Designing a Robot Control System

Aim:

The objective of this project is to implement a robot control system on STM32F107 development board having ARM Cortex M3 processor and running on Micrum's uC/OS-III real time operating system.

Introduction:

The idea is to control 13 robots on a given factory floor with commands being sent by Robot Control Center over UART communication. The Job of the Control System is to accept input from the user and forward appropriate commands to the Robot Manager. The Robot Manager interprets these commands and in turn pass the commands to robots as required.

The idea behind this project is to efficiently use concept of preemptive multitasking, synchronization between tasks using semaphores and mutex, communication between tasks using queues and mailbox. The algorithm can spawn up to 16 tasks if required.

General Program Design:

The control center, bots and robot manager communicate with each other over RS-232 and hence a custom designed interrupt based UART driver is developed. Each component on the network (robots, manager, control center) has a unique 8-bit address (0-255). Address zero is a broadcast packet which is intended for everyone on the network, address 1 means instruction packet is intended for Control Center, address 2 means the instruction packet is intended for Robot Manager and addresses 3-15 dedicated to 13 robots as shown in Fig. 1.

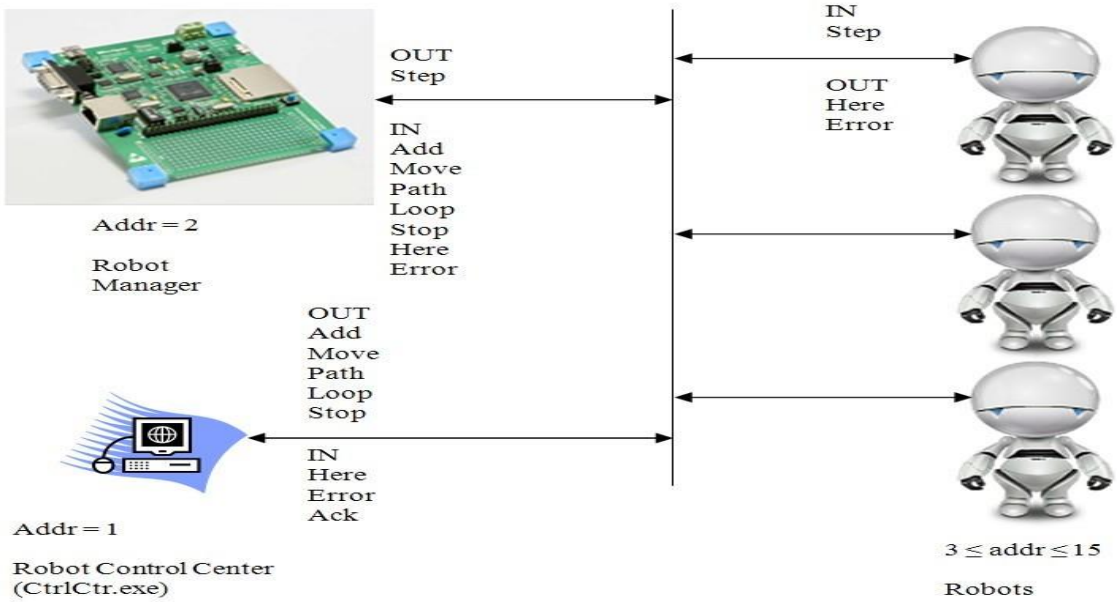


Fig.1 Robot Control System

The factory floor is organized as shown in Figure 2. The floor is organized as a rectangular grid. Each cell on the grid will be occupied by only one robot at a time. In Figure 2 below 5 robots are shown in green squares. Robot locations are given in cartesian coordinates. For example, there is a robot at location (17,3). The lower left cell represents the origin (0,0).

