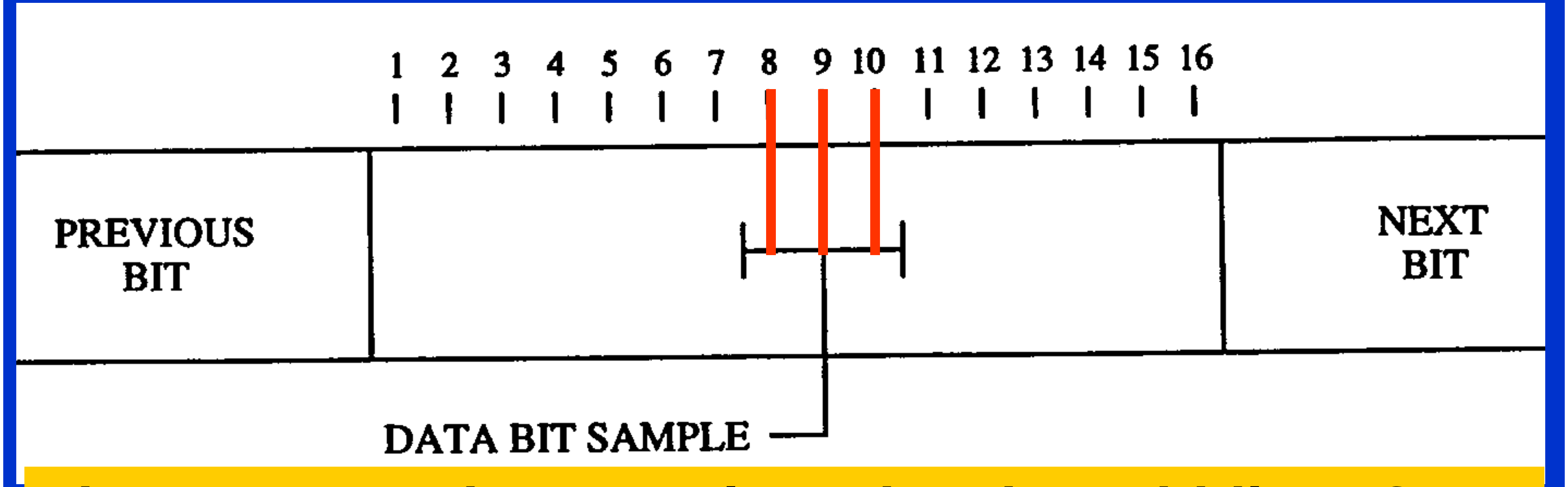


Transmitter

Receiver

SCI Features

- Illustration of ***data bit sampling***



Three successive samples taken in “middle” of each data bit; data bit is declared “valid” if two out of three samples agree (but noise flag “NF” is set to indicate that the “vote” was not “unanimous”)

SCI Features

- **Wired-OR Mode:** Used in single-wire systems with several SCI devices connected together (output pin is open-drain, requiring an external pull-up resistor)
- **Single-Wire Mode:** Transmit/receive are from a single pin (of Port S); TxD pin is open-drain and can therefore be used for sending or receiving data
- **Loop Mode:** Used for testing serial I/O software when the external device may not be available

SCI Registers

- **SCIDRL (SCI data) register**
  - **Read: receive data register (RDR)**
  - **Write: transmit data register (TDR)**
- **SCIDRH (SCI data high) register**
  - **R8 (bit 7) – 9th serial bit received**
  - **T8 (bit 6) – 9th serial bit transmitted**

Note: Our device drivers will use SCIDRL as the receive/transmit data register (RDR/TDR)

SCI Registers

- **SCICR1 (SCI Control) register**
  - **LOOPS (bit 7) – loop mode enable**
    - ***“0” – normal operation***
    - **“1” – loop mode enabled**
  - **WOMS (bit 6) – wired OR mode**
    - ***“0” – normal operation***
    - **“1” – output pins are open drain**

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCICR1 (SCI Control) register**
  - **RSRC (bit 5) – receiver source (for loop mode)**
    - ***“0” – receiver input connected internally to transmitter output***
    - **“1” – receiver input connected to TxD pin**
  - **M (bit 4) – mode (character format)**
    - ***“0” – one start, 8 data, one stop***
    - **“1” – one start, 9 data, one stop**

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCICR1 (SCI Control) register**
  - **ILT (bit 3) – idle line type**
    - ***"0" – short idle line mode enabled***
    - **"1" – long idle line mode enabled**
  - **WAKE (bit 2) – wakeup mode**
    - ***"0" – wake up by idle line recognition***
    - **"1" – wake up by address mark**

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCICR1 (SCI Control) register**
  - **PE (bit 1) – parity enable**
    - ***“0” – parity is disabled***
    - **“1” – parity is enabled**
  - **PT (bit 0) – parity type**
    - ***“0” – even parity is selected***
    - **“1” – odd parity is selected**

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCICR2 (SCI Control) register**
  - **TIE (bit 7) – transmit interrupt enable**
    - ***“0” – transmit interrupts disabled***
    - **“1” – transmit interrupts enabled**
  - **TCIE (bit 6) – transmit complete interrupt enable**
    - ***“0” – transmit complete interrupt disabled***
    - **“1” – transmit compete interrupt enabled**

🟩 ***indicates default mode after RESET***

SCI Registers

- SCICR2 (SCI Control) register
  - RIE (bit 5) – receive interrupt enable
    - *“0” – receive interrupts disabled*
    - “1” – receive interrupts enabled
  - ILE (bit 4) – idle interrupt enable
    - *“0” – idle interrupt disabled*
    - “1” – idle interrupt enabled

🟩 *indicates default mode after RESET*

SCI Registers

- **SCICR2 (SCI Control) register**
  - **TE (bit 3) – transmitter enable**
    - ***“0” – transmitter disabled***
    - **“1” – transmitter enabled**
  - **RE (bit 2) – receiver enable**
    - ***“0” – receiver disabled***
    - **“1” – receiver enabled**

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCICR2 (SCI Control) register**
  - **RWU (bit 1) – receiver wakeup control**
    - ***“0” – exit receiver wake-up***
    - **“1” – enter receiver wake-up**
  - **SBK (bit 0) – break control**
    - ***“0” – terminate break***
    - **“1” – send break**

A break character contains all logic 0s and has no start, stop, or parity bit

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCISR1 (SCI Status) register**
  - **TDRE (bit 7) – transmit data register empty flag**
    - **“0” – not ready for new data**
    - ***“1” – ready for new data***

Note: New data will *not be transmitted* unless SCISR1 is *read* before *writing* to the transmit data register (TDR)

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCISR1 (SCI Status) register**
  - **TC (bit 6) – transmit complete flag**
    - **“0” – transmitter busy**
    - ***“1” – transmitter idle***

Note: The TC bit is *cleared* by reading SCISR1 followed by writing to the transmit data register (TDR)

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCISR1 (SCI Status) register**
  - **RDRF (bit 5) – receive data register full**
    - ***"0" – no new data available***
    - **"1" – new receive data available**

Note: RDRF is *cleared* by reading SCISR1 followed by reading RDR

  - **IDLE (bit 4) – idle line detected flag**
    - ***"0" – receive data line idle***
    - **"1" – receive data line active**

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCISR1 (SCI Status) register**
  - **OR (bit 3) – receiver overrun error flag**
    - ***“0” – no overrun detected***
    - **“1” – overrun error detected**
  - **NF (bit 2) – noise flag**
    - ***“0” – no noise detected in data bit sample***
    - **“1” – noise detected in data bit sample**

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCISR1 (SCI Status) register**
  - **FE (bit 1) – framing error flag**
    - ***“0” – no framing error detected***
    - **“1” – framing error detected**
  - **PF (bit 0) – parity error flag**
    - ***“0” – parity correct (or disabled)***
    - **“1” – parity error detected**

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCISR2 (SCI Status) register**
  - **RAF (bit 0) – receiver active flag**
    - **“0” – *no character is being received***
    - **“1” – character is being received**

🟩 ***indicates default mode after RESET***

SCI Registers

- Baud rate control
  - SCIBDH (divisor high nibble)
  - SCIBDL (divisor low byte)

BR DIV = CLK / (16 x baud rate)

here, CLK = 24 MHz

Example: For operation at 9600 baud, a *baud rate divisor* of 156 (base 10) should be loaded into SCIBDL (leave SCIBDH in reset state)

Clicker Quiz

1. A possible cause of the noise flag (NF) being set in the SCI is:
   A. a bad cable or connector
   B. mismatch in baud rates
   C. error in the transmit side software
   D. error in the receive side software
   E. none of the above

2. A possible cause of the receiver overrun flag (OR) being set in the SCI is:

   A. a bad cable or connector

   B. mismatch in baud rates

   C. error in the transmit side software

   D. error in the receive side software

   E. none of the above

3. If the receiver overrun flag is *not* being set, a possible cause of "dropped characters" is:

   A. a bad quarterback and/or wide receiver

   B. mismatch in baud rates

   C. error in the transmit side software

   D. error in the receive side software

   E. none of the above

4. The SCI’s TDRE flag is cleared by:

   A. writing a character to SCIDR
   B. reading SCISR1 and then writing a character to SCIDR
   C. reading SCISR1
   D. writing a “1” to the TDRE bit of SCISR1
   E. none of the above

5. The SCI’s RDRF flag is cleared by:

A. reading a character from SCIDR

B. reading SCISR1 and then reading the character from SCIDR

C. reading SCISR1

D. writing a “1” to the RDRF bit of SCISR1

E. none of the above

6. The SCI *baud rate divisor* that should be used in a 24 MHz system to transmit 11-bit character frames at 9600 baud is:

A. 26

B. 52

C. 104

D. 156

E. none of the above

7. The *number* of 11-bit character frames that can be transmitted each second at 9600 baud is approximately:

A. 873

B. 960

C. 8730

D. 9600

E. none of the above

8. For a 24 MHz system, the *local clock error* associated with transmitting 10-bit character frames at 9600 baud using the SCI is approximately:

A. 5.0%

B. 1.6%

C. 0.5%

D. 0.16%

E. none of the above

SCI Program-Driven Operation

- **Declarations**

```
rxdrf   equ   $20   ; RDRF mask
txdre   equ   $80   ; TDRE mask
tron    equ   $0C   ; transmitter and
                    ; receiver enable mask
```

SCI Program-Driven Initialization

- **sci_pini**

```
; Initializes SCI for program-driven operation


; STEP (1)  Set transmission rate to 9600 baud
sci_pini    movb    #156t,scibdl

; STEP (2)  Select character format (M bit) and parity
   (bits 1 & 0) via SCI Control Register #1
           clr      scicr1

; STEP (3)  Enable transmitter and receiver via SCI
   Control Register #2 (with interrupts off)
            movb    #tron,scicr2
            rts
```

SCI Program-Driven Initialization

- **sci_pini**

```
; Initializes SCI for program-driven operation


; STEP (1)  Set transmission rate to 9600 baud
sci_pini   movb   #156t,scibdl

; STEP (2)  Select character format (M bit) and parity
   (bits 1 & 0) via SCI Control Register #1
          clr     scicr1

; STEP (3)  Enable transmitter and receiver via SCI
   Control Register #2 (with interrupts off)
           movb   #tron,scicr2
           rts
```

SCI Registers

- Baud rate control
  - SCIBDH (divisor high nibble)
  - SCIBDL (divisor low byte)

BR DIV = CLK / (16 x baud rate)
here, CLK = 24 MHz

Example: For operation at 9600 baud, a *baud rate divisor* of 156 (base 10) should be loaded into SC0BDL (leave SC0BDH in reset state)

SCI Program-Driven Initialization

- **sci_pini**

```
; Initializes SCI for program-driven operation

; STEP (1)  Set transmission rate to 9600 baud
sci_pini    movb    #156t,scibdl

; STEP (2)  Select character format (M bit) and parity
   (bits 1 & 0) via SCI Control Register #1
            clr      scicr1

; STEP (3)  Enable transmitter and receiver via SCI
   Control Register #2 (with interrupts off)
            movb    #tron,scicr2
            rts
```

SCI Registers

- **SCICR1 (SCI Control) register**
  - **RSRC (bit 5) – receiver source (for loop mode)**
    - ***“0” – receiver input connected internally to transmitter output***
    - **“1” – receiver input connected to TxD pin**
  - **M (bit 4) – mode (character format)**
    - ***“0” – one start, 8 data, one stop***
    - **“1” – one start, 9 data, one stop**

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCICR1 (SCI Control) register**
  - **PE (bit 1) – parity enable**
    - ***“0” – parity is disabled***
    - **“1” – parity is enabled**
  - **PT (bit 0) – parity type**
    - ***“0” – even parity is selected***
    - **“1” – odd parity is selected**

🟩 ***indicates default mode after RESET***

SCI Program-Driven Initialization

- **sci_pini**

```
; Initializes SCI for program-driven operation

; STEP (1)  Set transmission rate to 9600 baud
sci_pini    movb    #156t,scibdl

; STEP (2)  Select character format (M bit) and parity
   (bits 1 & 0) via SCI Control Register #1
           clr      scicr1

; STEP (3)  Enable transmitter and receiver via SCI
   Control Register #2 (with interrupts off)
            movb    #tron,scicr2
            rts
```

SCI Registers

- **SCICR2 (SCI Control) register**
  - **TIE (bit 7) – transmit interrupt enable**
    - ★ ***"0" – transmit interrupts disabled***
    - **"1" – transmit interrupts enabled**
  - **TCIE (bit 6) – transmit complete interrupt enable**
    - ★ ***"0" – transmit complete interrupt disabled***
    - **"1" – transmit compete interrupt enabled**

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCICR2 (SCI Control) register**
  - **RIE (bit 5) – receive interrupt enable**
    - ★ ***“0” – receive interrupts disabled***
    - **“1” – receive interrupts enabled**
  - **ILE (bit 4) – idle interrupt enable**
    - ★ ***“0” – idle interrupt disabled***
    - **“1” – idle interrupt enabled**

🟩 ***indicates default mode after RESET***

SCI Registers

- **SCICR2 (SCI Control) register**
  - **TE (bit 3) – transmitter enable**
    - ***“0” – transmitter disabled***
    - ★ **“1” – transmitter enabled**
  - **RE (bit 2) – receiver enable**
    - ***“0” – receiver disabled***
    - ★ **“1” – receiver enabled**

🟩 ***indicates default mode after RESET***

SCI Program-Driven Device Drivers

- **inchar**

```
; Inputs ASCII character from SCI port and
;    returns it in the A register
; No condition code bits or registers affected

rxdrf   equu   $20    ; RDRF mask

; STEP (1)  Check RDRF flag (wait until set)
inchar  brclr scisr1,rxdrf,inchar

; STEP (2)  Load A with character in RDR
        ldaa  scidrl
        rts
```

SCI Program-Driven Device Drivers

- **inchar**

```
; Inputs ASCII character from SCI port and
;    returns it in the A register
; No condition code bits or registers affected

rxdrf    equu   $20    ; RDRF mask

; STEP (1)  Check RDRF flag (wait until set)
inchar   brclr scisr1,rxdrf,inchar

; STEP (2)  Load A with character in RDR
         ldaa  scidrl
         rts
```

SCI Registers

- **SCISR1 (SCI Status) register**
  - **RDRF (bit 5) – receive data register full**
    - ***“0” – no new data available***
    - **“1” – new receive data available**

Note: RDRF is *cleared* by reading SCISR1 followed by reading RDR

  - **IDLE (bit 4) – idle line detected flag**
    - ***“0” – receive data line idle***
    - **“1” – receive data line active**

🟩 ***indicates default mode after RESET***

SCI Program-Driven Device Drivers

- **outchar**

```
; Outputs character passed in A to the SCI port
; No condition code bits or registers affected

txdre   equ   $80   ; TDRE mask


; STEP (1)  Check TDRE flag (wait until set)
outchar brclr scisr1,txdre,outchar

; STEP (2)  Load TDR with character in A register
        staa  scidrl
        rts
```

SCI Program-Driven Device Drivers

- **outchar**

```
; Outputs character passed in A to the SCI port
; No condition code bits or registers affected

txdre    equu   $80    ; TDRE mask

; STEP (1)  Check TDRE flag (wait until set)
outchar brclr scisr1,txdre,outchar

; STEP (2)  Load TDR with character in A register
        staa  scidrl
        rts
```

SCI Registers

- **SCISR1 (SCI Status) register**
  - **TDRE (bit 7) – transmit data register empty flag**
    - **“0” – not ready for new data**
    - **“1” – *ready for new data***

Note: New data will *not be transmitted* unless SCISR1 is *read* before *writing* to the transmit data register (TDR)

🟩 ***indicates default mode after RESET***

SCI Program-Driven Device Drivers

- **outchar**

; Outputs character passed in A to the SCI port
; No condition code bits or registers affected

```
txdre   equ   $80   ; TDRE mask
```

; STEP (1) ***Check TDRE flag (wait until set)***

```
outchar brclr scisr1,txdre,outchar
```

; STEP (2) ***Load TDR with character in A register***

```
        staa  scidrl
        rts
```

Motivation for Interrupt-Driven Mode

- **What does an SCI *receive* interrupt *mean*?**

"The next character is available to read...and please do so before the next one comes in!"

Note: SCI receive interrupts should be *continuously* enabled

Motivation for Interrupt-Driven Mode

- Why ***buffer*** data ***received*** by the SCI?
  - an ***entire string*** of characters may be needed to form a “command” (i.e., for the microcontroller to carry out some function)
  - a “burst” of characters (representing a string of commands or an array of data) may come in ***faster*** than the microcontroller can interpret them (or execute “in-line”)

Motivation for Interrupt-Driven Mode

- What does an SCI ***transmit*** interrupt ***mean***?

“*Please send me the next character!*”

Problem: The application program *may* not always have a character ready to send, resulting in an “*unrelenting*” interrupt request

Solution: Enable SCI transmit interrupts *only* when characters are available to send

Note: This means SCI transmit interrupts should *initially* be *disabled*

Motivation for Interrupt-Driven Mode

- **Why *buffer* data *transmitted* by the SCI?**
  - **using a buffer effectively *decouples* the CPU from the *speed limitations* imposed by the SCI character transmission rate**
  - **note that at 9600 baud, it takes about *one millisecond* to transmit *each character***

SCI Register Review

- **SCIDRL (SCI data) register**
  - **Read: receive data register (RDR)**
  - **Write: transmit data register (TDR)**

Note: Our device drivers will use SCIDRL as the receive/transmit data register (RDR/TDR)

SCI Register Review

- SCICR1 (SCI Control) register
  - PE (bit 1) – parity enable
    - *“0” – parity is disabled*
    - “1” – parity is enabled
  - PT (bit 0) – parity type
    - *“0” – even parity is selected*
    - “1” – odd parity is selected

🟩 *indicates default mode after RESET*

SCI Register Review

- **SCICR2 (SCI Control) register**
  - **TIE (bit 7) – transmit interrupt enable**
    - ***“0” – transmit interrupts disabled***
    - **“1” – transmit interrupts enabled**
  - **RIE (bit 5) – receive interrupt enable**
    - ***“0” – receive interrupts disabled***
    - **“1” – receive interrupts enabled**

🟩 ***indicates default mode after RESET***

SCI Register Review

- **SCICR2 (SCI Control) register**
  - **TE (bit 3) – transmitter enable**
    - ***"0" – transmitter disabled***
    - **"1" – transmitter enabled**
  - **RE (bit 2) – receiver enable**
    - ***"0" – receiver disabled***
    - **"1" – receiver enabled**

🟩 ***indicates default mode after RESET***

SCI Register Review

- **SCISR1 (SCI Status) register**
  - **TDRE (bit 7) – transmit data register empty flag**
    - **“0” – not ready for new data**
    - **“1” – *ready for new data***

Note: New data will *not* be transmitted unless SCISR1 is read *before* writing to the transmit data register (TDR)

🟩 ***indicates default mode after RESET***

SCI Register Review

- **SCISR1 (SCI Status) register**
  - **RDRF (bit 5) – receive data register full**
    - **“0” – *no new data available***
    - **“1” – new receive data available**

Note: RDRF is *cleared* by a read of the Receive Data Register (RDR)

🟩 ***indicates default mode after RESET***

SCI Register Review

- Baud rate control
  - SCIBDH (divisor high nibble)
  - SCIBDL (divisor low byte)

BR DIV = CLK / (16 x baud rate)

here, CLK = 24 MHz

Example: For operation at 9600 baud, a baud rate divisor of 156 (base 10) should be loaded into SC0BDL (leave SC0BDH in reset state)

SCI Interrupt Mode Operation

- **Register declarations**

```
rxdrf   equ  $20   ; RDRF mask
txdre   equ  $80   ; TDRE mask
tron    equ  $0C   ; transmit/receive enable
rimask  equ  $20   ; receive  interrupt mask
timask  equ  $80   ; transmit interrupt mask
```

SCI Interrupt-Mode Initialization

- **Buffer and pointer declarations**

```
rsize  equu 40    ; receive buffer size
rbuf   rmb  rsize
rin    fcb  0     ; receive buffer IN pointer
rout   fcb  0     ; receive buffer OUT pointer

tsize  equ  20    ; transmit buffer size
tbuf   rmb  tsize
tin    fcb  0     ; transmit buffer IN pointer
tout   fcb  0     ; transmit buffer OUT pointer
```

Useful Macro

Macro for incrementing B register mod BUFSIZE

<u>Example</u>: incBmod 20 ; increments B modulo 20

```
incBmod    MACRO
           incb
           cmpb   #\1
           bne    $+3
           clrb
          ENDM
```

SCI Interrupt-Mode Initialization

; Initializes SCI for interrupt-driven operation

; STEP (1) *Set transmission rate to 9600 baud*

```
scbini movb  #156,scibdl
```

; STEP (2) *Select character format (M bit) and*
; parity enable (and type, if parity enabled)

```
       clr   scicr1
```

; STEP (3) *Enable transmitter and receiver,*
; initially with receive interrupts enabled and
; transmit interrupts disabled

```
       movb  #tron,scicr2  ;enable trans/recv
       bset  scicr2,rimask ;enable recv intrs
       cli                 ;enable IRQ intrs
       rts
```

Buffered Character Input "BCI" Routine

- Application program interface that returns the next character available from the receive buffer RBUF in the A register

  1. Check status of receive buffer RBUF
  2. If EMPTY, wait for character
  3. Else, access character from RBUF[ROUT] and return it in (A)
  4. ROUT = (ROUT+1) mod RSIZE
  5. Exit

Buffered Character Input “BCI” Routine

; Returns next character from RBUF in A register

; STEP (1) *Check RBUF status*

```
bci     ldab  rout
```

; STEP (2) *If EMPTY, wait for character*

```
rwait  cmpb  rin      ; perform EMPTY check
       beq   rwait    ; wait if RBUF empty
```

Buffered Character Input “BCI” Routine

```
; STEP (3) Access next character from RBUF
        ldx      #rbuf
        ldaa     b,x      ;(A) = RBUF[ROUT]

; STEP (4) Increment ROUT modulo RSIZE
        incBmod  rsize
        stab     rout

; STEP (5) Exit
        rts
```

Buffered Character Output “BCO” Routine

- Application program interface that places character passed in A register into transmit buffer TBUF

  1. Check status of transmit buffer TBUF
  2. If FULL, wait for space
  3. Else, store character passed in the A register at TBUF[TIN]
  4. TIN = (TIN + 1) mod TSIZE
  5. Enable SCI transmit interrupts
  6. Exit

Buffered Character Output “BCO” Routine

; Places character passed in A into TBUF

; STEP (1) Check TBUF status

```
bco     ldab    tin
        incBmod tsize  ;(B) = (TIN+1) mod TSIZE
        pshb           ;save on stack
```

; STEP (2) If FULL, wait for space

```
twait   cmpb    tout   ;perform FULL check
        beq     twait  ;wait if TBUF full
```

Buffered Character Output “BCO” Routine

```
; STEP (3) Place character in TBUF
       ldx    #tbuf
       ldab   tin
       staa   b,x        ; TBUF[TIN] = (A)

; STEP (4) Increment TIN mod TSIZE
       pulb
       stab   tin

; STEP (5) Enable SCI transmit interrupts
       bset   scicr2,timask

; STEP (6) Exit
       rts
```

SCI Interrupt Service Routine

- Receive section

  R1. Check RDRF flag

  R2. If RDRF=0, go to transmit section;
  else, continue with receive section

  R3. Check status of receive buffer RBUF

  R4. If FULL, error exit (via “SWI”)

  R5. Else, input character from SCI RDR

  R6. Store character at RBUF[RIN]

  R7. RIN = (RIN + 1) mod RSIZE

SCI Interrupt Service Routine

- **Transmit section**

T1. Check TDRE flag

T2. If TDRE=0, exit; else, continue

T3. Check status of transmit buffer TBUF

T4. If EMPTY, disable SCI transmit interrupts and exit; else, continue

T5. Access character from TBUF[TOUT]

T6. Output character to SCI TDR

T7. TOUT = (TOUT + 1) mod TSIZE

T8. Exit

SCI Interrupt Service Routine

```
; Handles both receive and transmit sections of SCI

; STEP (R1) Check RDRF flag
; STEP (R2) If RDRF=0, go to transmit section;
;                   else, continue with receive section

sciisr brclr   scisr1,rxdrf,trans

; STEP (R3) Check status of RBUF
       ldab    rin
       incBmod rsize   ;(B) = (RIN+1) mod RSIZE

; STEP (R4) If FULL, error exit; else, continue
       cmpb    rout
       bne     rok
       swi
```

SCI Interrupt Service Routine

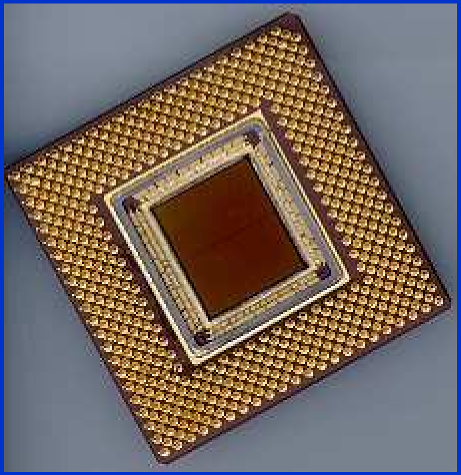
```
; STEP (R5) Input character from SCI RDR
rok    pshb          ; save (RIN+1) mod RSIZE
       ldaa  scidrl  ; read SCI RDR

; STEP (R6) Store character at RBUF[RIN]
       ldx   #rbuf
       ldab  rin
       staa  b,x     ; RBUF[RIN] = (RDR)

; STEP (R7) Increment RIN mod RSIZE
       pulb          ; restore from stack
       stab  rin     ; RIN = (RIN+1) mod RSIZE
```

SCI Interrupt Service Routine

```
; STEP (T1) Check TDRE flag
; STEP (T2) If TDRE=0, exit; else, continue
trans  brclr scisr1,txdre,scexit

; STEP (T3) Check status of TBUF
       ldab  tout
       cmpb  tin

; STEP (T4) If EMPTY, disable SCI transmit
;            interrupts and exit; else, continue
       bne   tok
       bclr  scicr2,timask ;disable trans intrs
       rti
```

SCI Interrupt Service Routine

```
; STEP (T5) Access character from TBUF
tok    ldx    #tbuf
       ldaa   b,x        ; (A) = TBUF[TOUT]

; STEP (T6) Output character to SCI TDR
       staa   scidrl     ; (TDR) = TBUF[TOUT]

; STEP (T7) Increment TOUT mod TSIZE
       incBmod  tsize
       stab     tout     ; TOUT=(TOUT+1) mod TSIZE

; STEP (T8) Exit
scexit rti
```



Microcontroller-Based Digital System Design

Module 3-C Serial Peripheral Interface (SPI)

Learning Objectives

- describe the key features of synchronous serial communication
- identify SPI features and operating modes
- distinguish among MISO, MOSI, SISO, and MOMI entities as they pertain to the SPI module
- configure the SPI to operate in a prescribed mode
- create an SPI device driver routine for a prescribed application

Outline

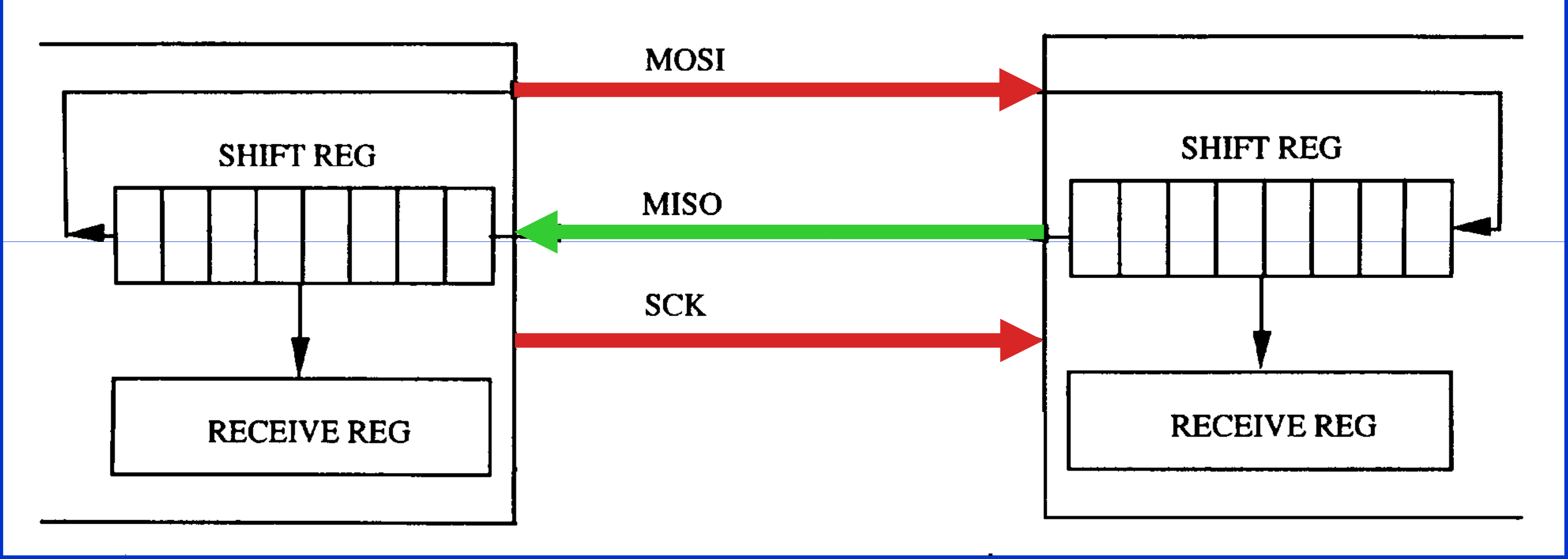
- Introduction
- Features
- Block Diagrams
- Registers
- Applications

Reference: Serial Peripheral Interface (SPI) Block User Guide

Introduction

- Definition: The **Serial Peripheral Interface** provides a high speed, ***synchronous*** serial interface with external ("peripheral") devices
- Description: A "master" and "slave" communicate by shifting bits to each other's registers; upon completion of eight bit shifts, a status flag is set and an interrupt is generated (if enabled)
- Where used:
  - communicating with simple peripherals such as LCD drivers, D/A converters, etc.
  - communicating with other microcontrollers

