UART DRIVER APIS
and
TEST APPLICATION

RECAP: UART header file assignment

Come up with function declarations/signatures for the following functions:

```
configure_data_width
configure_stop_bit
configure_parity
configure_data_packet
transmit_data
receive_data
```

in the same manner as given for **`configure_baud_rate`** in the previous slide.

Questions to be asked before you start on the design:

1. Should I pass parameters to the function? If so:
   - what should they represent?
   - what values can they take? Based on which what will be the data type
2. Should there be a return value? What values should it take and how should the calling function interpret this?

given <name>_uart.h

```
/**
 * @brief Configures the baud rate of the UART
 * @param baud : the baud rate of the UART data to be transmitted
 * valid values: 50 ...128000
 * @param clock: the input clock frequency in MHz is from the crystal clock.
 * valid values: 1.8432MHz, 3.072MHz, 18.432MHz
 * @return none
 */
void configure_baudrate (unsigned int baud, float clock);

/**
 * @brief ??
 *
 * @param ?? : ??
 * valid values: ??
 *
 * @return ??
 */
?? configure_data_width (??);

/**
 * @brief ??
 *
 * @param ?? : ??
 * valid values: ??
 *
 * @return ??
 */
?? configure_stop_bit (??);
```

```
/**
 * @brief ??
 *
 * @param ?? : ??
 * valid values: ??
 *
 * @return ??
 */
?? configure_parity (??);

/**
 * @brief ??
 *
 * @param ?? : ??
 * valid values: ??
 *
 * @return ??
 */
?? configure_data_packet (??);
```

foo_uart.h : part 1

UART Driver APIs/functions

```
/**
 * @brief  configures the length of the data
 *
 * @param data_length : length of the data that needs to be set
 * valid values: 5, 6, 7, 8
 *
 * @return none
 */
void configure_data_width (unsigned int data_length);

/**
 * @brief configures the stop bit of the UART.
 *
 * @param s_bit : the stop bit of UART for transmission and reception
 * valid values: 1, 1.5, 2
 *
 * @return none
 */
void configure_stop_bit (int s_bit);
/**
 * @brief configures the parity of the UART data.
 *
 * @param parity : the parity of data to be transmitted or received
 * valid values: odd, even
 *
 * @return parity
 */
int configure_parity (char parity);
```

Test application: main function

```
#include "foo_uart.h"

int main()
{
    configure_baud_rate(9000, 3.072);
    configure_data_width(5);
    configure_stop_bit (1.5);
    configure_parity (0);
}
```

foo_uart.h : part 2

UART Driver APIs/functions	Test application: main function

/**
* @brief **Configures the data packet format of the UART**
*
* @param **data_packet_format : the data packet format code for UART data transmission**
• valid values:**1 (8 data bits, even parity, 1 stop bit),**
• **0 (8 data bits, odd parity, 1 stop bit)**
*
* @return none
*/
void configure_data_packet(**int data_packet**);
	`#include "foo_uart.h"`
	
	`Int main()`
	`{`
	`configure_baud_rate(9000, 3.072);`
	`configure_data_packet(1);`
	`}`

foo_uart.h : part 2

UART Driver APIs/functions	Test application: main function

```
/**
 * @brief Configures the data packet format of the UART
 *
 * @param data_length : length of the data that needs to be set
 * valid values: 5, 6, 7, 8
 *
 * @param s_bit : the stop bit of UART for transmission and reception
 * valid values: 1, 1.5, 2
 *
* @param parity : the parity of data to be transmitted or received
 * valid values: 0 = odd,
 *                1 = even
*
* @return none
*/
void configure_data_packet(unsigned int data_length,
                           float s_bit,
                           char parity);
```

```
#include "foo_uart.h"

int main()
{
    configure_baud_rate(9000, 3.072);
    configure_data_packet(5, 1.5, 0);
}
```