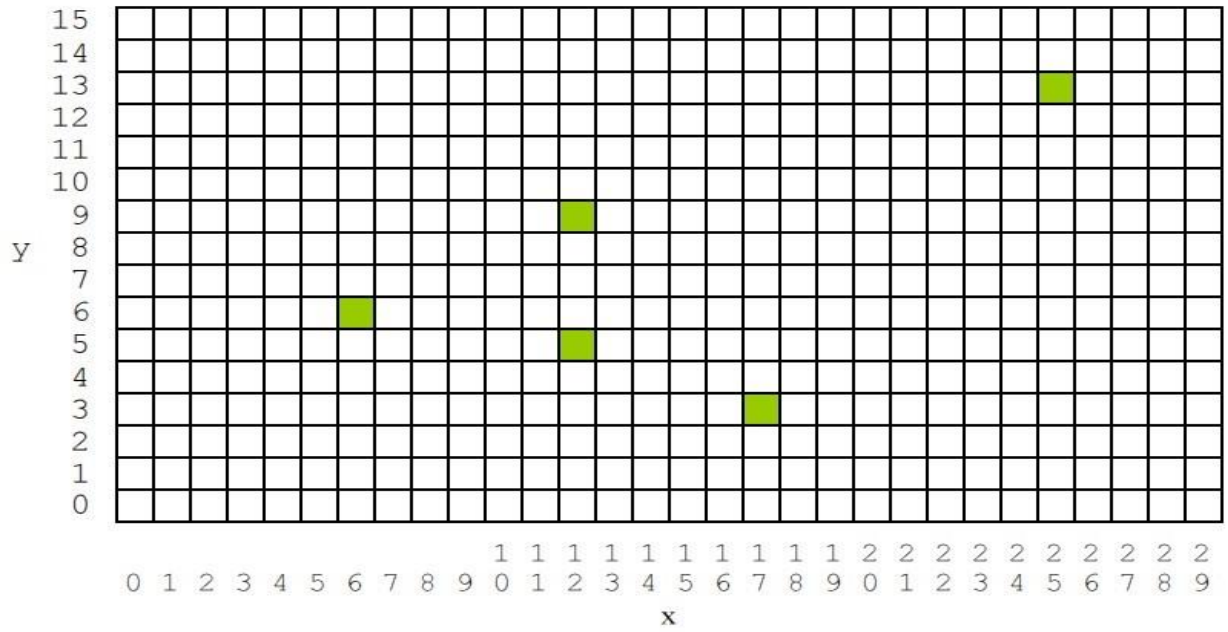


fig. 2 Factory floor mapping using grid coordination

Robots have limited capabilities, i.e. a robot can only move one step at a time to an adjacent cell. There are 8 surrounding directions in which robot may step and when asked to move if a cell in the direction is occupied by another robot, then it will avoid that direction and choose another adjacent direction thus avoiding collision.

fig. 3 Directions of Robot Steps

Input/ Output Packet Structure:

The packets are transferred via UART serial port. The general packet structure format is shown in Figure 4. The first 3 bytes of the packet constitute of 3 sync bytes. These are called as preamble bytes which enables synchronization with the start of new packet. In case of a bad packet find the beginning of the next packet by searching forward in the input stream until correct preamble byte is reached.

Byte Location in Packet	Packet	Payload	Byte in Payload
0	0x03		
1	0xAF		
2	0xEF	Payload	Byte in Payload
3	Packet Length	Payload Length	0
4	packet data byte [0]	Destination Address	1
5	packet data byte [1]	Source Address	2
6	packet data byte [2]	Message Type	3
7	packet data byte [3]	payload data byte [0]	4
8	…	payload data byte [1]	5
9	…	payload data byte [2]	6
…	…	…	
Packet Length – 2	packet data byte [Packet Length – 6]	payload data byte [Payload Length – 5]	Payload Length - 1
Packet Length - 1	Checksum Byte (XOR of all bytes in the packet except this byte)		

Fig 4: General Packet Structure

Listed below are the command packets that the control center can send to the Robot Manager.

ROBOT MANAGER COMMAND PACKETS
(Source = 1, Destination = 2)

Command Packet 0x00 - "Reset"

Reset the microcontroller, restarting the Robot Manager. This is accomplished by calling the BSP function NVIC_GenerateCoreReset().

Byte Location in Payload	Value
0	Payload Length in Bytes = 4
1	2
2	1
3	0x00

Command Packet 0x01 - "Add Robot"

Add a new robot to the factory floor at the indicated starting position (x, y).

Byte Location in Payload	Value
0	Payload Length in Bytes = 7
1	2
2	1
3	0x01
4	robot address (addr > 2)
5	x (x-coordinate of the robot's initial location)
6	y (y-coordinate of the robot's initial location)

Command Packet 0x02 - "Move Robot"

Move a robot to a new location (x, y). Note that this must result in an appropriate sequence of "Step" commands to the specified robot such that the robot moves to the destination with the fewest possible steps.

Byte Location in Payload	Value
0	Payload Length in Bytes = 7
1	2
2	1
3	0x02
4	robot address (addr > 2)
5	x (x-coordinate of the robot's destination)
6	y (y-coordinate of the robot's destination)

Command Packet 0x03 - "Follow Path"

Make a robot trace out a path visiting several consecutive (x, y) locations using the fewest possible steps.