SPI Communication Block Diagram



MASTER

SLAVE

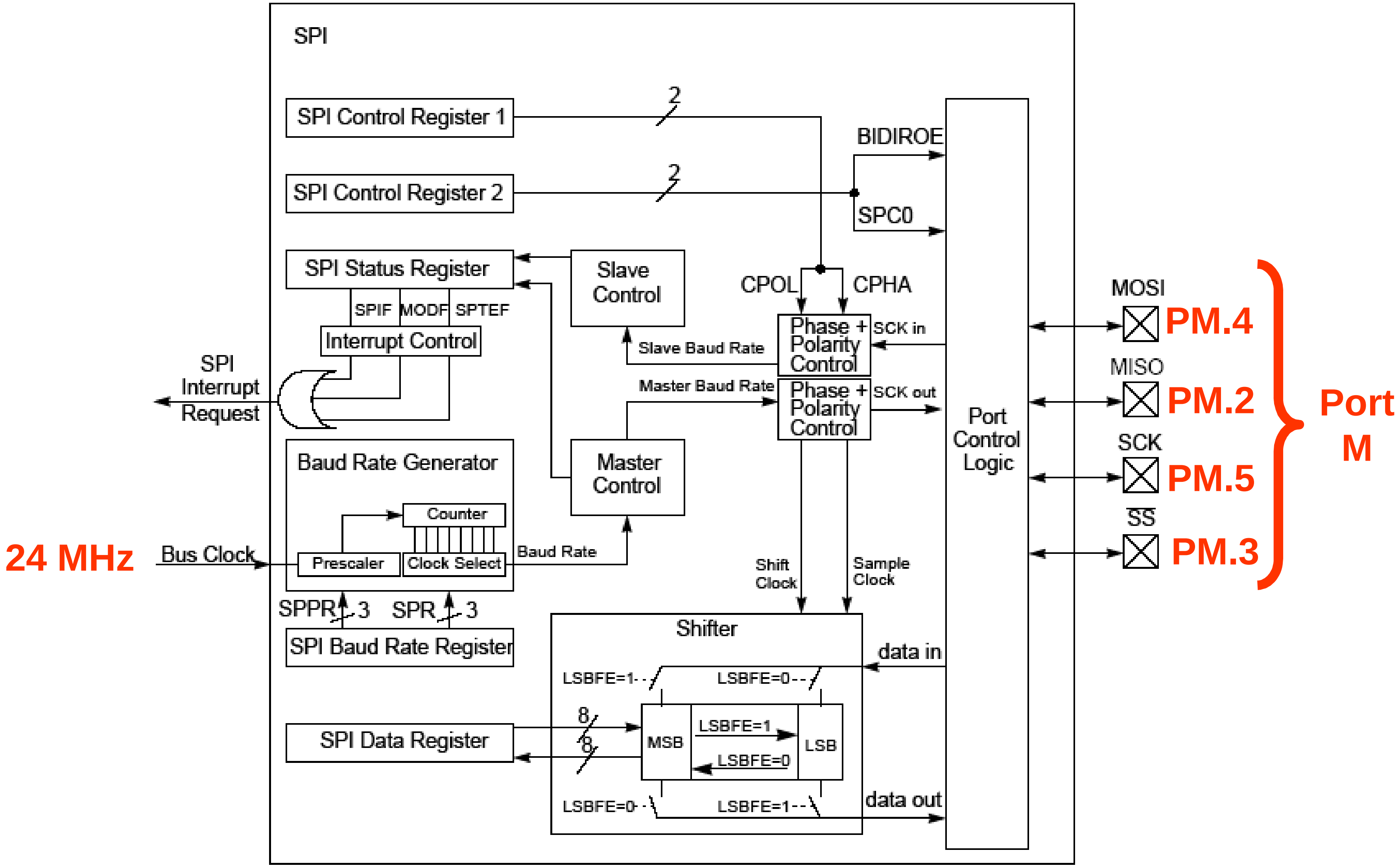
MOSI – Master Out, Slave In

MISO – Master In, Slave Out

SPI Features

- High speed synchronous serial interface
- May be used for inter-processor communication
- Flexible clock format
- Full duplex operation
- May be program- or interrupt-driven

SPI Block Diagram



SPI Registers

- **SPICR1 (SPI control register 1)**
  - **SPIE (bit 7)**
    - ***“0” – SPI interrupts disabled***
    - **“1” – SPI interrupts enabled**
  - **SPE (bit 6)**
    - ***“0” – SPI system disabled***
    - **“1” – SPI system enabled**
  - **SPTIE (bit 5)**
    - ***“0” – SPTEF (transmit empty) interrupt disabled***
    - **“1” – SPTEF *(transmit empty)* interrupt enabled**

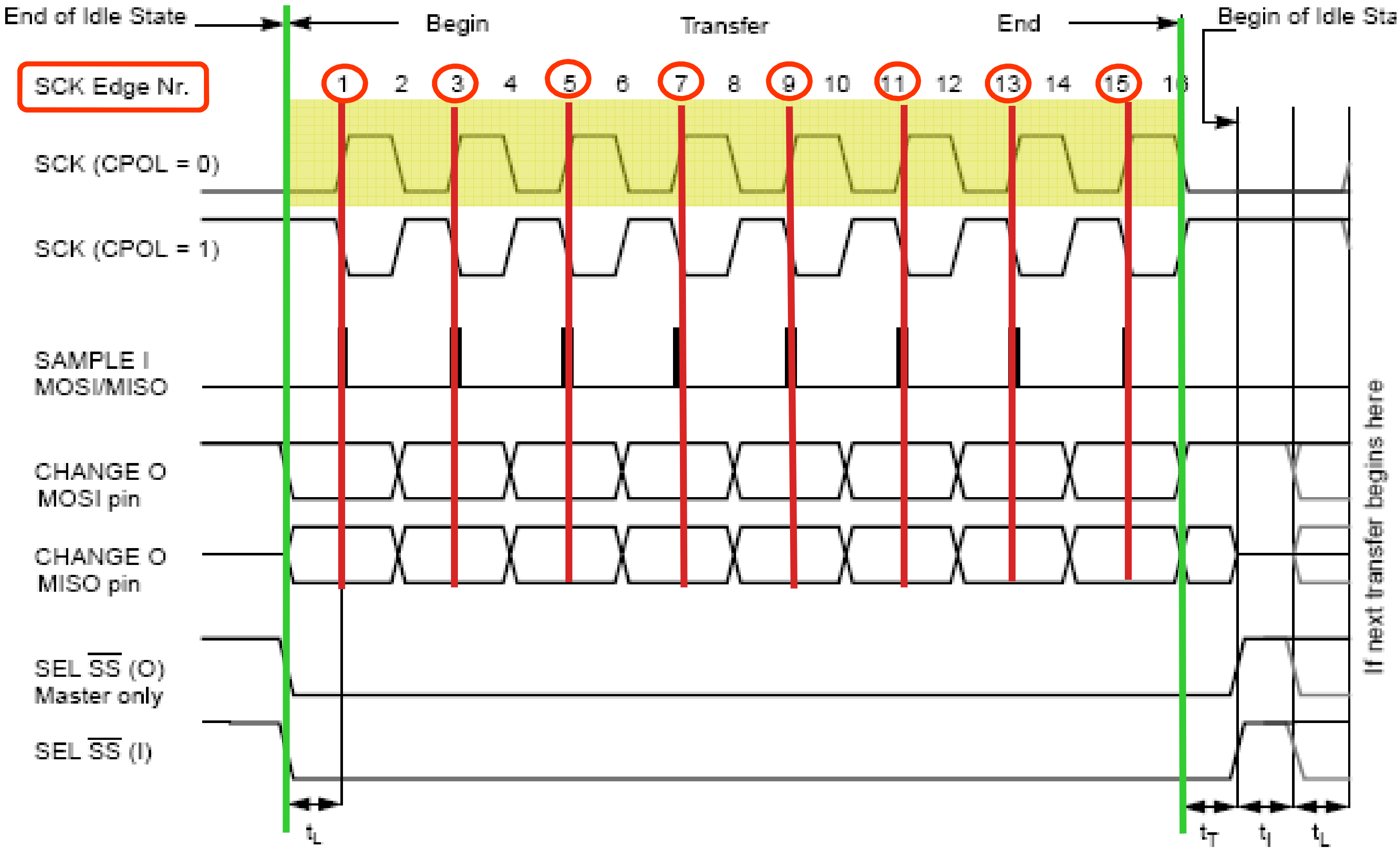
🟩 ***indicates default mode after RESET***

SPI Registers

- **SPICR1 (SPI control register 1)**
  - **MSTR (bit 4)**
    - ***"0" – slave mode enabled***
    - **"1" – master mode enabled**
  - **CPOL (bit 3)**
    - ***"0" – active high clock (SCK low in idle state)***
    - **"1" – active low clock (SCK high in idle state)**
  - **CPHA (bit 2)**
    - **"0" – data sampling occurs at odd edges of SCK**
    - ***"1" – data sampling occurs at even edges of SCK***

🟩 ***indicates default mode after RESET***

SPI Clock Format with CPOL=0, CPHA=0



MSB first (LSBFE = 0):	MSB	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	LSB	Minimum 1/2 SCK for t_T, t_I, t_L
LSB first (LSBFE = 1):	LSB	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	MSB	

t_L = Minimum leading time before the first SCK edge

t_T = Minimum trailing time after the last SCK edge

t_I = Minimum idling time between transfers (minimum $\overline{SS}$ high time)

t_L, t_T, and t_I are guaranteed for the master mode and required for the slave mode.

SPI Registers

- **SPICR1 (SPI control register 1)**
  - **SSOE (bit 1)**
    - ***"0" – slave select output disabled***
    - **"1" – slave select output enabled (in master mode, provided MODFEN bit is also "1")**
  - **LSBFE (bit 0)**
    - ***"0" – data transferred most significant bit first***
    - **"1" – data transferred least significant bit first**

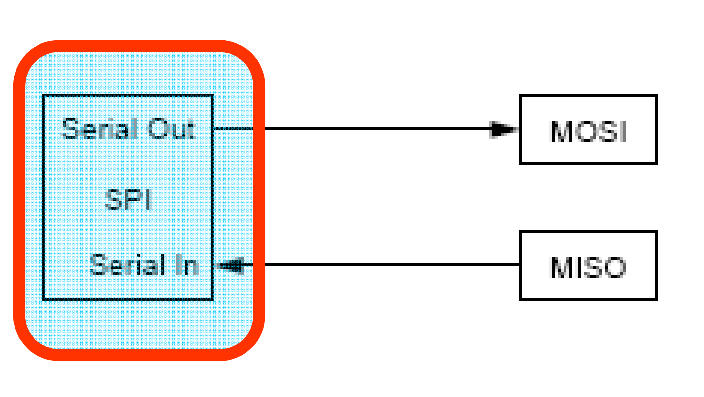
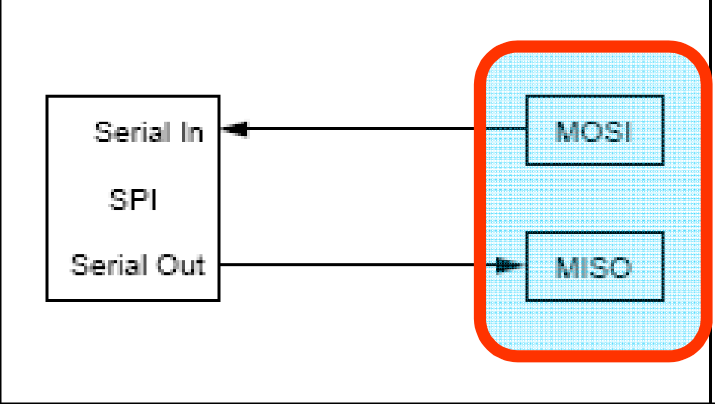
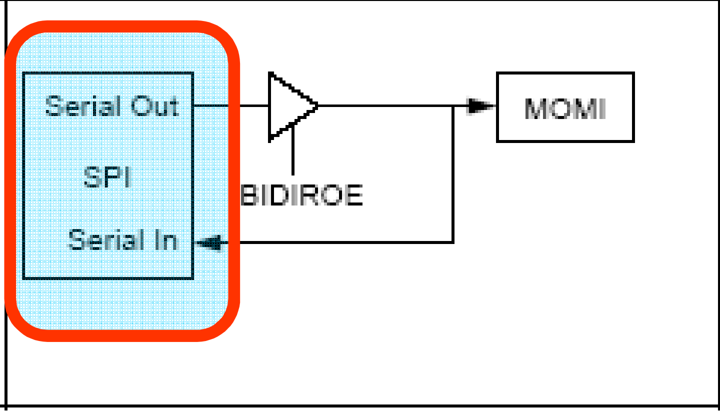
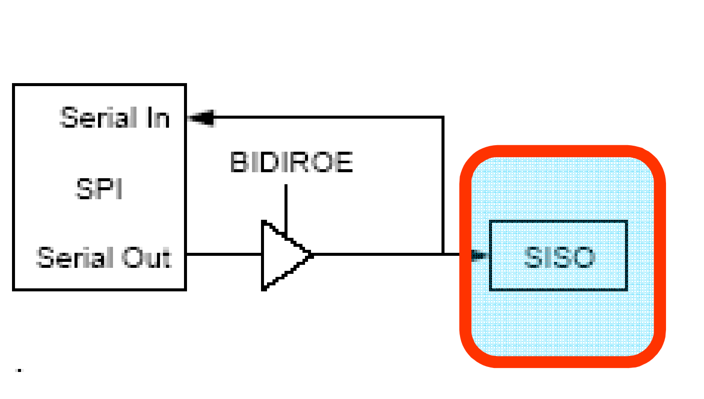
🟩 ***indicates default mode after RESET***

SPI Registers

- **SPICR2 (SPI control register 2)**
  - **MODFEN (bit 4) – allows detection of MODF “mode fault” failure (master mode only)**
    - ***“0” – slave select (SS) port pin not used by SPI***
    - **“1” – slave select (SS) port pin enabled with MODF feature**
  - **BIDIROE (bit 3) – MOSI/MISO output enable in bi-directional mode**
    - ***“0” – output buffer disabled***
    - **“1” – output buffer enabled**
  - **SPC0 (bit 0) – serial pin control bit**
    - ***“0” – bi-directional mode is disabled***
    - **“1” – bi-directional mode enabled**

🟩 ***indicates default mode after RESET***

SPI Modes **Master**

When SPE = 1	Master Mode MSTR = 1	Slave Mode MSTR = 0
Normal Mode SPC0 = 0		
Bidirectional Mode SPC0 = 1		

Key: Master Slave Input Output

SPI Registers

- **SPIBR (baud rate)**
  - **SPRx (bits 2-0) – baud rate pre-selection bits**
  - **SPPRx (bits 6-4) – baud rate selection bits**

$$\text{BaudRateDivisor} = (\text{SPPR} + 1) * 2^{(\text{SPR} + 1)}$$

$$\text{BaudRate} = \text{BusClock} / \text{BaudRateDivisor}$$

Example: If SPR=000 and SPPR=000 (default), then BaudRateDivisor = 2 → BaudRate = BusClock/2

SPI Registers

- **SPISR (status)**
  - **SPIF (bit 7) – set after a received data byte is copied into the SPI data register**
    - ***“0” – transfer not yet complete***
    - **“1” – received data copied to SPIDR**

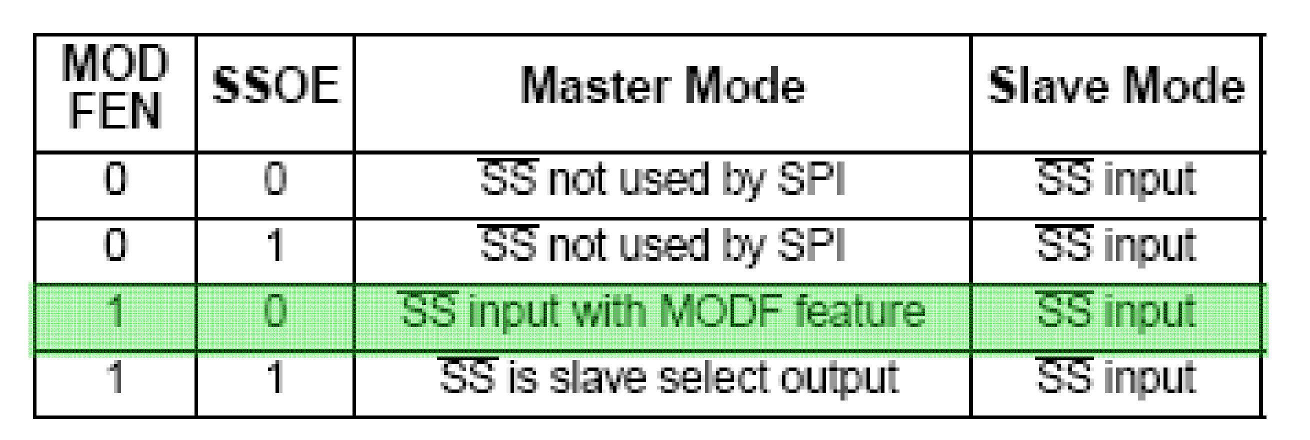
Cleared **by reading SPISR (status) register followed by reading SPIDR (data) register**

  - **SPTEF (bit 5) – set when the transmit data register is empty**
    - ***“0” – SPI transmit data register not empty***
    - **“1” – SPI transmit data register is empty**

🟩 ***indicates default mode after RESET***

SPI Registers

- **SPISR (status)**
  - **MODF (bit 4)**
    - ***“0” – mode fault has not occurred***
    - **“1” – mode fault has occurred (MSTR control bit set and slave select input asserted, which is *not permitted* in normal operation)**



MOD FEN	SSOE	Master Mode	Slave Mode
0	0	$\overline{\text{SS}}$ not used by SPI	$\overline{\text{SS}}$ input
0	1	$\overline{\text{SS}}$ not used by SPI	$\overline{\text{SS}}$ input
1	0	$\overline{\text{SS}}$ input with MODF feature	$\overline{\text{SS}}$ input
1	1	$\overline{\text{SS}}$ is slave select output	$\overline{\text{SS}}$ input

🟩 ***indicates default mode after RESET***

SPI Registers

- SPIDR (data register)
  - serves as both the input and the output data register for the SPI
- PORTM (Port M data register)
  - MISO (bit 2)
  - SS (bit 3)
  - MOSI (bit 4)
  - SCK (bit 5)

Clicker Quiz

1. Transmission of serial data using the SPI is inherently faster than using the SCI because:

   A. synchronous transmission does not require start/stop bits

   B. synchronous transmission requires an accompanying clock signal

   C. synchronous transmission does not require local synchronization

   D. all of the above

   E. none of the above

2. The “synchronous” aspect of an SPI interface means:

   A. data can only be transmitted in one direction
   B. data can be transmitted in both directions, but not at the same time
   C. data can be transmitted in both directions simultaneously
   D. data transmission requires a clock signal
   E. none of the above

3. The minimum SPI baud rate divisor possible is:

A. 1

B. 2

C. 4

D. 8

E. none of the above

4. Based on a 24 MHz system clock, the **maximum** SPI data transfer rate possible (in bits/second) is:

A. 2,000,000 bps

B. 4,000,000 bps

C. 8,000,000 bps

D. 12,000,000 bps

E. none of the above

5. The maximum SPI baud rate divisor possible is:

A. 128

B. 896

C. 2048

D. 4608

E. none of the above

6. Based on a 24 MHz system clock, the minimum SPI data transfer rate possible (in bits/second) is approximately:

A. 8,000 bps

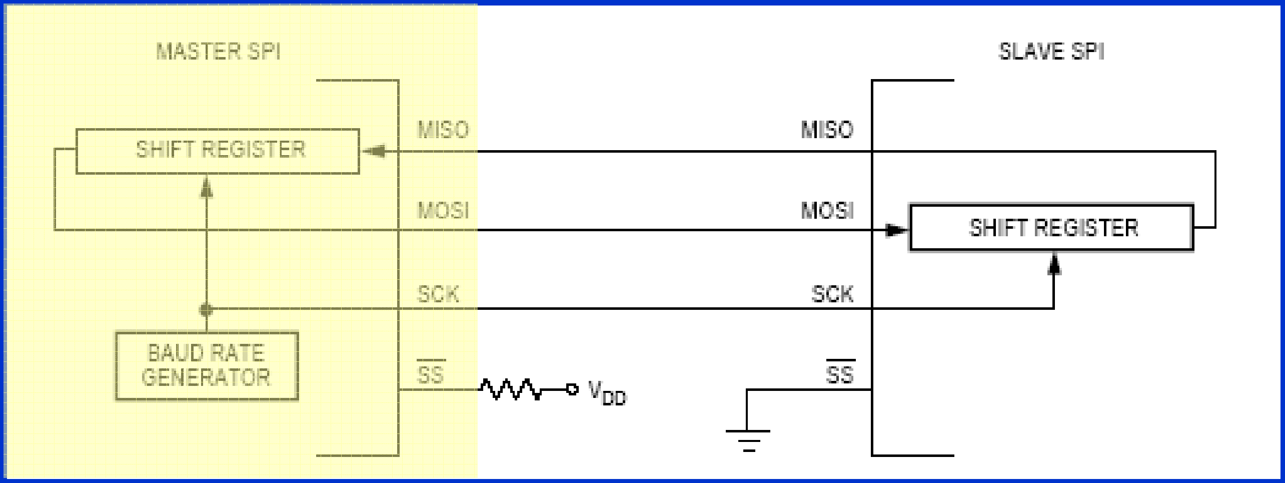
B. 11,719 bps

C. 93,750 bps

D. 1,500,000 bps

E. none of the above

SPI Initialization – Program-Driven Mode



```
; Select 1.5 Mbps baud rate (24 MHz bus)
spini  movb #$12,spibr
```

SPI Registers

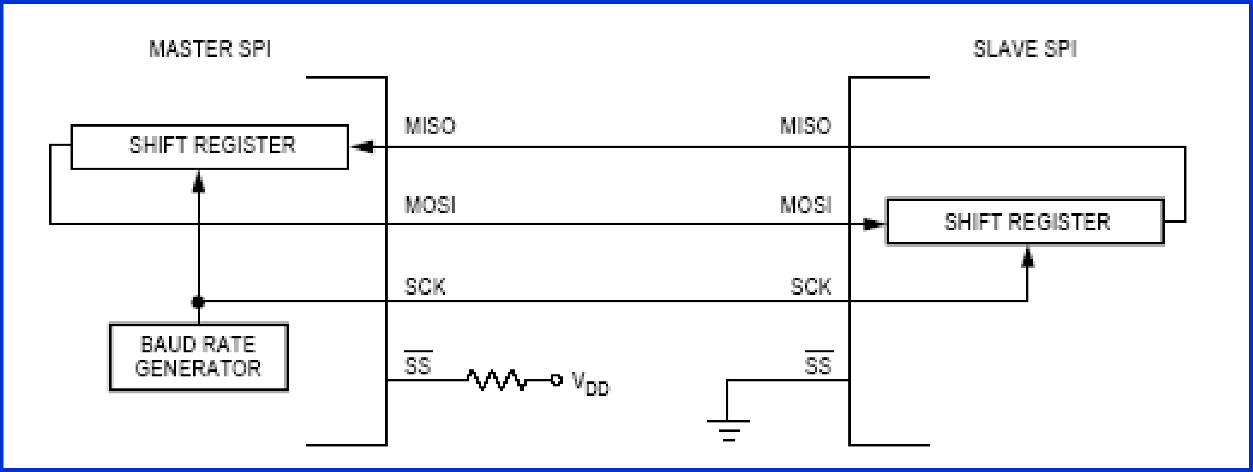
- **SPIBR (baud rate)**
  - **SPRx (bits 2-0) – baud rate pre-selection bits = 010**
  - **SPPRx (bits(6-4) – baud rate selection bits = 001**

BaudRateDivisior = (1 + 1) $*$ 2 $^{(2 + 1)}$ = 16

BaudRate = 24 MHz / 16 = 1.5 Mbps

SPIBR = x 001 x 010 = $12

SPI Initialization – Program-Driven Mode



```
; Select 1.5 Mbps baud rate (24 MHz bus)
 spini  movb #$12,spibr
; Master mode, Interrupts off, CPOL=0,
; CPHA=0, slave select disabled, data
; transferred most significant bit first
        movb #$50,spicr1
```

SPI Registers

- **SPICR1 (SPI control register 1)**
  - **SPIE (bit 7)**
    - ***"0" – SPI interrupts disabled***
    - **"1" – SPI interrupts enabled**
  - **SPE (bit 6)**
    - ***"0" – SPI system disabled***
    - **"1" – SPI system enabled**
  - **SPTIE (bit 5)**
    - ***"0" – SPTEF interrupt disabled***
    - **"1" – SPTEF interrupt enabled**

🟩 ***indicates default mode after RESET***

SPI Registers

- **SPICR1 (SPI control register 1)**
  - **MSTR (bit 4)**
    - ***“0” – slave mode enabled***
    - **“1” – master mode enabled**
  - **CPOL (bit 3)**
    - ***“0” – active high clock (SCK low in idle state)***
    - **“1” – active low clock (SCK high in idle state)**
  - **CPHA (bit 2)**
    - **“0” – data sampling occurs at odd edges of SCK**
    - ***“1” – data sampling occurs at even edges of SCK***

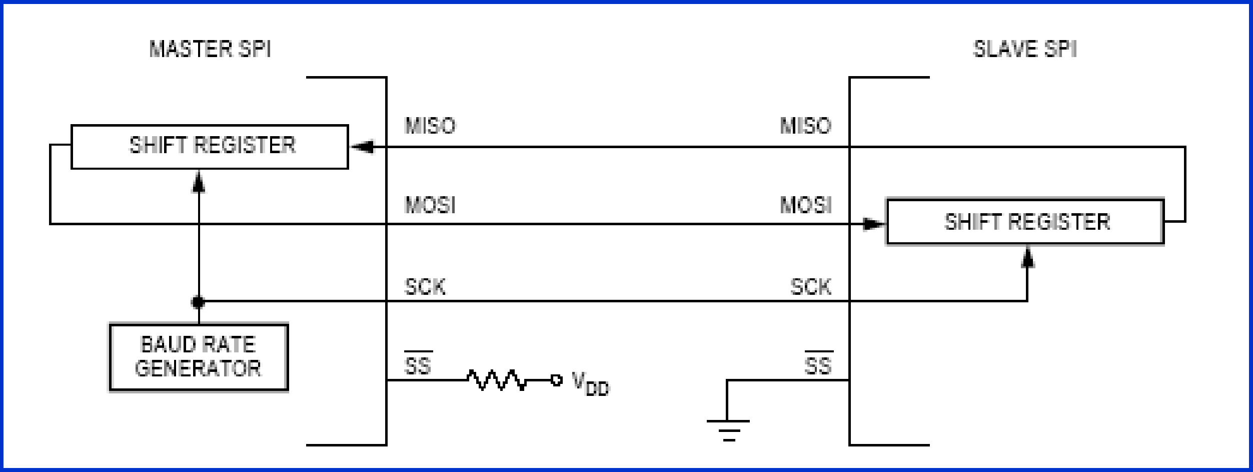
🟩 ***indicates default mode after RESET***

SPI Registers

- **SPICR1 (SPI control register 1)**
  - **SSOE (bit 1)**
    - ***"0" – slave select output disabled***
    - **"1" – slave select output enabled (in master mode, provided MODFEN bit is also "1")**
  - **LSBFE (bit 0)**
    - ***"0" – data transferred most significant bit first***
    - **"1" – data transferred least significant bit first**

🟩 ***indicates default mode after RESET***

SPI Initialization – Program-Driven Mode



```
; Select 1.5 Mbps baud rate (24 MHz bus)
 spini  movb #$12,spibr
; Master mode, Interrupts off, CPOL=0,
; CPHA=0, slave select disabled, data
; transferred most significant bit first
        movb #$50,spicr1
; Normal (non-bidirectional) mode
        clr  spicr2
        rts
```

SPI Registers

- SPICR2 (SPI control register 2)
  - MODFEN (bit 4) – allows detection of MODF “mode fault” failure (master mode only)
    - *“0” – slave select (SS) port pin not used by SPI*
    - “1” – slave select (SS) port pin enabled with MODF feature
  - BIDIROE (bit 3) – MOSI/MISO output enable in bi-directional mode
    - *“0” – output buffer disabled*
    - “1” – output buffer enabled
  - SPC0 (bit 0) – serial pin control bit
    - *“0” – bi-directional mode is disabled*
    - “1” – bi-directional mode enabled

🟩 *indicates default mode after RESET*

SPI Transmit/Receive (I/O) Routine

```
;Transmit data passed in A register
;Return data read in B register

;Transmit - wait for SPTEF to set
spio  brclr spisr,$20,spio
      staa  spidr     ;transmit data

;Receive - wait for SPIF to set
sprlp brclr spisr,$80,sprlp
      ldab  spidr    ;read data
      rts
```

SPI Registers

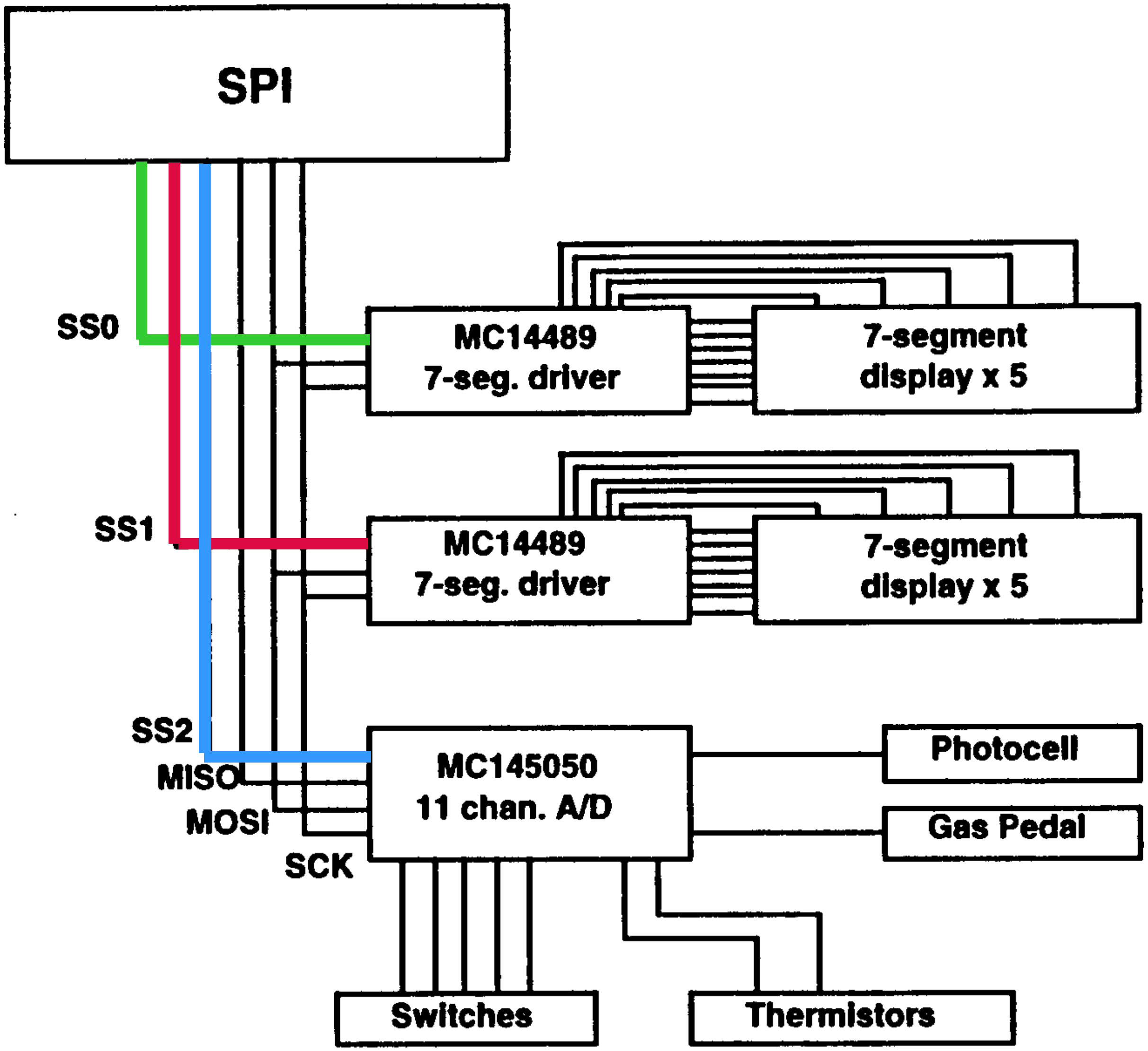
- **SPISR (status)**
  - **SPIF (bit 7) – set after a received data byte is copied into the SPI data register**
    - ***"0" – transfer not yet complete***
    - **"1" – received data copied to SPIDR**

Cleared **by reading SPISR (status) register followed by reading SPIDR (data) register**

  - **SPTEF (bit 5) – set when the transmit data register is empty**
    - ***"0" – SPI transmit data register not empty***
    - **"1" – SPI transmit data register is empty**