UART HEADER

(expected to be submitted as version : 1)

uart.h – version 1

```
/**
 * @brief Configures the baud rate of the UART
 *
 * @param baud : the baud rate of the UART
 *               data to be transmitted
 * valid values: 50 ...128000
 * @param clock: the input clock frequency in
 *               MHz.
 * valid values: 1.8432MHz,
 *               3.072MHz,
 *               18.432MHz
 *
 * @return none
 */
void configure_baudrate (unsigned int baud,
                         float clock);

/**
 * @brief Configures the data transmission
 *.       of the UART
 * @param word_len : the word length of the
 *.                 UART data to be sent
 * valid values    : 5, 6, 7, 8
 * @return none
 */
void configure_data_width(
                     unsigned char word_len);
```

```
/**
 * @brief Configures the stop bit of the data
 *        transmission.
 * @param stop_bit : Number of stop bits.
 * valid value : 1 for 1 stop bit
 *               2 sets 1.5 stop bit when data
 *                 length is 5.
 *               2 sets 2 stop bit when data
 *                 length is 6, 7, 8.
 * @return none
 */
void configure_stop_bit (
                     unsigned char stop_bit);

/**
 * @brief Configures the parity bit of the data.
 * @param parity   : the parity setting
 * valid values    : 0 = even parity,
 *                   1 = odd parity,
 *                   2 = no parity
 * @return none
 */
void configure_parity (unsigned char par);
```

uart.h – version 1

```
/**
 * @brief Configures the data transmission of
 *        the UART
 * @param data_len : the word length of the
 *                   uart data to be sent
 * valid values    : 5, 6, 7, 8
 * @param stop_bit : Number of stop bit of the
 *                   data.
 * valid values    : 1, 2
 * @param parity   : the parity setting
 * valid values    : 0 = even parity,
 *                   1 = odd parity,
 *                   2 = no parity
 * @return none
 */
void configure_data_packet(
                    unsigned char data_len,
                    unsigned char stop_bit,
                    unsigned char parity);
```

```
/**
 * @brief transmit data of the UART
 * @param data : one byte of data to be
 *               transmitted
 * @return : none
 */
void transmit_data (unsigned char data);
```

```
/**
 * @brief receive data of the UART
 * @return : one byte of received data
 *
 */
unsigned char receive_data();
```

UART HEADER
(improvements to the APIS)
(version : 2)

Transmit data

- We know that the data to be transmitted has to written into the THR register of the UART
- How do we know if THR is empty of not?

Transmit data

- We know that the data to be transmitted has to written into the THR register of the UART
- How do we know if THR is empty of not?
- There are 2 fields in the LSR register:
  - THRE field (bit 5): 1 in this bit indicates that the UART is ready to accept new character for transmission.
  - TEMT field (bit 6): 1 in this bit indicates that the transmitter holding and shift registers are empty.

Transmit data

- We know that the data to be transmitted has to written into the THR register of the UART
- How do we know if THR is empty of not?
- There are 2 fields in the LSR register:
  - THRE field (bit 5): 1 in this bit indicates that the UART is ready to accept new character for transmission.
  - TEMT field (bit 6): 1 in this bit indicates that the transmitter holding and shift registers are empty.
- How does this impact the declaration of function transmit_data?

transmit_data – version 1

- The function declaration of transmit_data is as follows:

```
/**
 * @brief transmit data of the UART
 * @param data : one byte of data to be transmitted
 * @return : none
 */
void transmit_data (unsigned char data);
```

- What this means is that when we make a call to `transmit_data` we assume that it will definitely succeed to transmit the passed argument.
- But what if THR has un-transmitted data?
- So how do we handle this?
  - Do we wait in transmit_data until the THRE or TEMT bits are high and write into THR? If so how long do we wait?
  - Do we return from transmit_data with a value that indicates a failure to transmit the data?
- Which is a better scheme?

transmit_data – version 1

- For now we will go with first option, i.e “we wait in transmit_data until the THRE or TEMT bits are high …”. Since this will be done as part of `transmit_data` function, the declaration of transmit_data is unchanged.

```
/**
 * @brief transmit data of the UART
 * @param data : one byte of data to be transmitted
 * @return : none
 */
void transmit_data (unsigned char data);
```

Receive data

- We know that the data to be received is read from RBR register of the UART.
- How do we know that RBR has data?

