Introduction to Microcontrollers

中山 大 学

数据科学与计算机学院

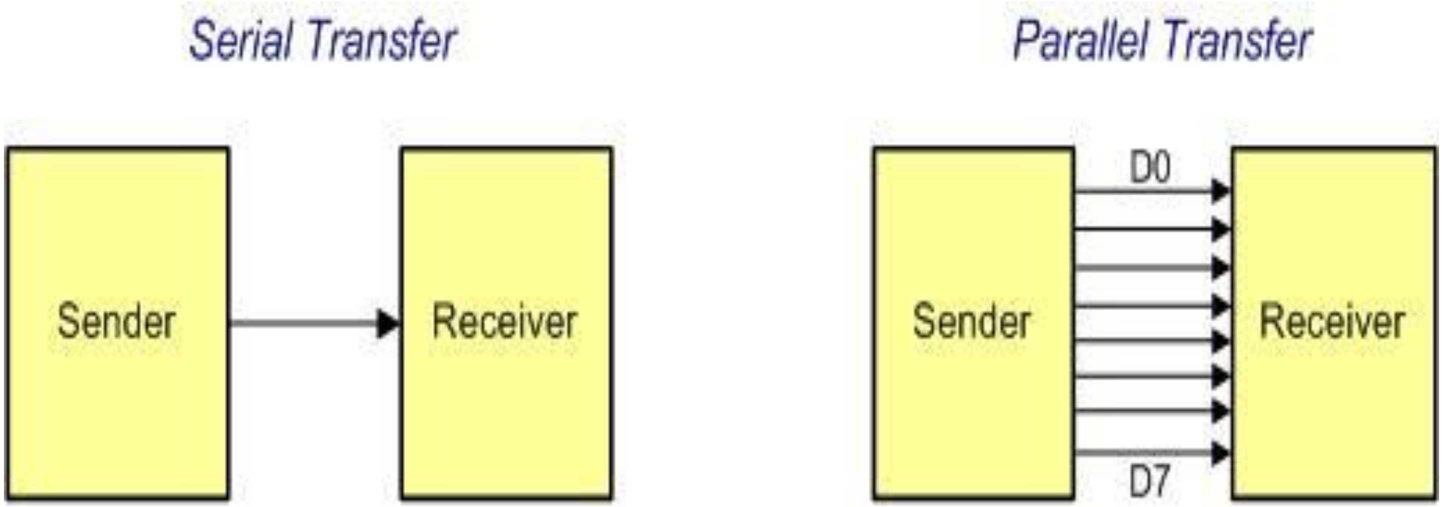
郭雪梅

Tel:39943108

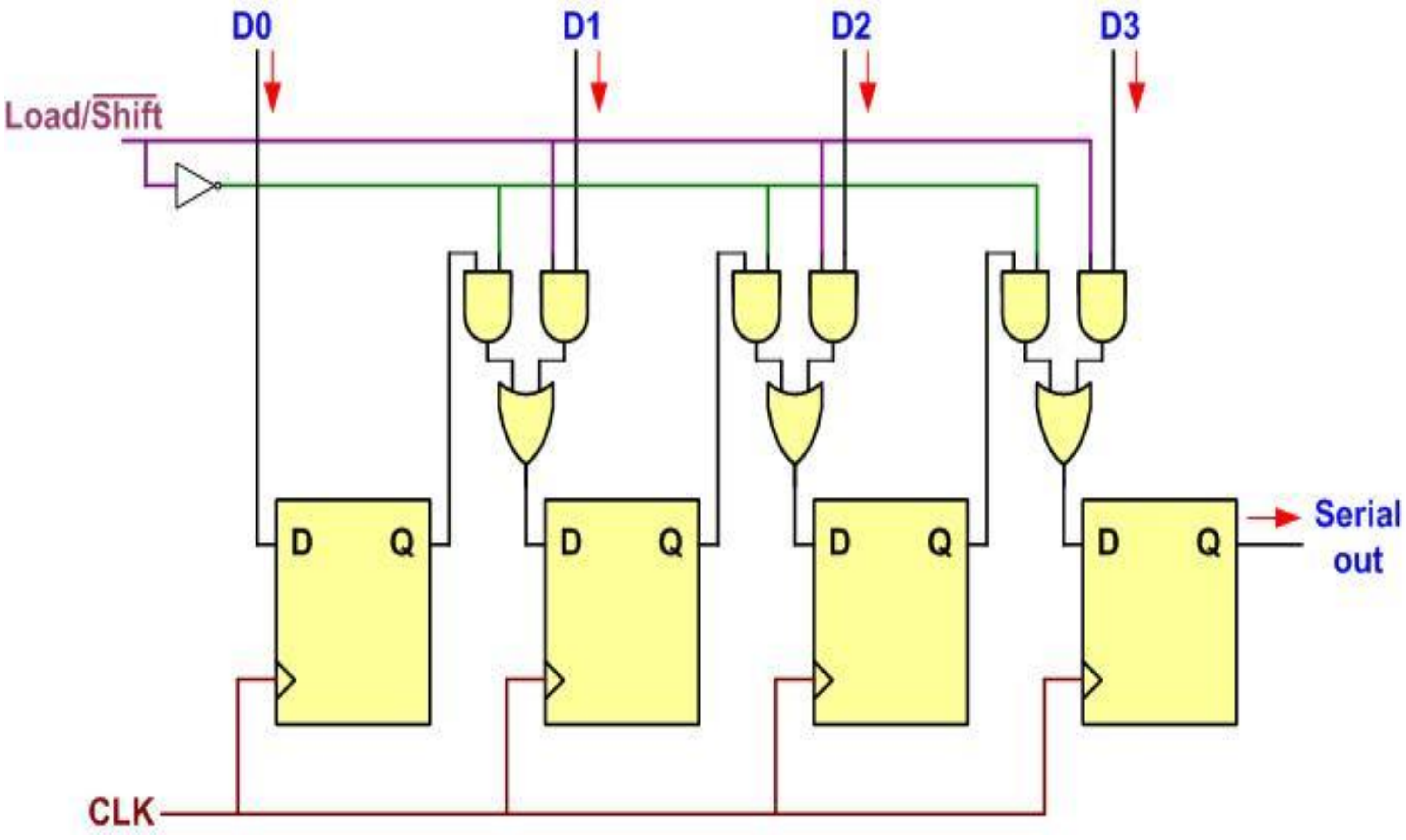
Email:guoxuem@mail.sysu.edu.cn

URL1：http://human-robot.sysu.edu.cn/course

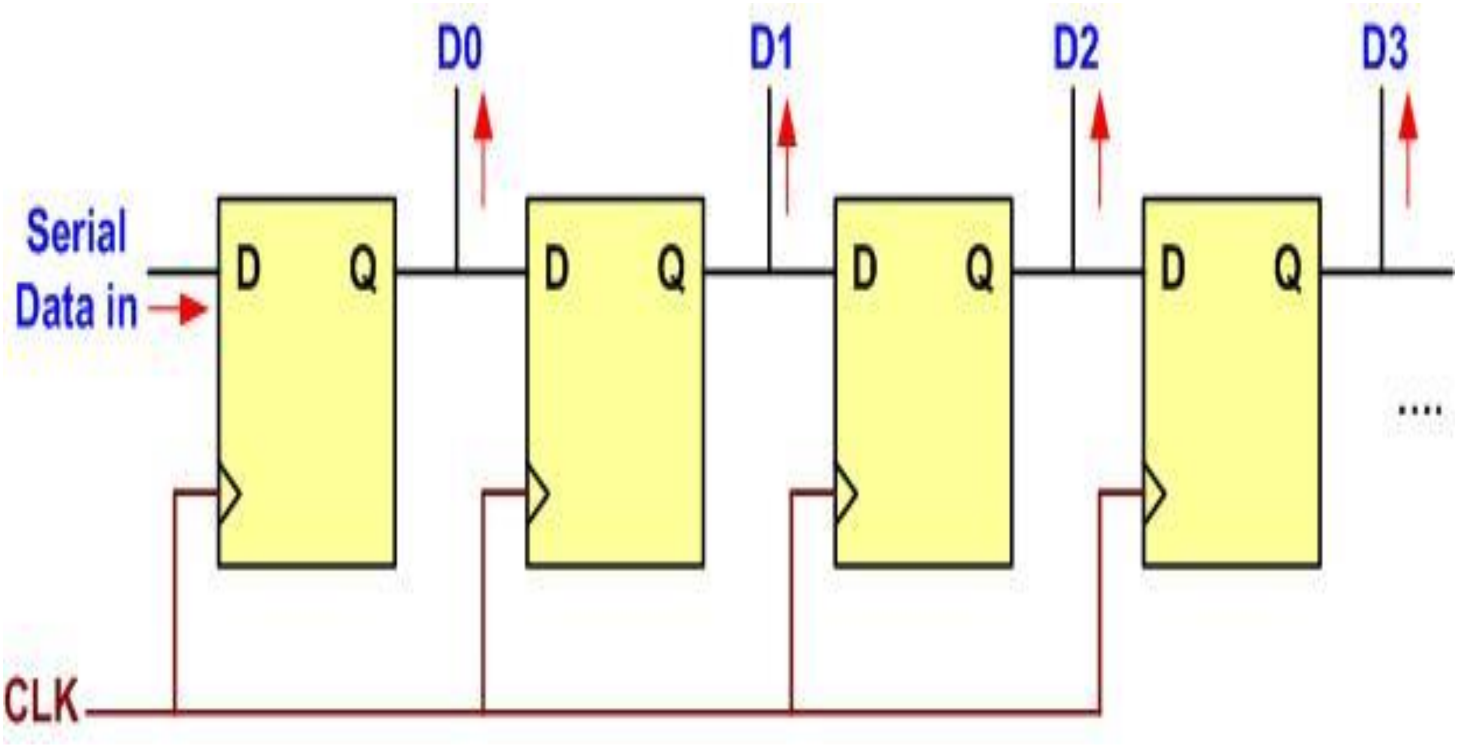
Serial vs. Parallel Data Transfer



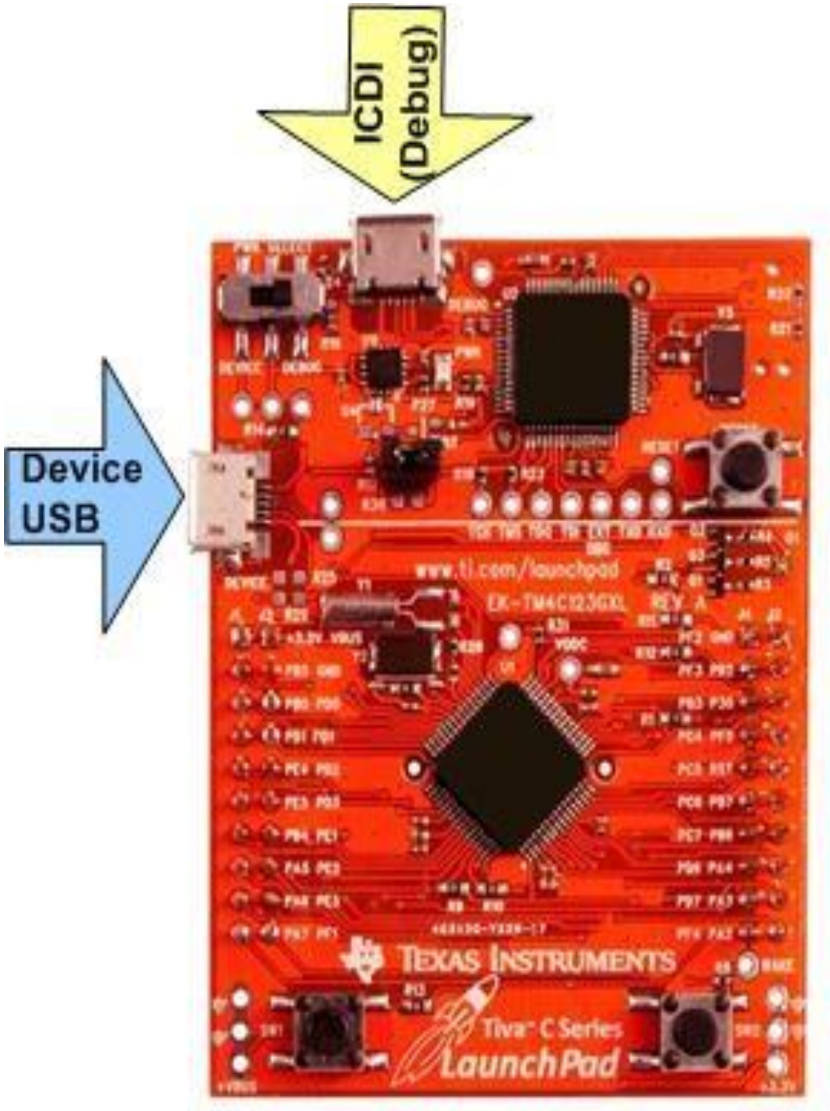
Parallel In Serial Out



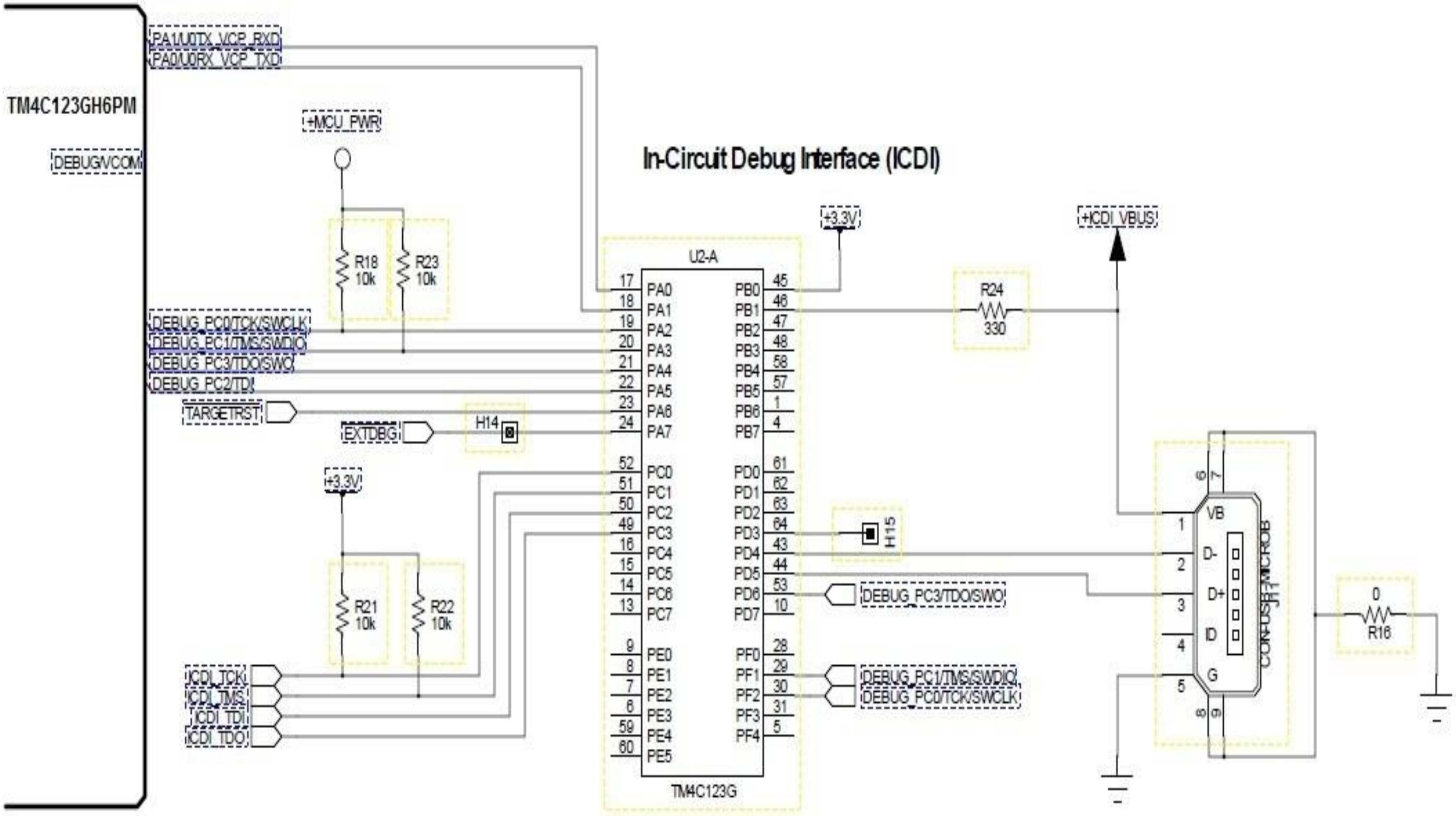
Serial In Parallel Out



Tiva LaunchPad