🟩 ***indicates default mode after RESET***

SPI Application – Multiple I/O Devices



SPI Application – MC14489 Interface

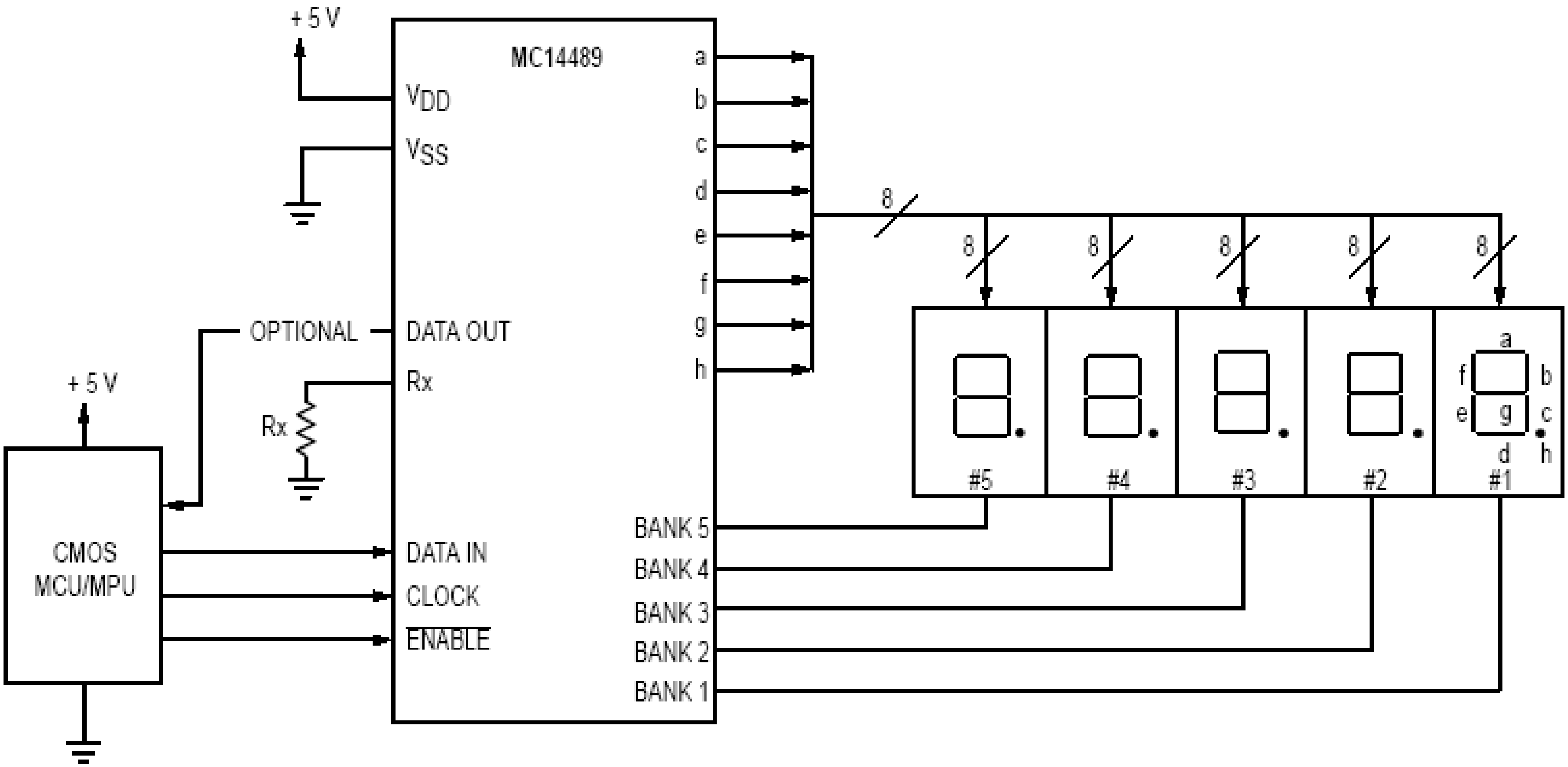


Figure 9. Non–Cascaded Application Example: 5 Character Common Cathode LED Display with Two Intensities as Controlled via Serial Port

SPI Application – MC145050 Interface

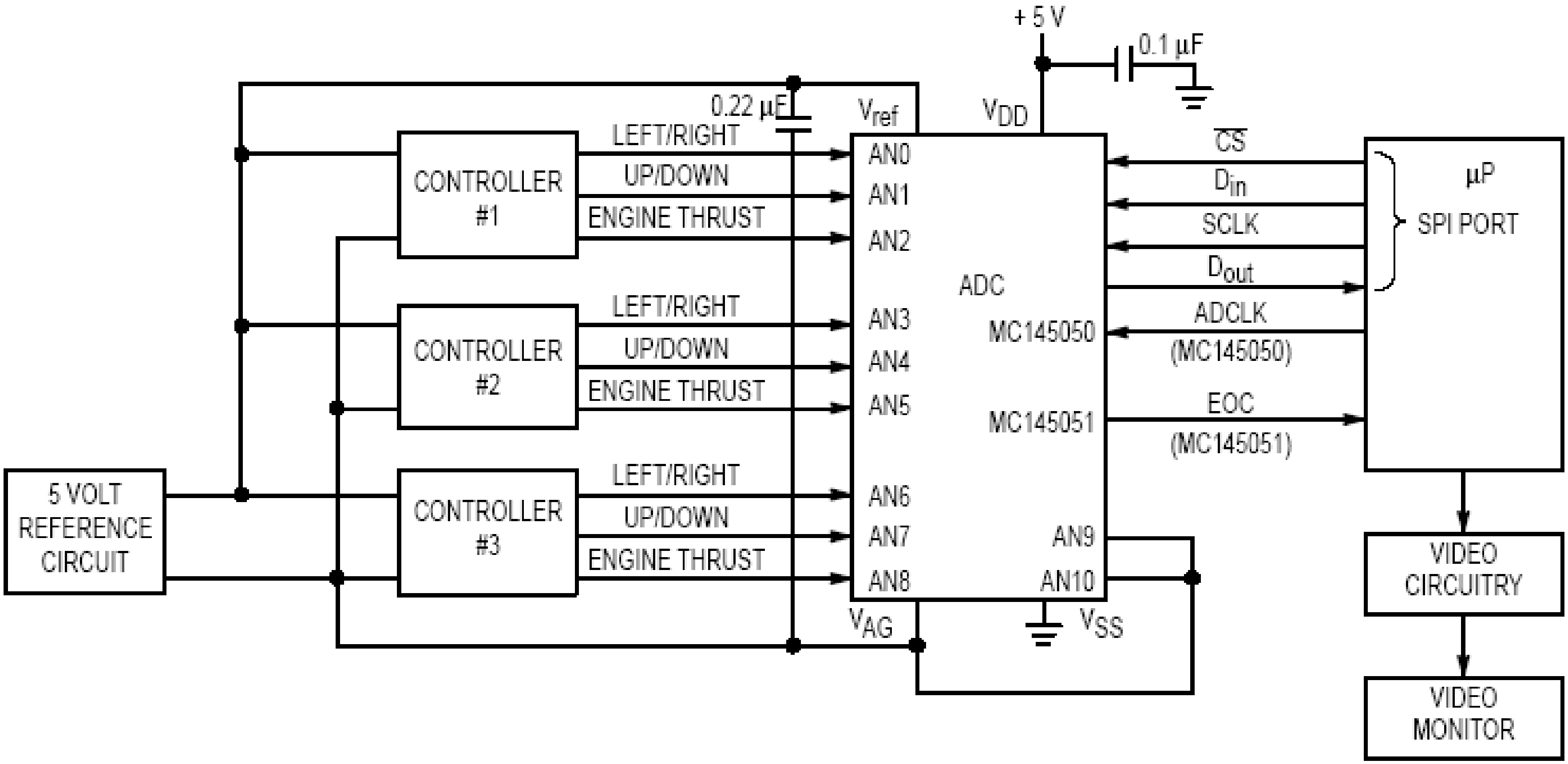
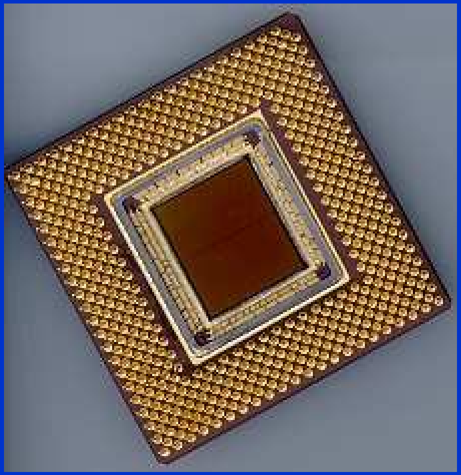


Figure 15. Joystick Interface



Microcontroller-Based Digital System Design

Module 3-D Timer Module (TIM)

Learning Objectives

- **describe the function of the TIM input capture mechanism**
- **describe the function of the TIM output compare mechanism**
- **describe the function of the TIM pulse accumulator mechanism**
- **distinguish among input capture, output compare, and pulse accumulator timer applications**
- **identify TIM features and operating modes**
- **configure the TIM to operate in a prescribed mode**
- **create a TIM device driver routine for a prescribed application**
- **discuss why the TIM module can serve as a higher precision reference for timer applications than the RTI subsystem**

Outline

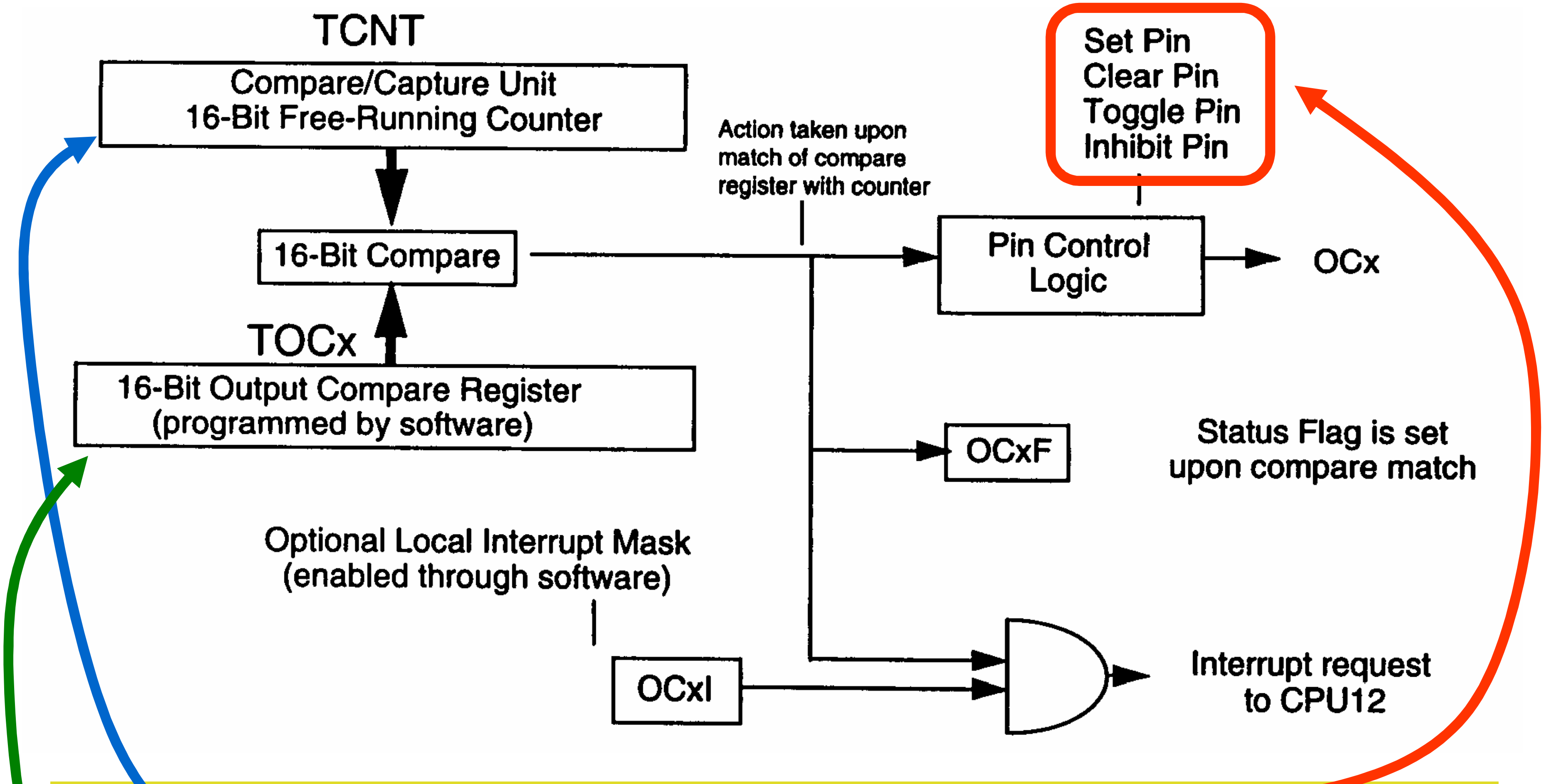
- Introduction
- Features
- Block Diagrams
- Registers
- Applications

Reference: **Timer (TIM) Block User Guide**

Introduction

- **Definition: The OUTPUT COMPARE function provides a mechanism to *output a signal* at a *specific time*, without *CPU intervention***
- **Where used:**
  - **waveform generation (e.g., PWM)**
  - **fuel injection/ignition timing**

Output Compare Function

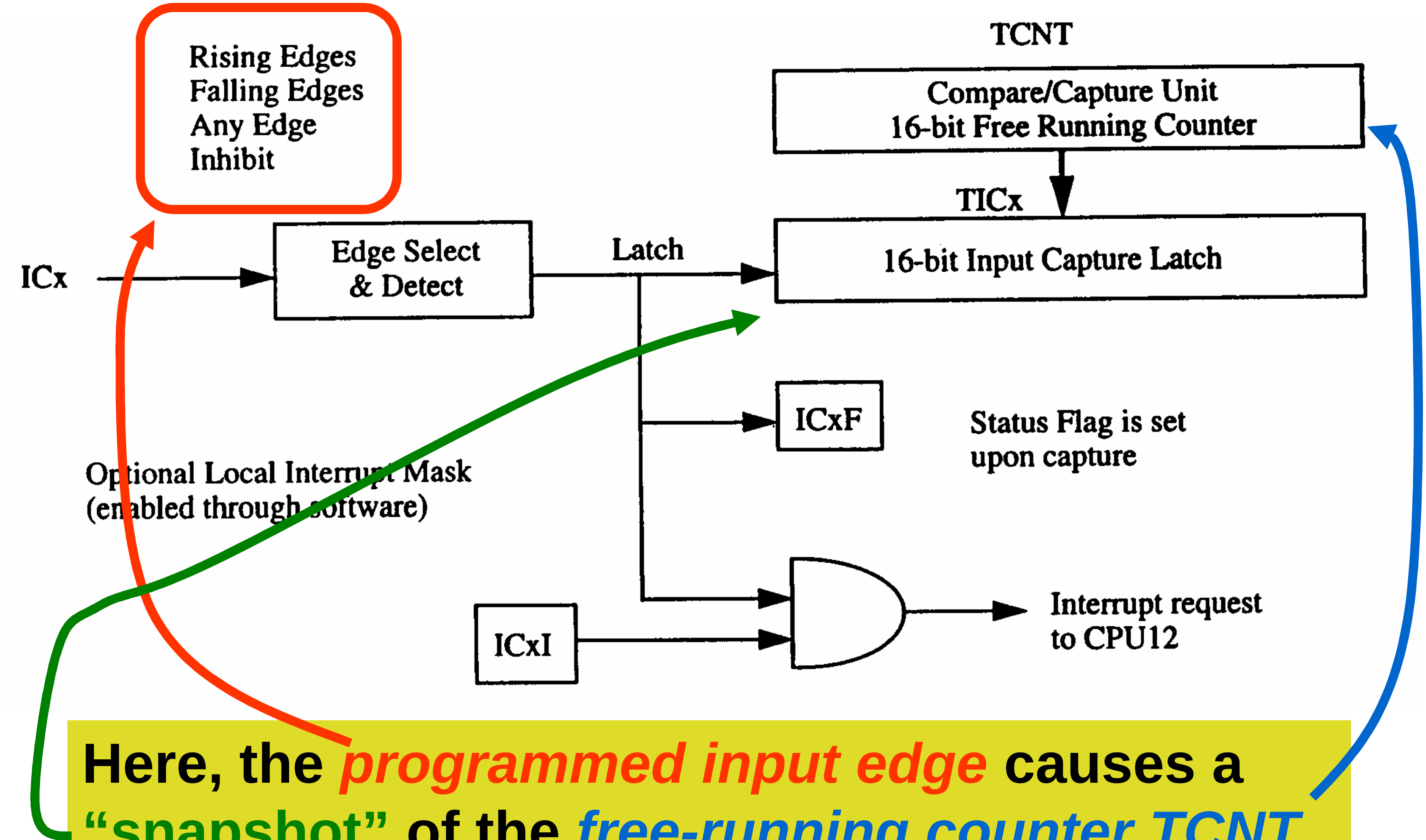


Here, the *programmed output transition* occurs when the *free-running counter TCNT* <u>matches</u> the *programmed output compare value*

Introduction

- Definition: The INPUT CAPTURE function provides a mechanism to ***capture the time*** at which an ***external event*** occurs, without ***CPU intervention***
- Where used:
  - waveform measurement
  - acceleration/velocity measurement

Input Capture Function

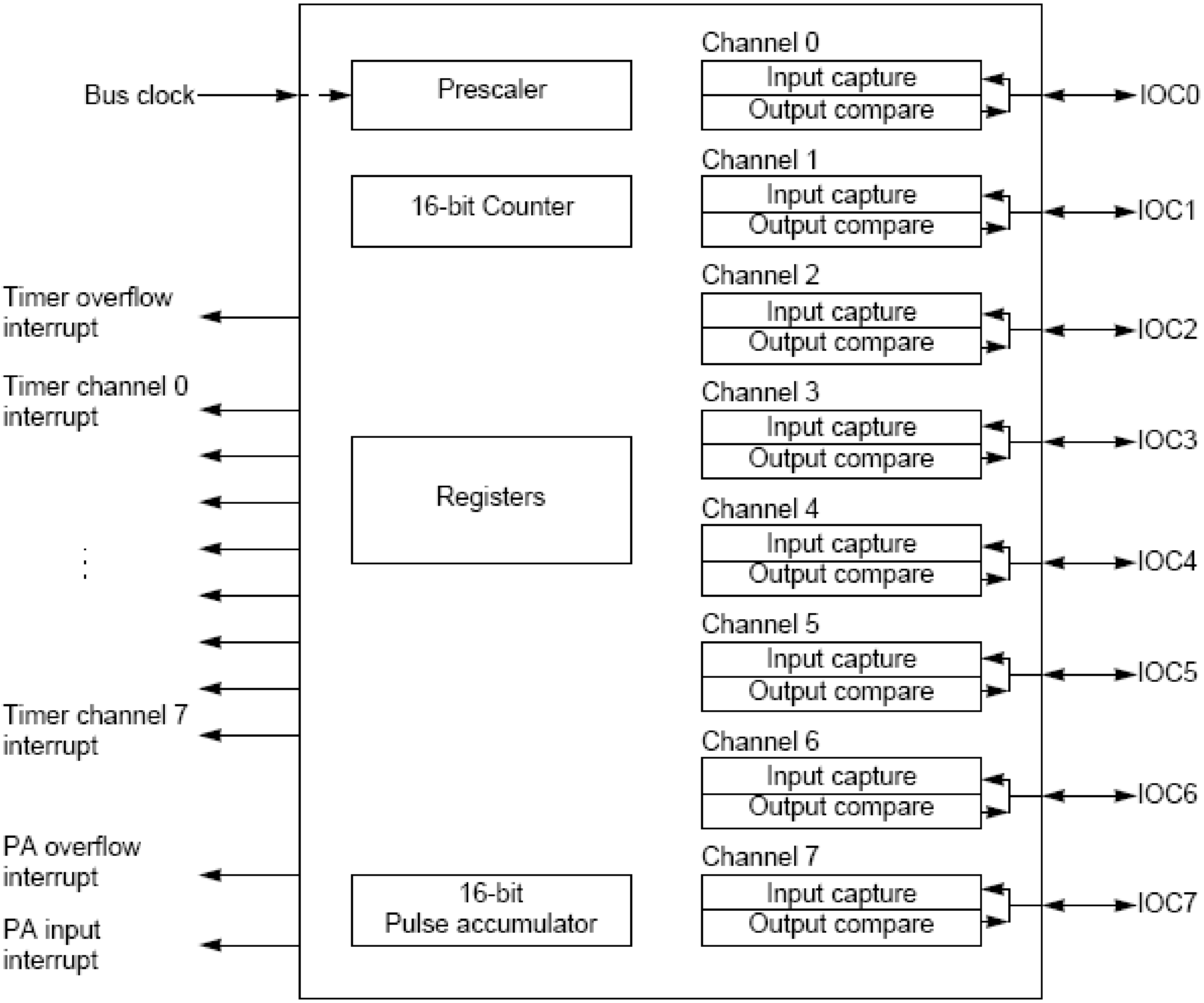


Here, the *programmed input edge* causes a "snapshot" of the *free-running counter TCNT* to be taken to serve as an event time stamp

Introduction

- **Definition: The PULSE ACCUMULATOR function provides a mechanism to *count external events*, without *CPU intervention***
- **Where used:**
  - **speed measurement**
  - **event counting**

TIM & PA Module Block Diagram



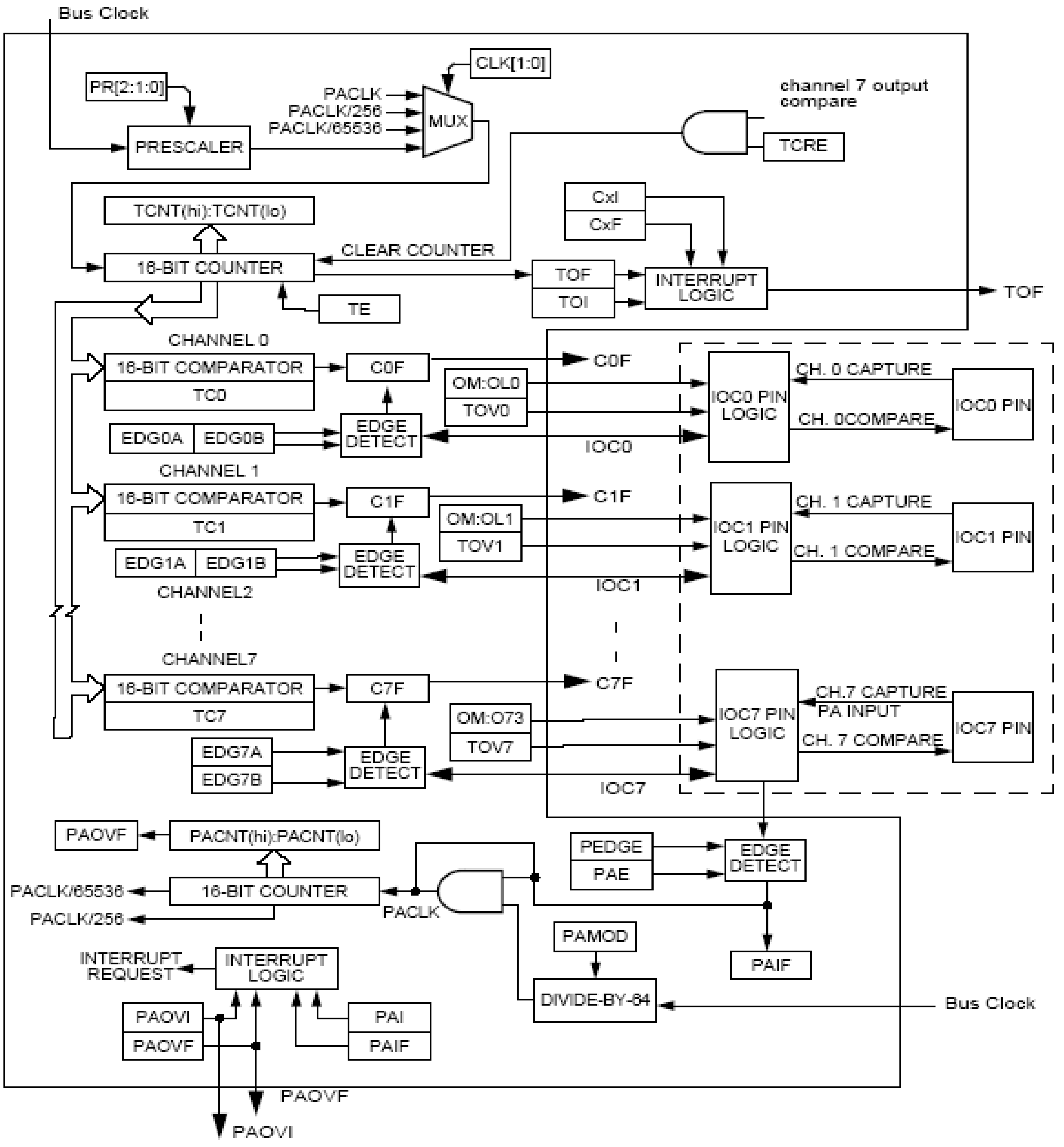
TIM Features and Functions

- TIM supports ***any mixture*** of 8 input capture channels and/or output compare channels
- There is a unique interrupt vector for each TIM channel (plus a few extras)
- Actions (time stamps for input captures, output pin assertions for output compares) are based on the state of a 16-bit free-running counter ("TCNT")
- **Channel 7 is "special"** – it can be configured to reset **TCNT** (and generate an interrupt) upon a successful output compare, thereby providing the capability to produce ***very precise periodic interrupts***

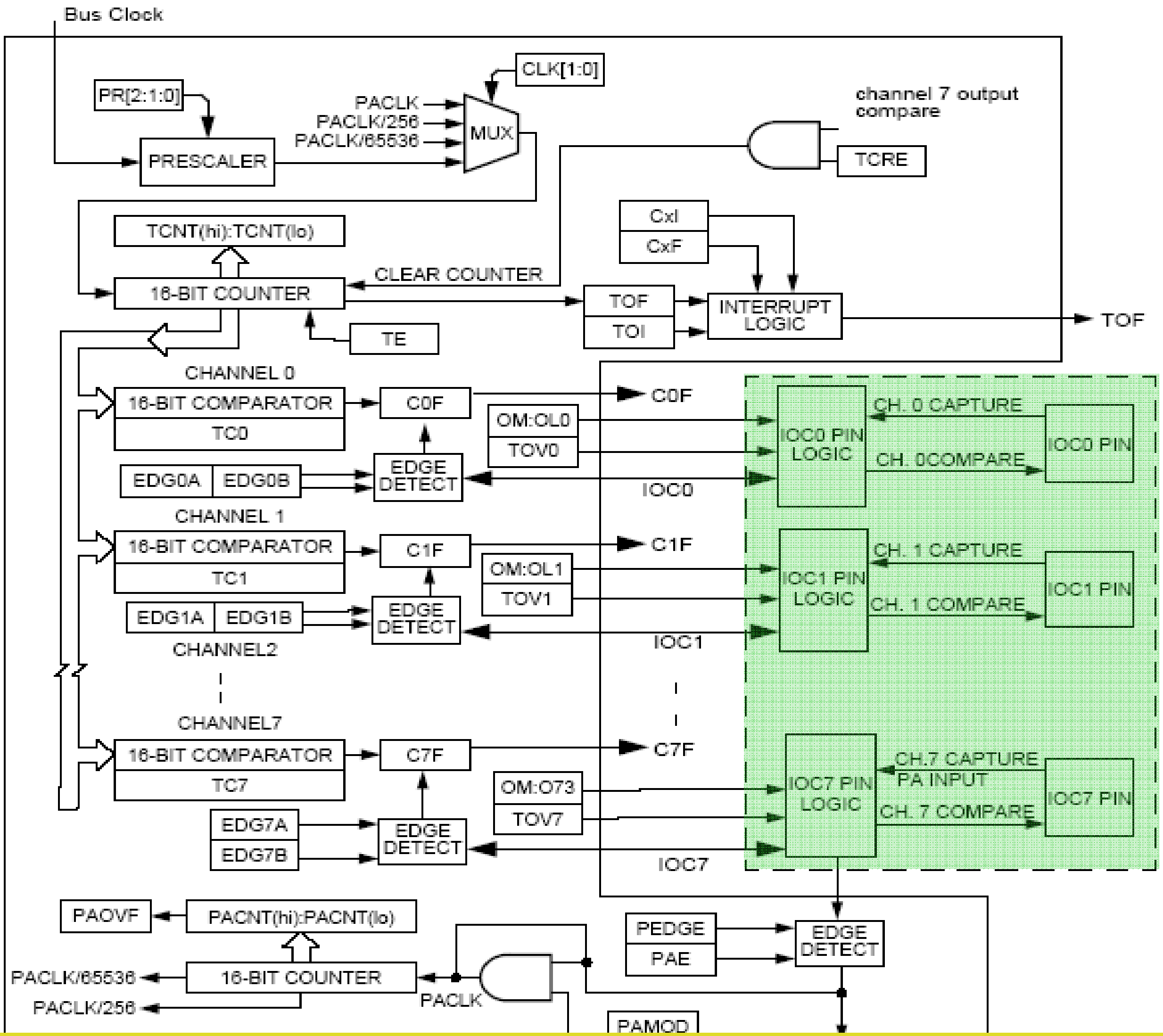
TIM Features and Functions

- The (single) pulse accumulator (PA) channel shares a port pin with **TIM Channel 7**
- The PA and the TIM Ch 7 output compare functions can be used ***simultaneously*** by **"disconnecting"** the TIM pad logic from the port pin
- The **"force output compare"** mechanism provides an application program the ability to force the set of actions associated with an output compare to occur ***immediately*** (instead of waiting for the 16-bit counter to reach the programmed value)

TIM/PA Block Diagram Details

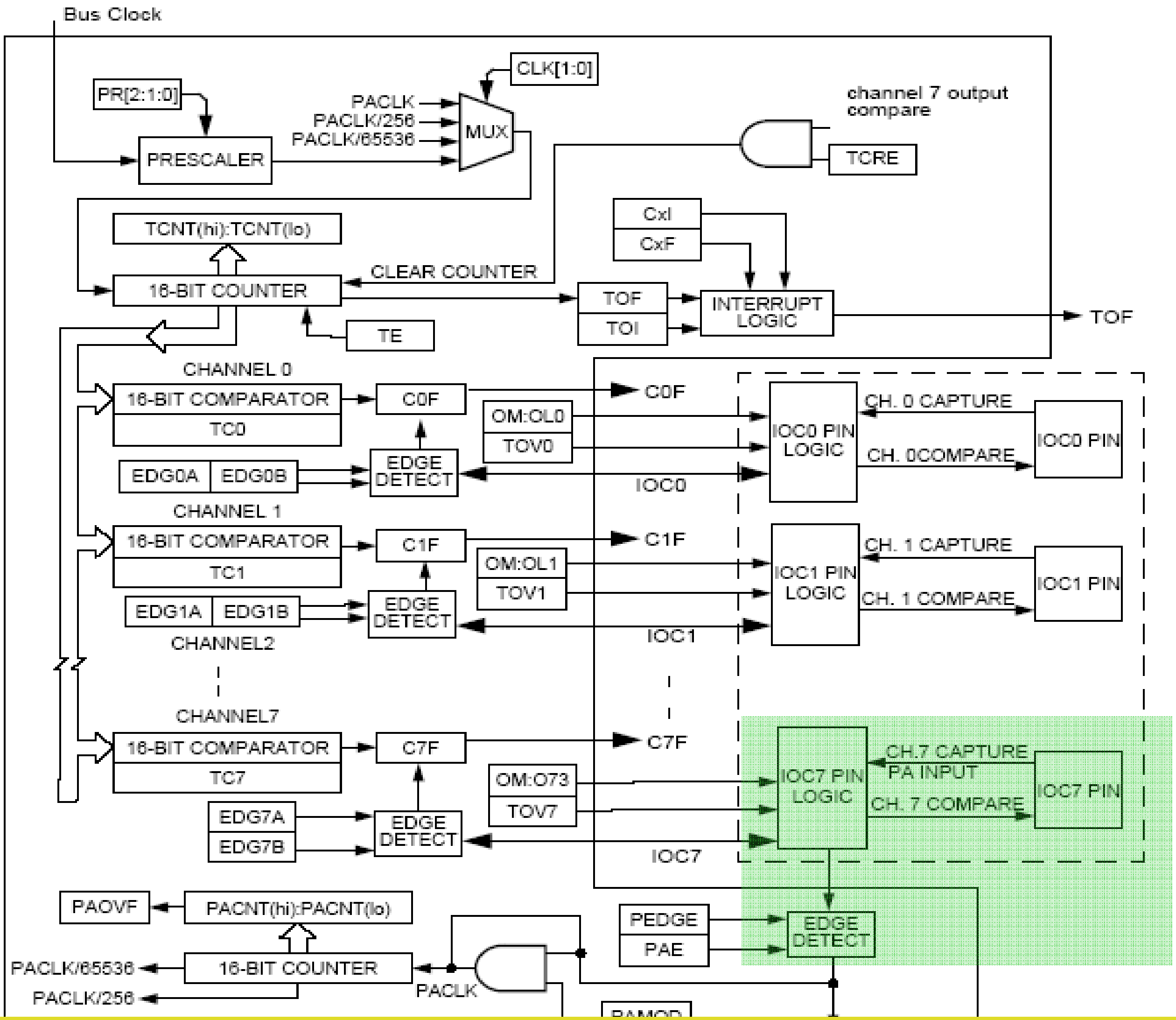


TIM/PA Block Diagram Details



Port T pins that are not being used for TIM functions can be “disconnected” from the TIM module and accessed via the Port T register (requires appropriate Port T DDR settings)