Byte Location in Payload	Value
0	Payload Length in Bytes
1	2
2	1
3	0x03
4	robot address (addr > 2)
5	x (x-coordinate of the robot's first destination)
6	y (y-coordinate of the robot's first destination)
7	x (x-coordinate of the robot's second destination)
8	y (y-coordinate of the robot's second destination)
…	…
k – 2	x (x-coordinate of the robot's last destination)
k – 1	y (y-coordinate of the robot's last destination)

Command Packet 0x04 - "Loop" Command

Make a robot continually follow a looping path, visiting the indicated points locations using the fewest possible steps. When the robot reaches its last destination location, it then moves back to the starting location and repeats the loop.

Byte Location in Payload	Value
0	Payload Length in Bytes
1	2
2	1
3	0x04
4	robot address (addr > 2)
5	x (x-coordinate of the robot's first destination)
6	y (y-coordinate of the robot's first destination)
7	x (x-coordinate of the robot's second destination)
8	y (y-coordinate of the robot's second destination)
…	…
k – 2	x (x-coordinate of the robot's last destination)
k – 1	y (y-coordinate of the robot's last destination)

Command Packet 0x05 - "Stop Looping" Command

Stop a looping robot in its tracks.

Byte Location in Payload	Value
0	Payload Length in Bytes = 5
1	2
2	1
3	0x05
4	robot address (addr > 2)

ROBOT COMMAND PACKETS
(Source = 2, Destination = robot address)

Command Packet 0x07 - "Step"

Command the addressed robot to take a single step in one of the 8 possible directions, or remain stationary if direction = NoStep.

Byte Location in Payload	Value
0	Payload Length in Bytes = 5
1	Robot Address
2	2
3	0x07
4	Direction in which to step (0 to 8 for NoStep to NW, clockwise)

OUTPUT PACKETS
(Source = robot address, destination = 0 for broadcast)

Output Packet 0x09 – Here I Am

Every time that a robot takes a step, it replies with a "Here I Am" packet giving its new location.

Byte Location in Payload	Value
0	Payload Length in Bytes = 6
1	0
2	Robot Address
3	0x09
4	x (x-coordinate of the robot's current location)
5	y (y-coordinate of the robot's current location)

OUTPUT PACKETS
(Source = 2, destination = 1)

Output Packet 0x0A – Command Acknowledgement

A command acknowledgement packet is sent in response to any *valid* command.

Byte Location in Payload	Value
0	Payload Length in Bytes = 5
1	1
2	2
3	0x0A
4	Command Type Acknowledged

Output Packet 0x0B – Error Code packet

An Error Code packet is sent in addition to a command acknowledgement packet when an invalid command is received.

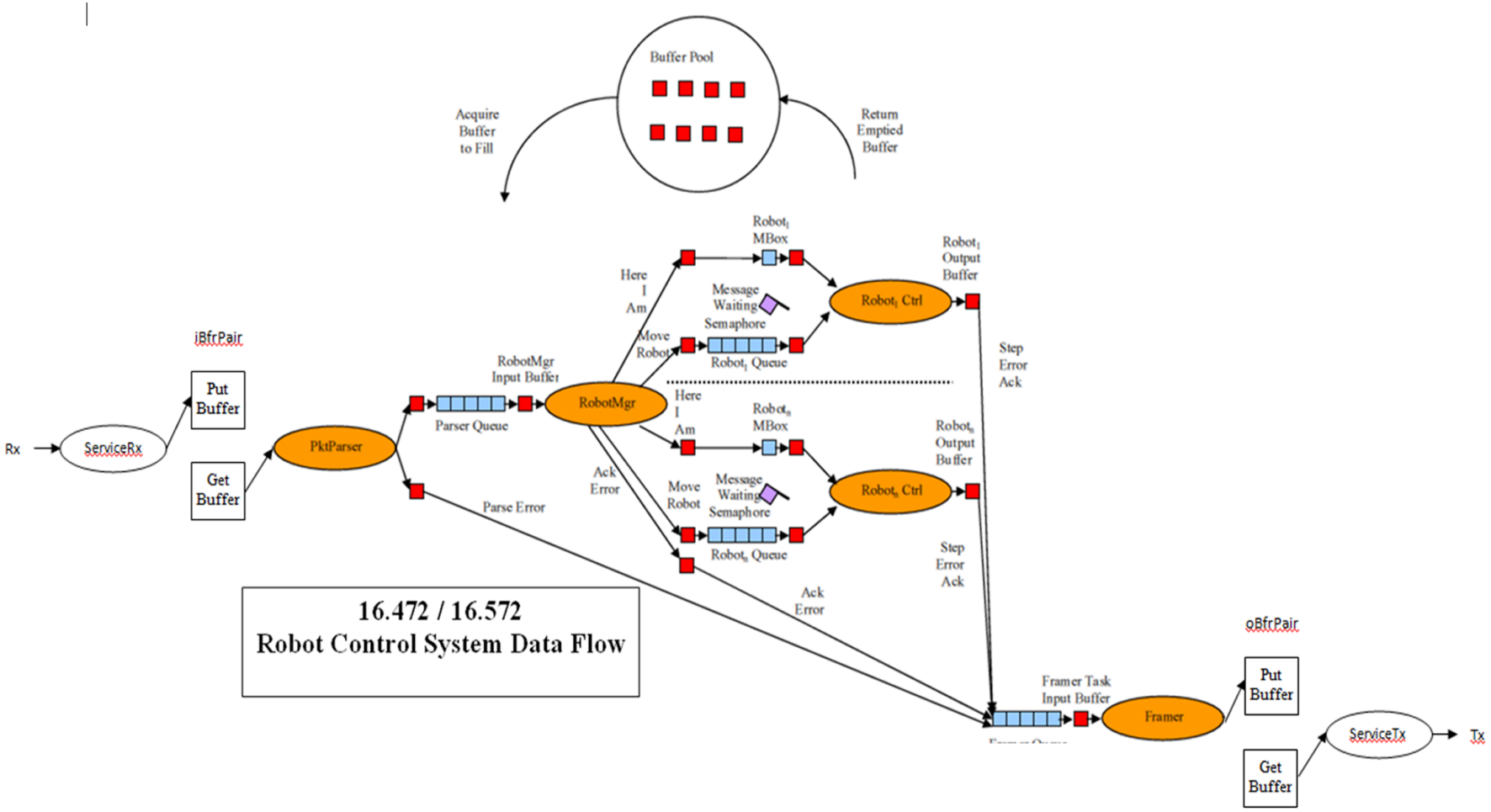
Byte Location in Payload	Value
0	Payload Length in Bytes = 5
1	1
2	2
3	0x0B
4	Error Code