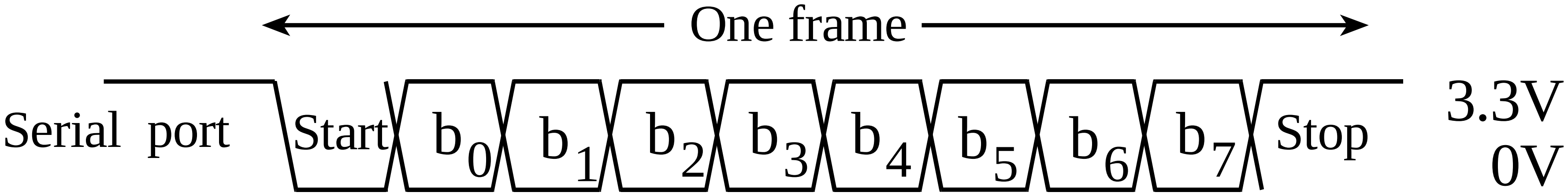


ICDI USB Port



Universal Asynchronous Receiver/Transmitter (UART)

UART (Serial Port) Interface



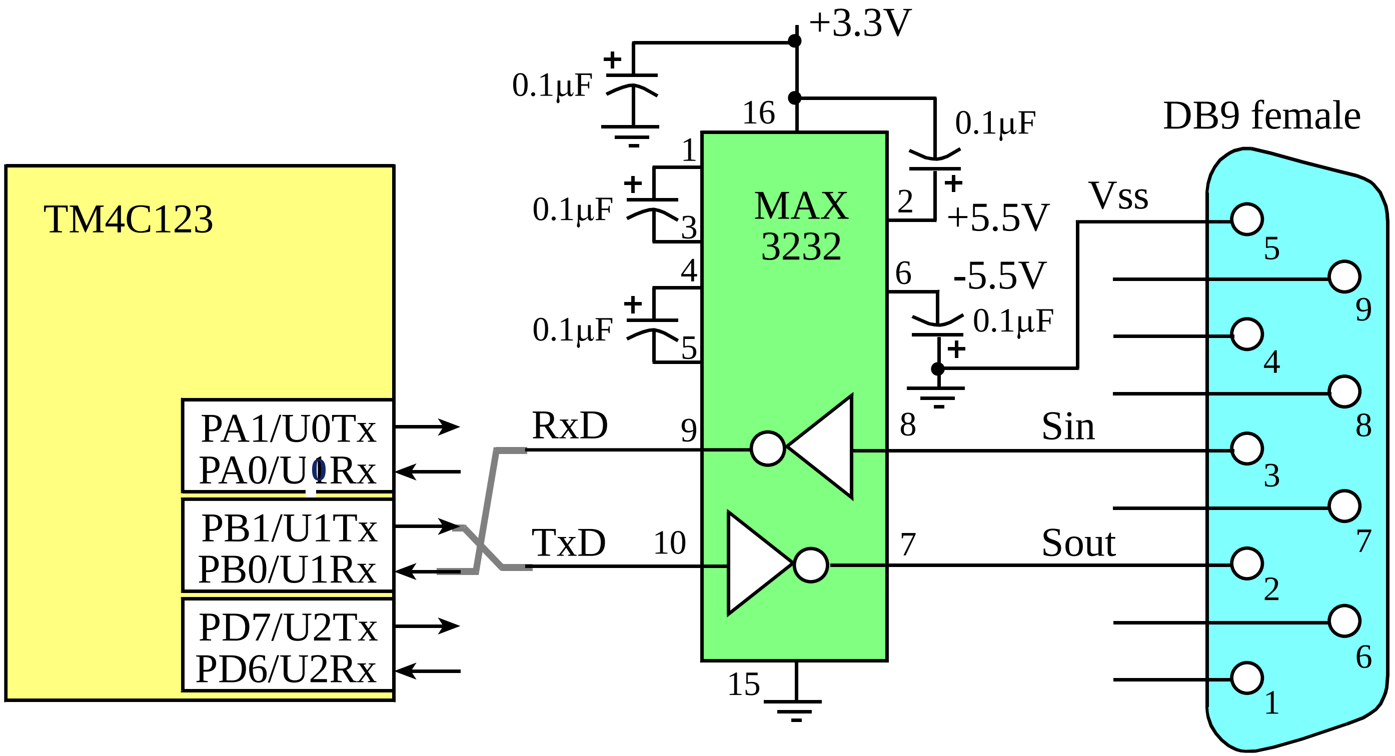
- Send/receive a *frame* of (5-8) data bits with a single (start) bit prefix and a 1 or 2 (stop) bit suffix
- Baud rate is total number of bits per unit time
  - Baudrate = 1 / bit-time
- Bandwidth is data per unit time
  - Bandwidth = (data-bits / frame-bits) * baudrate