Receive data

- We know that the data to be received is read from RBR register of the UART.
- How do we know that RBR has data?
  - By reading the DR (bit 0) of the register LSR.

Receive data

- We know that the data to be received is read from RBR register of the UART.
- How do we know that RBR has data?
  - By reading the DR (bit 0) of the register LSR.
- However there are also the bits 1 – 4 (OE, PE, FE, and BI) of the register LSR that indicate errors in the received data.
- What if data is received, but contains errors? What should we do? And how does this impact `receive_data`?

receive_data – version 1

- The function declaration of `receive_data` is as follows:

```
/**
 * @brief receive data of the UART
 * @return : one byte of received data
 *
 */
unsigned char receive_data();
```

- What this means is that when we make a call to `receive_data`, we are assuming that the received data will not have any errors.
- But we know that may not always be the case. OE, FE, PE or BI could be set 1 in LSR.
- So how do we handle this?
  - Should we ignore the received data when there is an error
  - Should we return the received data, but also return a status indicating that the data has errors or no errors to the main function?
- Which is a better scheme?

receive_data – version 2

- Returning the status and the received data is the better approach, as the driver function does not know the criticality of the data being received. Only the main function knows, hence `receive_data` would be as follows:

```
/**
 * @brief receive data of the UART
 * @return : 0 for failure
 *           else one byte of received data
 */
unsigned char receive_data ();
```

receive_data – version 2

- Returning the status and the received data is the better approach, as the driver function does not know the criticality of the data being received. Only the main function knows, hence `receive_data` would be as follows

```
/**
 * @brief receive data of the UART
 * @return : 0 for failure
 *           else one byte of received data
 */
unsigned char receive_data ();
```

- Returning of status should not be confused with the actual received data as the data can be any value. Hence using a pointer argument to return the data and status as return value.

```
/**
 * @brief receive data of the UART
 * @param data : one byte of data to be received
 * @return : 0 for success
 *           1 for failure
 */
int receive_data (unsigned char *data);
```

TEST APPLICATION

(The main function that makes calls to the Driver APIS)