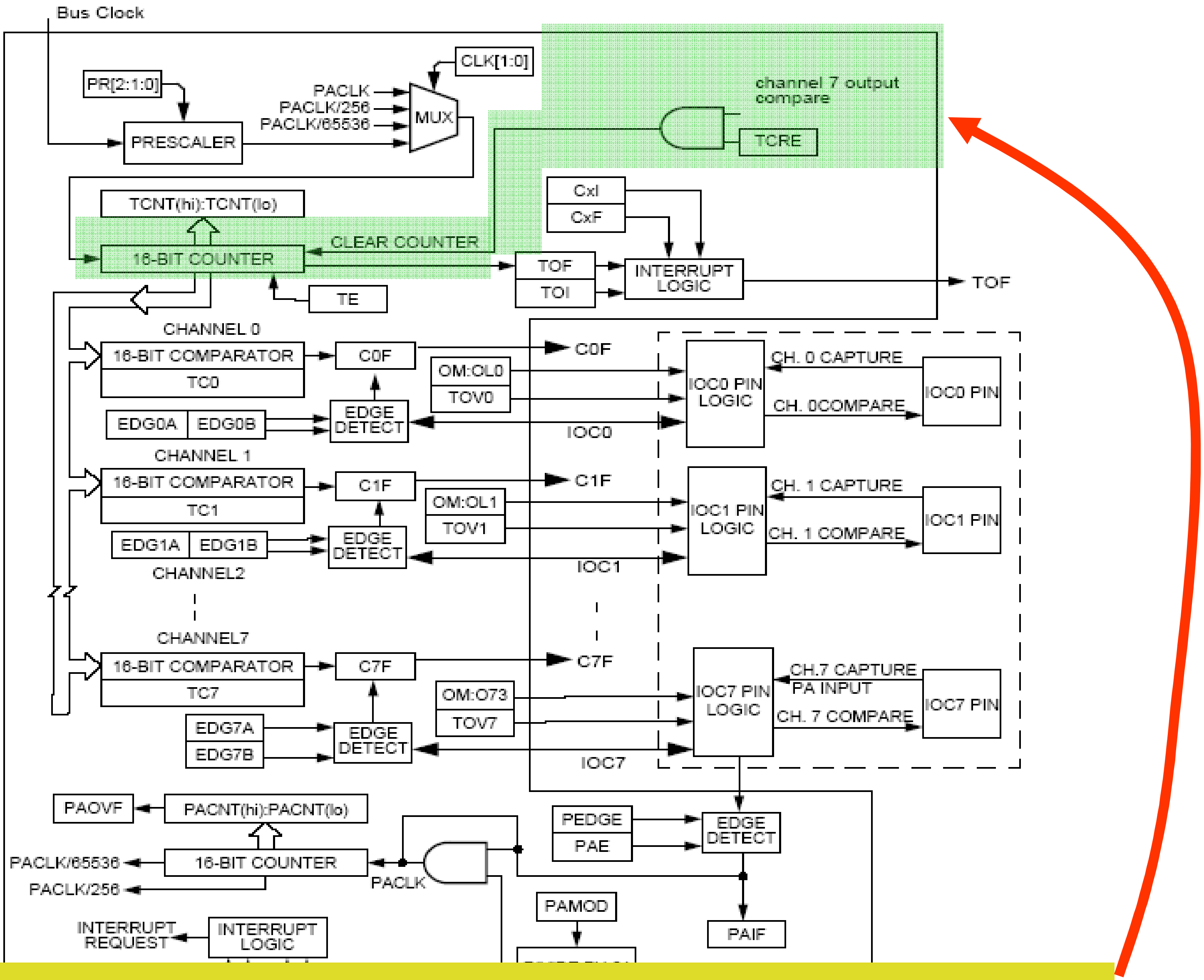
TIM/PA Block Diagram Details



The port pin for TIM Channel 7 is shared with the PA – pin logic is provided that allows the TIM to be disconnected from the port pin so that an Output Compare 7 and the PA can be used independently

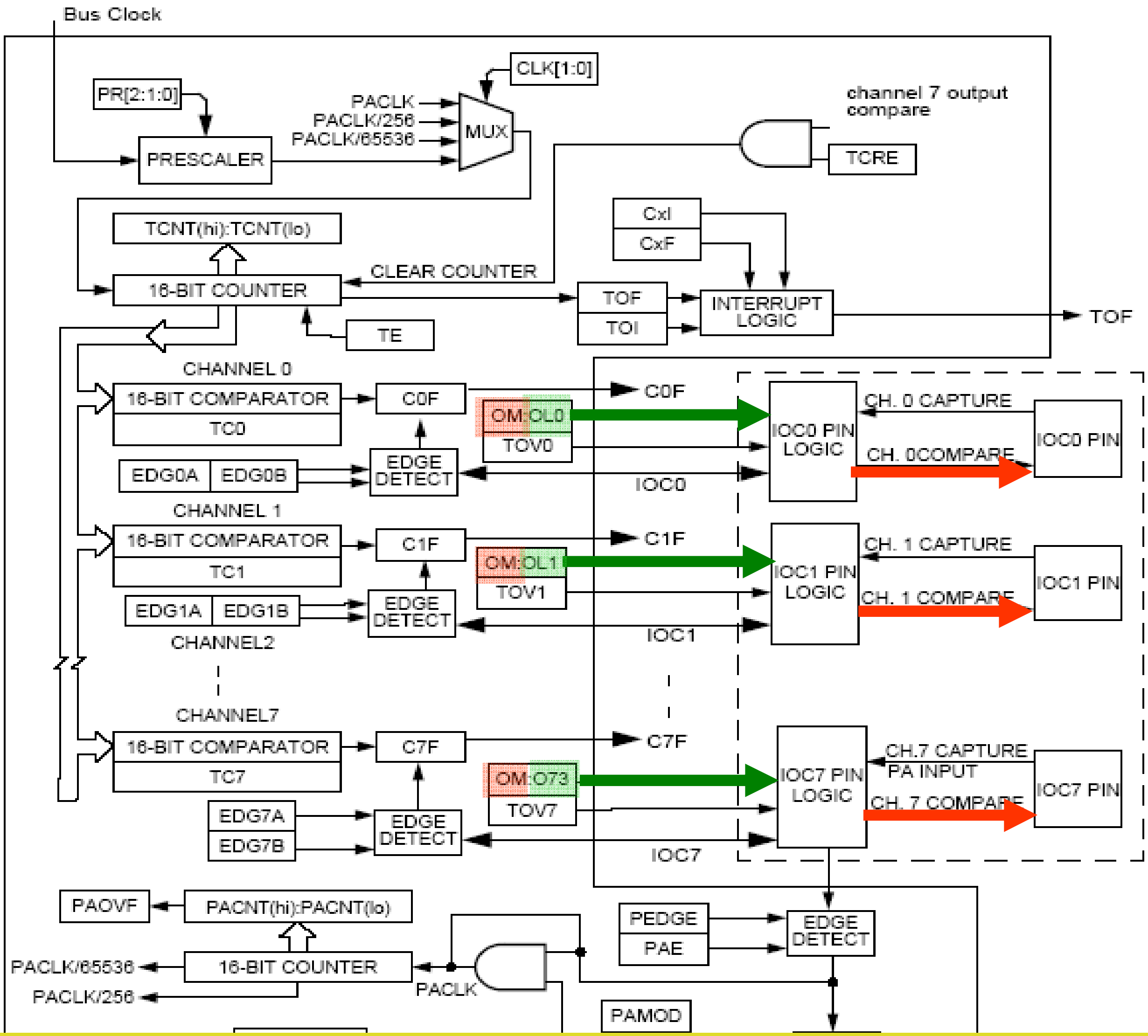
PAOVI

TIM/PA Block Diagram Details



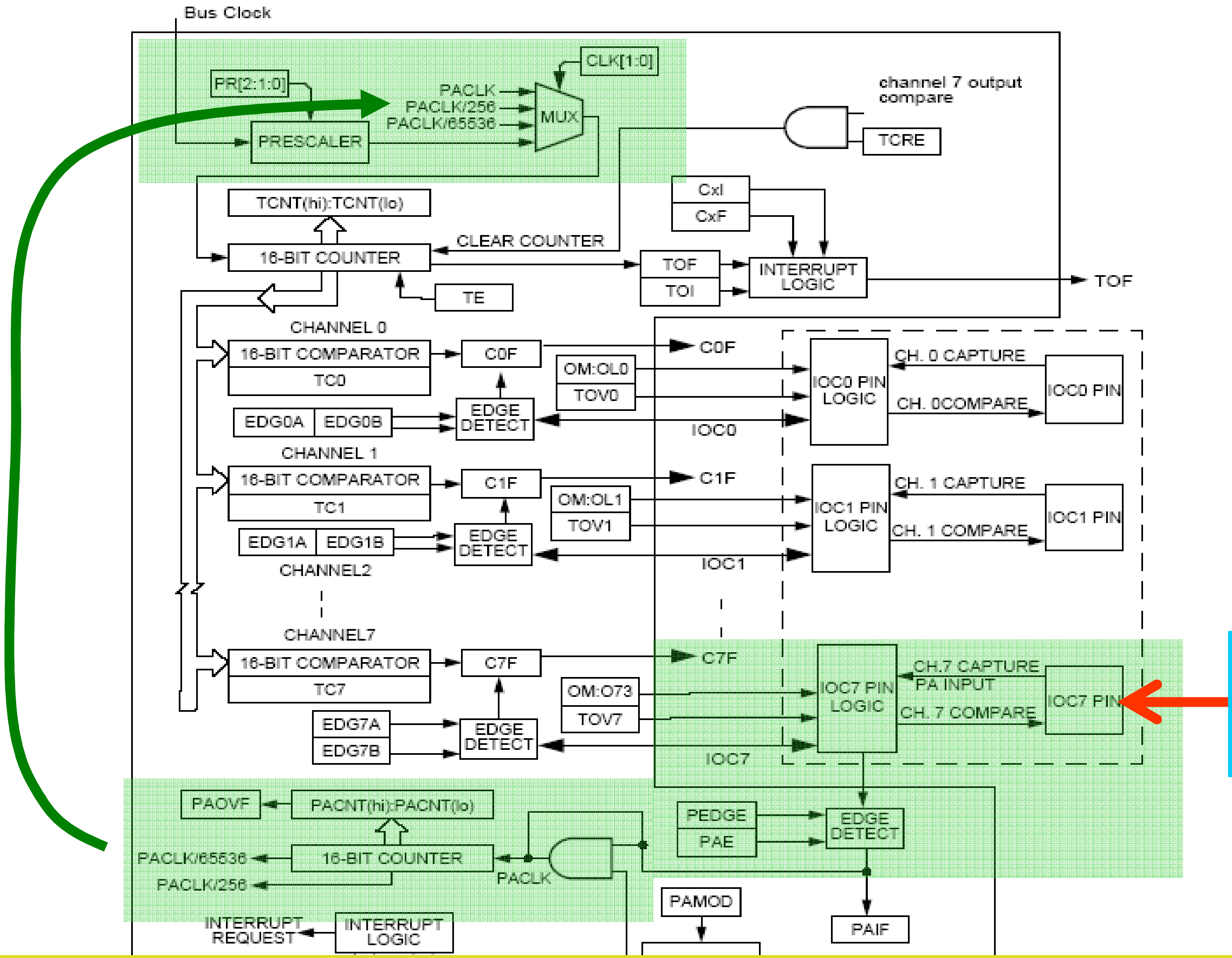
A **Channel 7** output compare is **"special"** because it can be enabled to reset TCNT when it occurs

TIM/PA Block Diagram Details



Another thing "special" about TIM Channel 7 is that when an Output Compare 7 occurs, a programmed bit pattern (in reg OC7D) can be output on selected (in reg OC7M) pins

TIM/PA Block Diagram Details



TIM clock source

Normally, the clock for TCNT is derived from the bus clock, but it can also be derived from the PA input pin

TIM Registers

- **TIOS (IC/OC select)**
  - **IOSx (bits 7-0)**
    - ***“0” – channel x acts as input capture***
    - **“1” – channel x acts as output compare**

🟩 ***indicates default mode after RESET***

TIM Registers

- TSCR1 (timer system control register 1)
  - TEN (bit 7) – timer enable
    - *“0” – timer subsystem disabled*
    - “1” – timer functions enabled
  - TFFCA (bit 4) – fast flag clear all
    - *“0” – normal flag clear mode*
    - “1” – enables fast flag clear mode (input capture read -or- output compare write *automatically clears* corresponding channel flag – helps reduce software overhead)

🟩 *indicates default mode after RESET*

TIM Registers

- **TSCR2 (timer system control register 2)**
  - **TOI (bit 7) – timer overflow interrupt enable (when TCNT “wraps around”)**
    - ***“0” – interrupt inhibited***
    - **“1” – interrupt generated when TOF set**
  - **TCRE (bit 3) – timer counter reset enable**
    - ***“0” – reset inhibited (free-running)***
    - **“1” – counter reset by successful output compare 7**

🟩 ***indicates default mode after RESET***

TIM Registers

- **TSCR2 (timer system control register 2)**
  - **PR2, PR1, PR0 (bits 2-0) – prescalar**
    - **“000” – *prescale factor: 1***
    - **“001” – prescale factor: 2**
    - **“010” – prescale factor: 4**
    - **“011” – prescale factor: 8**
    - **“100” – prescale factor: 16**
    - **“101” – prescale factor: 32**
    - **“110” – prescale factor: 64**
    - **“111” – prescale factor: 128**

🟩 ***indicates default mode after RESET***

TIM Registers

- TCTL1 (timer control register 1)
- TCTL2 (timer control register 2)
  - OMx – output mode
  - OLx – output level

    for output compare

    - *“00” – timer disconnected from pin*
    - “01” – toggle OCx output line
    - “10” – clear OCx output line to zero
    - “11” – set OCx output line to one

🟩 *indicates default mode after RESET*

TIM Registers

- **TCTL3 (timer control register 3)**
- **TCTL4 (timer control register 4)**
  - **EDGExB – input capture edge control**
  - **EDGExA – input capture edge control**
    - **“00” – *capture disabled***
    - **“01” – capture on rising edges only**
    - **“10” – capture on falling edges only**
    - **“11” – capture on rising and falling edges**

🟩 ***indicates default mode after RESET***

TIM Registers

- **TIE (timer interrupt enable)**
  - **CxI (bits 7-0) – input capture/output compare interrupt enable**
    - ***“0” – interrupt disabled***
    - **“1” – interrupt enabled**

🟩 ***indicates default mode after RESET***

TIM Registers

- **TFLG1 (timer interrupt flag 1)**
  - **CxF (bits 7-0)**
    - ***"0" – channel x flag not set***
    - **"1" – channel x flag set**

Flag CxF is *cleared* by writing a "1" to *bit x* of this register (unless the "fast flag clear" mode is used, in which case *just reading* (input capture) or *writing* (output compare) the channel register will *automatically clear* these flags

🟩 ***indicates default mode after RESET***

can be used to trigger the increment of the high byte/word of a time stamp

TIM Registers

- TFLG2 (timer interrupt flag 2)
  - TOF (bit 7) – timer overflow flag
    - *“0” – flag is not set*
    - “1” – flag is set

The TOF flag is *set* when the free-running timer *overflows* from $FFFF to $0000

Writing a “1” to *bit 7* of this register *clears* the TOF flag

🟩 *indicates default mode after RESET*

TIM Registers

- **CFORC (timer compare force)**
  - **FOCx (bits 7-0)**
    - ***"0" – no action***
    - **"1" – output compare occurs immediately, i.e., a write to this register with the corresponding bit(s) set causes the action programmed for that output compare to occur immediately (instead of at the "programmed time")**

🟩 ***indicates default mode after RESET***

TIM Registers

- **OC7M (output compare 7 mask)**
  - **OC7Mx (bits 7-0)**
    - ***"0" – Port T in default state***
    - **"1" – Port T pin set for output mode**
- **OC7D (output compare 7 data)**

When a successful *OC7 compare* occurs, for each bit set in OC7M, the *corresponding* bit in OC7D is *output* to Port T

🟩 ***indicates default mode after RESET***

TIM Registers

- **TTOV (timer toggle on overflow)**
  - **TOVx (bits 7-0)**
    - **“0” – *feature is disabled***
    - **“1” – in output compare mode, channel port pin toggles when TCNT overflows**

🟩 ***indicates default mode after RESET***

TIM Registers

- TC0 (IC/OC register 0)
- TC1 (IC/OC register 1)
- TC2 (IC/OC register 2)
- TC3 (IC/OC register 3)
- TC4 (IC/OC register 4)
- TC5 (IC/OC register 5)
- TC6 (IC/OC register 6)
- TC7 (IC/OC register 7)

All are 16-bit

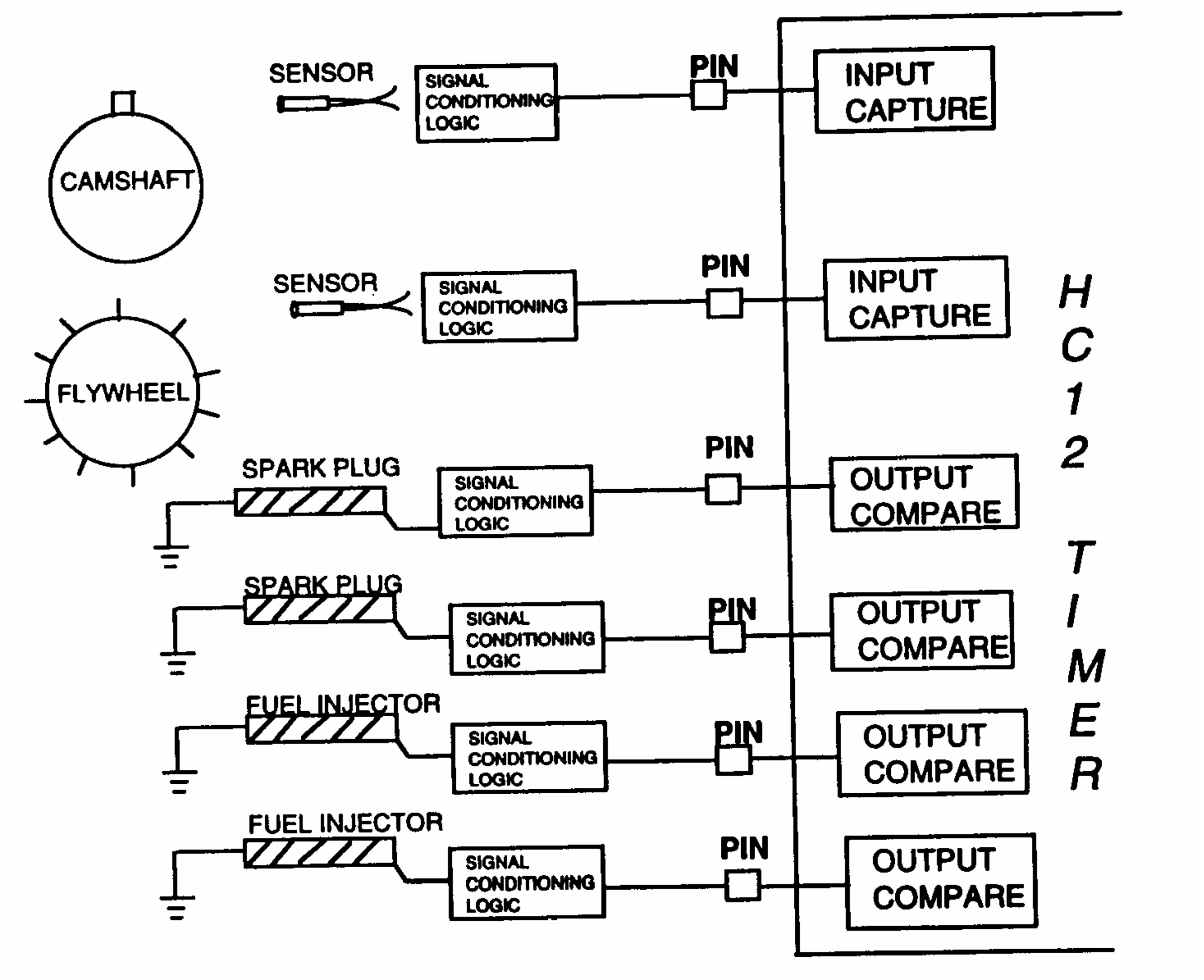
TIM Registers

- PORTT (timer data port)
- DDRT (data direction register)
  - “0” – port pin configured for input
  - “1” – port pin configured for output

Note: The timer forces the state of DDRT to be “output” for each pin associated with an *enabled* OUTPUT COMPARE function

🟩 *indicates default mode after RESET*

TIM Application – Car Engine Control



Clicker Quiz

1. An application for which the *input capture* mode of the TIM module could be used is:
   A. generating a precisely timed sequence of output pulses
   B. precisely measuring the time interval between two input pulses
   C. counting the number of input pulses
   D. all of the above
   E. none of the above

2. An application for which the *output compare* mode of the TIM module could be used is:
   A. generating a precisely timed sequence of output pulses
   B. precisely measuring the time interval between two input pulses
   C. counting the number of input pulses
   D. all of the above
   E. none of the above

3. **The TIM module can potentially provide a more accurate time base than the RTI module because:**

   A. **the TIM module uses a more accurate system clock than the RTI module**

   B. **the TIM module interrupt rates are based on powers-of-two divisions of the system clock, while the RTI allows arbitrary divisions of the system clock**

   C. **the RTI interrupt rates are based on powers-of-two divisions of the system clock, while the TIM module allows arbitrary divisions of the system clock**

   D. **hidden behind the RTI is a Micruhsoft Winduhs Vister operating system, notorious for taking as much time as it pleases to do anything useful and socially redeeming**

   E. **none of the above**

TIM Application – Digital Clock

- Basic idea: Create a highly accurate digital clock using the TIM module as a ***time base***
  - use prescaler of 16 to divide 24 MHz bus clock down to 1.5 MHz (will cause TCNT register to increment every *0.667 μs*)
  - use Timer Channel 7 so that the TCNT register can be ***automatically reset*** when the Output Compare occurs
  - set TC7 Output Compare register to 15,000 → interrupt will occur every 10 ms
  - call “clock” routine when 100 of these interrupts accumulate

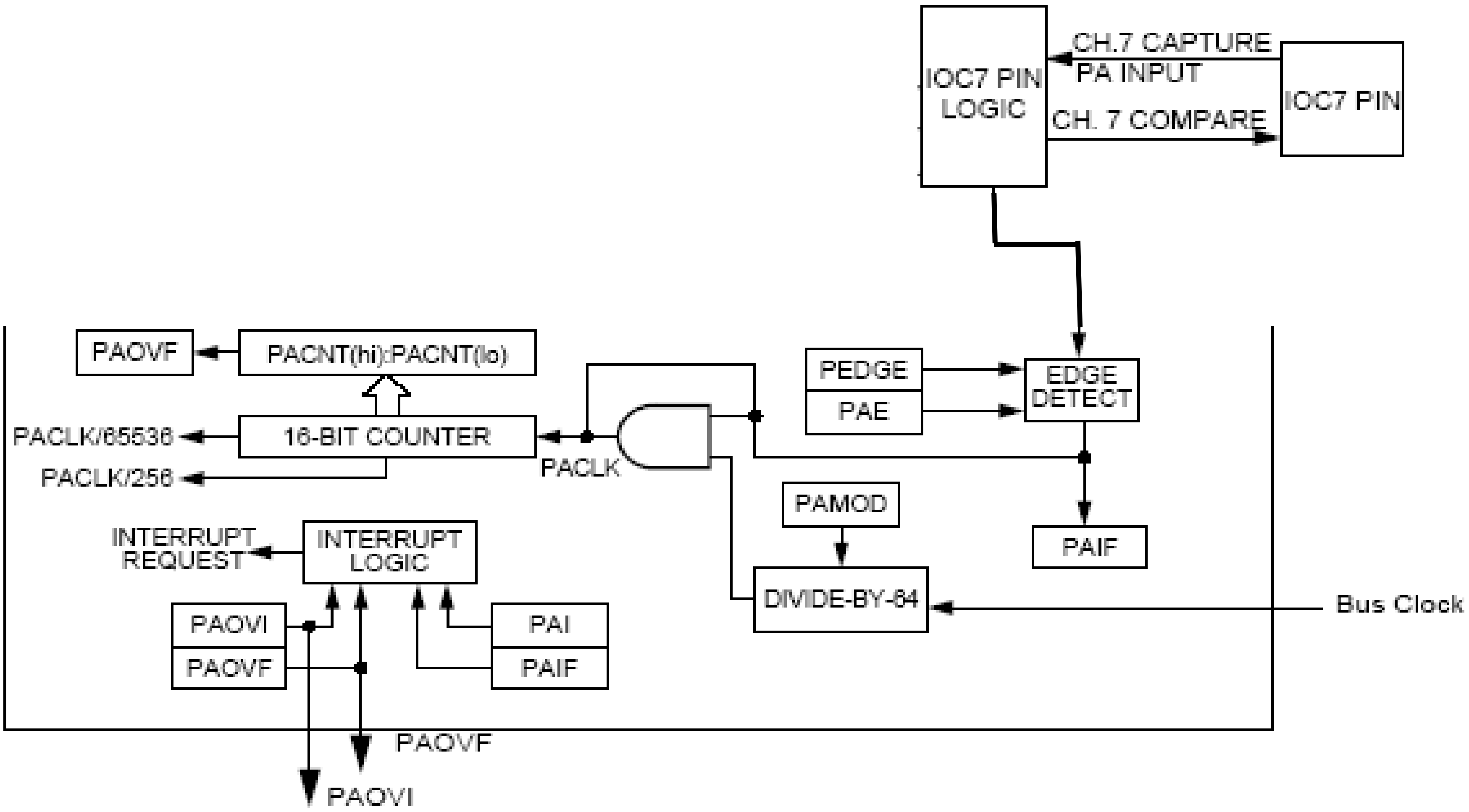
TIM Application – Digital Clock

```
; Initializes TIM for Digital Clock application
; STEP (1) Enable TIM subsystem
tim_ini    movb    #$80,tscr1
; STEP (2) Set prescale factor to 16, and enable
;                  counter reset after OC7
           movb    #$0C,tscr2
; STEP (3) Set Ch 7 for Output Compare
           movb    #$80,tios
; STEP (4) Enable Ch 7 interrupt
           movb    #$80,tie
; STEP (5) Set up Ch 7 to generate 10 ms interrupt rate
           movw    #15000,tch7
; STEP (6) Enable system IRQ interrupts
           cli
           rts
```

PA Features and Functions

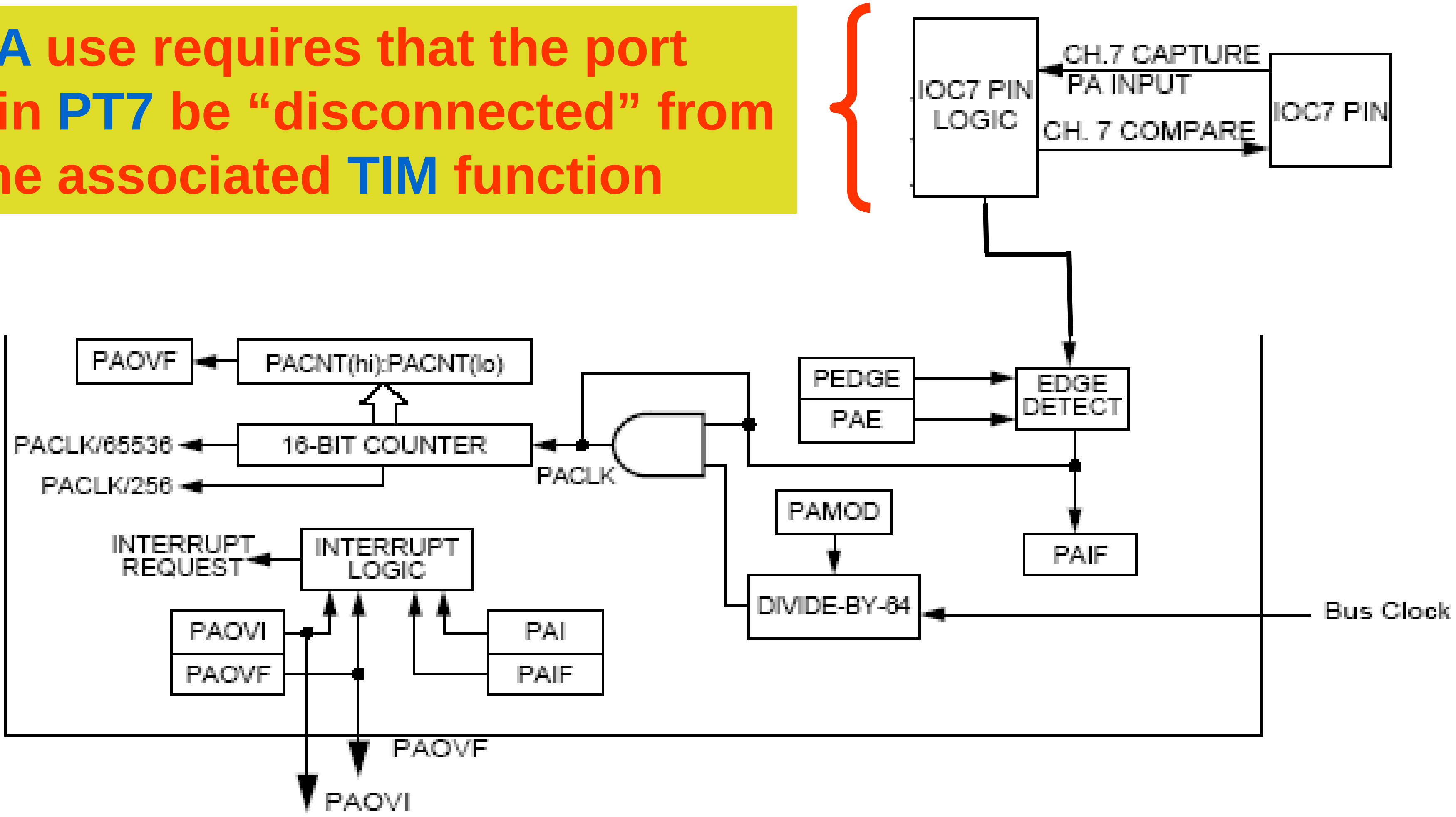
- The (single) pulse accumulator (PA) channel shares a port pin with **TIM Channel 7**
- The PA and the TIM Ch 7 output compare functions can be used ***simultaneously*** by **“disconnecting”** the TIM pad logic from the port pin)
- There are two basic PA modes:
  - ***event counting*** (programmed transition on PT7 increments the 16-bit PACNT register)
  - ***gated accumulation*** (PACNT incremented by scaled bus clock, assertion level on PT7 enables/disables counter)

PA Module Block Diagram



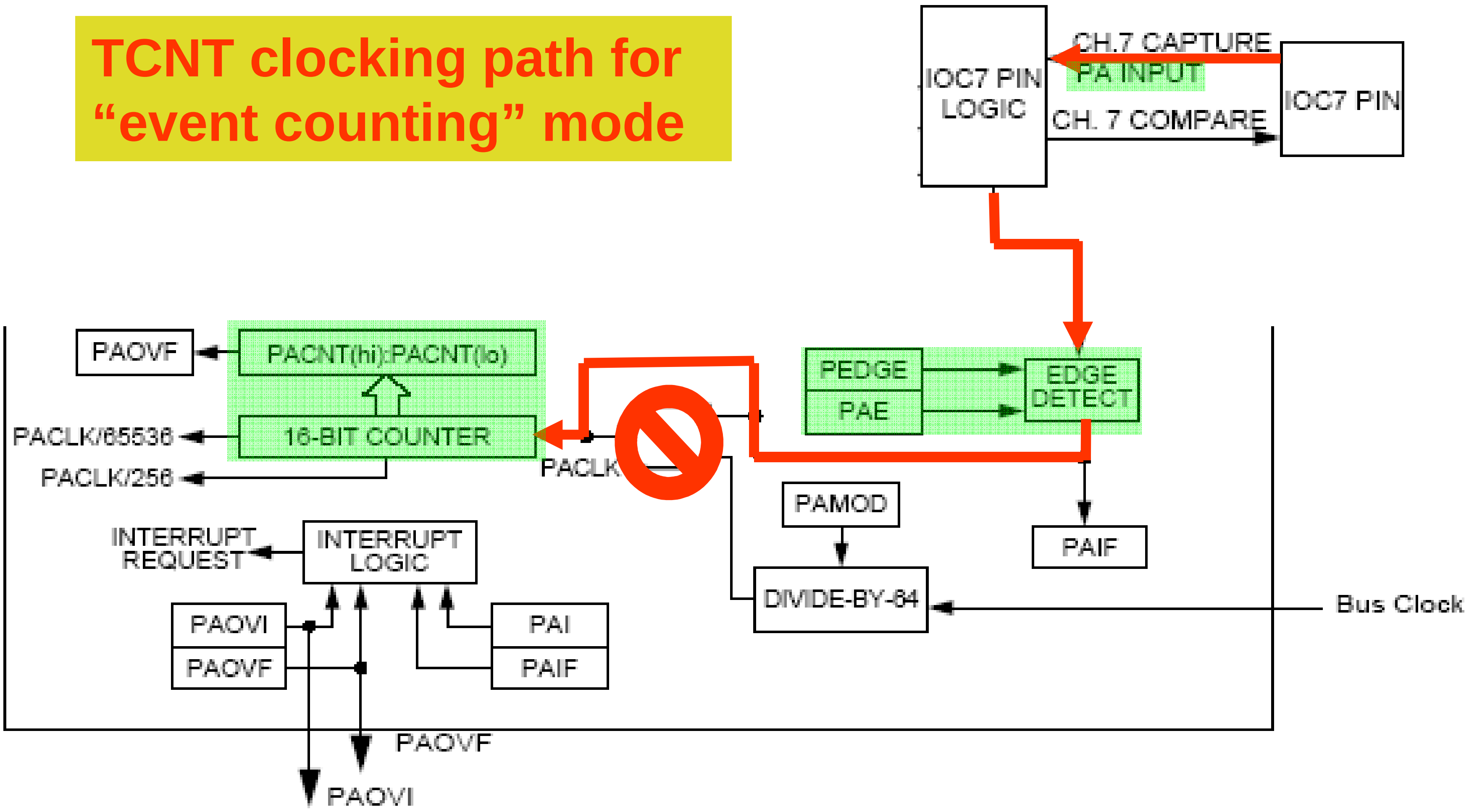
PA Module Block Diagram

PA use requires that the port pin PT7 be “disconnected” from the associated TIM function