TM4C123 LaunchPad I/O Pins

IO	Ain	0	1	2	3	4	5	6	7	8	9	14
PA2		Port		SSI0Clk								
PA3		Port		SSI0Fss								
PA4		Port		SSI0Rx								
PA5		Port		SSI0Tx								
PA6		Port			I$_2$C1SCL		M1PWM2					
PA7		Port			I$_2$C1SDA		M1PWM3					
PB0		Port	U1Rx						T2CCP0			
PB1		Port	U1Tx						T2CCP1			
PB2		Port			I$_2$C0SCL				T3CCP0			
PB3		Port			I$_2$C0SDA				T3CCP1			
PB4	Ain10	Port		SSI2Clk		M0PWM2			T1CCP0	CAN0Rx		
PB5	Ain11	Port		SSI2Fss		M0PWM3			T1CCP1	CAN0Tx		
PB6		Port		SSI2Rx		M0PWM0			T0CCP0			
PB7		Port		SSI2Tx		M0PWM1			T0CCP1			
PC4	C1-	Port	U4Rx	U1Rx		M0PWM6		IDX1	WT0CCP0	U1RTS		
PC5	C1+	Port	U4Tx	U1Tx		M0PWM7		PhA1	WT0CCP1	U1CTS		
PC6	C0+	Port	U3Rx					PhB1	WT1CCP0	USB0epen		
PC7	C0-	Port	U3Tx						WT1CCP1	USB0pflt		
PD0	Ain7	Port	SSI3Clk	SSI1Clk	I$_2$C3SCL	M0PWM6	M1PWM0		WT2CCP0			
PD1	Ain6	Port	SSI3Fss	SSI1Fss	I$_2$C3SDA	M0PWM7	M1PWM1		WT2CCP1			
PD2	Ain5	Port	SSI3Rx	SSI1Rx		M0Fault0			WT3CCP0	USB0epen		
PD3	Ain4	Port	SSI3Tx	SSI1Tx				IDX0	WT3CCP1	USB0pflt		
PD6		Port	U2Rx			M0Fault0		PhA0	WT5CCP0			
PD7		Port	U2Tx					PhB0	WT5CCP1	NMI		
PE0	Ain3	Port	U7Rx									
PE1	Ain2	Port	U7Tx									
PE2	Ain1	Port										
PE3	Ain0	Port										
PE4	Ain9	Port	U5Rx		I$_2$C2SCL	M0PWM4	M1PWM2			CAN0Rx		
PE5	Ain8	Port	U5Tx		I$_2$C2SDA	M0PWM5	M1PWM3			CAN0Tx		
PF0		Port	U1RTS	SSI1Rx	CAN0Rx		M1PWM4	PhA0	T0CCP0	NMI	C0o	
PF1		Port	U1CTS	SSI1Tx			M1PWM5	PhB0	T0CCP1		C1o	TRD1
PF2		Port		SSI1Clk		M0Fault0	M1PWM6		T1CCP0			TRD0
PF3		Port		SSI1Fss	CAN0Tx		M1PWM7		T1CCP1			TRCLK
PF4		Port					M1Fault0	IDX0	T2CCP0	USB0epen		

back

RS-232 Serial Port



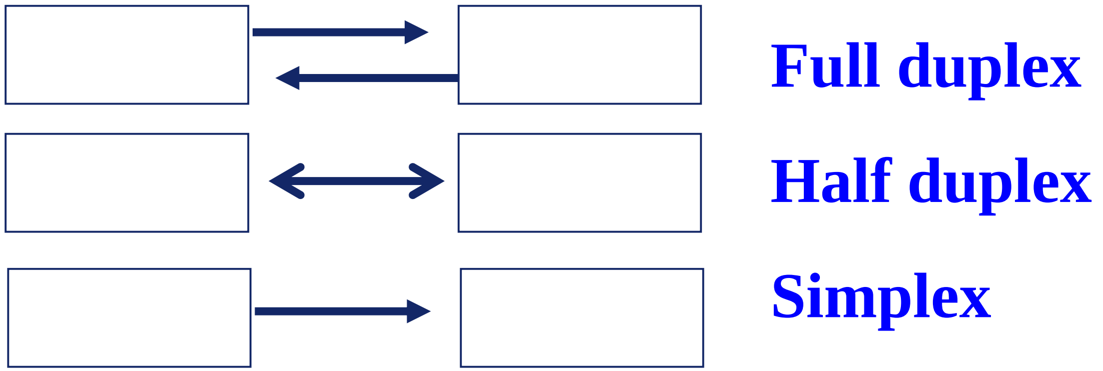
DB25 Pin	RS232 Name	DB9 Pin	EIA-574 Name	Signal	Description	True	DTE	DCE
2	BA	3	103	TxD	Transmit Data	-5.5V	out	in
3	BB	2	104	RxD	Receive Data	-5.5V	in	out
7	AB	5	102	SG	Signal Ground			

// this U1Tx PD3 not connected

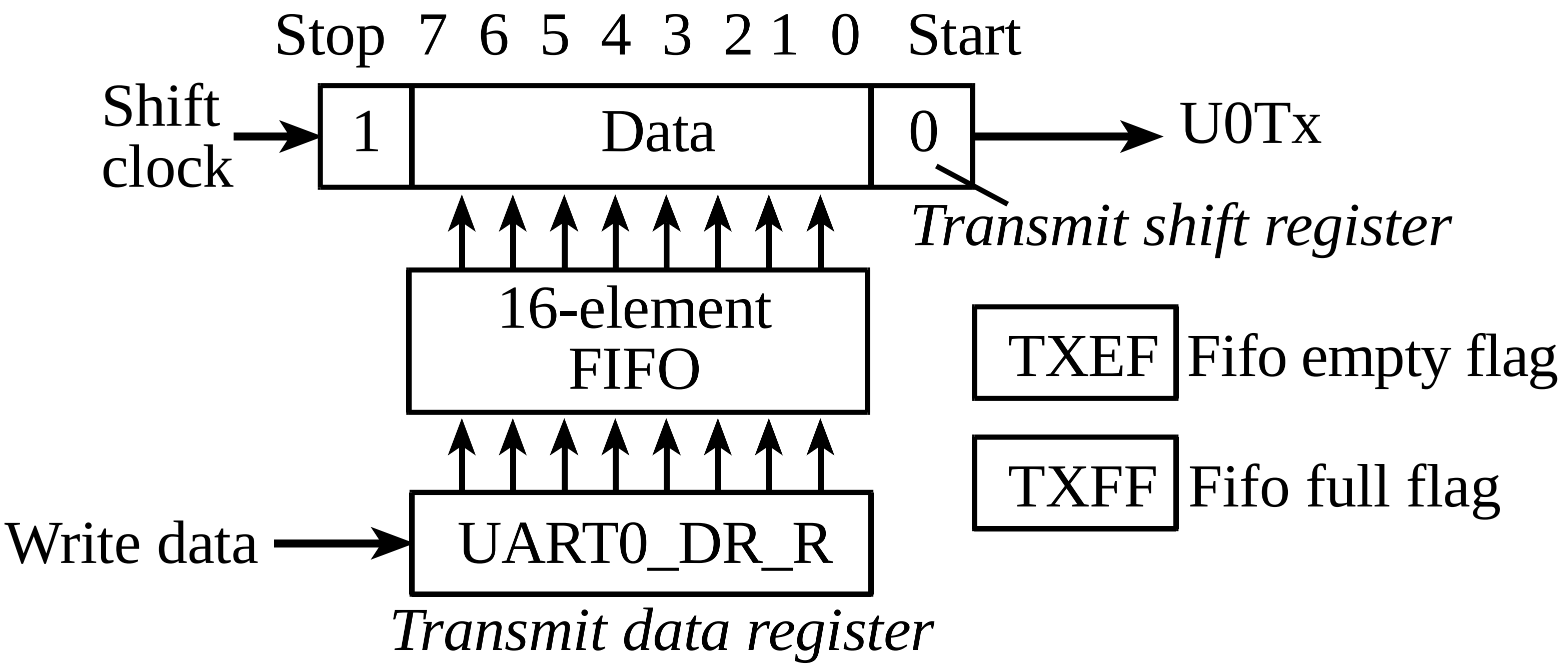
// this U1Rx PD2 tied to U1Tx PD3 of other microcontroller

Serial I/O

- Serial communication
  - Transmit Data (TxD), Receive Data (RxD), and Signal Ground (SG) implement *duplex* communication link
  - Both communicating devices must operate at the same bit rate
  - Least significant bit sent first



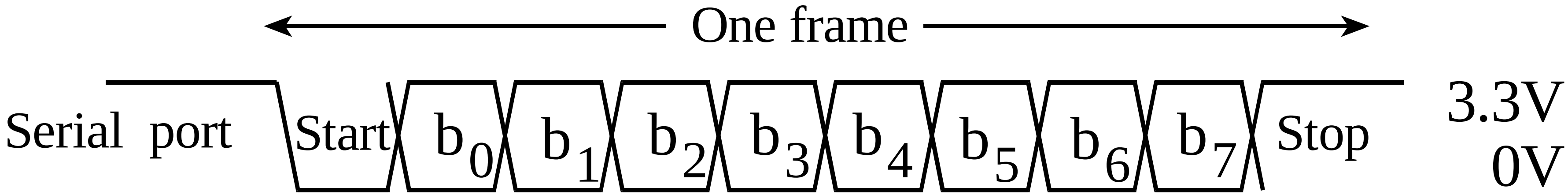
UART - Transmitter



发送器保持寄存器UART0_DR_R，写入该寄存器的值保存到发送FIFO中，当该字节到达FIFO底部时，它将被送入发送移位寄存器（U0TSR）进行发送。

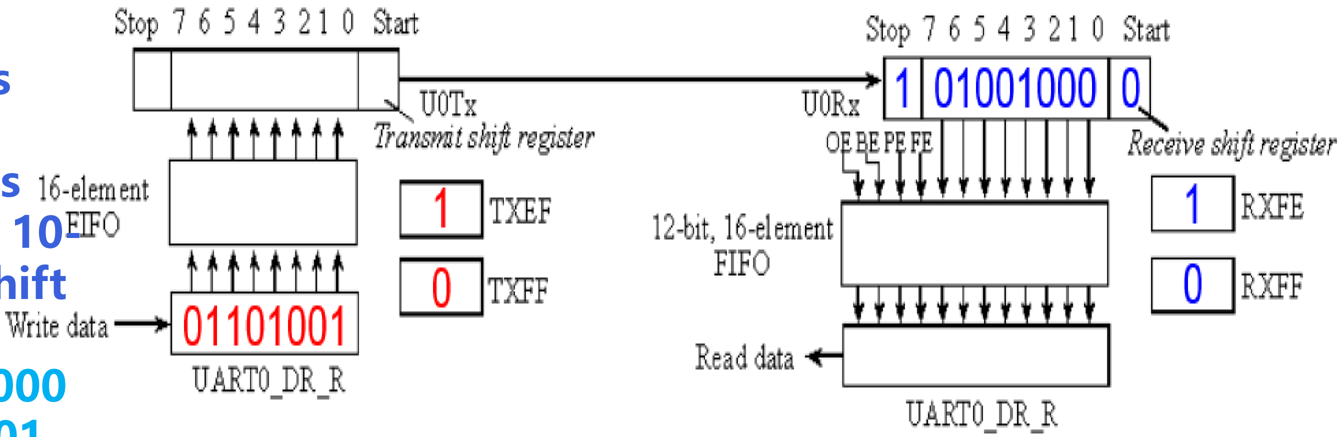
UART - Transmitter

- Tx Operation
  - Data written to UART0_DR_R
    - passes through 16-element FIFO
    - permits small amount of data rate matching between processor and UART
  - Shift clock is generated from 16x clock
    - permits differences in Tx and Rx clocks to be reconciled



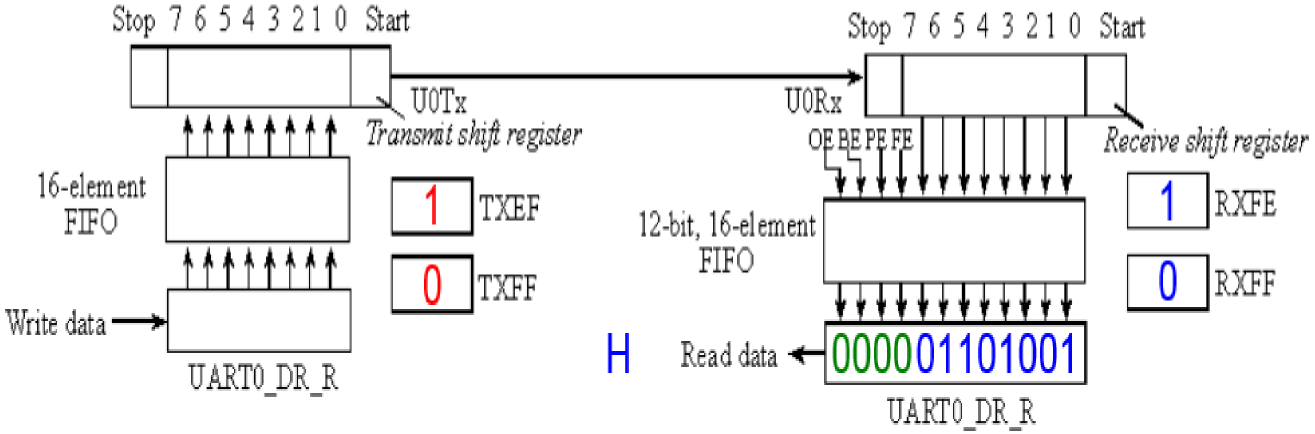
the UART gets data from the FIFO and loads them into the 10-bit transmit shift register