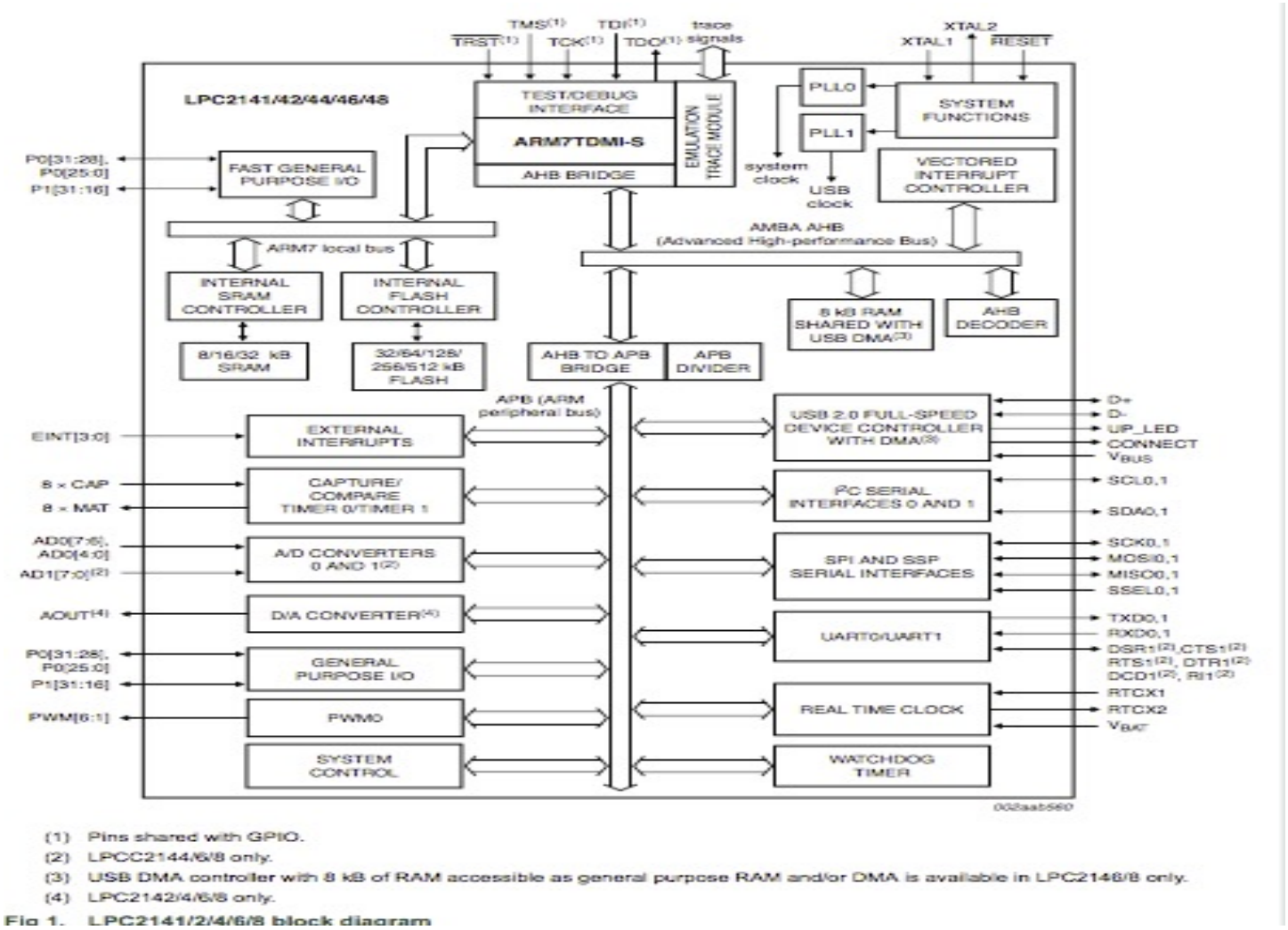
From 4th sem microntroller lab

```
#include<LPC21xx.h>

void delay(void);
void serial(void);

unsigned char mg;

int main()
{
  unsigned int i;
  unsigned char msg[]={"BVB"};

  serial();

  while(1)
  {
   for(i=0;i<3;i++)
    {
        while(!(U0LSR & 0x20));
        U0THR = msg[i];
    }
   while(!(U0LSR & 0x01));

   mg=U0RBR;
   U0THR=mg;


  delay();
  }
}
```

```
void serial()
{
  PINSEL0 = 0x00000005;

  U0LCR = 0x83;

  U0DLL = 0x61;

  U0LCR = 0x03;

  U0IER = 0x01;

}



void delay()
{
  unsigned int i;
  for(i=0;i<10000;i++);
}
```

PINSEL0 – a digression
(explanation on why PINSEL0 is set 5
in the main function)

