PROJECT-8

ARM to ARM Serial Communication

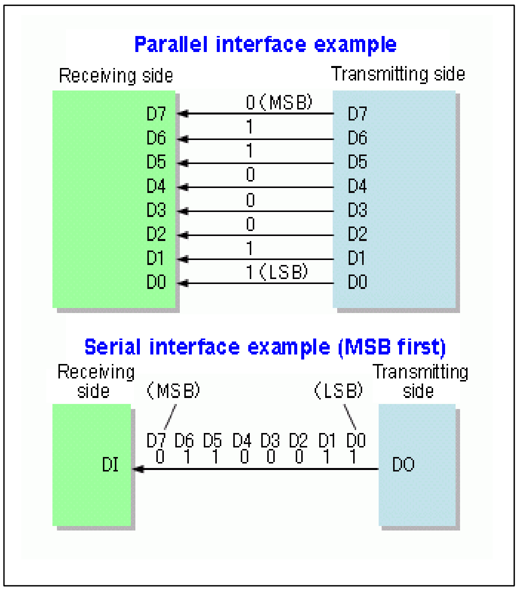
EE18B022	EE18B030	EE18B040	EE18B050
EE18B062	EE18B064	EE18B066	EE18B070
EE18B128	EE18B136	EE18B148	EE18B158

TA: BHANASHREE, PRAMOD

Introduction:-

Serial Communication:

In telecommunication and data transmission, **serial communication** is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus. This is in contrast to parallel communication, where several bits are sent as a whole with several parallel channels. Keyboard and mouse cable, cables that carry digital video, Ethernet cable and many other cables use serial communication. Practically all long-distance communication transmits data one bit at a time, rather than in parallel, because it reduces the cost of the cable.



Motivation:

Since serial communication is so widely used, through this project we hope to understand the underlying concepts and protocols, used in serial communication and demonstrate serial communication in ARM through UART Port.

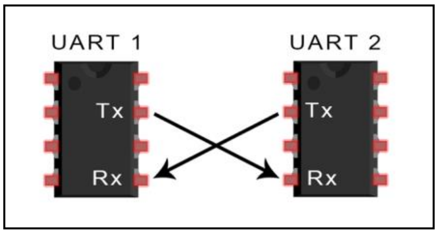
Our prototype definition:

We consider a gray scale image of 32X32 pixels, which can be represented as 2 dimensional matrix of pixel intensities. This matrix data is transferred serially byte by byte through UART0 Port of ARM kit 1 and stored in Flash Memory, this data is again transmitted through UART0 Port of ARM kit 1 to UART0 Port of ARM kit2 only.

SERIAL COMMUNICATION using UART:

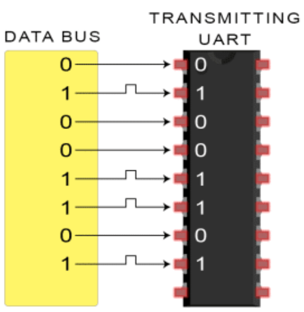
Serial Communication between two devices like CPU, memory or microcontroller is implemented using universal asynchronous receiver transmitter (**UART**). It is used for transmitting and receiving serial data. The **UART** acts as an intermediary between parallel and serial interfaces. One of the best things about **UART** is that it only uses two wires to transmit data between devices.

In UART communication, two UARTs communicate directly with each other. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART. Data bus is used to send data to the UART by another device like a CPU, memory, or microcontroller.

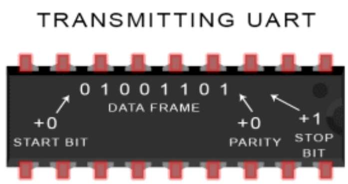


I. Data is transferred from the data bus to the transmitting UART in parallel form.
II. After the transmitting UART gets the parallel data from the data bus, it adds a start bit, a parity bit, and a stop bit, creating the data packet.
III. The entire packet is sent serially from the transmitting UART to the receiving UART bit by bit through Tx pin.
IV. The Receiving UART reads the data packet bit by bit at its Rx pin at a pre-configured baud rate. It removes the start bit, parity bit, and stop bits.
V. The receiving UART converts the serial data back into parallel and transfers it to the data bus on the receiving end.