‘H’ =01001000
‘i’ =01101001



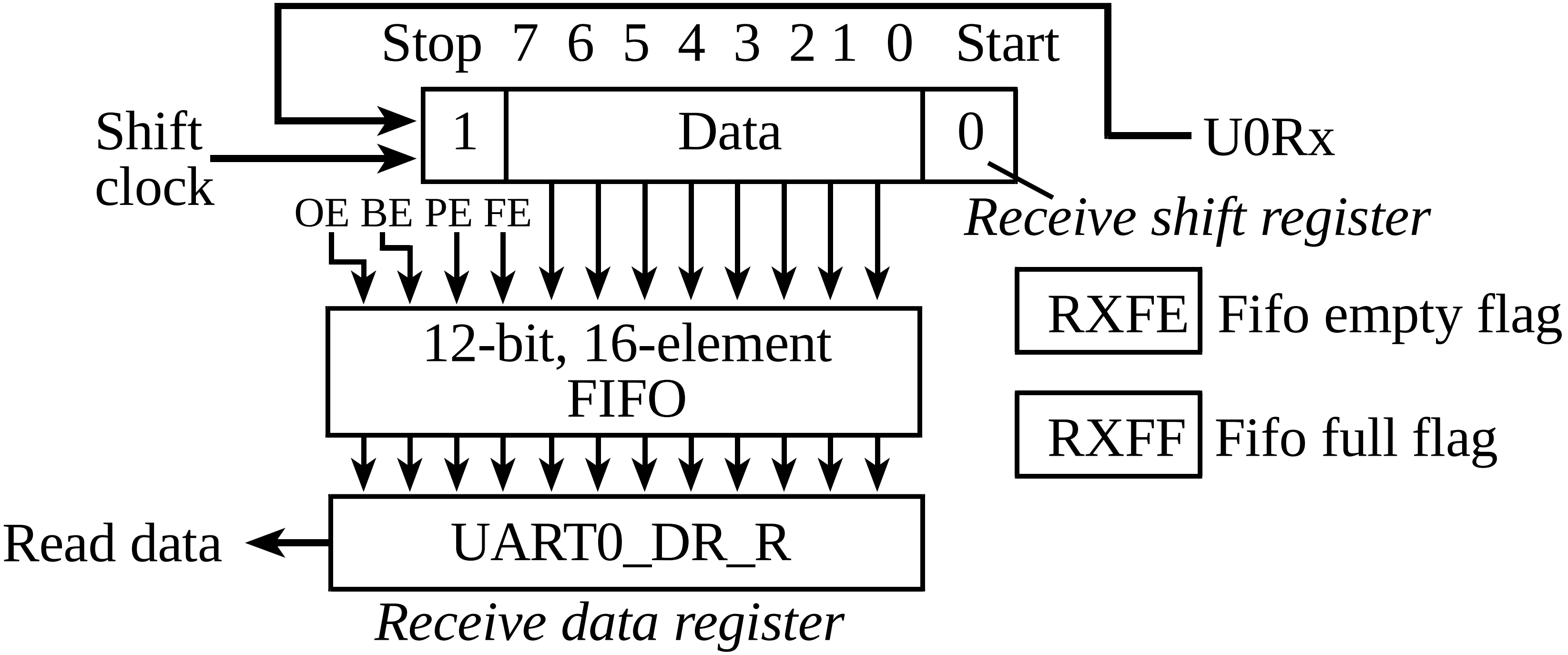
a new byte is written to UART0_DR_R it is put into the transmit FIFO

shift register includes a start bit, 8 data bits, and 1 stop bit.



The receiver recognizes a new frame by its start bit. The bits are shifted in using the same order as the transmitter shifted them out: start, b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, and then stop.

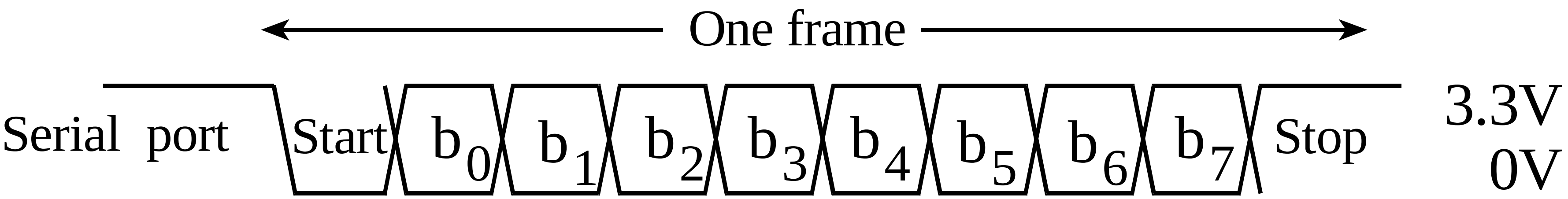
UART - Receiver



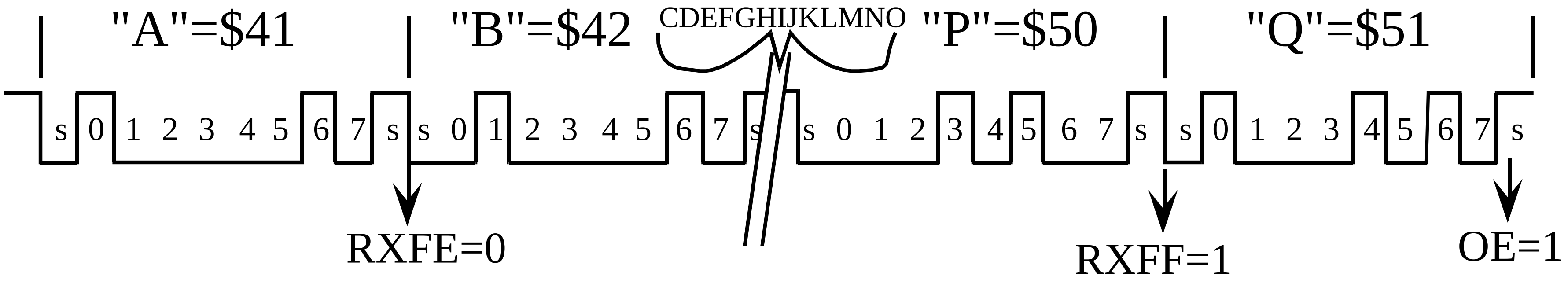
U0RSR移位寄存器从**RxD0**引脚接收的有效数据将被送到接收**FIFO**中。通过读取**U0RDR**寄存器可以将接收**FIFO**中最早接收到的字节读出，当**FIFO**中不再包含有效数据时，该寄存器反映接收到的最后一个有效字节数据。接收的数据不足**8**位时，高位用**0**填充。

UART - Receiver

- Rx Operation
  - RXFE is 0 when data are available
  - RXFF is 1 when FIFO is full
  - FIFO entries have four control bits
    - BE set when Tx signal held low for more than one frame (break)
    - OE set when FIFO is full and new frame has arrived
    - PE set if frame parity error
    - FE set if stop bit timing error



UART – Overrun Error



17 frames transmitted and none read => overrun error

TM4C UART0 – Registers

	31–12	11	10	9	8	7–0	Name
$4000.C000		OE	BE	PE	FE	DATA	UART0_DR_R

	31–3	3	2	1	0	
$4000.C004		OE	BE	PE	FE	UART0_RSR_R

	31–8	7	6	5	4	3	2–0	
$4000.C018		TXFE	RXFF	TXFF	RXFE	BUSY		UART0_FR_R

	31–16	15–0	
$4000.C024		DIVINT	UART0_IBRD_R

	31–6	5–0	
$4000.C028		DIVFRAC	UART0_FBRD_R

	31–8	7	6 – 5	4	3	2	1	0	
$4000.C02C		SPS	WPEN	FEN	STP2	EPS	PEN	BRK	UART0_LCRH_R

	31–10	9	8	7	6–3	2	1	0	
$4000.C030		RXE	TXE	LBE		SIRLP	SIREN	UARTEN	UART0_CTL_R

	31–6	5-3	2-0	
$4000.C034		RXIFLSEL	TXIFLSEL	UART0_IFLS_R

	31-11	10	9	8	7	6	5	4		
$4000.C038		OEIM	BEIM	PEIM	FEIM	RTIM	TXIM	RXIM		UART0_IM_R
$4000.C03C		OERIS	BERIS	PERIS	FERIS	RTRIS	TXRIS	RXRIS		UART0_RIS_R
$4000.C040		OEMIS	BEMIS	PEMIS	FEMIS	RTMIS	TXMIS	RXMIS		UART0_MIS_R
$4000.C044		OEIC	BEIC	PEIC	FEIC	RTIC	TXIC	RXIC		UART0_IC_R

back

The **UART0_IBRD_R** and **UART0_FBRD_R**

Specify the baud rate. The baud rate **divider** is a 22-bit binary fixed-point value with a resolution of 2^-6. The **Baud16** clock is created from the system bus clock, with a frequency of (Bus clock frequency)/**divider**. The baud rate is 16 times slower than **Baud16**

Baud rate = **Baud16/16** = (Bus clock frequency)/(16***divider**)

For example, if the bus clock is 8 MHz and the desired baud rate is 19200 bits/sec, then the **divider** should be 8,000,000/16/19200 or 26.04167, As a binary fixed-point number, this number is about 11010.000011. We can establish this baud rate by
putting the 11010 into **UART0_IBRD_R**
and the 000011 into **UART0_FBRD_R** .

TM4C UART Setup

UART0 operation

- UART clock started in SYSCTL_RCGCUART_R
- Digital port clock started in SYSCTL_RCGCGPIO_R
- UART0_CTL_R contains UART enable (UARTEN), Tx (TXE), and Rx enable (RXE)
  - set each to 1 to enable
  - UART disabled during initialization
- UART0_IBRD_R and UART_FBRD_R specify baud rate
  - bit rate = (bus clock frequency)/(16*divider)
  - ex: want 19.2 kb/s and bus clock is 8 MHz
  - 8 MHz/(16*19.2 k) = 26.04167 = 11010.000011_2
  - Tx and Rx clock rates must be within 5% to avoid errors
- GPIO_PORTB_AFSEL_R to choose alternate function
- Write appropriate values to GPIO_PORTB_PCTL_R (link)
- GPIO_PORTB_DEN_R Enable digital I/O on pins 1-0
- GPIO_PORTB_AMSEL_R no Analog I/O on pins 1-0
- write to UART0_LCRH_R to activate

UART 初始化

❿ **UARTLCRH**寄存器是线控寄存器。串行参数，例如数据长度、奇偶校验位和停止位的选择都是在该寄存器中完成的。

❿ 在更新波特率除数（**UARTIBRD**和/或**UARTIFRD**）时，还必须写**UARTLCRH**寄存器。波特率除数寄存器的写入选通（strobe）信号与**UARTLCRH**寄存器相连。