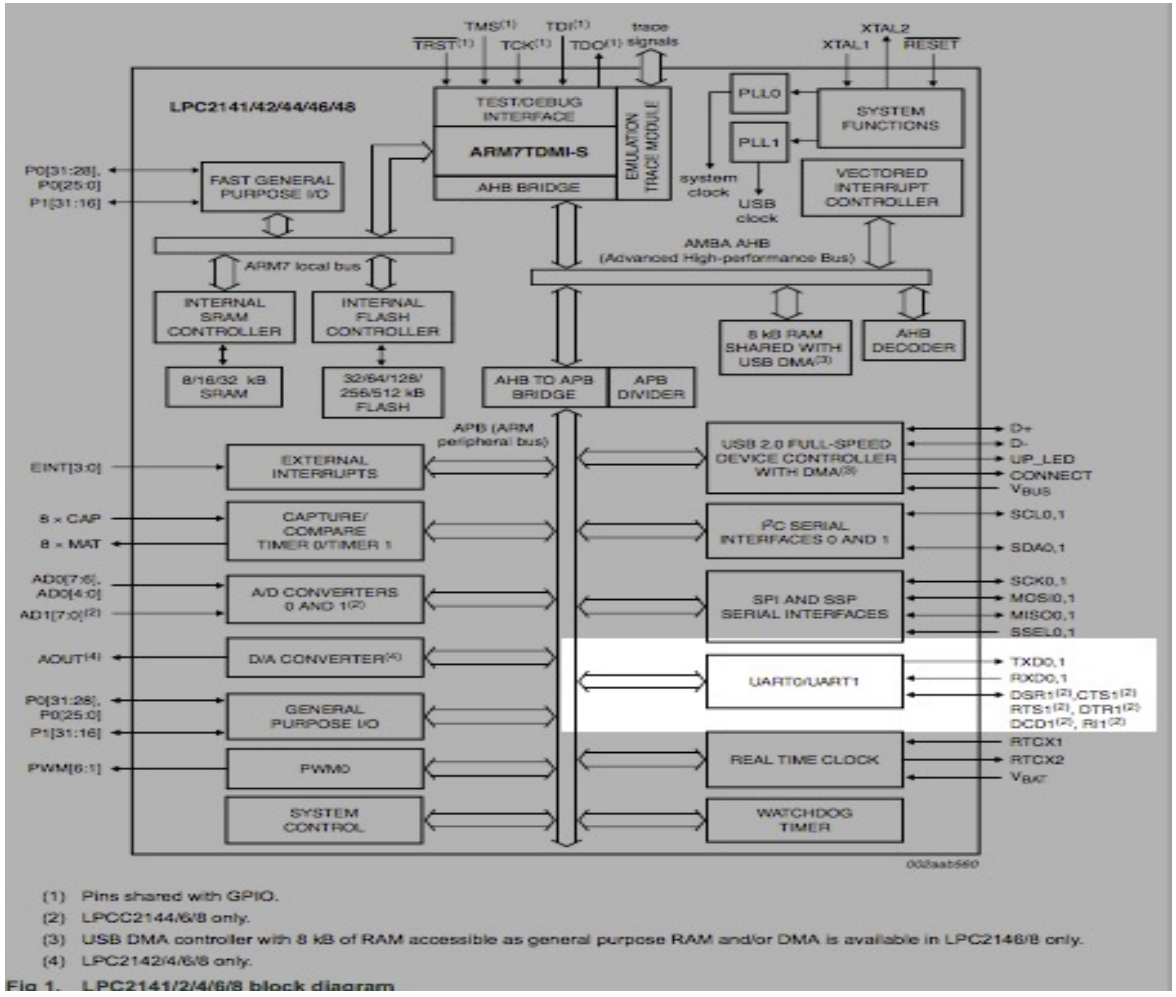
UART in LPC2148 : block diagram



(1) Pins shared with GPIO.
(2) LPC2144/6/8 only.
(3) USB DMA controller with 8 kB of RAM accessible as general purpose RAM and/or DMA is available in LPC2146/8 only.
(4) LPC2142/4/6/8 only.