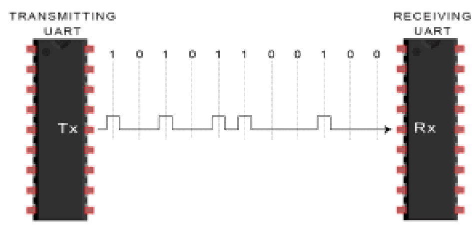
I. Parallel to Serial Data Conversion



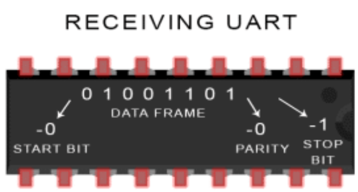
II. Addition of start bit, parity bit and the stop bit



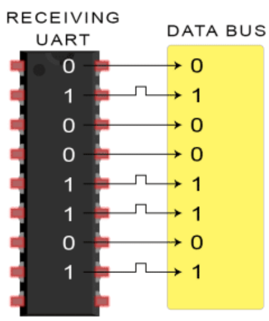
UARTs transmit data *asynchronously*,



III. Serial Transmission from transmitting UART to receiving UART



IV. Removal of start bit, parity bit and stop bit



V. Serial to Parallel Data Conversion

UARTs transmit data asynchronously, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.

When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the *baud rate.* Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps). Both UARTs must operate at about the same baud rate.

LPC 2378 has 4 inbuilt UARTs. Due to technical constraints, we are using only UART0.

UART Registers:

Register	Description
U0RBR	Contains the recently received Data
U0THR	Contains the data to be transmitted
U0FCR	FIFO control register
U0LCR	Controls the UART frame formatting (Number of Data Bits, Stop bits, parity selection)
U0DLL	Least Significant Byte of the UART baud rate generator value.
U0DLM	Most Significant Byte of the UART baud rate generator value.

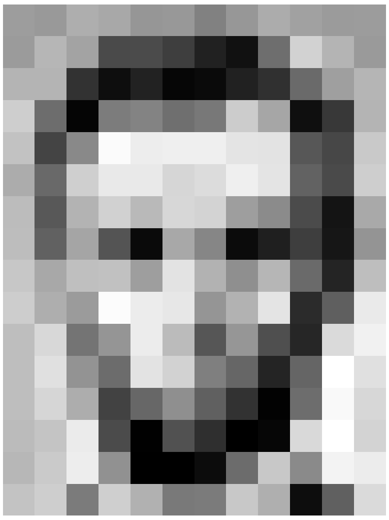
$$UARTn_{baudrate} = \frac{pclk}{16 \times (256 \times UnDLM + UnDLL)}$$

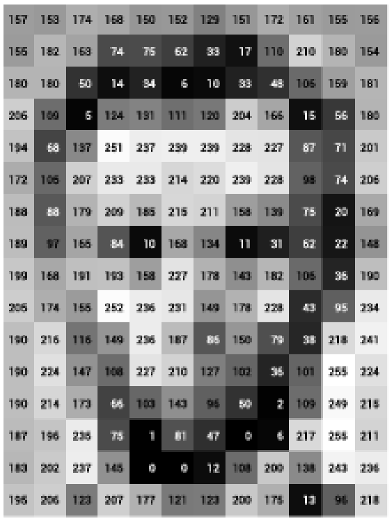
Steps for Configuring UART0:

I. Configure the GPIO pin for UART0 function using PINSEL register.
PINSEL0=0x00000050 has to be used for making P0.2 /P0.3 as Tx and Rx.
II. Configure LCR for 8-data bits, 1 Stop bit, Disable Parity and Enable DLAB.
U0LCR=0x83
III. Calculate the DLM, DLL values for required baud rate from PCLK.
IV. Update the DLM, DLL with the calculated values.
U0DLM = Fdiv/256
U0DLL = Fdiv%256
V. Finally clear DLAB to disable the access to DLM, DLL.
U0LCR=0x03

Grayscale Image to Matrix:

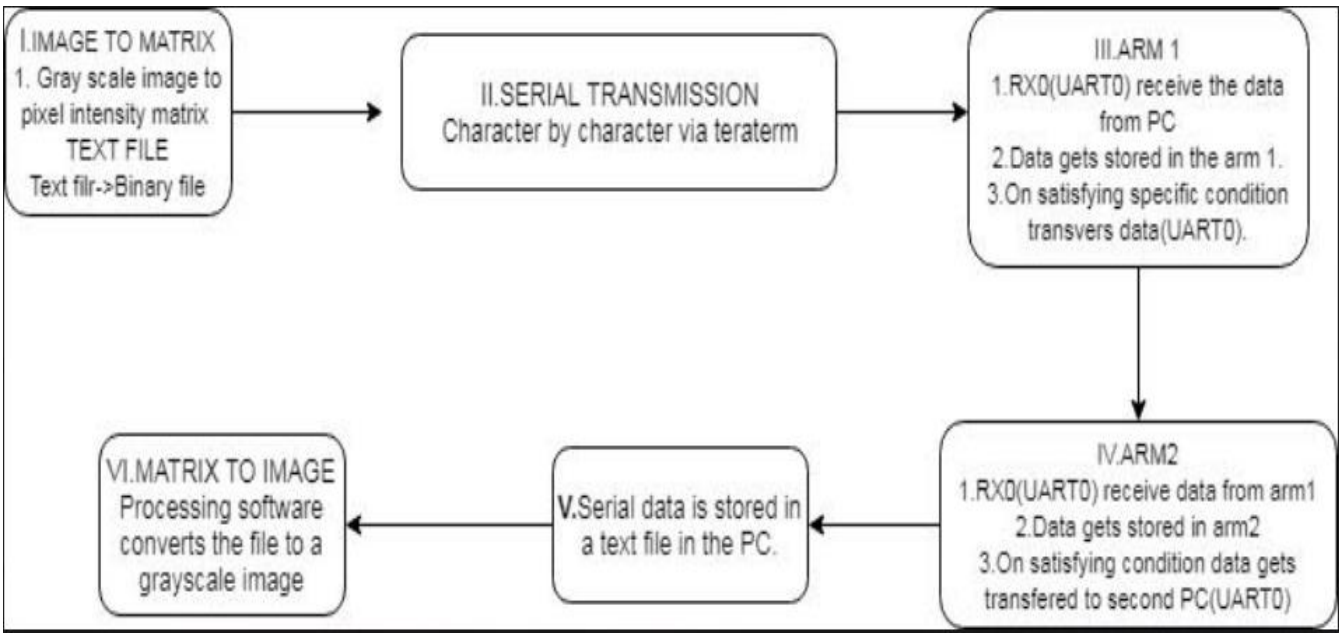
- Greyscale image is converted into a matrix with each entry representing pixel intensity, using software like MATLAB and stored as text file.
- But text file sees it’s content as ASCII characters and not as actual data.
- To overcome this, we convert the stored text file to binary file.





157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

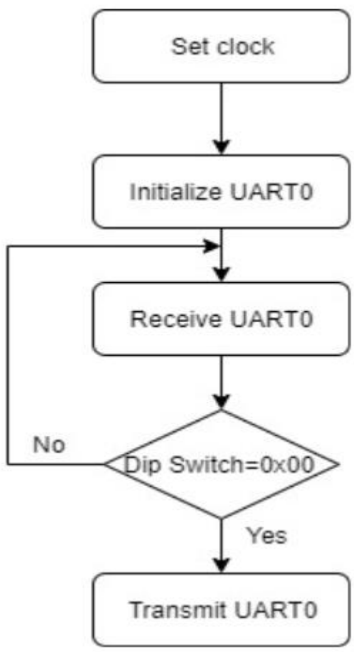
OVERVIEW OF THE PROCESS:



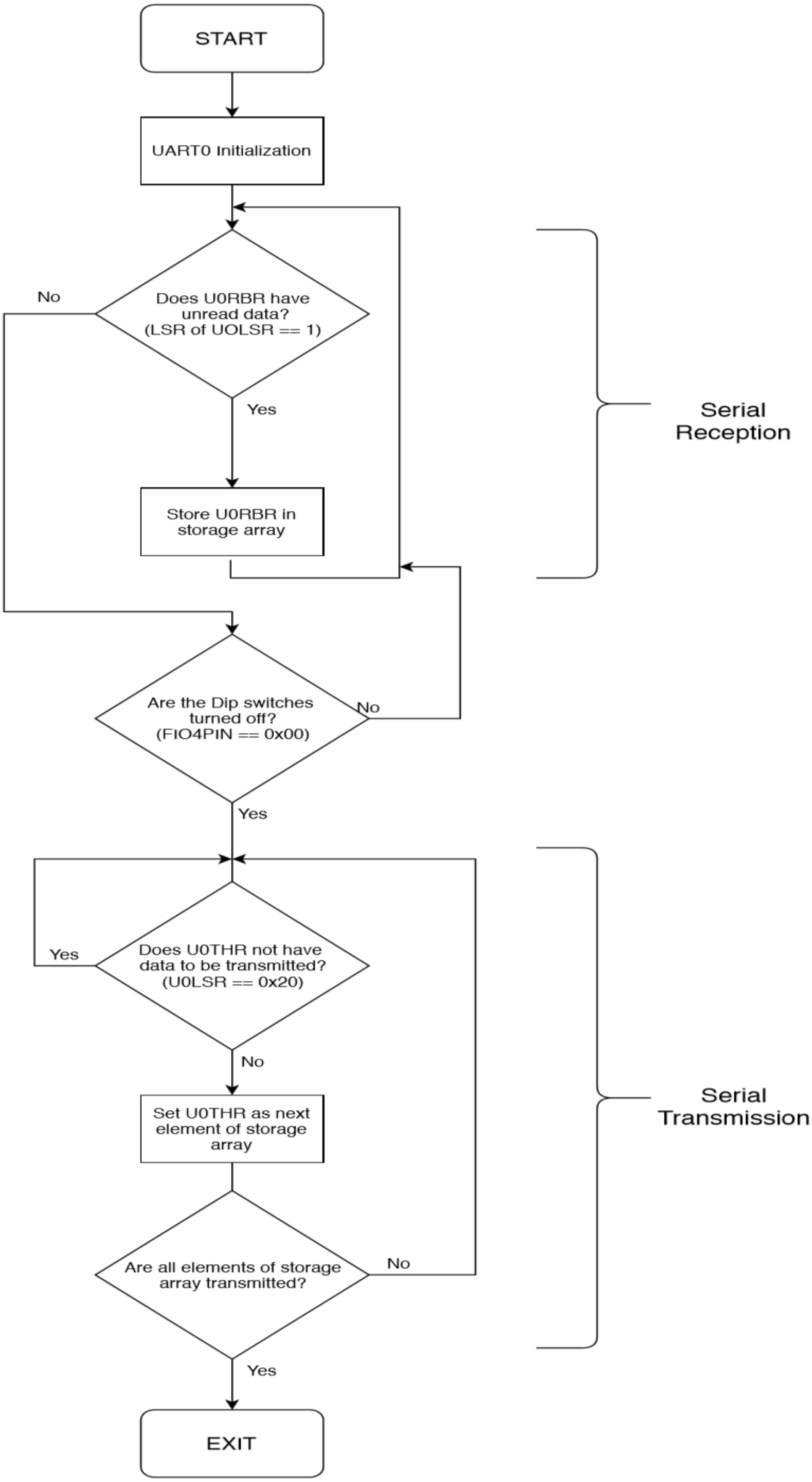
DETAILED EXPLANATION:

Initially, Binary file containing the pixel intensity data is transferred from PC to ARM1 using Teraterm through UART0, which is stored into an array. After the data is completely received, RS 232 cable connected to PC and ARM1 is manually removed and inserted into UART0 of the ARM2 and when the specified condition is satisfied, ARM1 starts transferring data to ARM2 and stored in it(In our code, when dip switches are positioned to 0x00 data gets transferred). Then again, RS 232 cable is to removed from ARM1 and connected to second PC. Similarly, when the dip switches are positioned to 0x00, ARM2 starts transferring data to second PC and stored as a text file. This file is converted back to image using Processing Software.