0x3 8位 0x2 7位
0x1 6位 0x0
5位（默认）

该位设为1，
停止位2位

该位设为1，那么奇偶校验使能
0奇校验，1偶校验

7	[6：5]	4	3	2	1	0
奇偶黏贴	字长	FIFO使能	双停止位选择	奇偶设置	奇偶校验位	发送终止

UART控制（**UARTCTL**），发送使能 (TXE、8位)和接受使能 (RXE、9位)位**0**位UART使能。

UART1 Device Driver on PC5 and PC4

IO	Ain	0	1	2	3	4	5	6	7	8	9	14
PC4	C1-	Port	U4Rx	U1Rx		M0PWM6		IDX1	WT0CCP0	U1RTS		
PC5	C1+	Port	U4Tx	U1Tx		M0PWM7		PhA1	WT0CCP1	U1CTS		

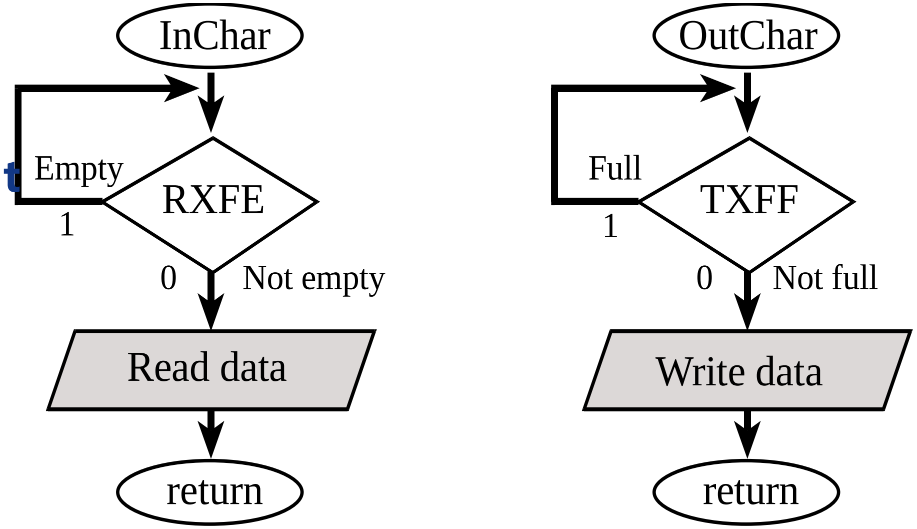
PCTL bits 5-4 are set to 0x22 to select U1Tx and U1Rx on PC5 and PC4. GPIO_PORTC_PCTL_R&0xFF00FFFF)+0x00220000

```
// Assumes a 80 MHz bus clock, creates 115200 baud rate
void UART_Init(void){           // should be called only once
  SYSCTL_RCGC1_R |= 0x00000002;  // activate UART1
  SYSCTL_RCGC2_R |= 0x00000004;  // activate port C
  UART1_CTL_R &= ~0x00000001;    // disable UART
  UART1_IBRD_R = 43;    // IBRD = int(80,000,000/(16*115,200)) = int(43.40278)
  UART1_FBRD_R = 26;     // FBRD = round(0.40278 * 64) = 26
  UART1_LCRH_R = 0x00000070;  // 8 bit, no parity bits, one stop, FIFOs
  UART1_CTL_R |= 0x00000001;     // enable UART
  GPIO_PORTC_AFSEL_R |= 0x30;    // enable alt funct on PC5-4
  GPIO_PORTC_DEN_R |= 0x30;      // configure PC5-4 as UART1
  GPIO_PORTC_PCTL_R = (GPIO_PORTC_PCTL_R&0xFF00FFFF)+0x00220000;
  GPIO_PORTC_AMSEL_R &= ~0x30;   // disable analog on PC5-4
}
```

UART Synchronization

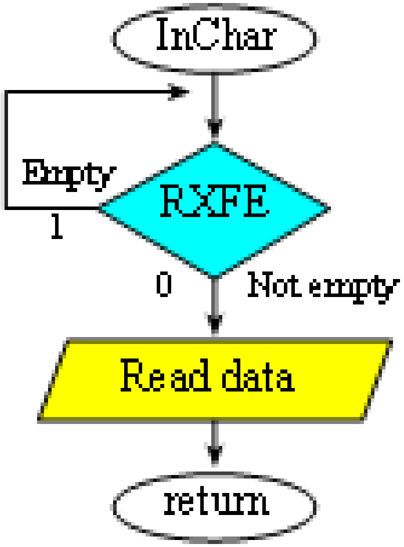
```
// Wait for new input, then return ASCII code
unsigned char UART_InChar(void){
  while((UART1_FR_R&0x0010) != 0);    // wait
until RXFE is 0
  return((unsigned
char)(UART1_DR_R&0xFF));
}
// Wait for buffer to be not full, then output
void UART_OutChar(unsigned char data){
  while((UART1_FR_R&0x0020) != 0);      //
wait until TXFF is 0
  UART1_DR_R = data;
}
// Immediately return input or 0 if no input
unsigned char
UART_InCharNonBlocking(void){
  if((UART1_FR_R&UART_FR_RXFE) == 0){
    return((unsigned char)(UART1_DR_R&0xFF));
  } else{
    return 0;
  }
}
```

❿ Busy-wait operation

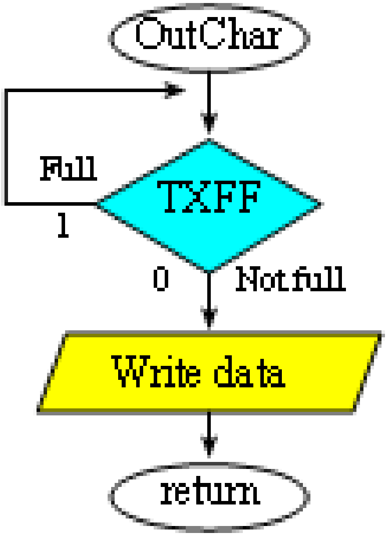


UART Busy-Wait Send/Recv

```
// Wait for new input,
// then return ASCII code
uint8_t UART_InChar(void) {
  while((UART1_FR_R&0x0010) != 0);
  // wait until RXFE is 0
  return((uint8_t)(UART1_DR_R&0xFF));
}
```



```
// Wait for buffer to be not full,
// then output
void UART_OutChar(uint8_t data) {
  while((UART1_FR_R&0x0020) != 0);
  // wait until TXFF is 0
  UART1_DR_R = data;
}
```



UART Interrupts

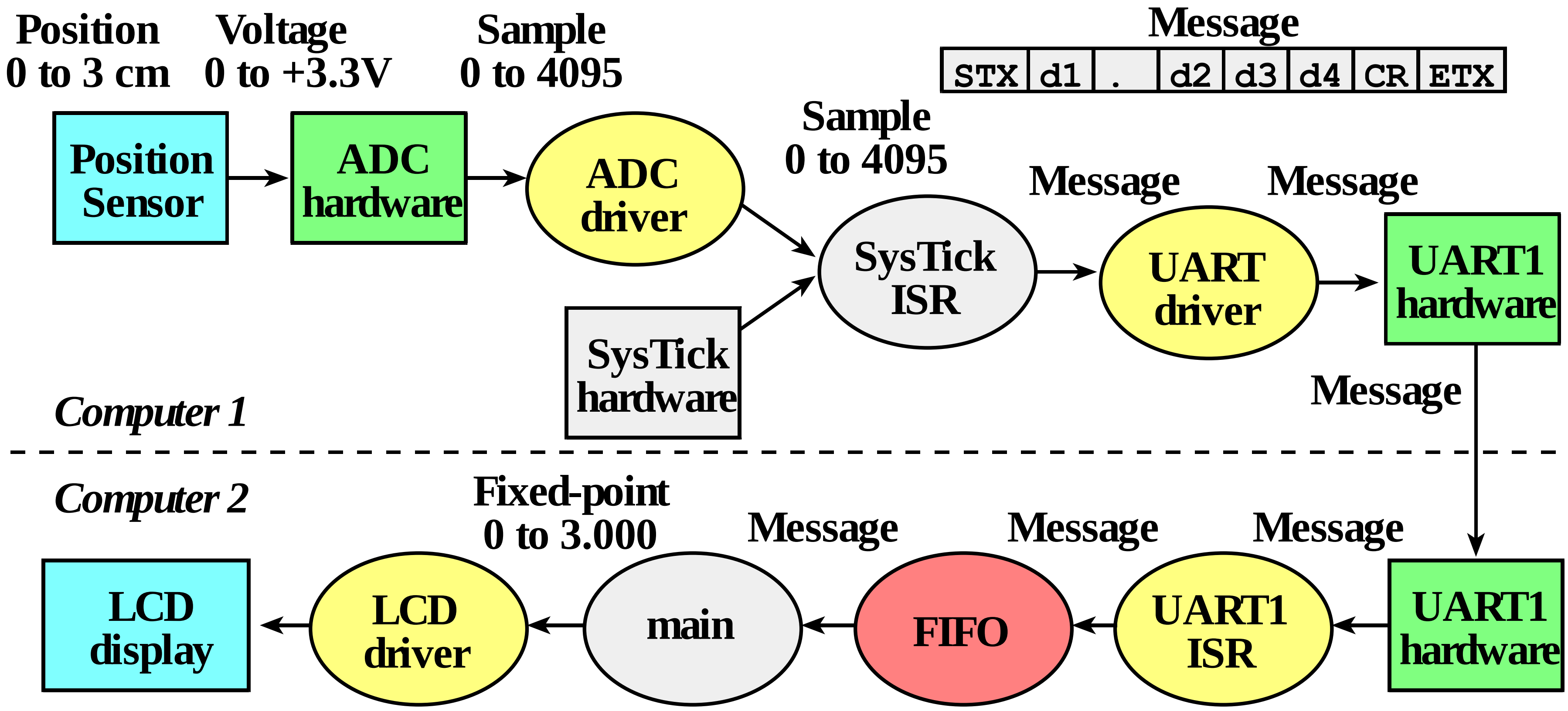
❿ UARTx_IFLS_R register (bits 5,4,3)

RXIFLSEL	RX FIFO	Set RXRIS interrupt trigger when
0x0	≥ ⅛ full	Receive FIFO goes from 1 to 2 characters
0x1	≥ ¼ full	Receive FIFO goes from 3 to 4 characters
0x2	≥ ½ full	Receive FIFO goes from 7 to 8 characters
0x3	≥ ¾ full	Receive FIFO goes from 11 to 12 characters
0x4	≥ ⅞ full	Receive FIFO goes from 13 to 14 characters

TXIFLSEL	TX FIFO	Set TXRIS interrupt trigger when
0x0	≤ ⅞ empty	Transmit FIFO goes from 15 to 14 character
0x1	≤ ¾ empty	Transmit FIFO goes from 13 to 12 character
0x2	≤ ½ empty	Transmit FIFO goes from 9 to 8 characters
0x3	≤ ¼ empty	Transmit FIFO goes from 5 to 4 characters
0x4	≤ ⅛ empty	Transmit FIFO goes from 3 to 2 characters

UART中断的**FIFO**深度选择 （**UARTx_IFLS_R**）

Lab 9 – Distributed Measurement



Position
0 to 3 cm

Voltage
0 to +3.3V

Sample
0 to 4095

Message

STX	d1	.	d2	d3	d4	CR	ETX

Position Sensor

ADC hardware

ADC driver

Sample
0 to 4095

SysTick ISR

SysTick hardware

Message

UART driver

Message

UART1 hardware

Message

Computer 1

Computer 2

Fixed-point
0 to 3.000

Message

Message

Message

LCD display

LCD driver

main

FIFO

UART1 ISR

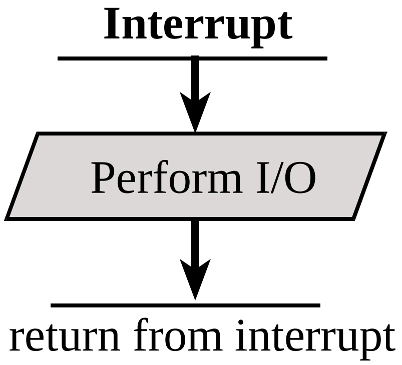
UART1 hardware

Lab9: Transmitter SysTick ISR

- ❑ Toggle heartbeat
- ❑ Sample ADC
- ❑ Toggle heartbeat
- ❑ Convert to integer part of fixed point
- ❑ Send message, 8 calls to `UART_OutChar`
  - ❖ STX
  - ❖ Ones digit
  - ❖ Decimal point
  - ❖ Tenths digit
  - ❖ Hundreds digit
  - ❖ Thousandth digit
  - ❖ CR
  - ❖ ETX
- ❑ Toggle heartbeat