Fig 1. LPC2141/2/4/6/8 block diagram

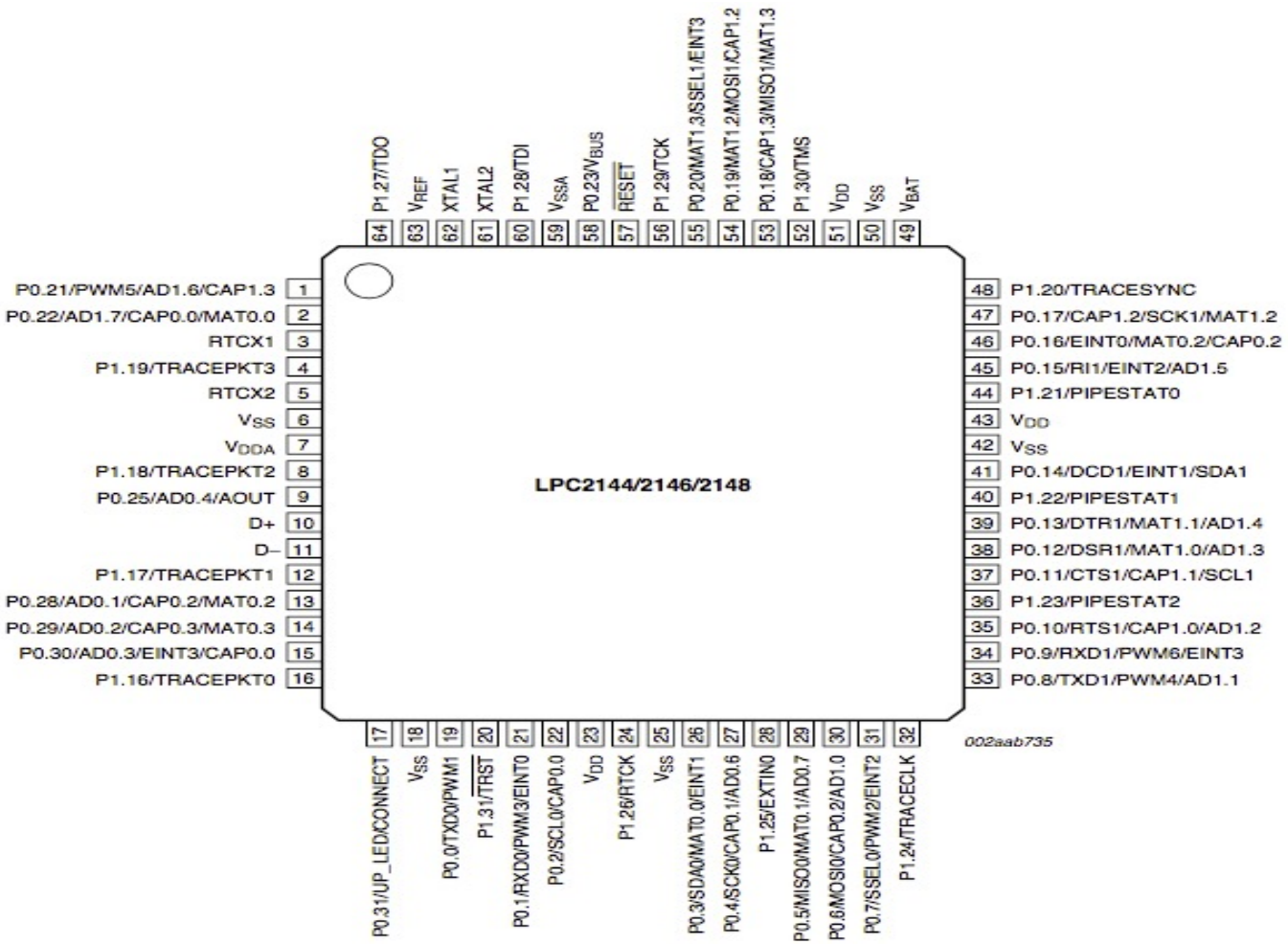
UART in LPC2148 : block diagram



(1) Pins shared with GPIO.
(2) LPC2144/6/8 only.
(3) USB DMA controller with 8 kB of RAM accessible as general purpose RAM and/or DMA is available in LPC2146/8 only.
(4) LPC2142/4/6/8 only.