OVERVIEW of the ARM code:



Flow of the program:



CODE:

```
#include "LPC23xx.h"

/******
Routine to set processor and pheripheral clock
*****/


void TargetResetInit(void)
{
   // 72 Mhz Frequency
    if ((PLLSTAT & 0x02000000) > 0)
  {
     /* If the PLL is already running */
     PLLCON &= ~0x02; /* Disconnect the PLL */
     PLLFEED = 0xAA; /* PLL register update sequence, 0xAA, 0x55 */
     PLLFEED = 0x55;
  }
    PLLCON &= ~0x01; /* Disable the PLL */
    PLLFEED = 0xAA; /* PLL register update sequence, 0xAA, 0x55 */
    PLLFEED = 0x55;
    SCS &= ~0x10; /* OSCRANGE = 0, Main OSC is between 1 and 20 Mhz */
    SCS |= 0x20; /* OSCEN = 1, Enable the main oscillator */
    while ((SCS & 0x40) == 0);
    CLKSRCSEL = 0x01; /* Select main OSC, 12MHz, as the PLL clock source */
    PLLCFG = (24 << 0) | (1 << 16); /* Configure the PLL multiplier and divider */
    PLLFEED = 0xAA; /* PLL register update sequence, 0xAA, 0x55 */
    PLLFEED = 0x55;
    PLLCON |= 0x01; /* Enable the PLL */
    PLLFEED = 0xAA; /* PLL register update sequence, 0xAA, 0x55 */
    PLLFEED = 0x55;
    CCLKCFG = 3; /* Configure the ARM Core Processor clock divider */
    USBCLKCFG = 5; /* Configure the USB clock divider */
    while ((PLLSTAT & 0x04000000) == 0);
    PCLKSEL0 = 0xAAAAAAAA; /* Set peripheral clocks to be half of main clock */
    PCLKSEL1 = 0x22AAA8AA;
    PLLCON |= 0x02; /* Connect the PLL. The PLL is now the active clock source */
    PLLFEED = 0xAA; /* PLL register update sequence, 0xAA, 0x55 */
    PLLFEED = 0x55;
    while ((PLLSTAT & 0x02000000) == 0);
    PCLKSEL0 = 0x55555555; /* PCLK is the same as CCLK */
    PCLKSEL1 = 0x55555555;
}
```

```
char img[1024];

// serial Reception routine
int serial_rx(void)
{
  while (!(U0LSR & 0x01))
 {
  int a;
   a =FIO4PIN & 0xFF;
             if(!a)                    //if a is not equal to 0x00 it will recieve data
             break;                    //if equal break out of loop
}
return (U0RBR);
}

//serial transmission routine
void serial_tx(int ch)
{
while ((U0LSR & 0x20)==0); //if U0THR doesn't have data to be transmitted stay
                                              //here
 U0THR = ch;
}

/* main routine****/
int main ()
{
    unsigned int Fdiv,a=1,k=0;
    char value;
    TargetResetInit();

    /** uart0 initialization **/
    PINSEL0 = 0x00000050;
    U0LCR = 0x83; // 8 bits, no Parity, 1 Stop bit
    Fdiv = ( 72000000 / 16 ) / 19200 ; //baud rate
    U0DLM = Fdiv / 256;
    U0DLL = Fdiv % 256;
    U0LCR = 0x03; // DLAB = 0
    FIO3DIR=0xFF;
    FIO4DIR=0x00;
    //Reception
   for(int i=0;i<1024;i++)
    {
             img[i]=serial_rx();
              k++;                    //count number of bytes received
    }
```

```
 //wait between reception and transmission (Code = 00000000)
while(a)
 {
     a =FIO4PIN&0xFF;
 }

 //Transmission
for(int i=0;i<k;i++)
 {
   serial_tx(img[i]);
 }

   //wait after sending(until reset)
  while(1);
  return 0;
 }
```

OUTPUT:

CONCLUSION:

Thus, we have successfully transmitted 32x32 pixel image serially from one PC to another PC using two Vi-ARM LPC2378 processors. We have not encountered any loss in data. A few recommendations that we have are:

- Transmission using UART 1 also would be more efficient but the lab ARM boards are not able to support it.
- Usage of SPI could also be looked upon.
- The number of people in the group could be reduced.

Overall the project was successful and very helpful in our understanding of the course.