Busy-wait version

Busy-wait version



Lab9: UART Rx Interrupt

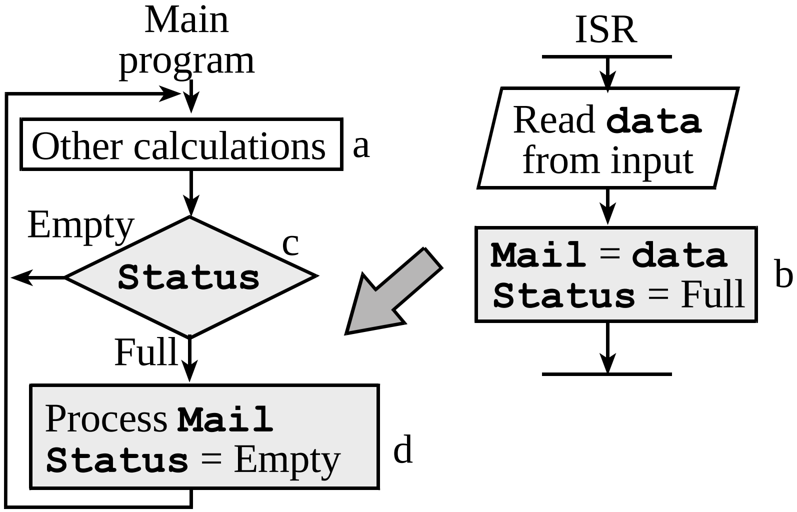
- **Interrupt Trigger, sets RXRIS**
  - **Receive FIFO has gone from 7 to 8 elements (1/2 full)**
- **Initialization (add these)**
  - **Arm RXRIS** `UART1_IM_R |= 0x10;`link
  - **Set** `UART1_IFLS_R` **bits 5,4,3 to 010 (1/2 full)**
  - `NVIC_PRI1_R   // bits 21-23`
  - `NVIC_EN0_R // enable interrupt 6 in NVIC`
- **Interrupt vector in startup.s**
  - **Name ISR** `UART1_Handler`
- **Acknowledge (in ISR)**
  - `UART1_ICR_R = 0x10;`

Lab9: Interrupt+Mailbox?

Background thread

❿ RXRIS ISR

- Read UART1_DR_R
- Store in RXmail
- Set RXstatus



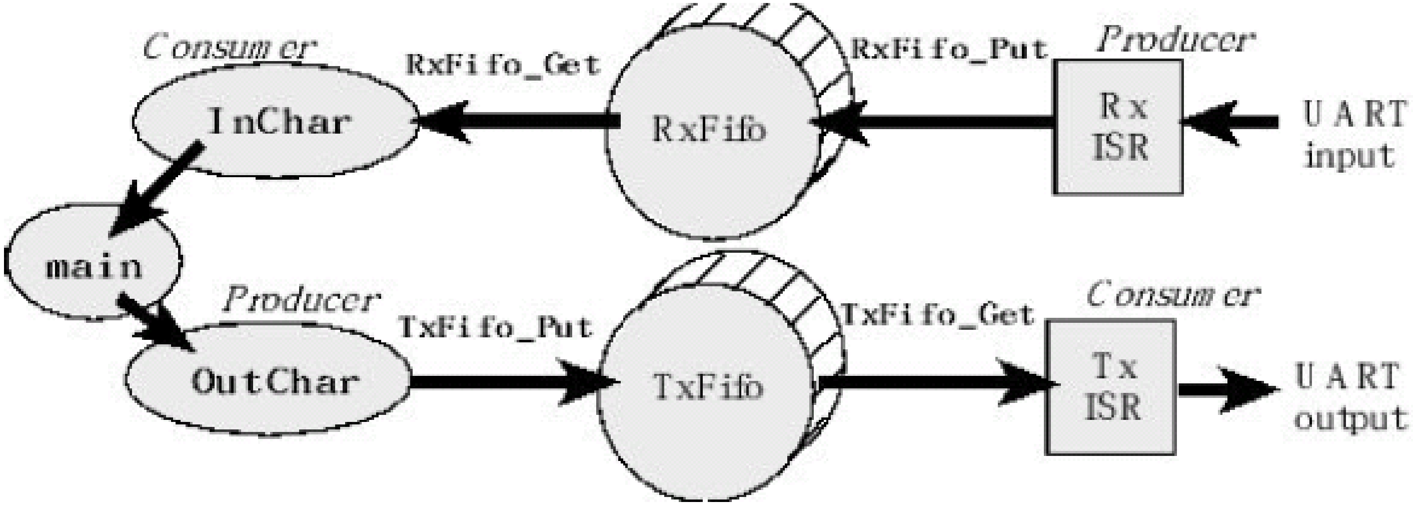
Foreground thread

❑ Main loop

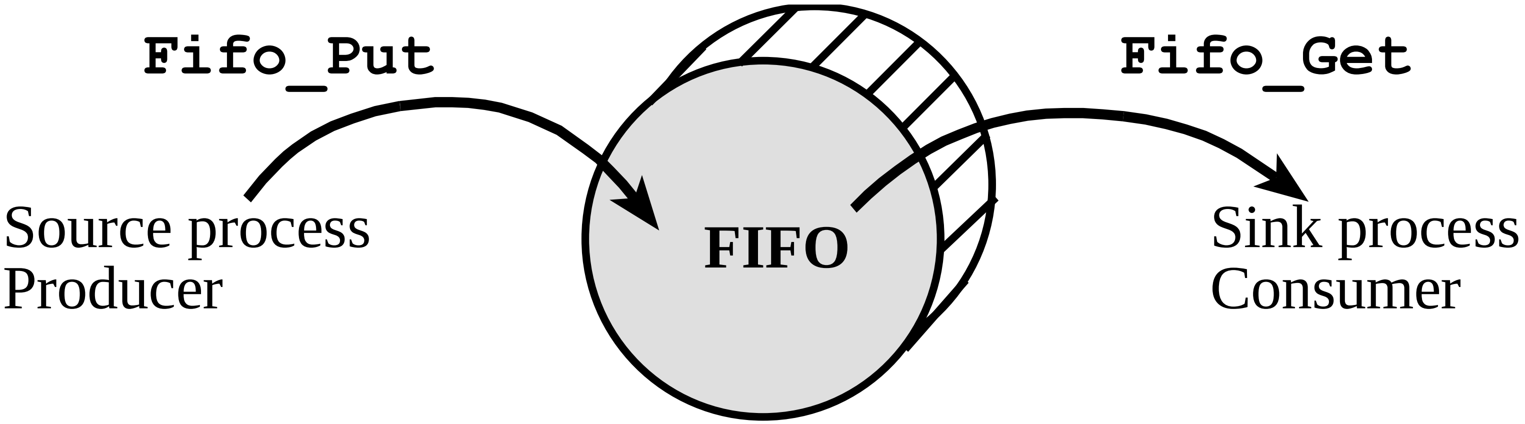
- ❖ Wait for RXstatus
- ❖ Read RXmail
- ❖ Clear RXstatus
- ❖ Convert to distance
- ❖ Display on LCD

What can go wrong?

Without the **FIFO** we would have to produce one piece of data, then process it, produce another piece of data, then process it. With the **FIFO**, the **producer** thread can continue to produce data without having to wait for the **consume**r to finish processing the previous data.

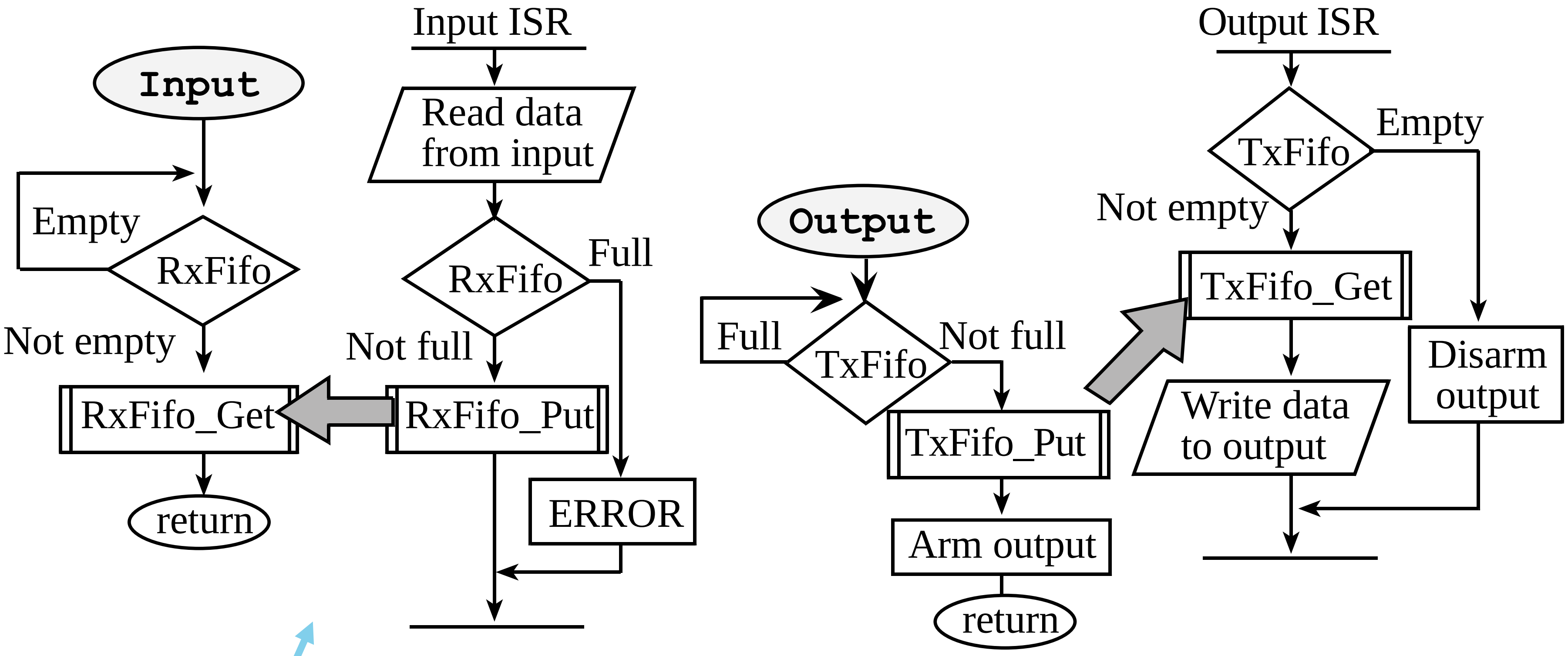


First-In/First-Out (FIFO) Queues



- ❑ **Order preserving**
- ❑ **Producer(s) put (on tail end)**
- ❑ **Consumer(s) get (from head end)**
- ❑ **Buffer decouples producer & consumer**
  - ❖ **Even out temporary mismatch in rates**