Fig 1. LPC2141/2/4/6/8 block diagram

LPC2148 : Pin configuration



LPC2148 : Pin description

Table 35. Pin description

Symbol	Pin	Type	Description
P0.0 to P0.31		I/O	**Port 0:** Port 0 is a 32-bit I/O port with individual direction controls for each bit. Total of 28 pins of the Port 0 can be used as a general purpose bi-directional digital I/Os while P0.31 provides digital output functions only. The operation of port 0 pins depends upon the pin function selected via the pin connect block. Pins P0.24, P0.26 and P0.27 are not available.
P0.0/TXD0/ PWM1	19[1]	I/O	**P0.0** — General purpose digital input/output pin
		O	**TXD0** — Transmitter output for UART0
		O	**PWM1** — Pulse Width Modulator output 1
P0.1/RxD0/ PWM3/EINT0	21[2]	I/O	**P0.1** — General purpose digital input/output pin
		I	**RxD0** — Receiver input for UART0
		O	**PWM3** — Pulse Width Modulator output 3
		I	**EINT0** — External interrupt 0 input

LPC2148 : Pin Selection

- Pin connect block allows the selection of pins of the microcontroller that have more than one functionality.
- configuration registers PINSEL0, PINSEL1 and PINSEL2 allow for the selection of functionality.
- selection of one function excludes the other functionality.
- There are 2 identical UARTs on LPC2148:
  - UART0
  - UART1
- PINSEL0 configures UART0's TX and RX functionality.
- PINSEL1 configures UART1's TX and RX functionality.

LPC2148 : Pin Selection for UART0 and UART1

6.4.1 Pin function Select register 0 (PINSEL0 - 0xE002 C000)

The PINSEL0 register controls the functions of the pins as per the settings listed in Table 40. The direction control bit in the IO0DIR register is effective only when the GPIO function is selected for a pin. For other functions, direction is controlled automatically.

Table 37. Pin function Select register 0 (PINSEL0 - address 0xE002 C000) bit description

Bit	Symbol	Value	Function	Reset value
1:0	P0.0	00	GPIO Port 0.0	0
		01	TXD (UART0)	
		10	PWM1	
		11	Reserved	
3:2	P0.1	00	GPIO Port 0.1	0
		01	RxD (UART0)	
		10	PWM3	
		11	EINT0	
17:16	P0.8	00	GPIO Port 0.8	0
		01	TXD UART1	
		10	PWM4	
		11	Reserved[1][2] or AD1.1[3]	
19:18	P0.9	00	GPIO Port 0.9	0
		01	RxD (UART1)	
		10	PWM6	
		11	EINT3	

LPC2148 : PINSEL0 values

- The value to be put in PINSEL0 to select TX and RX of UART0: 0x00000005

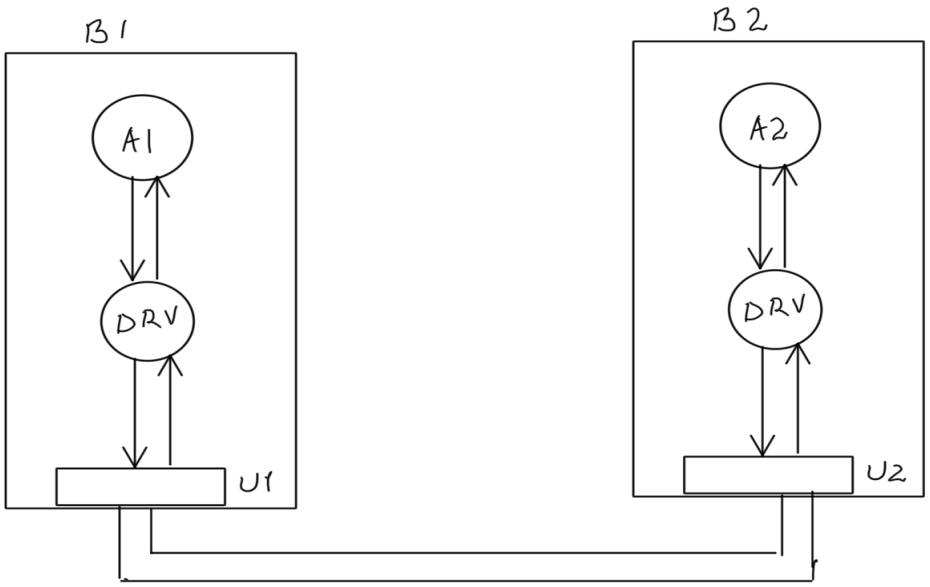
bit position	3	2	1	0
Values to select TX and RX functionality of UART0	0	1	0	1