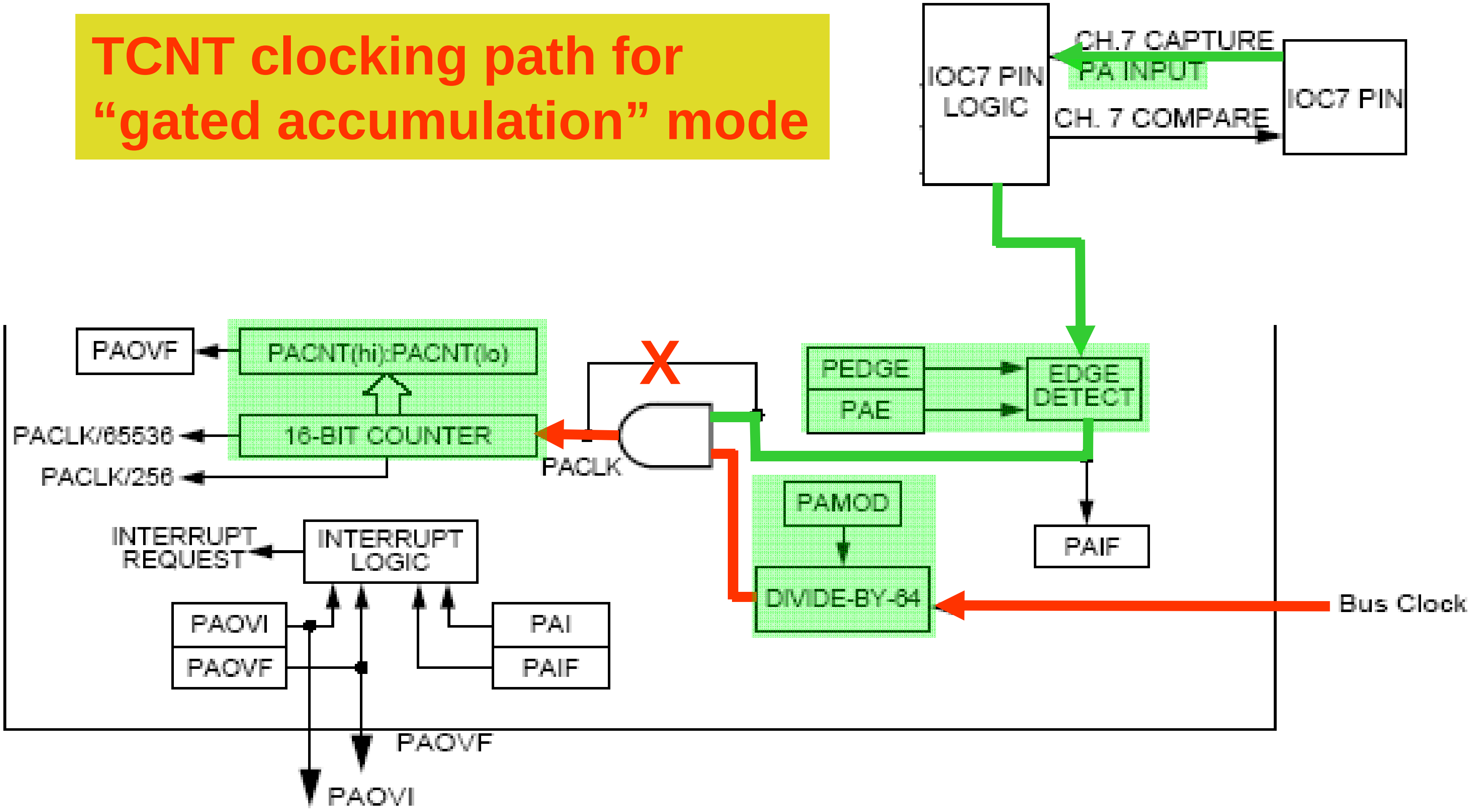
PA Module Block Diagram

TCNT clocking path for “event counting” mode

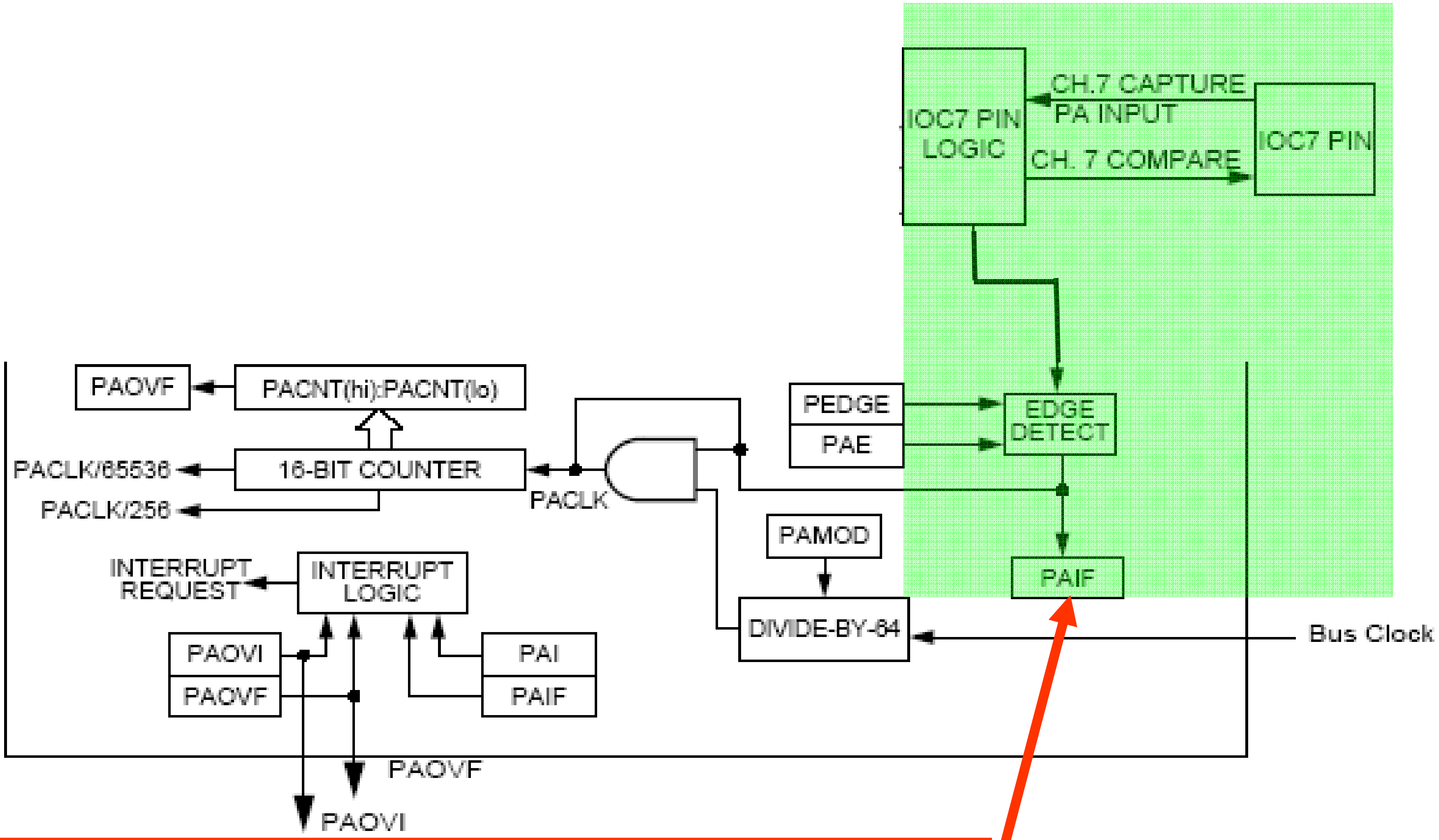


PA Module Block Diagram

TCNT clocking path for “gated accumulation” mode

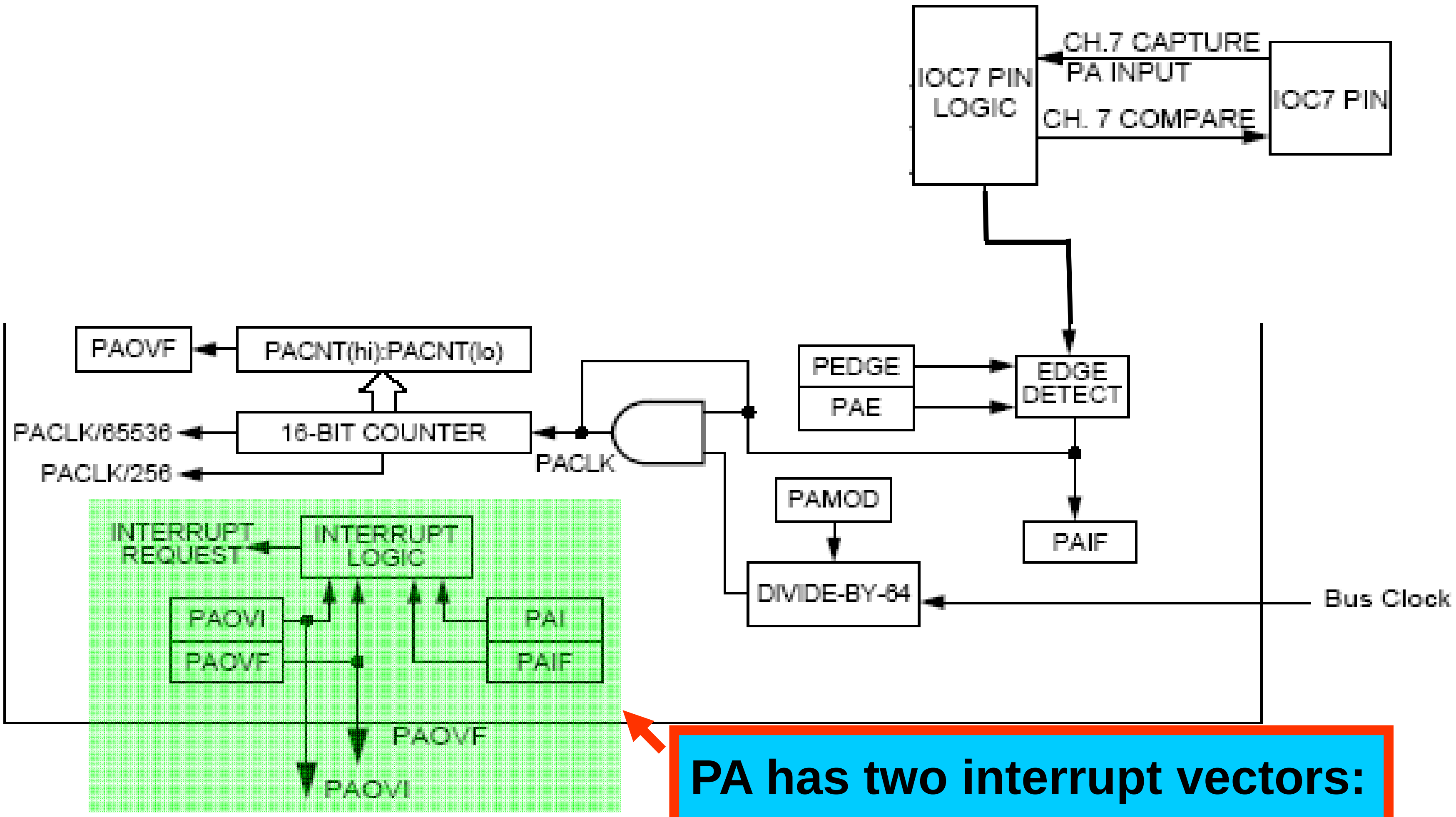


PA Module Block Diagram



PA can simply be used as an edge-sensitive interrupt device flag if desired

PA Module Block Diagram



PA has two interrupt vectors: input interrupt (PAI) and overflow interrupt (PAOVI)

PA Registers

- PACTL (pulse accumulator control)
  - PAEN (bit 6) – pulse accumulator system enable
    - “0” – *pulse accumulator system disabled*
    - “1” – pulse accumulator system enabled
  - PAMOD (bit 5) – pulse accum mode
    - “0” – *event counter mode*
    - “1” – gated time accumulation mode

🟩 *indicates default mode after RESET*

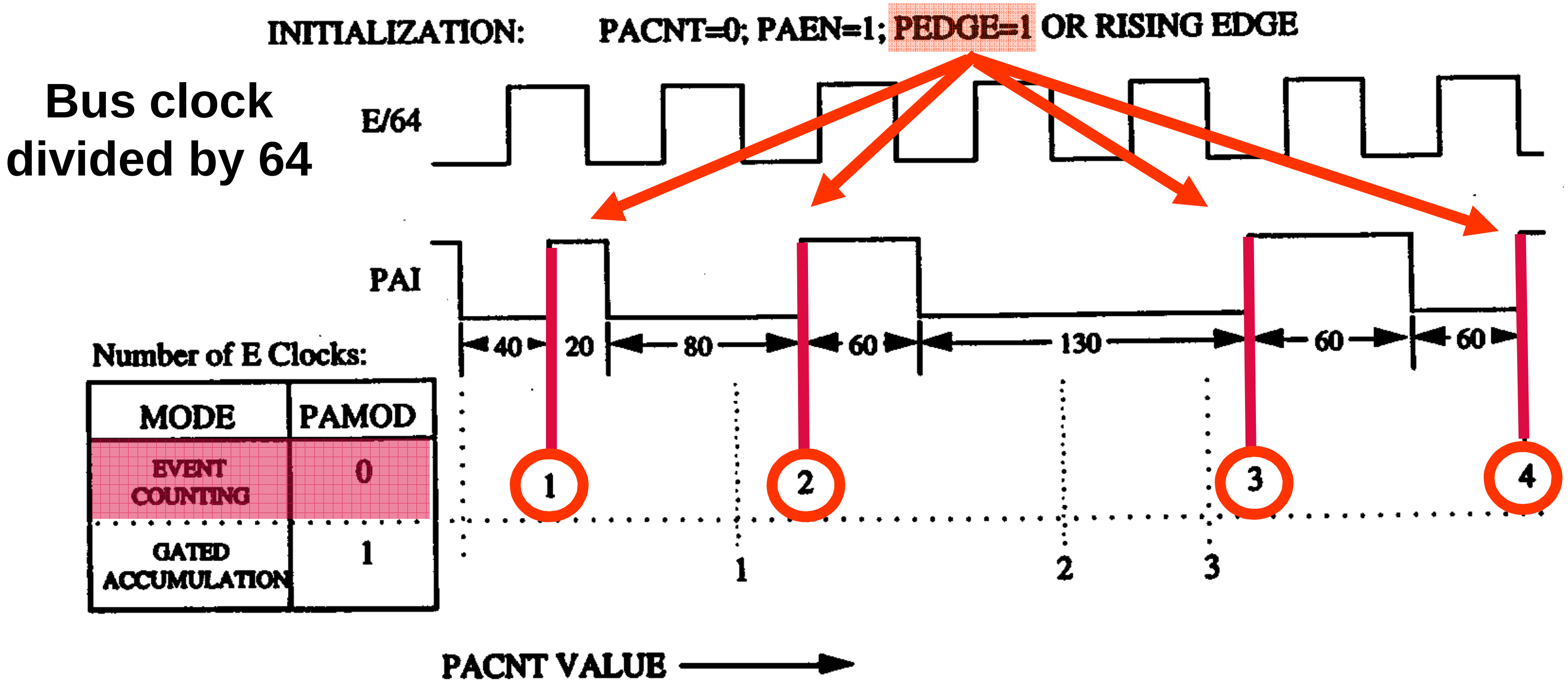
PA Registers

- **PACTL (pulse accumulator control)**
  - **PEDGE (bit 4) – edge control (event)**
    - ***"0" – count on falling edges***
    - **"1" – count on rising edges**
  - **PEDGE (bit 4) – edge control (gated)**
    - ***"0" – input pin high enables accumulation***
    - **"1" – input pin low enables accumulation**

🟩 ***indicates default mode after RESET***

Pulse Accumulator Modes

Here, PEDGE = 1 means that PACNT increments every time a rising edge on pin PT7 is detected

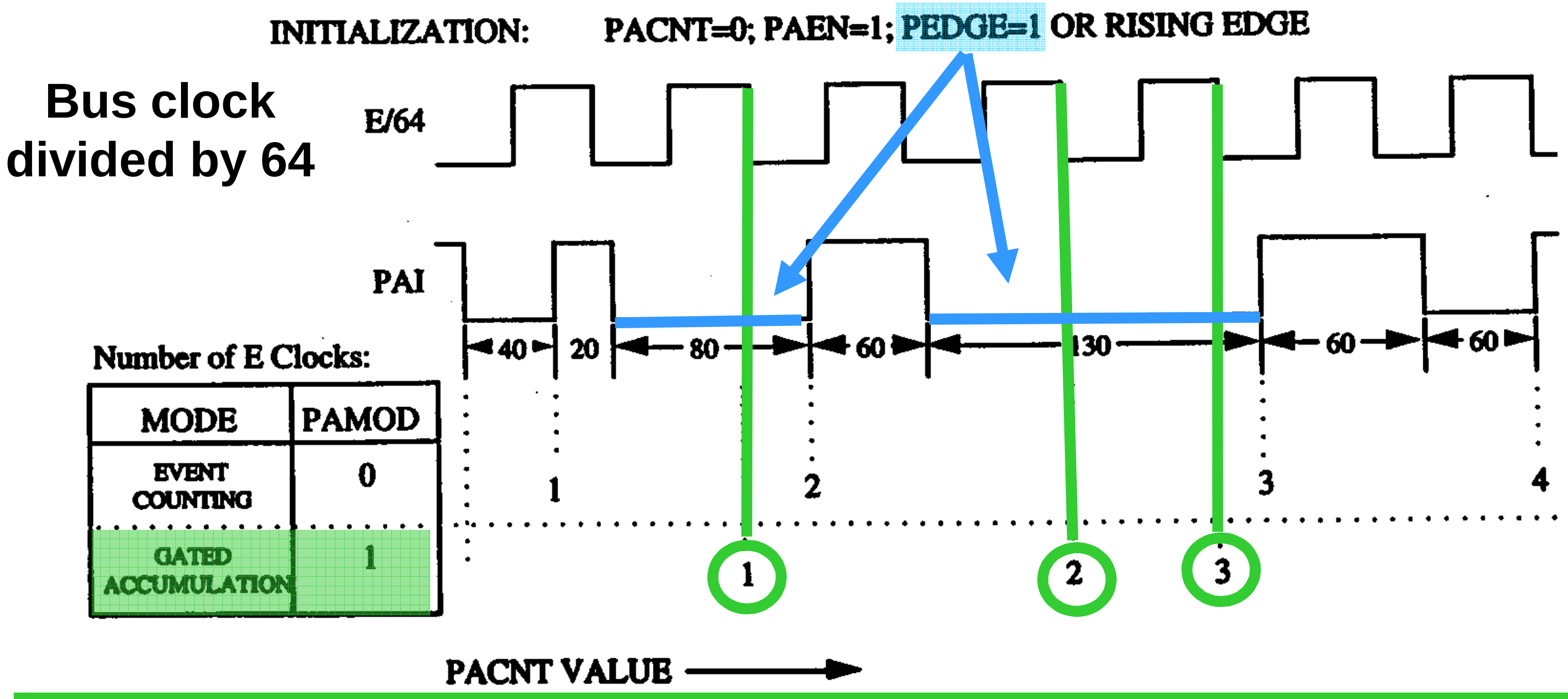


MODE	PAMOD
EVENT COUNTING	0
GATED ACCUMULATION	1

PAMOD = 0 selects “event counting” mode

Pulse Accumulator Modes

Here, PEDGE = 1 means that the accumulation is enabled on an active low assertion of pin PT7



MODE	PAMOD
EVENT COUNTING	0
GATED ACCUMULATION	1

PAMOD = 1 selects "gated accumulation" mode

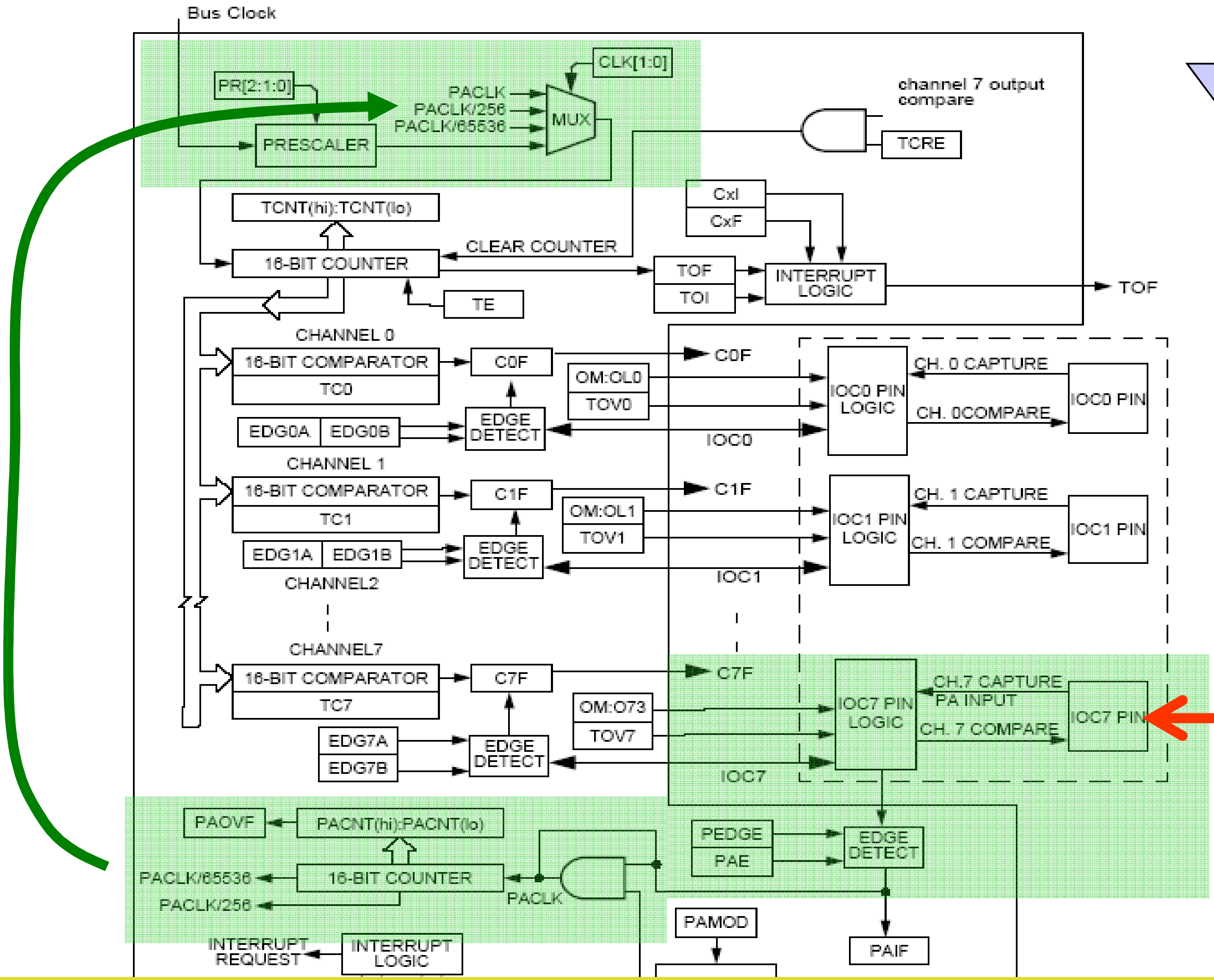
Note that PACNT always increments on the negative edge of the scaled bus clock

PA Registers

- PACTL (pulse accumulator control)
  - CLK1, CLK0 (bits 3,2) – clock select
    - *“00” – use timer prescalar clock as timer counter clock*
    - “01” – use PACLK as input to timer counter clock
    - “10” – use PACLK/256 as timer clock freq
    - “11” – use PACLK/65536 as timer clock frequency

🟩 *indicates default mode after RESET*

TIM/PA Block Diagram Details



TIM clock source

Normally, the clock for TCNT is derived from the bus clock, but it can also be derived from the PA input pin

PA Registers

- **PACTL (pulse accumulator control)**
  - **PAOVI (bit 1) – pulse accumulator overflow interrupt enable**
    - ***“0” – interrupt inhibited***
    - **“1” – interrupt enabled**
  - **PAI (bit 0) – pulse accumulator interrupt enable**
    - ***“0” – interrupt inhibited***
    - **“1” – interrupt enabled**

🟩 ***indicates default mode after RESET***

PA Registers

- **PAFLG (pulse accumulator flag)**
  - **PAOVF (bit 1) – pulse accumulator overflow flag**
    - **“0” – *no overflow has occurred***
    - **“1” – pulse accumulator has overflowed**

Writing a “1” to *bit 1* of this register *clears* the PAOVF flag

🟩 ***indicates default mode after RESET***

PA Registers

- **PAFLG (pulse accumulator flag)**
  - **PAIF (bit 0) – pulse accumulator interrupt flag**
    - **“0” – *no edge detected on PA input***
    - **“1” – edge detected on PA input**

Writing a “1” to *bit 0* of this register *clears* the PAIF flag

🟩 ***indicates default mode after RESET***

Clicker Quiz

1. An application for which the *pulse accumulator* mode of the TIM module could be used is:
   A. generating a precisely timed sequence of output pulses
   B. precisely measuring the time interval between two input pulses
   C. counting the number of input pulses
   D. all of the above
   E. none of the above

2. The TIM Ch 7 periodic interrupt generation capability and the pulse accumulator (PA) can be used simultaneously by:

   A. disconnecting the TIM Ch 7 output from the port pin

   B. re-routing the TIM Ch 7 output to a different port pin

   C. continuously changing the mode of the port pin

   D. re-routing the PA input to a different port pin

   E. none of the above

PA Application – Digital Tachometer

- **Basic idea: Estimate motor RPM based on number of pulses accumulated from a slotted disk over a specified integration period**
  - **PA used to accumulate pulses in *"event counting"* mode**
  - **either RTI or TIM module used to determine *integration period* for estimate**
  - **pulse count converted to BCD number in software**

Thought question: Why *doesn't it matter* which one (RTI or TIM) is used as the *time base* for the *RPM estimate*?

PA Application – Digital Tachometer

- Basic idea: Estimate motor RPM based on number of pulses accumulated from a slotted disk over a specified integration period
  - PA used to accumulate pulses in ***"event counting"*** mode
  - either RTI or TIM module used to determine ***integration period*** for estimate
  - pulse count converted to BCD number in software

Thought question: Why *doesn't it matter* which one (RTI or TIM) is used as the *time base* for the *RPM estimate*? *Time-base error doesn't accumulate (unlike clock)*

Not-So-Quick Clicker Quiz

```
;*************************************************************************
; In this exercise we will investigate using the TIM module as
;    a time base, and compare it to use of the RTI subsystem.
;*************************************************************************
;
; The objective of this problem is to implement stopwatch that can count
; in increments of 0.1 second up to 999.9 seconds.
; Here, both the RTI and TIM modules will be used as time bases, with the
; opportunity to compare the results obtained using each approach.
;
; The following docking board resources will be used:
; - left pushbutton (PAD7): stopwatch reset
; - right pushbutton (PAD6): stopwatch start/stop
; - left LED (PT1): stopwatch run/stop state
; - right LED (PT0): stopwatch maxed out (999.9) state
;
; The four-digit stopwatch value (NNN.N) is to be updated on the terminal
; screen every one-tenth second (display both RTI and TIM values).
;
;  1. Calculate the expected difference between the two stopwatches and
;     compare that with the actual values obtained
;
;  2. See if you can make the RTI-based stopwatch "more accurate" by
;     changing the interrupt rate (and modifying maximum RTICNT) while
      still successfully de-bouncing the pushbuttons
```

```
; ======================================================================
;
;  Variable declarations (SRAM)
;

   org  $3800
RFLG    rmb     1       ; RTI one-tenth second flag (1 -> set, 0 -> clear)
TFLG    rmb     1       ; TIM one-tenth second flag (1 -> set, 0 -> clear)
LEFTPB  rmb     1       ; left pushbutton flag (1 -> set, 0 -> clear)
RGHTPB  rmb     1       ; right pushbutton flag (1 -> set, 0 -> clear)
RUNSTP  rmb     1       ; run/stop flag (1 -> run, 0 -> stop)
rticnt  rmb     1       ; RTICNT (variable)
timcnt  rmb     1       ; TIMCNT (variable)
prevpb  rmb     1       ; previous state of pushbuttons (variable)
rtime   rmb     2       ; RTI stopwatch time NNN.N (variable)
ttime   rmb     2       ; TIM stopwatch time NNN.N (variable)
```

```
;************************************************************************
;
; RTI and TIM initializations given in the class notes
;
; set 8.192 ms RTI interrupt rate
   movb #$70,rtictl

; enable RTI interrupts
   bset crgint,$80

; enable TIM subsystem
   movb    #$80,tscr1

; set TIM prescale factor to 16, and enable counter reset after OC7
   movb    #$0C,tscr2

; set TIM Ch 7 for Output Compare
   movb    #$80,tios

; enable TIM Ch 7 interrupt
   movb    #$80,tie

; set TIM Ch 7 to generate 10.0 ms interrupt rate
   movw    #15000,tc7

; enable IRQ interrupts
   cli
```

```
;  If the "run/stop" flag is set, then
;    - If the “RFLG" flag is set, then
;      + clear the “RFLG" flag
;      + increment the RTI stopwatch value by one tenth (in BCD)
;    - Endif
;    - If the “TFLG" flag is set, then
;      + clear the “TFLG" flag
;      + increment the TIM stopwatch value by one tenth (in BCD)
;      + update the display
;    - Endif
;  Endif

main1

        brclr   RUNSTP,$01,main2

        brclr   RFLG,$01,main12
        clr     RFLG
        ldaa    rtime+1
        adda    #1
        daa
        staa    rtime+1
        ldaa    rtime
        adca    #0
        daa
        staa    rtime
```

```
;  If the "run/stop" flag is set, then
;    - If the "RFLG" flag is set, then
;      + clear the “RFLG" flag
;      + increment the RTI stopwatch value by one tenth (in BCD)
;    - Endif
;    - If the “TFLG" flag is set, then
;      + clear the “TFLG" flag
;      + increment the TIM stopwatch value by one tenth (in BCD)
;      + update the display
;    - Endif
;  Endif


main12
        brclr   TFLG,$01,main2
        clr     TFLG
        ldaa    ttime+1
        adda    #1
        daa
        staa    ttime+1
        ldaa    ttime
        adca    #0
        daa
        staa    ttime

        jsr     rtdisp
```

```
main2

;  If the left pushbutton ("reset stopwatch") flag is set, then:
;    - clear the left pushbutton flag
;    - clear the "run/stop" flag
;    - turn off both the "run/stop" and "maxed out" LEDs
;    - reset the terminal display to "000.0"
;  Endif

        brclr   LEFTPB,$01,main3
        clr     LEFTPB
        clr     RUNSTP
        clr     PTT
        clr     rtime
        clr     rtime+1
        clr     ttime
        clr     ttime+1
        jsr     rtdisp
```

```
main3

;  If the right pushbutton ("start/stop") flag is set, then
;    - clear the right pushbutton flag
;    - toggle the "run/stop" flag
;    - toggle the "run/stop" LED
;  Endif

        brclr   RGHTPB,$01,main4
        clr     RGHTPB
        ldaa    RUNSTP
        eora    #$01
        staa    RUNSTP
        ldaa    PTT
        eora    #$02
        staa    PTT
```

```
main4

;  If the TIM stopwatch has reached "999.9", then:
;    - clear the "run/stop" flag
;    - turn on the "time expired" LED
;    - turn off the "run/stop" LED
;  Endif

        ldaa    ttime
        cmpa    #$99
        bne     main41
        ldaa    ttime+1
        cmpa    #$99
        bne     main41
        clr     RUNSTP
        movb    #$01,PTT
main41
        jmp     main1   ; continue looping
```

```
;***********************************************************************
;
; TIM interrupt service routine
;
; Interrupt generated every 10 ms on TC7
;
; Use variable TIMCNT to count up to 10 of these
; and set TFLG when that happens
;
; NOTE: If the "runstp" flag is clear (stopwatch is "stopped")
;       this routine should DO NOTHING other than clear
;       the TC7 interrupt flag  WHY?

tim_isr
      bset      TFLG1,$80       ; clear TC7 interrupt flag
      brclr     RUNSTP,$01,tdone
      ldaa      timcnt
      inca
      staa      timcnt
      cmpa      #10             ; count to 100 ms (10 x 10ms)
      blt       tdone
      clr       timcnt
      movb      #$01,TFLG       ; set tenth-of-second flag
tdone
      rti
```

```
;***********************************************************************
; RTI interrupt service routine
;
;  Keeps track of when 0.1 second of RTI interrupts has accumulated
;    and sets RFLG
;
;  Also, samples state of pushbuttons (PTAD7 = left, PTAD6 = right)
;  If change in state from "high" to "low" detected, set pushbutton flag
;     LEFTPB (for PTAD7 H -> L), RGHTPB (for PTAD6 H -> L)
;     Recall that pushbuttons are momentary contact closures to ground

rti_isr

;  Using RTICNT, track when 0.1 second of RTI interrupts has accumulated
;  Set the “RFLG" flag when this occurs and clear RTICNT

        bset    crgflg,$80
        brclr   RUNSTP,$01,chkpb

        inc     rticnt
        ldaa    rticnt
        cmpa    #12     ; based on initial 8.192 ms interrupt rate
        blt     chkpb
        clr     rticnt
        movb    #$01,RFLG
```

```
;  Check the pushbuttons for a change of state (compare previous with current)
;  If no change detected, exit service routine

chkpb
        ldaa     PTAD     ; read current state of pushbuttons
        anda     #$C0     ; mask off PTAD7 and PTAD6
        cmpa     prevpb   ; compare with previous PB state
        bne      didchg   ; if current state != previous state
        rti               ; figure out which pushbutton involved
                          ; else, exit with no change

;  State of PB changed -- check if action necessary
;  Note: Not considering case when both PBs pressed simultaneously

didchg
        psha              ; save current state of PB

;  If previous state of left PB was "H" and current state is "L"
;  then set “LEFTPB" flag and proceed to exit; else, check right PB

chklft
        anda     #$80     ; isolate current left PB
        eora     #$80     ; complement current left PB
        psha
        ldab     prevpb
        andb     #$80     ; isolate previous left PB
        andb     1,sp+
        beq      chkrgt   ; if AND=0, left PB did not change from H->L so exit
                          ; else, set LEFTPB flag
        movb     #$01,LEFTPB
        bra     rdone
```

```
;  If previous state of right PB was "H" and current state is "L"
;  then set "rghtpb" flag and proceed to exit

chkrgt
        ldaa     0,sp     ; reload current PB state
        anda     #$40     ; isolate current right PB
        eora     #$40     ; complement current right PB
        psha
        ldab     prevpb
        andb     #$40     ; isolate previous right PB
        andb     1,sp+
        beq      rdone    ; if AND=0, right PB did not change from H->L so exit
                          ; else, set RGHTPB flag
        movb     #$01,RGHTPB


;  Since PB state changed, set PREVPB = current state read

rdone
        pula
        staa     prevpb    ; update PB state

        rti
```

Q1. A practical lower limit on the RTI interrupt rate to ensure successful de-bouncing of the pushbuttons is on the order of:

A. 0.01 ms

B. 0.1 ms

C. 1.0 ms

D. 10 ms

E. none of the above

Q2. The reason the RUNSTP flag needs to be checked in both the RTI and TIM interrupt service routines is because:

A. **there is no other way to start/stop the two stopwatches simultaneously**

B. **timing error would otherwise be introduced when the stopwatches are started/stopped**

C. **a race condition might otherwise occur if the RTI and TIM interrupts fire simultaneously**

D. **the TIM and RTI need to remain pending**

E. **none of the above**

Q3. When the stopwatch runs to completion (TTIME = 999.9 secs), the difference between RTIME and TTIME will be approximately:

A. 0.17 sec

B. 1.7 secs

C. 16.9 secs

D. 17.2 secs

E. none of the above

```
  set 8.192 ms RTI interrupt rate

       movb     #$70,rtictl     ; set 8.192 ms RTI interrupt rate

;**********************************************************************
rti_isr

;  Using RTICNT, track when one-tenth second of RTI interrupts accumulated
;  Set the “RFLG" flag when this occurs and clear RTICNT

       bset     crgflg,$80
       brclr    runstp,$01,chkpb
       inc      rticnt
       ldaa     rticnt
       cmpa     #12        ; 12 x 8.192 = 98.304 ms
       blt      chkpb
       clr      rticnt
       movb     #$01,rflg
```

RTIME runs “fast” (“1 sec” = 0.98304 sec actual)

TTIME / 0.98304 = RTIME (off by 17.2 after 1000 secs)

Q4. If the RTI interrupt rate is changed to 1.024 ms and the RTICNT comparison value is adjusted appropriately, the difference between RTIME and TTIME can be reduced to approximately ________ when the stopwatch runs to completion.