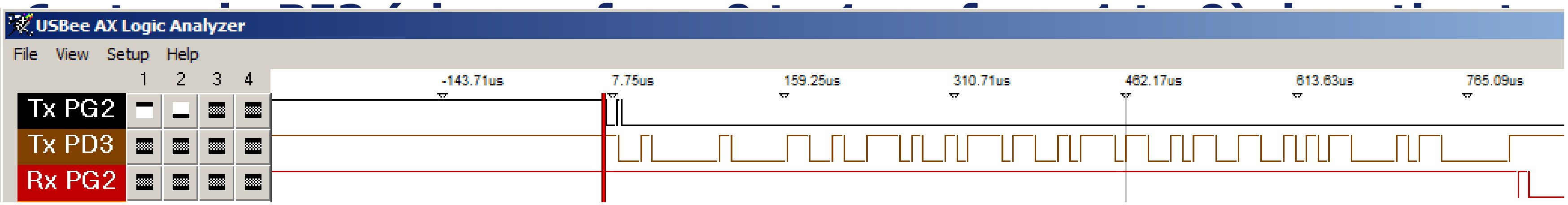
FIFO Queue Synchronization



Lab 9

Lab 9 - RXRIS ISR

1. **toggle PF2 (change from 0 to 1, or from 1 to 0), heartbeat**
2. **toggle PF2 (change from 0 to 1, or from 1 to 0), heartbeat**
3. **as long as the RXFE bit in the UART1_FR_R is zero**
    - **Read bytes from UART1_DR_R**
    - **Put all bytes into your software FIFO,** `RxFifo_Put`
    - **Should be exactly 8 bytes, but could be more possibly**
    - **If your software FIFO is full (data lost)**
      increment a global error count (but don’t loop back)
    - **The message will be interpreted in the main program**
4. **Increment a Counter,**
    - **debugging monitor of the number of UART messages received**
5. **acknowledge the interrupt by clearing the flag which requested it**
    - `UART1_ICR_R = 0x10;` **// clears bit 4 (RXRIS) in RIS register**
6. **toggle PF2 (change from 0 to 1, or from 1 to 0), heartbeat**



Interrupt-driven device driver for the UART uses two hardware FIFOs and two software FIFOs to buffer data

```
#define FIFOSIZE 16 // size of the FIFOs (must be power of 2)
#define FIFOSUCCESS 1 // return value on success
#define FIFOFAIL 0 // return value on failure
AddIndexFifo(Rx, FIFOSIZE, char, FIFOSUCCESS, FIFOFAIL)
AddIndexFifo(Tx, FIFOSIZE, char, FIFOSUCCESS, FIFOFAIL)
void UART_Init(void){ // should be called only once
SYSCTL_RCGCUART_R |= 0x01; // activate UART0
SYSCTL_RCGCGPIO_R |= 0x01; // activate port A
RxFifo_Init(); // initialize empty FIFOs
TxFifo_Init();
UART0_CTL_R &= ~UART_CTL_UARTEN; // disable UART
UART0_IBRD_R = 27; // IBRD=int(50,000,000/(16*115,200)) = int(27.1267)
UART0_FBRD_R = 8; // FBRD = round(0.1267 * 64) = 8
UART0_LCRH_R = (UART_LCRH_WLEN_8|UART_LCRH_FEN); // 8-bit word, FIFOs
UART0_IFLS_R &= ~0x3F; // clear TX and RX interrupt FIFO level fields
```

configure interrupt for TX FIFO <= 1/8 full
configure interrupt for RX FIFO >= 1/8 full

- UART0_IFLS_R += (UART_IFLS_TX1_8|UART_IFLS_RX1_8);
- // enable TX and RX FIFO interrupts and RX time-out interrupt
- UART0_IM_R |= (UART_IM_RXIM|UART_IM_TXIM|UART_IM_RTIM);
- UART0_CTL_R |= 0x0301; // enable RXE TXE UARTEN
- GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R&0xFFFFFF00)+0x00000011; // UART
- GPIO_PORTA_AMSEL_R &= ~0x03; // disable analog function on PA1-0
- GPIO_PORTA_AFSEL_R |= 0x03; // enable alt funct on PA1-0
- GPIO_PORTA_DEN_R |= 0x03; // enable digital I/O on PA1-0
- NVIC_PRI1_R = (NVIC_PRI1_R&0xFFFF00FF)|0x00004000; // UART0=priority 2
- NVIC_EN0_R = NVIC_EN0_INT5; // enable interrupt 5 in NVIC
- EnableInterrupts();
- }

- // copy from hardware RX FIFO to software RX FIFO
- // stop when hardware RX FIFO is empty or software RX FIFO is full
- void static copyHardwareToSoftware(void){ char letter;
- while(((UART0_FR_R&UART_FR_RXFE)==0)&&(RxFifo_Size() < (FIFOSIZE-1))){
- letter = UART0_DR_R;
- RxFifo_Put(letter);
- }
- } // copy from software TX FIFO to hardware TX FIFO
- // stop when software TX FIFO is empty or hardware TX FIFO is full
- void static copySoftwareToHardware(void){ char letter;
- while(((UART0_FR_R&UART_FR_TXFF) == 0) && (TxFifo_Size() > 0)){
- TxFifo_Get(&letter);
- UART0_DR_R = letter;
- }

```
} // input ASCII character from UART
// spin if RxFifo is empty
char UART_InChar(void){
char letter;
while(RxFifo_Get(&letter) == FIFOFAIL){};
return(letter);
} // output ASCII character to SCI
// spin if TxFifo is full
void UART_OutChar(char data){
while(TxFifo_Put(data) == FIFOFAIL){};
UART0_IM_R &= ~UART_IM_TXIM; // disable TX FIFO interrupt
copySoftwareToHardware();
UART0_IM_R |= UART_IM_TXIM; // enable TX FIFO interrupt
// at least one of three things has happened:
// hardware TX FIFO goes from 3 to 2 or less items
// hardware RX FIFO goes from 1 to 2 or more items
// UART receiver has timed out
```

```
void UART0_Handler(void){
if(UART0_RIS_R&UART_RIS_TXRIS){ // hardware TX FIFO <= 2 items
UART0_ICR_R = UART_ICR_TXIC; // acknowledge TX FIFO
// copy from software TX FIFO to hardware TX FIFO
copySoftwareToHardware();
if(TxFifo_Size() == 0){ // software TX FIFO is empty
UART0_IM_R &= ~UART_IM_TXIM; // disable TX FIFO interrupt}
}
if(UART0_RIS_R&UART_RIS_RXRIS){ // hardware RX FIFO >= 2 items
UART0_ICR_R = UART_ICR_RXIC; // acknowledge RX FIFO
// copy from hardware RX FIFO to software RX FIFO
copyHardwareToSoftware();}
if(UART0_RIS_R&UART_RIS_RTRIS){ // receiver timed out
UART0_ICR_R = UART_ICR_RTIC; // acknowledge receiver time out
// copy from hardware RX FIFO to software RX FIFO
copyHardwareToSoftware();
}
}
```

FIFO Full Errors

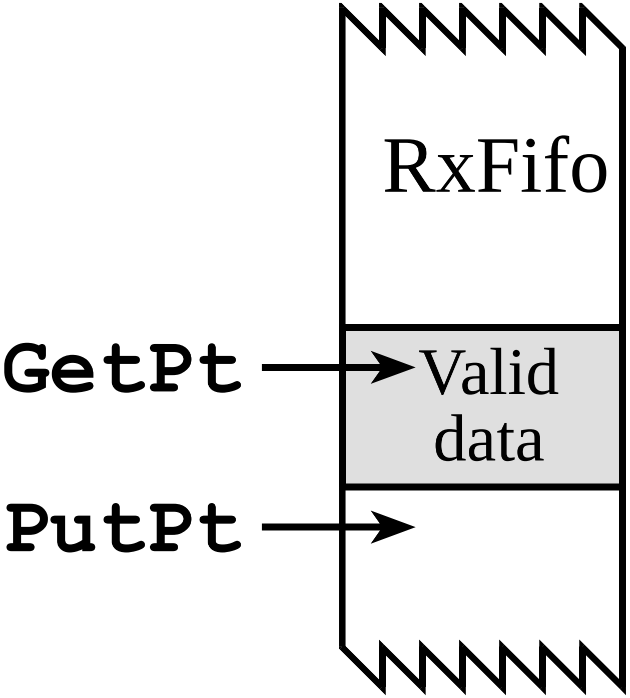
Average producer rate exceeds the average consumer rate

- Sample ADC every 50 ms
- Average time to process the sample is 51 ms
- Solution: decrease producer rate or increase consumer rate
  - Lower sampling rate
  - Faster computer
  - More efficient compiler
  - Rewrite time-critical code in assembly
  - More computers (distributed processing)

Producer rate temporarily exceeds the consumer rate

- ❑ Sample ADC every 50 ms
- ❑ Every 100th sample it takes 1 sec to process
- ❑ Solution: increase FIFO queue size