TEST APPLICATION
(continuation …)

From 4th sem microntroller lab

```
#include<LPC21xx.h>

void delay(void);
void serial(void);

unsigned char mg;

int main()
{
  unsigned int i;
  unsigned char msg[]={"BVB"};

  serial();

  while(1)
  {
   for(i=0;i<3;i++)
    {
        while(!(U0LSR & 0x20));

        U0THR = msg[i]; // Transmit
    }
   while(!(U0LSR & 0x01));

   mg=U0RBR; //Receive data
   U0THR=mg;


  delay();
  }
}
```

```
void serial()
{
  /* configuring the pin corresponding to GPIO port 0's
   * input/ouput pins to connect to UART0 TX and RX
   */
  PINSEL0 = 0x00000005;

  /* configuring the UART0's LCR register for:
   * data_width = ??
   * stop bit = ??
   * parity = ??
  */
  U0LCR = 0x83;

  /* configuring the baud rate to ?? */
  U0DLL = 0x61;

  U0LCR = 0x03;

  U0IER = 0x01;

}


/* What is the delay in microsecounds? */
void delay()
{
  unsigned int i;
  for(i=0;i<10000;i++);
}
```



B1 & B2	Boards 1 & 2
A1 & A2	Application software
DRV	Driver Software
U1 & U2	UART 1 & 2

A1 & A2 running on the CPUs of B1 & B2:

1. Initialize the baud rate to 9600 while working on input clock frequency of 1.843MHz.
2. configure the data length to 7, stop bit to 1 and enable even parity
3. write data to be transmitted on U1 which will be received on U2.
4. write data to be transmitted on U2 which will be received on U1.

DRV: Uart driver software:

1. Program registers DLL & DLM for UART to function at desired baud rate.
2. Program the bits 0 – 4 of register LCR to set UART to data_length = 7, stop_bit = 1 and even parity.
3. data to be sent is put in register THR of U1 which is received in RBR of U2.
4. data to be sent is put in register THR of U2 which is received in RBR of U1

pseudocode of uart.c – version 1

Application: main program	DRV: UART device driver functions

`int main ( )`
`{`
`  configure_baud_rate(9600,`
`  1.843);`
	`void configure_baud_rate(unsigned int baud,`
	`  float freq)`
	`{ /* calculate the divisor value`
	`  write 1 into dlab bit of LCR`
	`  program DLL and DLM with the 16 bit divisor value`
	`  write 0 into dlab bit of LCR`
	`  */`
	`}`

`configure_data_packet(7, 1, 1);`	`void configure_data_packet(unsigned char data_len,`
	`  unsigned char stp_bit,`
	`  unsigned char par)`
	`{ /* program LCR register to configure UART for:`
	`  data length = 7`
	`  stop bit = 1`
	`  parity to be even`
	`  */`
	`}`

pseudocode of uart.c – version 1

A1 is the main program running on B1:	**DRV: Uart driver software:**

`transmit_data(data);`	`void transmit_data(unsigned char dt) {`
	`/* check if THR is empty`
	`THR = data;`
	`*/`
	`}`

`data = receive_data();`
`}`
	`unsigned char receive_data()`
	`{`
	`/* check if data is available in RBR`
	`return (RBR);`
	`*/`
	`}`

Thank You

www.vayavyalabs.com