A. 0.35 sec
B. 1.7 secs
C. 3.5 secs
D. 7.2 secs
E. none of the above

```
; set 1.024 ms RTI interrupt rate

       movb     #$40,rtictl     ; set 1.024 ms RTI interrupt rate

;**********************************************************************
rti_isr

;  Using RTICNT, track when one-tenth second of RTI interrupts accumulated
;  Set the "rflg" flag when this occurs and clear RTICNT

       bset     crgflg,$80
       brclr    runstp,$01,chkpb
       inc      rticnt
       ldaa     rticnt
       cmpa     #98      ; 98 x 1.024 = 100.352 ms
       blt      chkpb
       clr      rticnt
       movb     #$01,rflg
```

RTIME runs “slow” (“1 sec” = 1.00352 sec actual)

TTIME / 1.00352 = RTIME (off by 3.5 after 1000 secs)

Q5. Through careful choice of RTI interrupt rate, the difference between RTIME and TTIME can be reduced to approximately ________ when the stopwatch runs to completion while still successfully de-bouncing the pushbuttons.

A. 0.03 sec

B. 0.3 sec

C. 1.5 secs

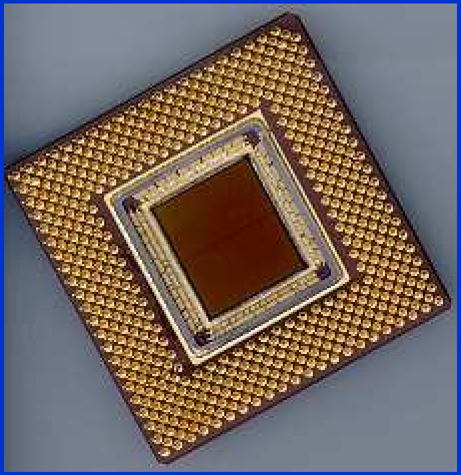
D. 3.0 secs

E. none of the above

```
; set 1.408 ms RTI interrupt rate

       movb     #$1A,rtictl     ; set 1.408 ms RTI interrupt rate *****

;**********************************************************************
rti_isr

;  Using RTICNT, track when one-tenth second of RTI interrupts accumulated
;  Set the "rflg" flag when this occurs and clear RTICNT

       bset     crgflg,$80
       brclr    runstp,$01,chkpb
       inc      rticnt
       ldaa     rticnt
       cmpa     #71      ; 71 x 1.408 = 99.968 ms
       blt      chkpb
       clr      rticnt
       movb     #$01,rflg
```

RTIME runs “fast” (“1 sec” = 0.99968 sec actual)

TTIME / 0.99968 = RTIME (off by 0.3 after 1000 secs)



Microcontroller-Based Digital System Design

Module 3-E
Pulse Width Modulation (PWM)

Learning Objectives

- define pulse width modulation (PWM) and describe potential applications
- define natural sampling and contrast it with pulse-code modulation (PCM)
- measure and compare the effects of different input and output sampling frequencies on signal integrity in an ATD-PWM digitization loop
- describe how a PWM channel can be used as a digital-to-analog (DTA) converter
- identify PWM features and operating modes
- configure the PWM to operate in a prescribed mode

Outline

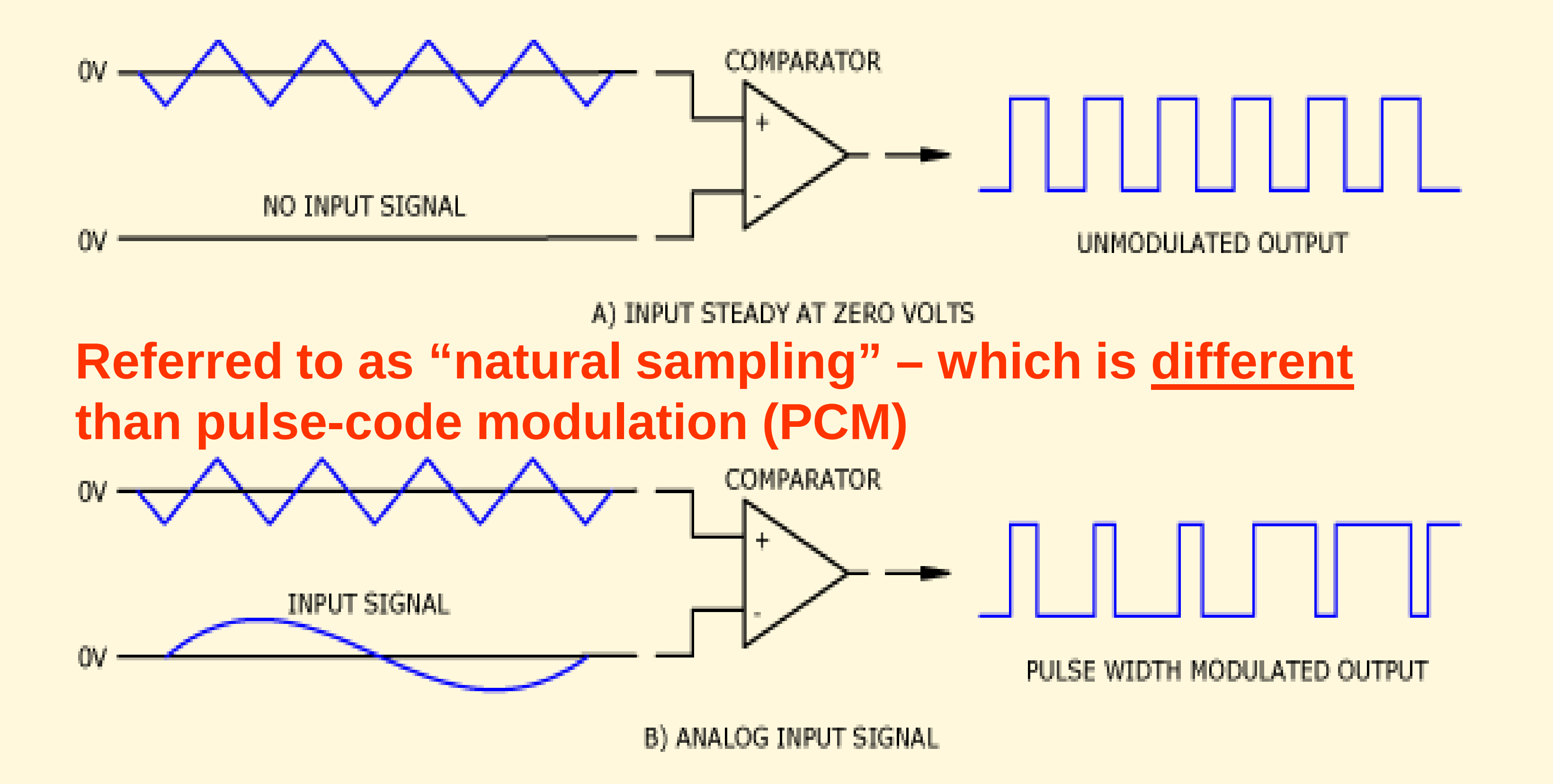
- Introduction
- PWM features
- PWM registers
- PWM initialization

Reference: **Pulse Width Modulation (PWM) Block User Guide**

Introduction

- Definition: The pulse width modulation unit provides the capability of producing a square wave output of varying *duty cycle* and *sampling frequency*
  - 0% duty cycle – *always off*
  - 50% duty cycle – *symmetric square wave*
  - 100% duty cycle – *always on*
- Where used
  - D.C. motor speed control
  - LED backlight intensity control
  - digital-to-analog conversion (in conjunction with low-pass filter)

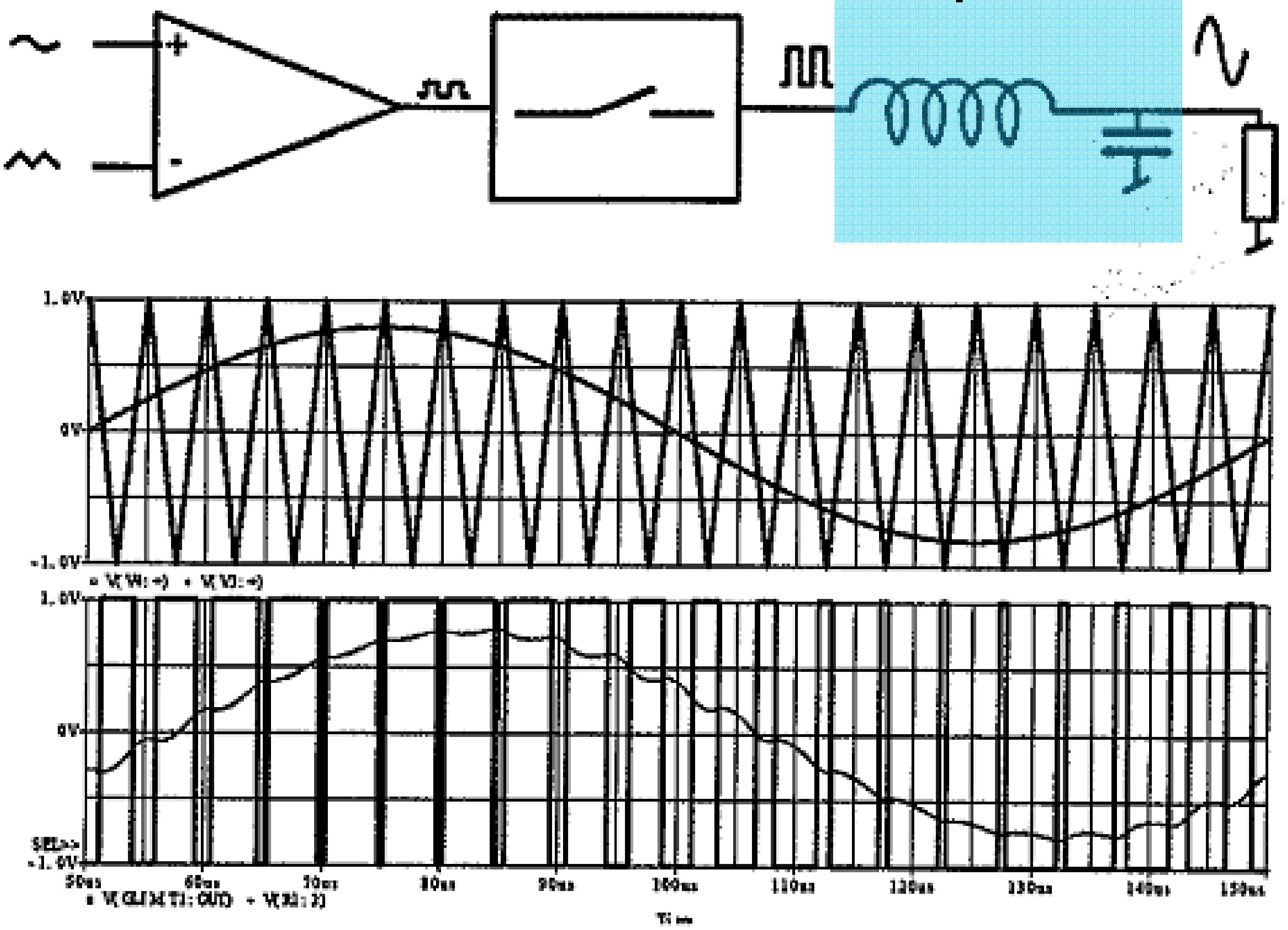
PWM Encoding of an Analog Input Signal



Referred to as “natural sampling” – which is <u>different</u> than pulse-code modulation (PCM)

Can be thought of as a one-bit A/D encoding system. The sampling frequency must be at least an order of magnitude higher than the highest frequency component of the input signal. The PWM output can be low-pass filtered to re-construct the analog signal.

PWM Encoding (Analog Modulator) and Signal Reconstruction

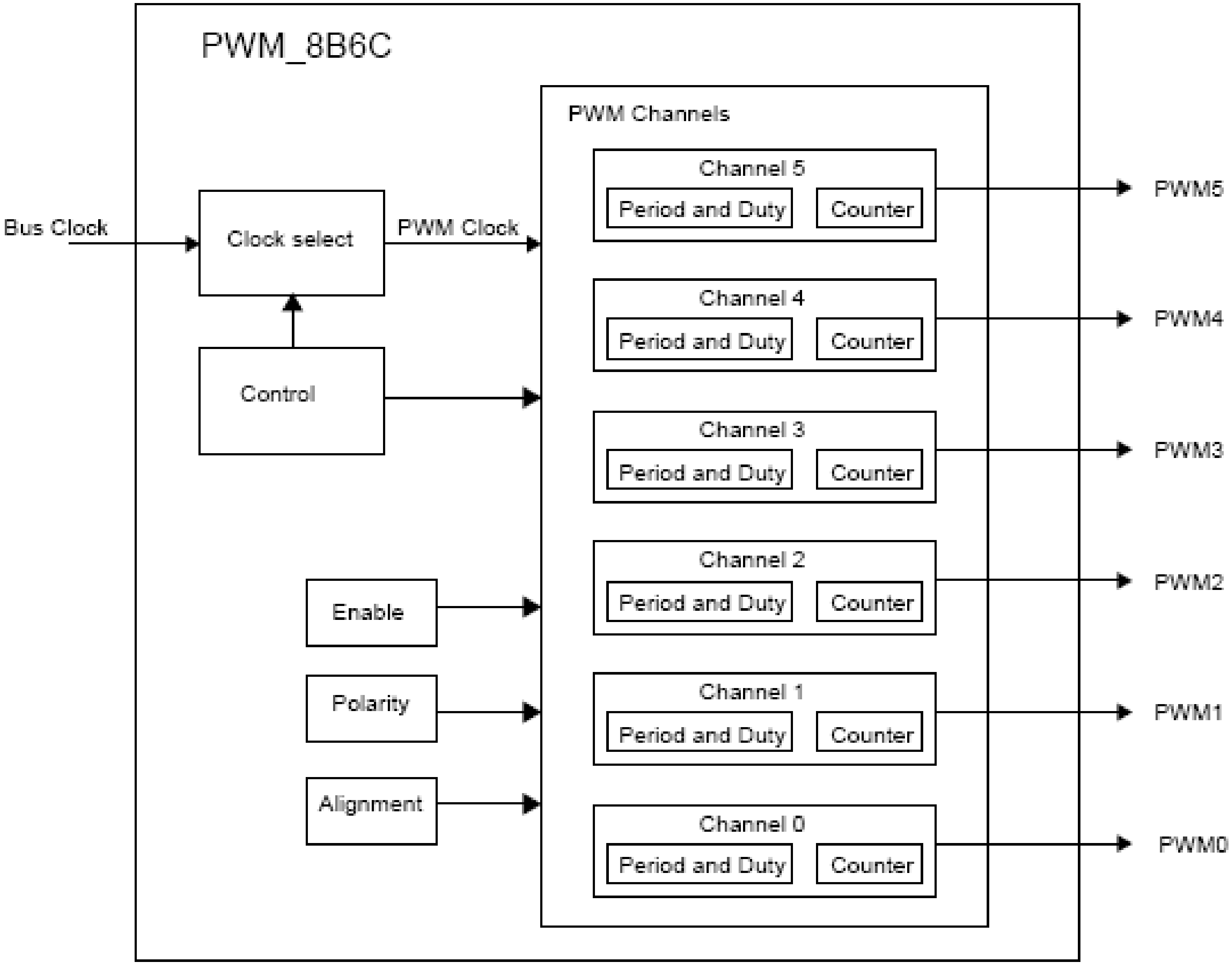


low-pass filter

PWM Features and Operating Modes

- **6 independent channels with programmable period and duty cycle**
- **8-bit resolution (6-channel mode) or 16-bit resolution (3-channel mode)**
- **Flexible clock generation – wide range of sampling frequencies**
- **Period and duty cycle registers are double-buffered (allows immediate PWM update)**
- **Output pulse polarity and alignment are programmable**
- **Emergency shutdown capability**

PWM Block Diagram



PWM Registers

- **PWME (PWM channel enable)**
  - **PWENx (bit x) – channel x enable**
    - ***"0" – disabled***
    - **"1" – enabled**
- **PWMPOL (PWM polarity select)**
  - **PPOLx (bit x) – channel x polarity**
    - ***"0" – active low***
    - **"1" – active high**

🟩 ***indicates default mode after RESET***

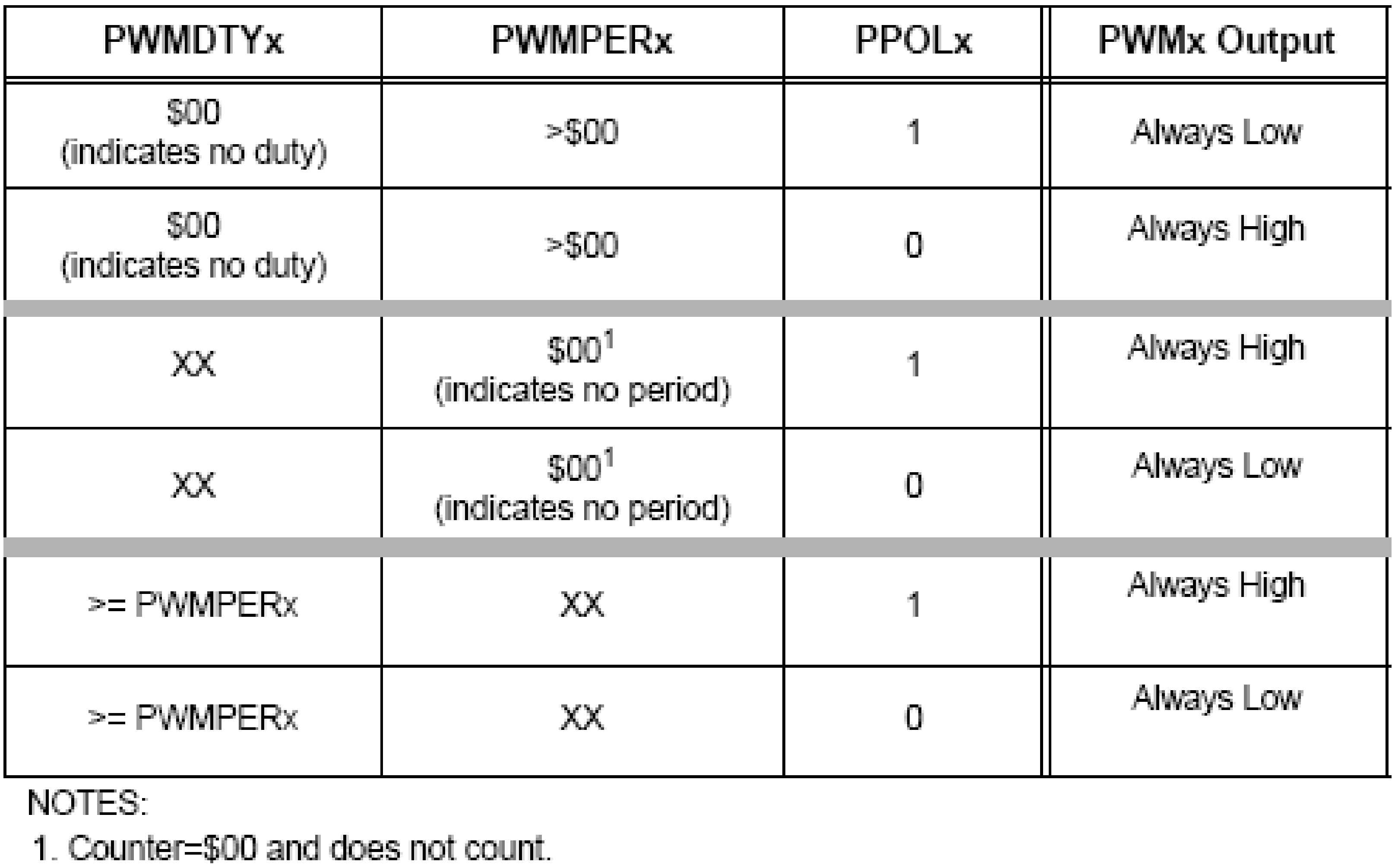
PWM Registers

- **PWMPERx (channel x period)**
  - the number of "clock ticks" that constitute ***one complete period*** of the PWM signal

- **PWMDTYx (channel x duty cycle)**
  - the number of "clock ticks" the PWM signal is ***asserted*** low* (PPOL=0) or high* (PPOL=1)

***In ECE 270, we define duty cycle as the percent of the period that a signal is asserted (independent of assertion level) – note that the definition used in the Freescale documentation is dependent on assertion level ("% of time signal is high").**

PWM Boundary Conditions

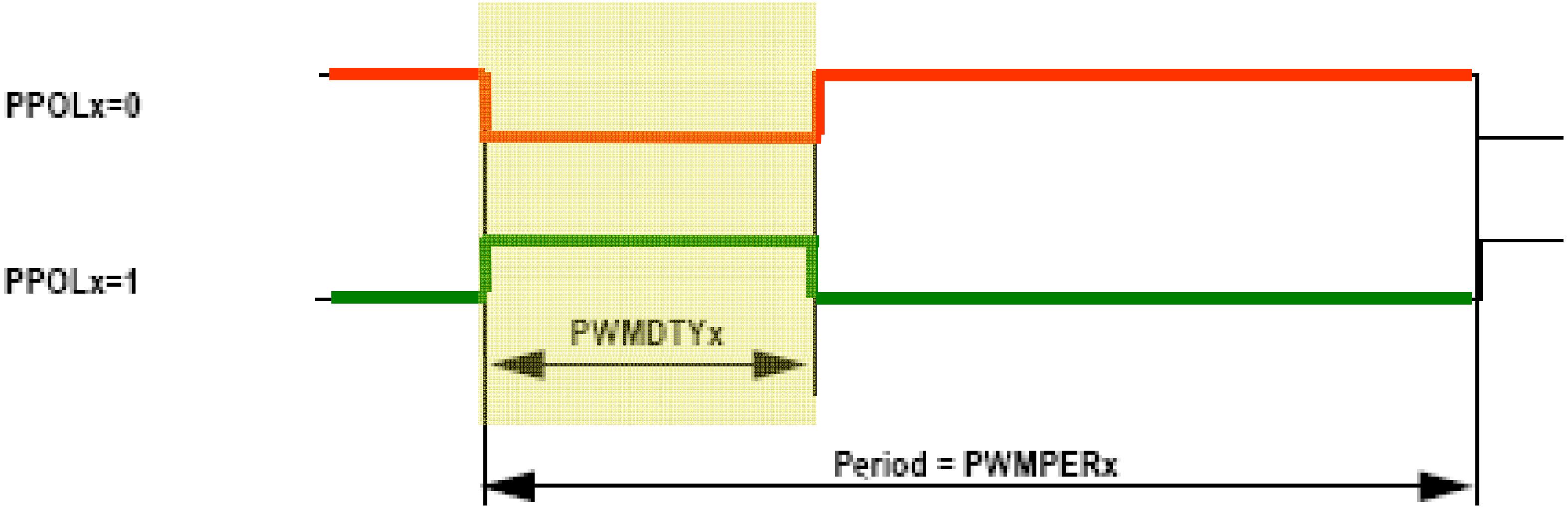


PWMDTYx	PWMPERx	PPOLx	PWMx Output
$00 (indicates no duty)	>$00	1	Always Low
$00 (indicates no duty)	>$00	0	Always High
XX	$00[1] (indicates no period)	1	Always High
XX	$00[1] (indicates no period)	0	Always Low
>= PWMPERx	XX	1	Always High
>= PWMPERx	XX	0	Always Low

NOTES:

[1] Counter=$00 and does not count.