FIFO Queue Implementation



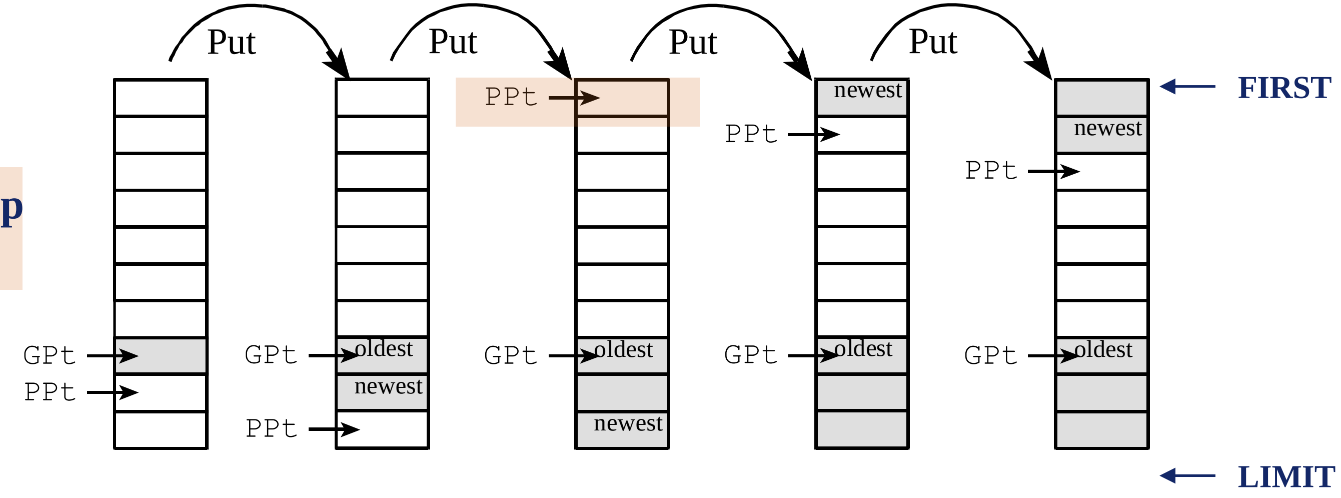
- ❑ PutPt
- ❑ GetPt

FIFO Full/Empty Conditions

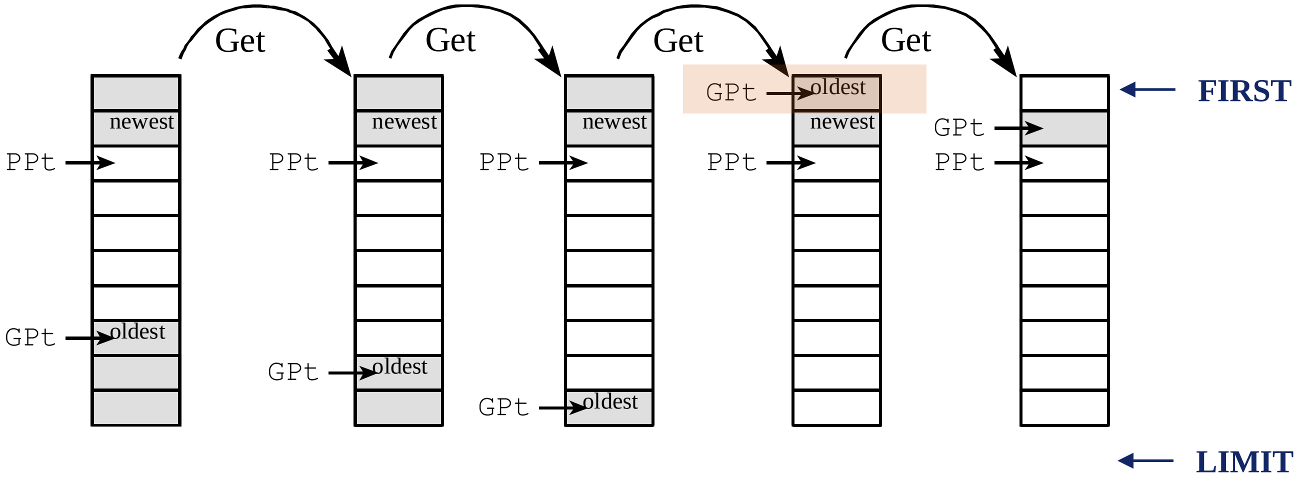
- FIFO Parameter Relations
  - Buffer is EMPTY
    - PutPt equals GetPt
  - Buffer is FULL
    - PutPt + 1 equals GetPt
      - note that there is no data stored at PutPt
      - as a result, if N locations are allocated for a buffer, only N-1 data elements will fill the buffer

FIFO Wrapping

Pointer wrap on 2nd put



Pointer wrap on 4th get



FIFO Queue

- FIFO Implementations
  - FIFO_Put
    - stores a single value on the FIFO queue
      - operates with interrupts disabled
      - updates PutPt
        - detects buffer full condition
      - handles transition from LIMIT-1 to FIRST
  - FIFO_Get
    - reads a single value from the FIFO queue
      - operates with interrupts disabled
      - updates GetPt
        - detects buffer empty condition
      - handles transition from LIMIT-1 to FIRST

FIFO in C

```
#define FIFO_SIZE 10
int32_t static *PutPt;
int32_t static *GetPt;
int32_t static Fifo[FIFO_SIZE];

void Fifo_Init(void){
  PutPt = GetPt = &Fifo[0];
}
```

`static` means private to this file

FIFO Routines in C

```
int Fifo_Put(int32_t data)
{
int32_t *tempPt;
  tempPt = PutPt+1;    // see if there is room
  if(tempPt==&Fifo[FIFO_SIZE]){
    tempPt = &Fifo[0];
  }
  if(tempPt == GetPt){
    return(0);         // full!
  }
  else{
    *(PutPt) = data;   // save
    PutPt = tempPt;    // OK
    return(1);
  }
}
```

Actually plus 4 bytes

FIFO Routines in C

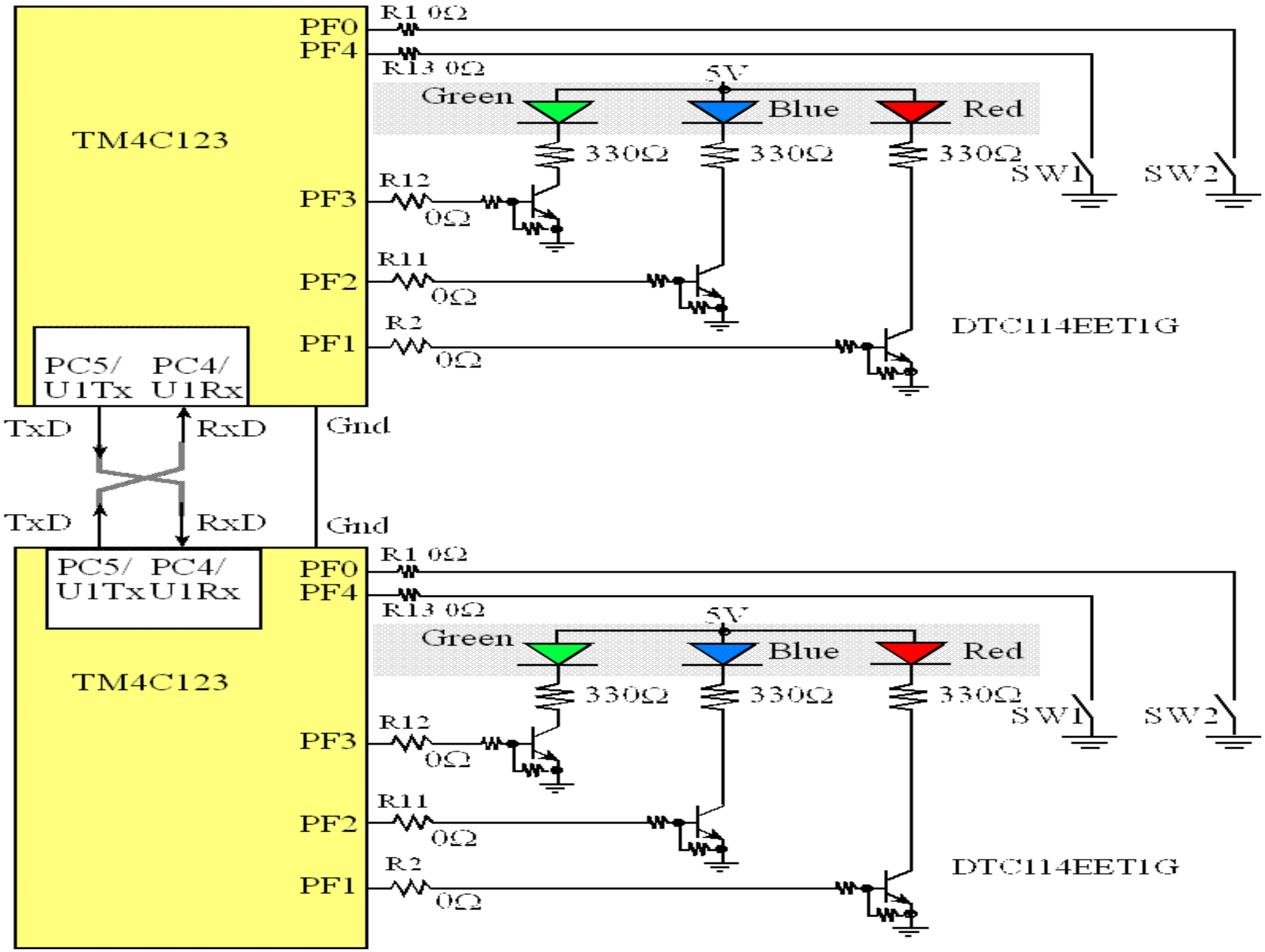
```
int Fifo_Get(int32_t *datapt){

  if(PutPt == GetPt){
    return(0);   // Empty
  }
  else{
    *datapt = *(GetPt++);
    if(GetPt==&Fifo[FIFO_SIZE]){
       GetPt = &Fifo[0];
    }
    return(1);
  }
}
```

Actually plus 4 bytes

UART_4C123

```
// U0Rx (VCP receive) connected to PA0
// U0Tx (VCP transmit) connected to PA1
#include <stdint.h>
#include "UART.h"
#include "inc/tm4c123gh6pm.h"
#define UART_FR_TXFF            0x00000020  // UART Transmit FIFO Full
#define UART_FR_RXFE            0x00000010  // UART Receive FIFO Empty
#define UART_LCRH_WLEN_8        0x00000060  // 8 bit word length
#define UART_LCRH_FEN           0x00000010  // UART Enable FIFOs
#define UART_CTL_UARTEN         0x00000001  // UART Enable
//------------UART_Init------------
// Initialize the UART for 115,200 baud rate (assuming 50 MHz UART clock),
// 8 bit word length, no parity bits, one stop bit, FIFOs enabled
// Input: none
// Output: none
```

```
void UART_Init(void){
  SYSCTL_RCGCUART_R |= 0x01;           // activate UART0
  SYSCTL_RCGCGPIO_R |= 0x01;           // activate port A
  while((SYSCTL_PRGPIO_R&0x01) == 0){};
  UART0_CTL_R &= ~UART_CTL_UARTEN;      // disable UART
  UART0_IBRD_R = 27;    // IBRD = int(50,000,000 / (16 * 115,200)) = int(27.1267)
  UART0_FBRD_R = 8;      // FBRD = int(0.1267 * 64 + 0.5) = 8
                                       // 8 bit word length (no parity bits, one stop bit, FIFOs)
  UART0_LCRH_R = (UART_LCRH_WLEN_8|UART_LCRH_FEN);
  UART0_CTL_R |= UART_CTL_UARTEN;       // enable UART
  GPIO_PORTA_AFSEL_R |= 0x03;           // enable alt funct on PA1-0
  GPIO_PORTA_DEN_R |= 0x03;             // enable digital I/O on PA1-0
                                        // configure PA1-0 as UART
  GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R&0xFFFFFF00)+0x00000011;
  GPIO_PORTA_AMSEL_R &= ~0x03;          // disable analog functionality on PA
}
char UART_InChar(void){
  while((UART0_FR_R&UART_FR_RXFE) != 0);
  return((char)(UART0_DR_R&0xFF));
}
void UART_OutChar(char data){
  while((UART0_FR_R&UART_FR_TXFF) != 0);
  UART0_DR_R = data;}
```



Distributed using two LaunchPads connected together by the UARTs.

U1Rx (PC4) connected to U1Tx (PC5) of other microcontroller

U1Tx (PC5) connected to U1Rx (PC4) of other microcontroller

Ground connected ground of other microcontroller

SW2 (send color) connected to PF0

Red LED connected to PF1

Blue LED connected to PF2

Green LED connected to PF3

SW1 (step color) connected to PF4

Program 11.8. High-level communication network (C11_Network).

```
//  red, yellow, green, light blue, blue, purple,  white,  dark
const long ColorWheel[8] = {0x02,0x0A,0x08,0x0C,0x04,0x06,0x0E,0x00};
int main(void){ unsigned long SW1,SW2;
  long prevSW1 = 0;        // previous value of SW1
  long prevSW2 = 0;        // previous value of SW2
  unsigned char inColor;   // color value from other microcontroller
  unsigned char color = 0; // this microcontroller's color value
  PLL_Init();              // set system clock to 80 MHz
  SysTick_Init();          // initialize SysTick
  UART_Init();             // initialize UART
  PortF_Init();            // initialize buttons and LEDs on Port F
  while(1){
    SW1 = GPIO_PORTF_DATA_R&0x10; // Read SW1
    if((SW1 == 0) && prevSW1){    // falling of SW1?
      color = (color+1)&0x07;     // step to next color
    }
```

```
 prevSW1 = SW1; // current value of SW1
    SW2 = GPIO_PORTF_DATA_R&0x01; // Read SW2
    if((SW2 == 0) && prevSW2){    // falling of SW2?
      UART_OutChar(color+0x30);   // send color as '0' - '7'
    }
    prevSW2 = SW2; // current value of SW2
    inColor = UART_InCharNonBlocking();
    if(inColor){ // new data have come in from the UART??
      color = inColor&0x07;     // update this computer's color
    }
    GPIO_PORTF_DATA_R = ColorWheel[color];  // update LEDs
    SysTick_Wait10ms(2);        // debounce switch
  }
}
```