Polarity Select and Duty Cycle

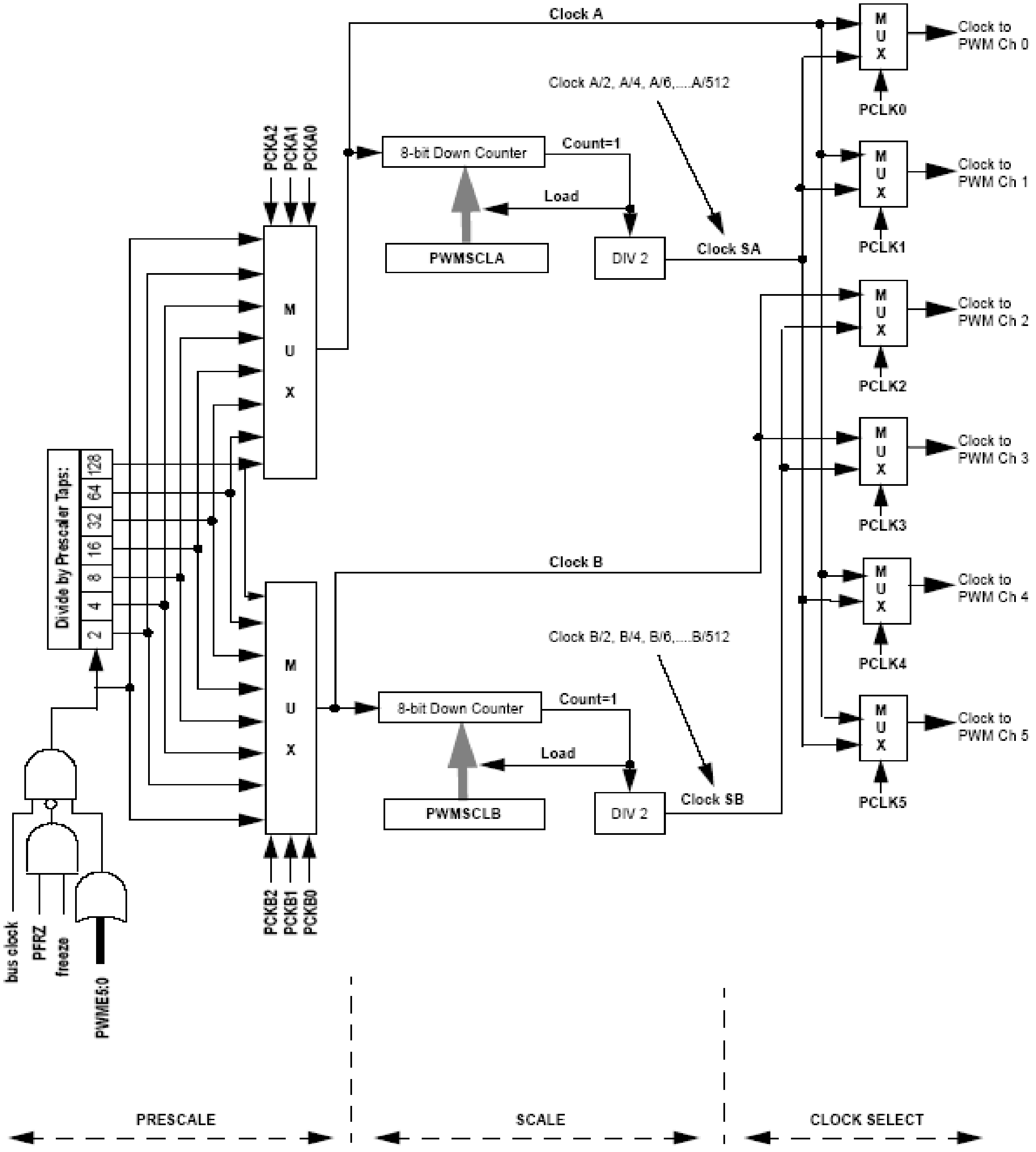


Duty Cycle (%) = 100 ∗ (PWMDTY)/(PWMPER)

Sampling Frequency = (Input Clock)/(PWMPER)

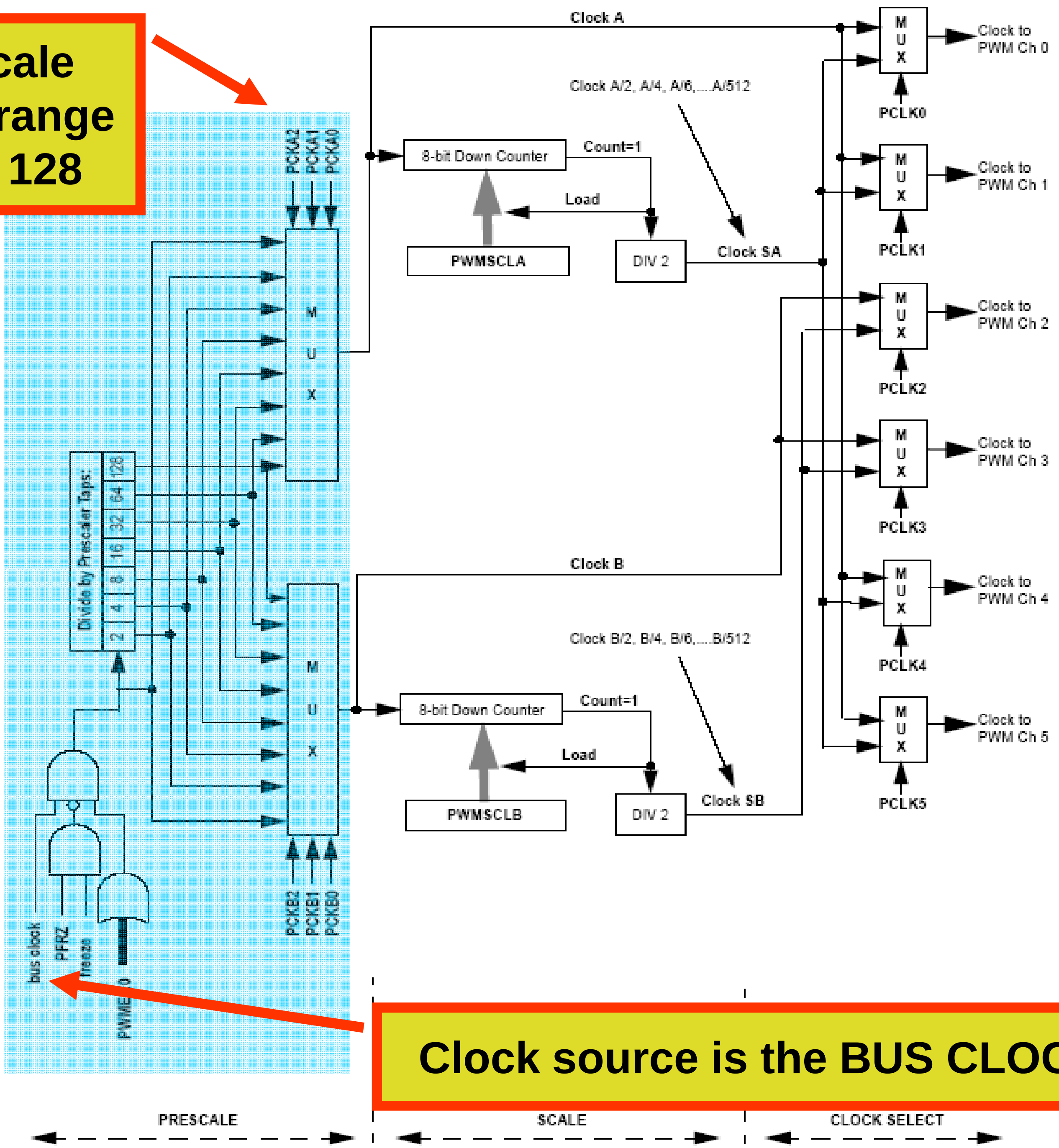
“Left Aligned” Mode (default)

PWM Clock Selection Diagram



PWM Clock Selection Diagram

Pre-scale divisor range is 1 to 128



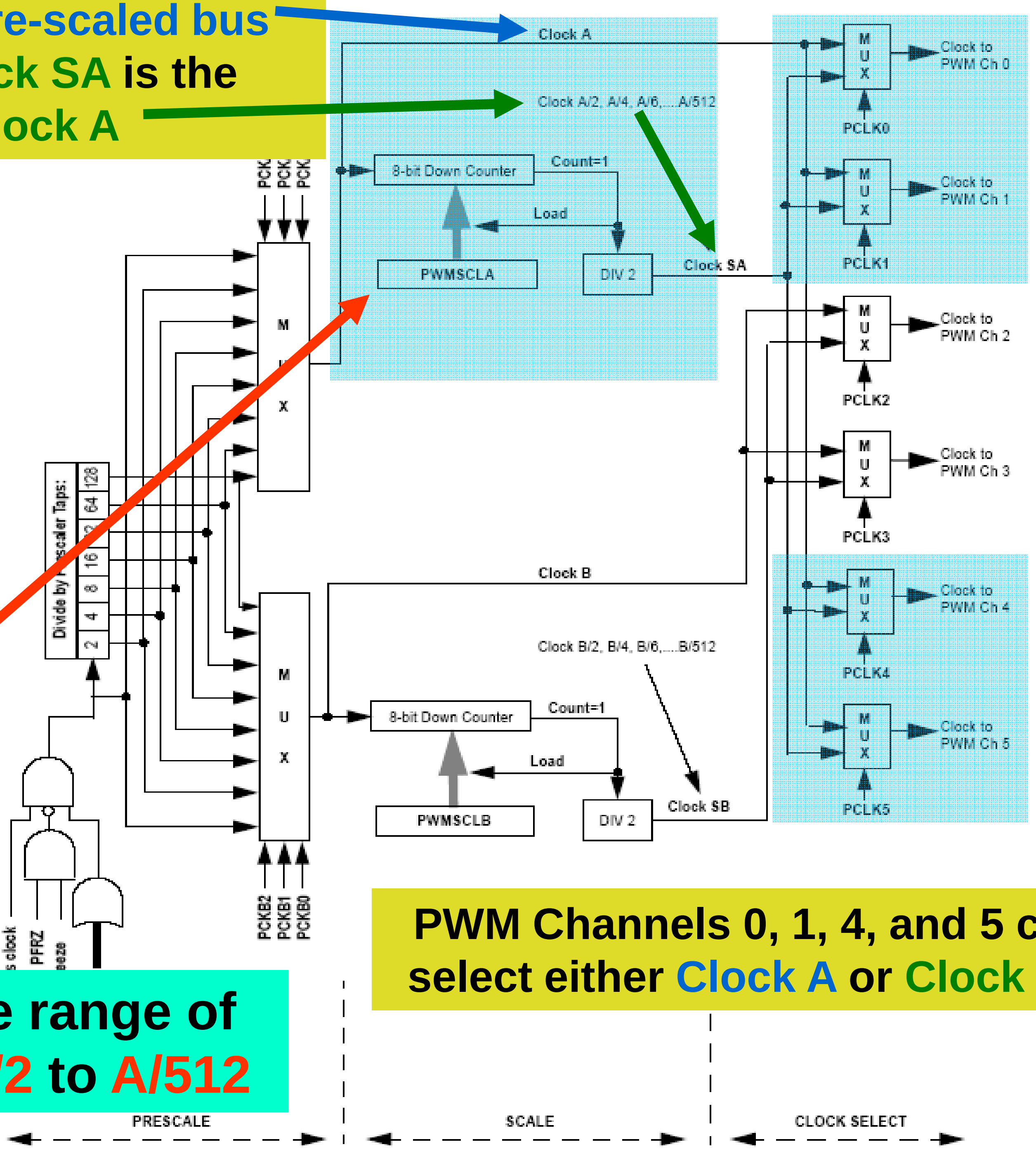
Clock source is the BUS CLOCK

PWM Registers

- **PWMPRCLK (prescale clock select)**
  - **– PCKB2- PCKB0 (bits 6-4) – Clock B prescaler**
  - **– PCKA2-PCKA0 (bits 2-0) – Clock A prescaler**
    - ***“000” – bus clock***
    - **“001” – bus clock/2**
    - **“010” – bus clock/4**
    - **“011” – bus clock/8**
    - **“100” – bus clock/16**
    - **“101” – bus clock/32**
    - **“110” – bus clock/64**
    - **“111” – bus clock/128**

🟩 ***indicates default mode after RESET***

Clock A is the pre-scaled bus clock, and Clock SA is the scaled Clock A



In general:

N = D/2

where N is the value to load in PWSCALA for clock divisor D

Note that the range of SA is from A/2 to A/512

PWM Channels 0, 1, 4, and 5 can select either Clock A or Clock SA

PWM Registers

- PWMCLK (clock source select)
  - PCLK5 (bit 5) – channel 5 clock select
    - *“0” – use “Clock A”*
    - “1” – use “Clock SA”
  - PCLK4 (bit 4) – channel 4 clock select
    - *“0” – use “Clock A”*
    - “1” – use “Clock SA”

🟩 *indicates default mode after RESET*

PWM Registers

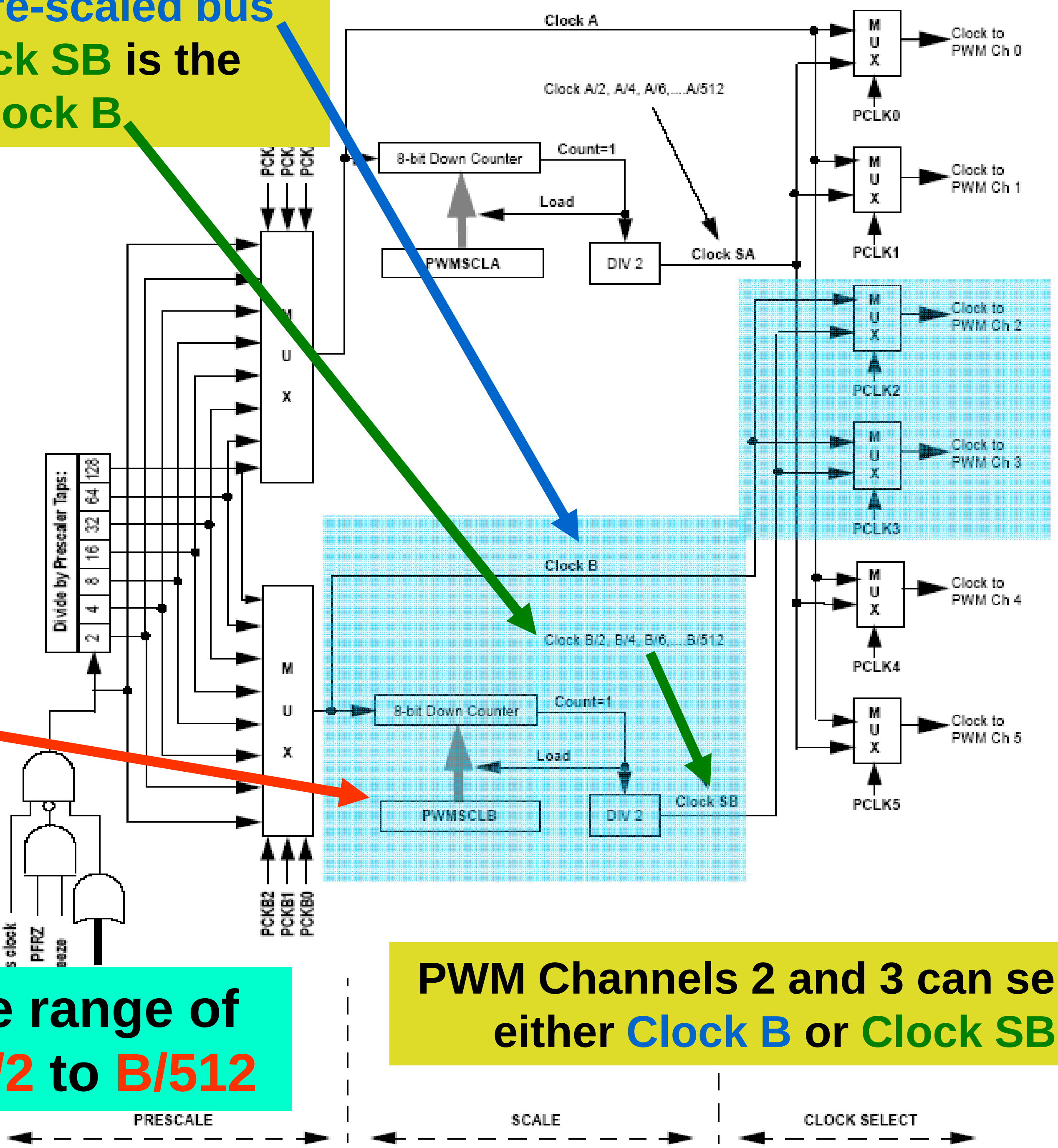
- **PWMCLK (clock source select)**
  - **PCLK1 (bit 1) – channel 1 clock select**
    - ***"0" – use "Clock A"***
    - **"1" – use "Clock SA"**
  - **PCLK0 (bit 0) – channel 0 clock select**
    - ***"0" – use "Clock A"***
    - **"1" – use "Clock SA"**

🟩 ***indicates default mode after RESET***

PWM Registers

- PWMSCLA (Scale A Register)
  - 8-bit programmable scale value for Clock A
  - Clock SA = Clock A / (2 $*$ PWMSCLA)
  - $00 is used to represent 256_{10}

Clock B is the pre-scaled bus clock, and Clock SB is the scaled Clock B



In general:

N = D/2

where N is the value to load in PWSCALB for clock divisor D

Note that the range of SB is from B/2 to B/512

PWM Channels 2 and 3 can select either Clock B or Clock SB

PWM Registers

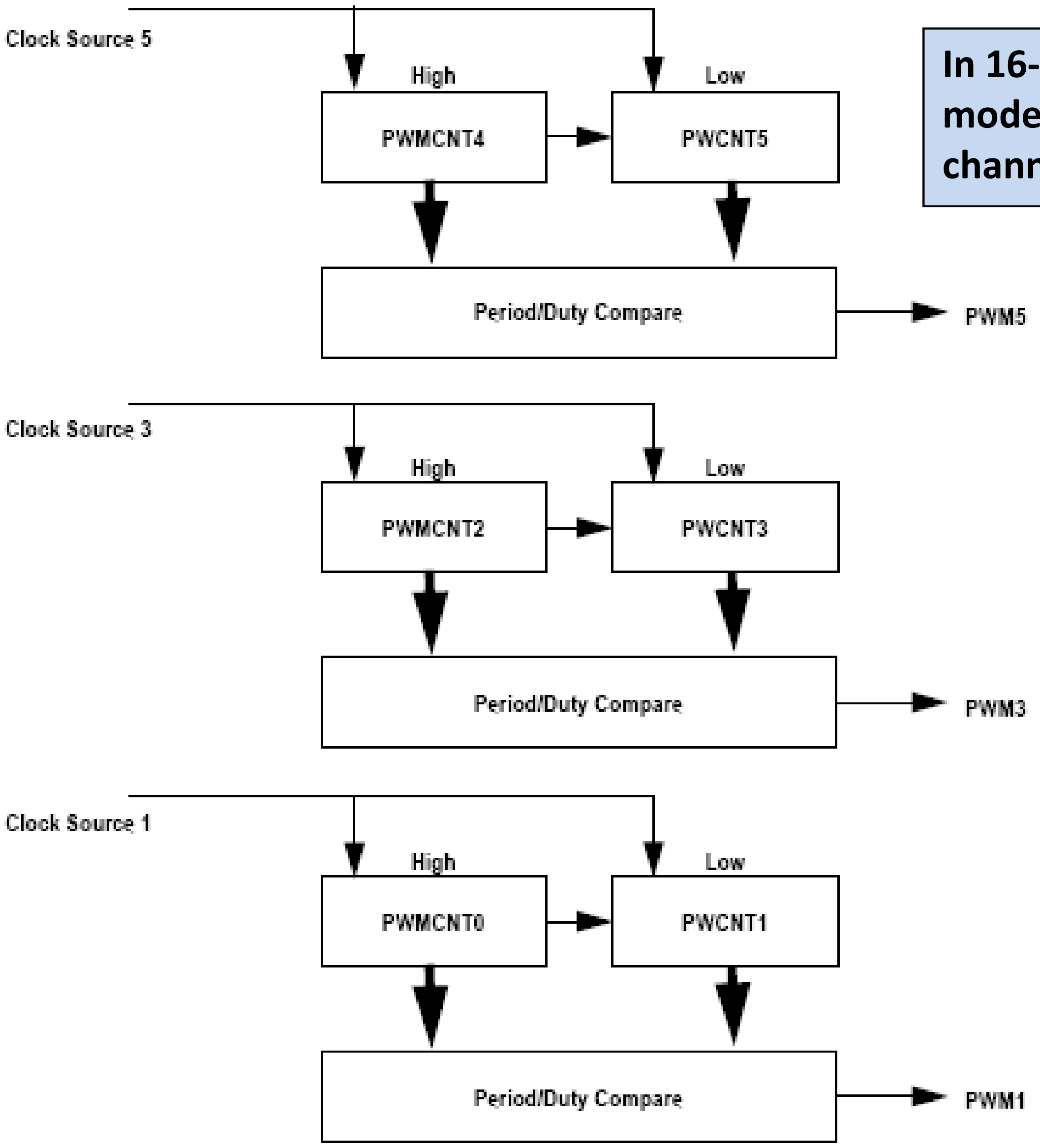
- **PWMCLK (clock source select)**
  - **PCLK3 (bit 3) – channel 3 clock select**
    - ***“0” – use “Clock B”***
    - **“1” – use “Clock SB”**
  - **PCLK2 (bit 2) – channel 2 clock select**
    - ***“0” – use “Clock B”***
    - **“1” – use “Clock SB”**

🟩 ***indicates default mode after RESET***

PWM Registers

- **PWMSCLB (Scale B Register)**
  - **8-bit programmable scale value for Clock B**
  - **Clock SB = Clock B / (2 $*$ PWMSCLB)**
  - **\$00 used to represent 256_{10}**

16-bit PWM Mode



In 16-bit ("concatenated") mode, the ***lower numbered*** channel is the **high byte**

PWM Registers

- **PWMCTL (PWM control)**
  - **CON45 (bit 6) – concatenate chs 4 & 5**
    - ***“0” – channels 4 & 5 separate 8-bit***
    - **“1” – channels 4 & 5 concatenated 16-bit**
  - **CON23 (bit 5) – concatenate chs 2 & 3**
    - ***“0” – channels 2 & 3 separate 8-bit***
    - **“1” – channels 2 & 3 concatenated 16-bit**
  - **CON01 (bit 4) – concatenate chs 0 & 1**
    - ***“0” – channels 0 & 1 separate 8-bit***
    - **“1” – channels 0 & 1 concatenated 16-bit**

🟩 ***indicates default mode after RESET***

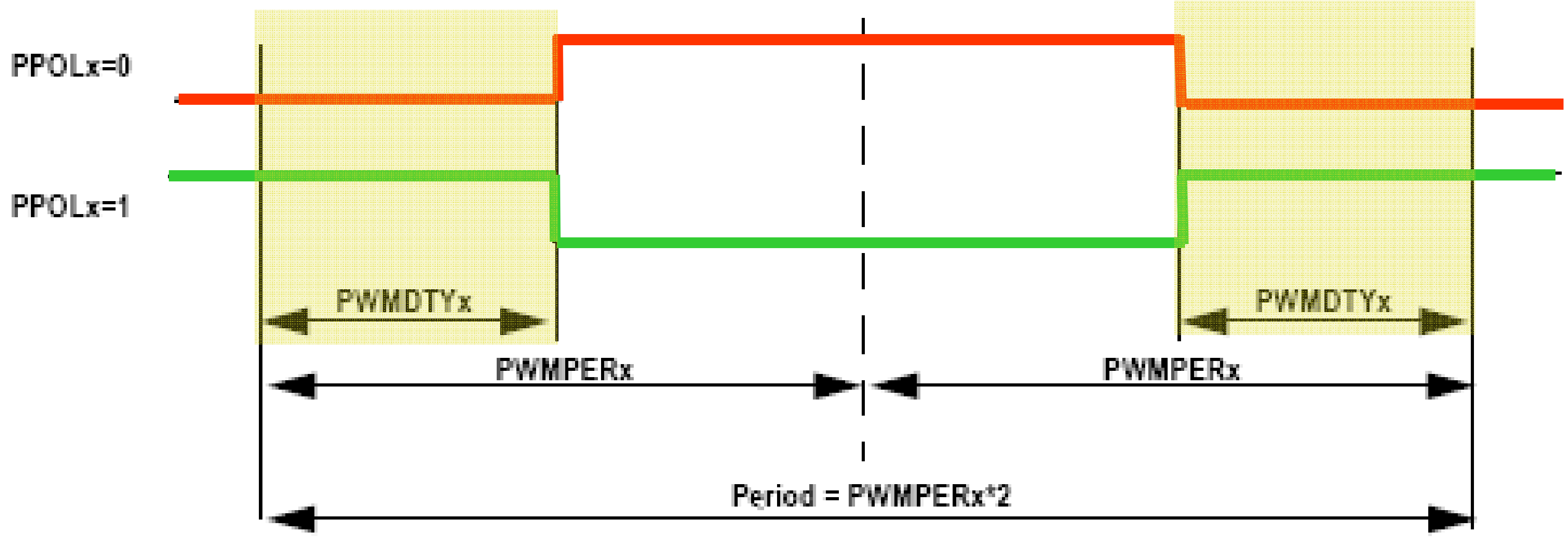
PWM Registers

- **PWMCAE (center align enable)**
  - **CAEx (bit x) – center align channel x**
    - ***"0" – operate ch x in left-aligned mode***
    - **"1" – operate ch x in center-aligned mode**

Note: The center-aligned mode is useful for asynchronous motor control (e.g., for brushless DC motors)

🟩 ***indicates default mode after RESET***

Center-Aligned Mode



Duty Cycle (%) = 100 ∗ (PWMDTY)/(PWMPER)

Sampling Frequency = (Input Clock)/(2∗ PWMPER)

PWM Registers

- **PWMSDN (PWM shutdown)**
  - **PWMIF (bit 0) – interrupt flag**
    - ***“0” – PWM state has not changed***
    - **“1” – PWM state has changed**
  - **PWMIE (bit 6) – interrupt enable**
    - ***“0” – PWM interrupt disabled***
    - **“1” – PWM interrupt enabled**
  - **PWMRSTRT (bit 5)**
    - **Used to restart PWM outputs after shutdown state is cleared**

🟩 ***indicates default mode after RESET***

PWM Registers

- **PWMSDN (PWM shutdown)**
  - **PWMLVL (bit 4) – shutdown output level**
    - ***“0” – PWM outputs all forced low***
    - **“1” – PWM outputs all forced high**
  - **PWM5IN (bit 2) – Ch 5 input status**
    - **Reflects current status of PWM5 pin**
  - **PWM5INL (bit 1) – shutdown active level**
    - ***“0” – Active level is low***
    - **“1” – Active level is high**
  - **PWM5ENA (bit 0) – shutdown enable**
    - ***“0” – PWM emergency shutdown disabled***
    - **“1” – PWM emergency shutdown enabled**

🟩 ***indicates default mode after RESET***

PWM Registers

- Port P (data register)
  - bits 0-5 used for PWM output channels 0-5
  - PWM takes precedence over general-purpose I/O when enabled
- DDRP (data direction register)
  - used to establish data direction of Port P bits when used for general-purpose I/O

Clicker Quiz

1. Useful applications of the PWM include:
   A. D.C. motor speed control
   B. digital-to-analog conversion
   C. controlling the intensity of an LED
   D. all of the above
   E. none of the above

2. The double buffering feature of the PWM unit:

   A. prevents a PWM output from changing the instant the period or duty register is written

   B. provides a larger window of time during which the PWM registers can be written

   C. provides a larger window of time during which the PWM registers can be read

   D. all of the above

   E. none of the above

3. Given a 24 MHz bus clock, in 8-bit mode the minimum frequency 50% duty cycle square wave that can be generated by the PWM unit is approximately:

   A. 0.15 Hz

   B. 1.50 Hz

   C. 15.00 Hz

   D. 150.0 Hz

   E. none of the above

4. Given a 24 MHz bus clock, in 8-bit mode the maximum frequency 50% duty cycle square wave that can be generated by the PWM unit is:

A. 12,000 Hz

B. 120,000 Hz

C. 1,200,000 Hz

D. 12,000,000 Hz

E. none of the above

5. Given a 24 MHz system clock with PWMPRCLK = $0110\ 0000_2$ and PWMCLK = $0000\ 0100_2$, the following combination of PWM register initializations will produce a 100 Hz (approx.), 50% duty cycle square wave on Channel 2 (assuming PWMPOL = $0000\ 0100_2$ and PWMEN = $0000\ 0100_2$):

   A. PWSCALB = 93_{10}, PMWPER2 = 20_{10}, PMWDTY2 = 10_{10}

   B. PWSCALB = 186_{10}, PWMPER2 = 20_{10}, PWMDTY2 = 10_{10}

   C. PWSCALB = 50_{10}, PWMPER2 = 100_{10}, PWMDTY2 = 50_{10}

   D. PWSCALB = 49_{10}, PWMPER2 = 100_{10}, PWMDTY2 = 50_{10}

   E. none of the above

PWM Initialization Example

Assume the bus clock is 24 MHz and that the PWM is used to generate four left-aligned active high square wave signals:

- **Ch 0 – 120,000 Hz – 10% duty cycle**
- **Ch 1 – 20,000 Hz – 30% duty cycle**
- **Ch 2 – 7,500 Hz – 70% duty cycle**
- **Ch 3 – 250 Hz – 85% duty cycle**

Determine the register initializations required to generate these four waveforms

PWM Initialization Example

For Ch 0 (and Ch 1), need PWMPER = 100 to have a "resolution" of 1% in specifying the duty cycle → need input clock frequency of at least 120,000 $*$ 100 = 12 MHz (conveniently, this is half the 9S212C32 bus clock freq.)

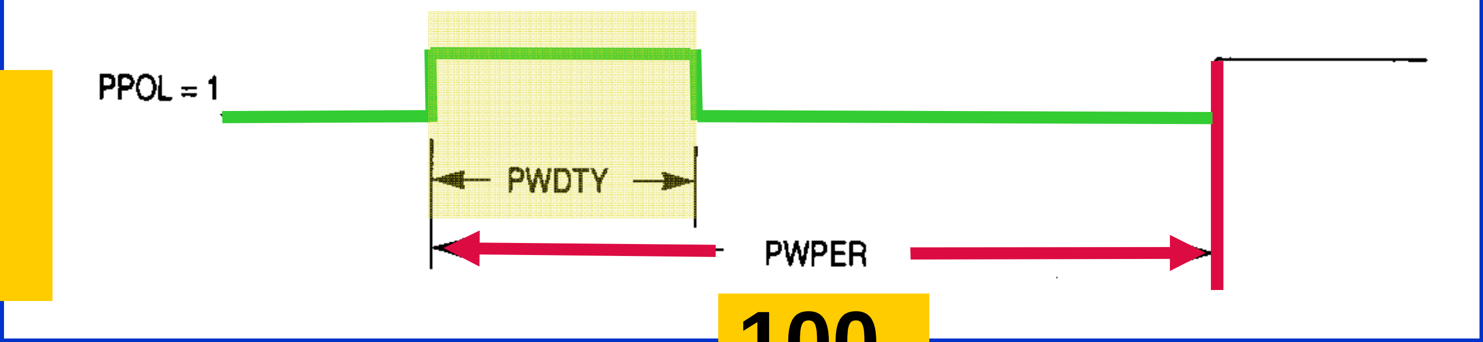
The "scale" register for Clock A can be used to produce the input clock for Ch 1, which is 20,000 $*$ 100 = 2 MHz → Clock SA = (Clock A)/6 → PWSCALA = 3

PWM Initialization Example

- PWME = 0Fh (enables PWM Chs 0-3)
- PWMPER0 (ch 0 period) = 100t
- PWMPER1 (ch 1 period) = 100t
- PWMPER2 (ch 2 period) = 100t
- PWMPER3 (ch 3 period) = 100t
  - the number of “clock ticks” that constitute one complete period of the PWM signal

Will allow resolution of 1%

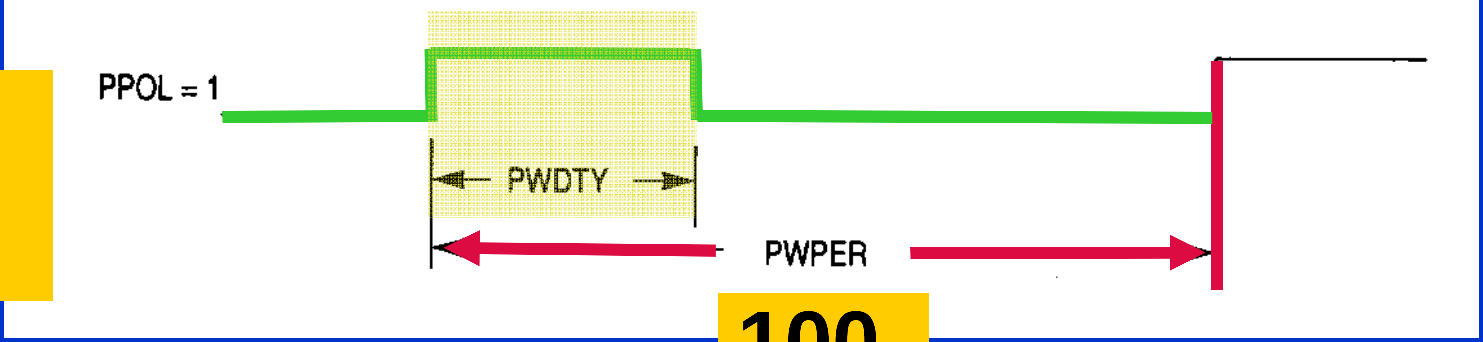
Left-aligned mode



100

PWM Initialization Example

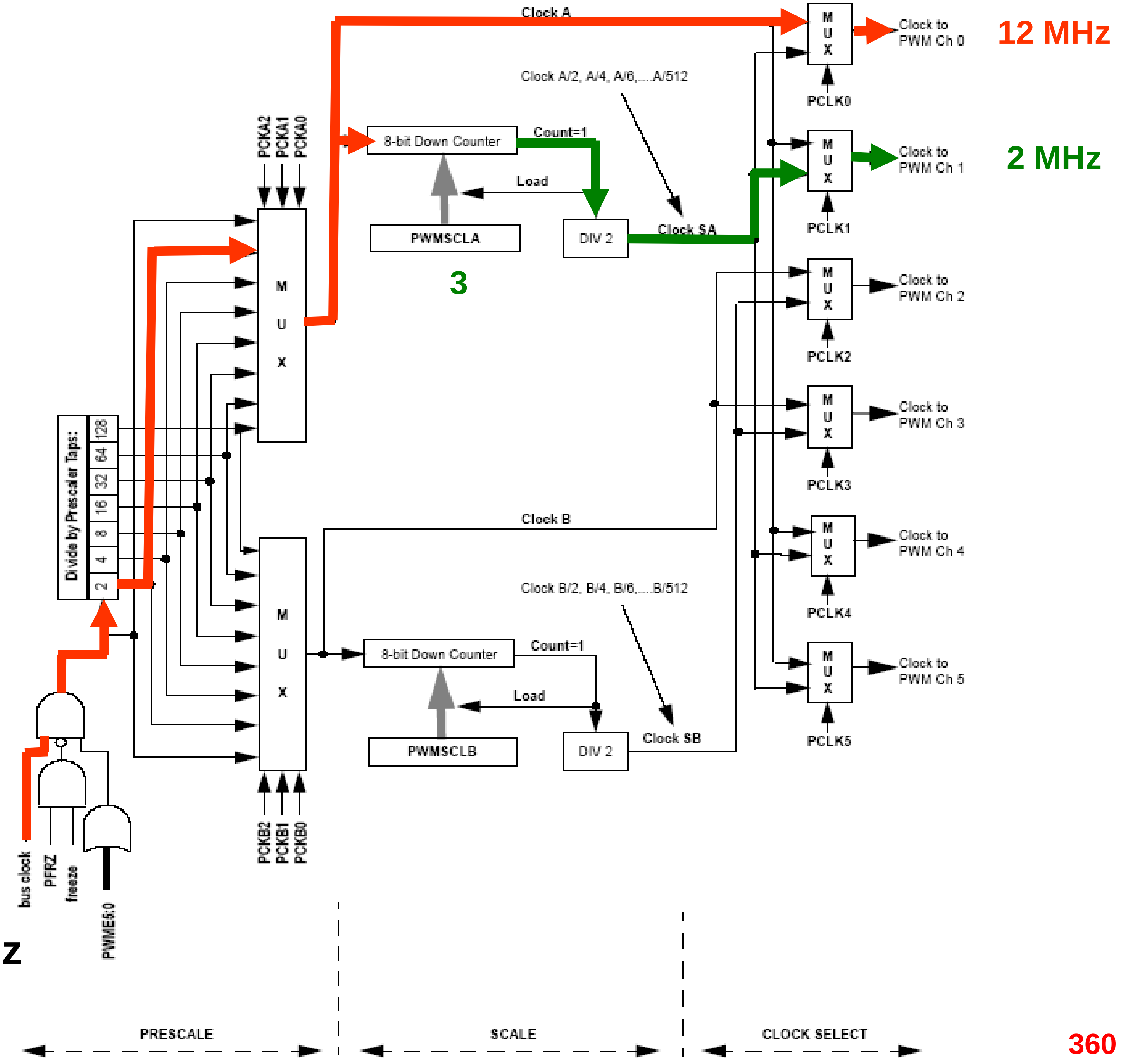
- **PWMPOL = 0Fh (Chs 0-3 active high polarity)**
- **PWMDTY0 (Ch 0 duty cycle) = 10t (10%)**
- **PWMDTY1 (Ch 1 duty cycle) = 30t (30%)**
- **PWMDTY2 (Ch 2 duty cycle) = 70t (70%)**
- **PWMDTY3 (Ch 3 duty cycle) = 85t (85%)**
  - **the number of "clock ticks" the PWM signal is asserted low (PPOL=0) or high (PPOL=1)**



Left-aligned mode

100

PWM Clock Selection Diagram



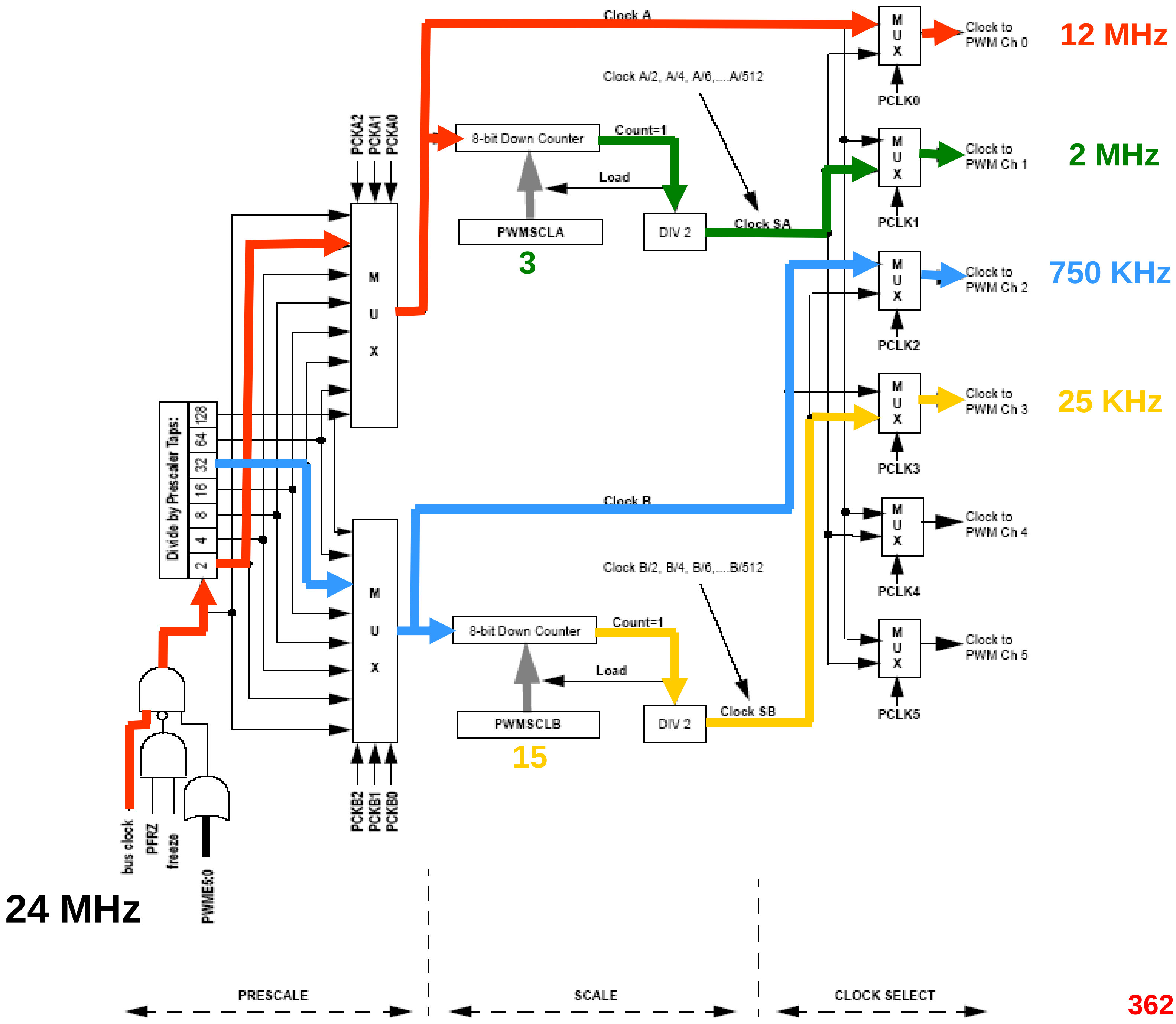
24 MHz

PWM Initialization Example

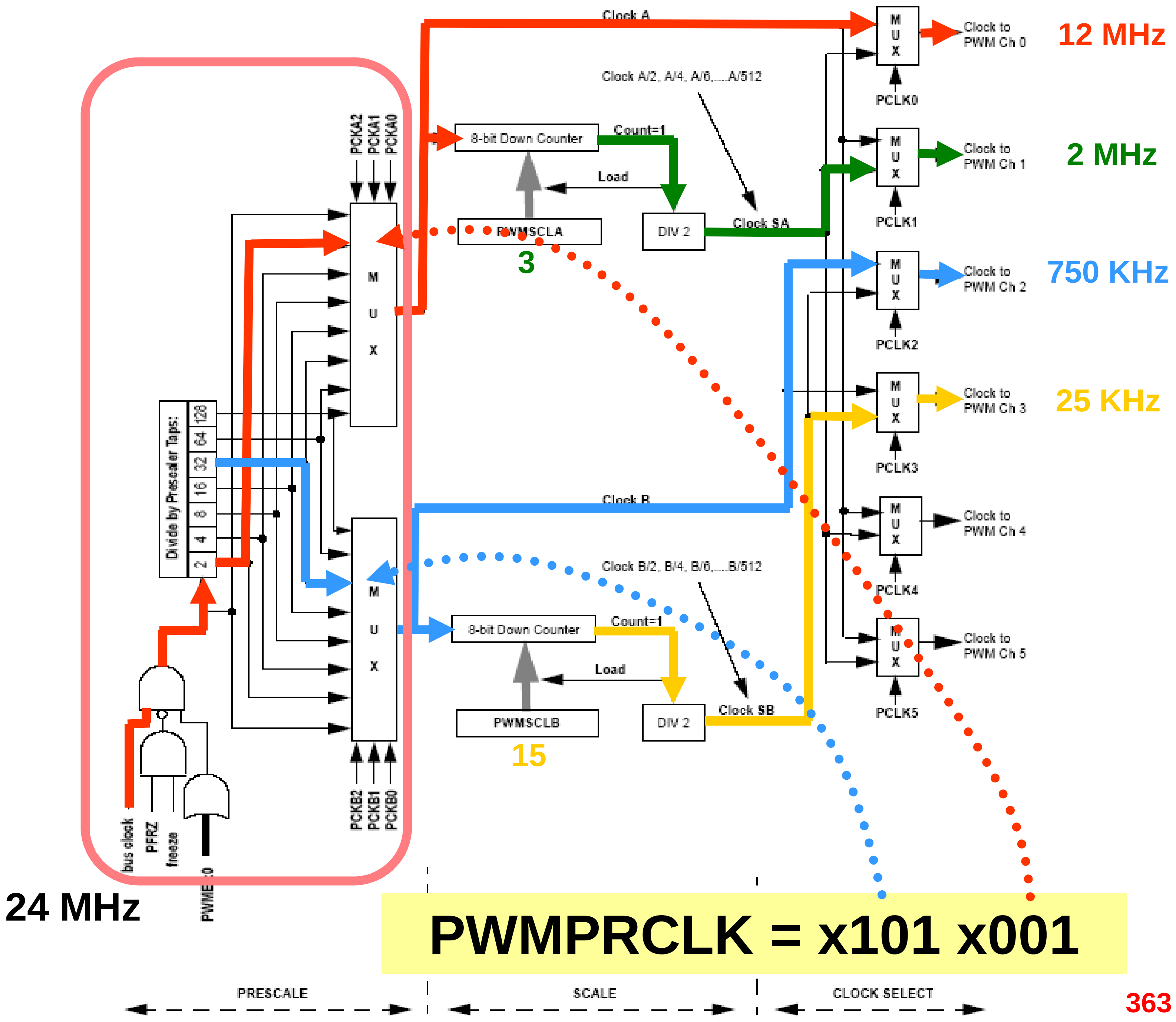
For Ch 2 (and Ch 3), again need PWMPER = 100 to have a “resolution” of 1% in specifying the duty cycle → need input clock frequency of at least 7,500 $*$ 100 = 750 KHz = 24 MHz / 32

The “scale” register for Clock B can be used to produce the input clock for Ch 3, which is 250 $*$ 100 = 25 KHz → Clock SB = (Clock B) / 30 → PWSCALB = 15

PWM Clock Selection Diagram

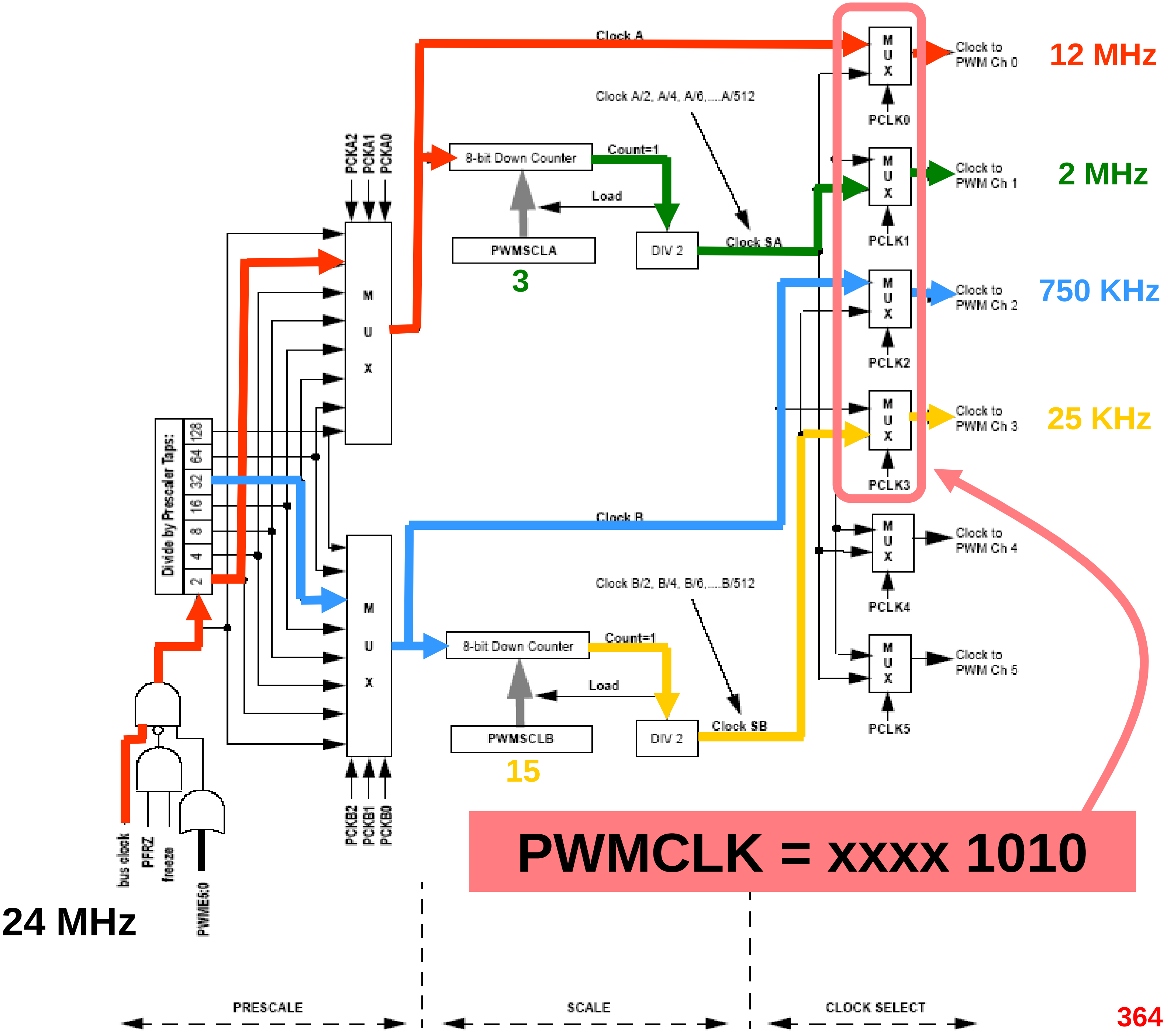


PWM Clock Selection Diagram



PWMPRCLK = x101 x001

PWM Clock Selection Diagram



12 MHz

2 MHz

750 KHz

25 KHz

3

15

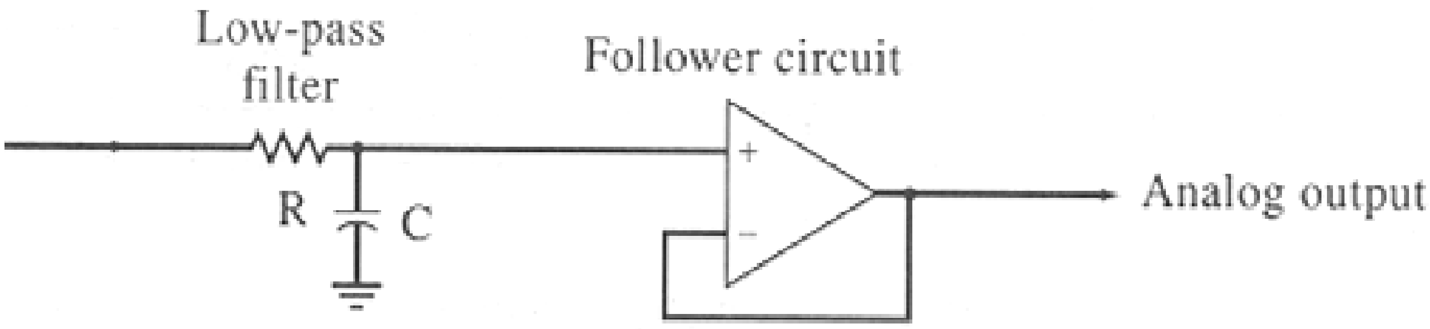
24 MHz

PWMCLK = xxxx 1010

PWM Applications/Interfaces

- **Simple “D/A converter” (single pole LPF)**

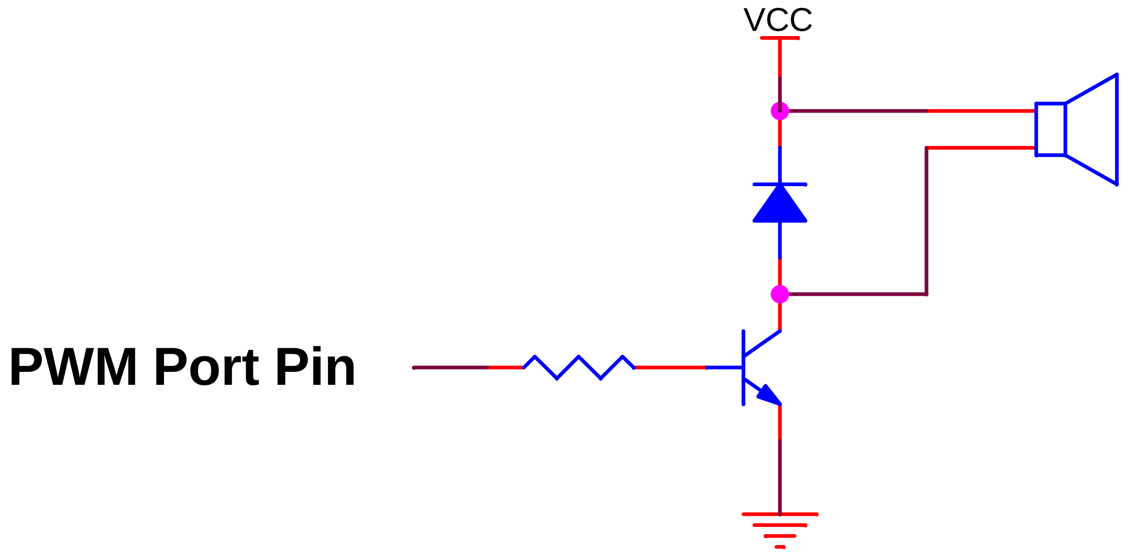
PWM Port Pin



A higher order (active) LPF can also be used – the OP AMP provides isolation and additional output drive capability (not necessary if output load is high impedance)

PWM Applications/Interfaces

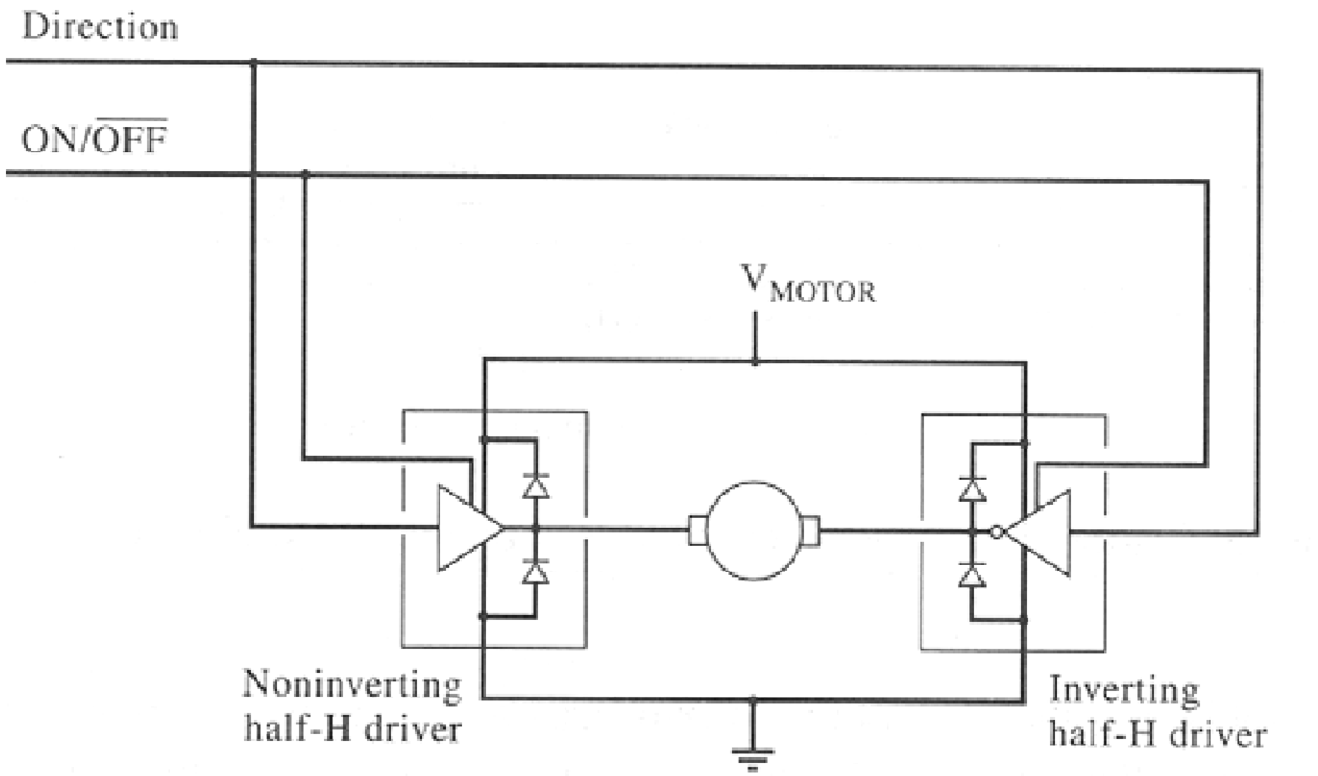
- **Driving a small loudspeaker**



The loudspeaker will "mechanically" low-pass filter the PWM signal

PWM Applications/Interfaces

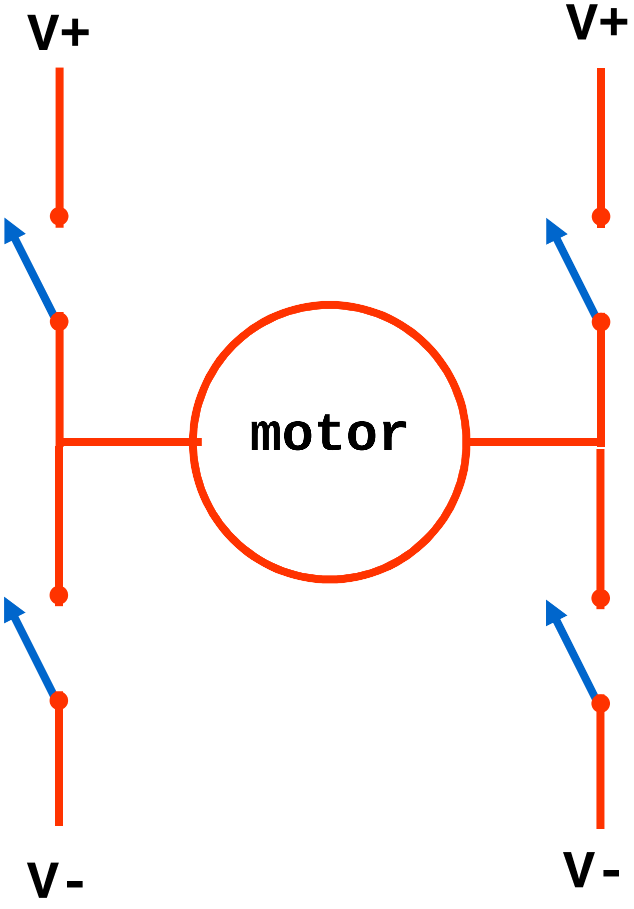
- **Motor speed and direction control using an “H”-bridge (here, a pair of “half-H” bridges)**



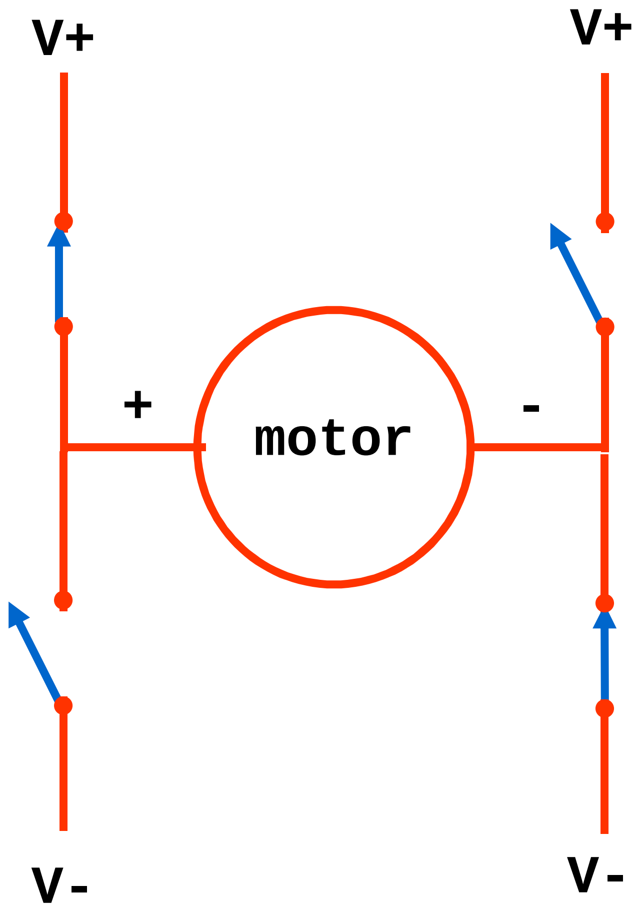
Output Port Pin

PWM Port Pin

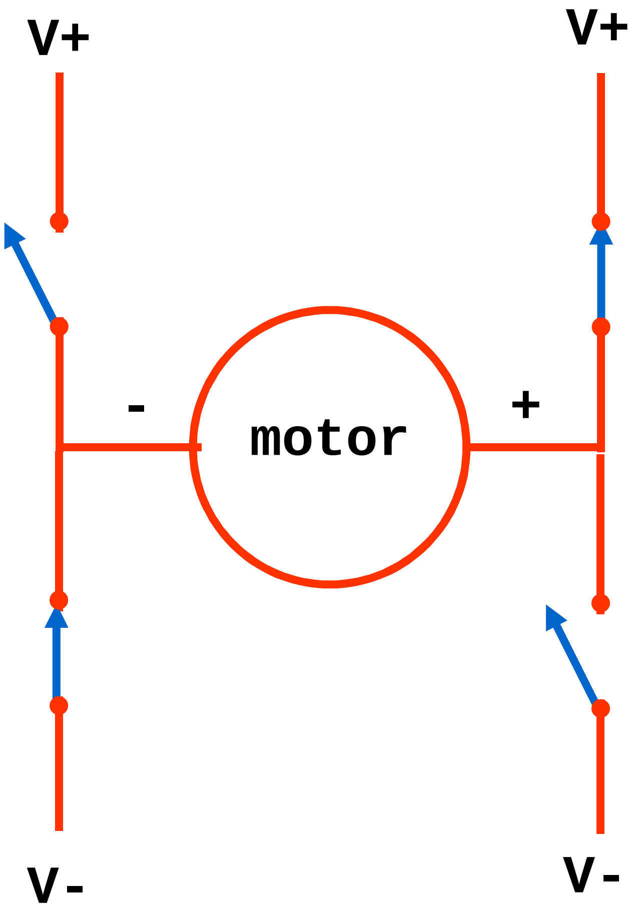
“H Bridge”



“H Bridge”



“H Bridge”



Not-So-Quick Clicker Quiz

```
;**********************************************************************
; Analyze the following code
;**********************************************************************
;
; initialize PWM Ch 0

        movb     #$01,MODRR        ; PT0 used as PWM Ch 0 output

        movb     #$01,PWME         ; enable PWM Ch 0
        movb     #$01,PWMPOL       ; set active high polarity
        movb     #$00,PWMCTL       ; no concatenate (8-bit)
        movb     #$00,PWMCAE       ; left-aligned output mode
        movb     #$FF,PWMPER0      ; set maximum 8-bit period (255)
        movb     #$7F,PWMDTY0      ; set 50% duty cycle
        movb     #$07,PWMPRCLK     ; set Clock A = 24 MHz / 128 (max pre-scalar)
        movb     #$01,PWMCLK       ; select Clock SA for Ch 0
        movb     #$00,PWMSCLA      ; set Clock SA scalar to 512 (max scalar)

; initialize TIM Ch 1

        movb    #$80,tscr1         ; enable TC1
        movb    #$07,tscr2         ; set TIM pre-scale factor to 128 (max)
        movb    #$02,tios          ; set TIM TC1 for Output Compare mode
        movw    #$0004,tctl1       ; toggle PT1 on successful output compare
        movw    #$0000,tc1         ; value for OC1

eloop   bra      eloop             ; infinite (do-nothing) loop
```

Q1. **The PT0 “toggle rate” (in milliseconds) for the code as written will be approximately:**

A. **1.36 ms**

B. **2.73 ms**

C. **696 ms**

D. **699 ms**

E. **none of the above**

Q2. The PT1 “toggle rate” (in milliseconds) for the code as written will be approximately:

A. 1.36 ms

B. 350 ms

C. 696 ms

D. 699 ms

E. none of the above

Q3. The PT0 and PT1 “toggle rates” could be made to be identical (i.e., exactly in sync) by:

A. changing the value in PWMDTY0 to $80

B. changing the value in PWMPER0 to $00

C. changing the value in PWMSCLA to $01

D. changing the value in TC1 to $0100

E. none of the above

Q4. Increasing the value loaded into TC1 would:

- **A.** **do nothing except change the (initial) relative phase of the PT0 and PT1 port pin toggling**
- **B.** **increase the PT1 port pin toggle rate**
- **C.** **decrease the PT1 port pin toggle rate**
- **D.** **disable port pin PT1 from toggling**
- **E.** **none of the above**

Not-So-Quick Clicker Quiz

(after DPS)

```
;************************************************************************
;
; In this exercise we will investigate ATD-PWM audio digitization
;   and signal reconstruction as well as a "digital volume control".
;
;************************************************************************
;
; Resources used:
;  - TIM TC7 - interrupt every 0.1 ms, used to initiate ATD conversion
;  - ATD Ch 0 - audio input (PIN 5 on BREADBOARD HEADER)
;  - ATD Ch 1 - potentiometer input for digital volume control (PIN 6 on
;               BREADBOARD HEADER)
;  - PWM Ch 0 - audio output (PIN 27 on BREADBOARD HEADER)

; CAUTION: Carefully adjust the function generator analog output for a
;          D.C. offset of 2 V and a peak voltage no greater than 4 V
;          (use the oscilloscope to confirm).

; Task list:
;   1.  Determine the input sampling frequency and the output sampling
;       frequency for the code as written.
;   2.  Try varying the input and output sampling frequencies
;       (independently), and note the effects.
;  START OF CODE
;***********************************************************************

; initialize ATD (8 bit, unsigned, nominal sample time, seq length = 2)

      movb      #$80,ATDCTL2    ; power up ATD
      movb      #$10,ATDCTL3    ; set conversion sequence length to TWO
      movb      #$85,ATDCTL4    ; select resolution and sample time

; initialize PWM Ch 0 (left aligned, positive polarity, max 8-bit period)

      movb      #$01,MODRR      ; PT0 used as PWM Ch 0 output
      movb      #$01,PWME       ; enable PWM Ch 0
      movb      #$01,PWMPOL     ; set active high polarity
      movb      #$00,PWMCTL     ; no concatenate (8-bit)
      movb      #$00,PWMCAE     ; left-aligned output mode
      movb      #$FF,PWMPER0    ; set maximum 8-bit period
      movb      #$00,PWMDTY0    ; initially clear DUTY register
      movb      #$00,PWMPRCLK   ; set Clock A = 24 MHz (prescaler = 1)
      movb      #$01,PWMCLK     ; select Clock SA for Ch 0
      movb      #$01,PWMSCLA    ; set Clock SA scalar to 2
                                ; **** change to modify output sampling rate
; initialize TIM Ch 7 (periodic 0.1 ms interrupt)

      movb    #$80,tscr1  ; enable TC7
      movb    #$0C,tscr2  ; set TIM prescale factor to 16
                          ; and enable counter reset after OC7
      movb    #$80,tios   ; set TIM TC7 for Output Compare
      movb    #$80,tie    ; enable TIM TC7 interrupt
      movw    #150,tc7  ; set up TIM TC7 to generate 0.1 ms interrupt rate
                        ; **** change this to modify input sampling rate

; enable IRQ interrupts

      cli               ; enable IRQ interrupts

; do nothing after initialization complete

eloop  bra      eloop   ; wait loop
```

```
; TIM interrupt service routine
; Interrupt generated every 0.1 ms (default - may be changed)
; Initiate ATD conversion on Chs 0 and 1
; Multiply analog input sample by potentiometer sample -> PWM Ch 0
;

tim_isr

      bset      tflg1,$80       ; clear TC7 interrupt flag

      movb      #$10,ATDCTL5    ; start conversion on ATD Ch 0
                                ; NOTE: "mult" (bit 4) needs to be set

await brclr     ATDSTAT,$80,await

      ldaa      ATDDR0          ; analog input sample
      ldab      ATDDR1          ; potentiometer sample
      mul                       ; multiply input sample X pot setting
      adca      #0              ; round upper 8 bits of result
      staa      PWMDTY0         ; copy result to PWM Ch 0

      rti
```

Q1. **The input sampling frequency, for the code as written, is approximately:**

A. 1 KHz

B. 2 KHz

C. 5 KHz

D. 10 KHz

E. none of the above

Q2. An appropriate choice for anti-aliasing low-pass filter cut-off frequency, for the application as written, would be:

A. **1 KHz**

B. **2 KHz**

C. **5 KHz**

D. **10 KHz**

E. **none of the above**

Q3. The **output sampling frequency**, for the code as written, is approximately:

A. 10 KHz

B. 12 KHz

C. 47 KHz

D. 94 KHz

E. none of the above

Q4. If the value loaded in TSCR2 was changed to $0B and the value loaded into TC7 was changed to $00FF, the new input sampling frequency would be (approximately):

A. 11,765 Hz

B. 23,530 Hz

C. 47,059 Hz

D. 94,118 Hz

E. none of the above

Q5. **If the value loaded into PWMSCLA was changed to 4, the new output sampling frequency would be (approximately):**

A. **11,765 Hz**

B. **23,530 Hz**

C. **47,059 Hz**

D. **94,118 Hz**

E. **none of the above**

Q6. **If the changes described in Q4 and Q5 were made, the reconstructed analog waveform would be ____ relative to the code as originally written.**

- **A.** **lower in amplitude**
- **B.** **less distorted**
- **C.** **more distorted**
- **D.** **higher in amplitude**
- **E.** **none of the above**

Q7. The change in the reconstructed analog output waveform referred to in Q6 is caused by:

A. uniformly sampling both the ATD input and the PWM output

B. uniformly sampling the ATD input, but naturally sampling the PWM output

C. naturally sampling the ATD input, but uniformly sampling the PWM output

D. naturally sampling both the ATD input and the PWM output

E. none of the above