Error Codes:

The table below lists the error codes, which may be reported by the Error Code Packet.

ERROR DISCOVERED BY	ERROR	ERROR CODE
Parser	P1 Error	1
	P2 Error	2
	P3 Error	3
	Checksum Error	4
	Bad Packet Length	5
Add Robot	Bad Robot Address	11
	Bad Location	12
	Location Occupied	13
	Robot Already Exists	14
Move Robot	Bad Robot Address	21
	Non-existent Robot	22
	Bad Location	23
Follow Path	Bad Robot Address	31
	Non-existent Robot	32
	Bad Location	33
Loop	Bad Robot Address	41
	Non-existent Robot	42
	Bad Location	43
Stop	Bad Robot Address	51
	Non-existent Robot	52
Robot Manager Task	Bad Message Type	61
Robot Controller Task	Robot Gave Up Trying to Reach Destination	0x1aa (where "aa" is the robot address)

Project Architecture:



Each of the ovals that are represented above are different tasks that are created with different priorities in uC/OS.

In the above data flow diagram, ServiceRx receives the data through ISR and puts the data into Put Buffer of iBfrPair. Once the complete data is filled into Put Buffer the address of Put Buffer and Get Buffer are swapped and semaphore closediBfr count is increased using OSSemPost() for next data to be read into iBfrPair. The data can be added if there is one empty buffer available in iBfrPair. Then using GetByte() the data form Get Buffer of iBfrPair is read in PktParser task. This task first checks for packet errors like preamble, packet length errors etc. If it encounters any error, error message type is forwarded to framer queue. The queue comprises of sequence of buffers of size poolsize having data waiting to be sent to framer task. If none of the data is incorrect, the payload is extracted into parser queue ready to be transferred to RobotMgr task. The RobotMgr task checks for logic errors like incorrect robot address, incorrect destination etc. given by the user.

This task can continue to perform if there is buffer available in parser queue, if not available the processor will suspend the RobotMgr task and allow any other high priority waiting task to complete its activities. If there is no logical error, the payload is then transferred to robot Ctrl task of a given robot address through robot queue and current address of robot is sent to Robot Ctrl task of that robot address through Robot Mailbox for the same task. Robot Ctrl task is set to re-entrent mode and can spawn upto 13 task if 13 robot calls are invoked. It is achieved by creating an array of size 13 for stack and an array of size 13 for 13 different Task Control Block. In the Robot Ctrl task, the robot payload directed to a given robot is created based on type of command request is sent by Ctrl Center. To move a robot through a shortest given path the change in x and y coordinate are calculated and based on the sign of the changes the final destination of the data is located in a quadrant with respect to current location.

To follow a shortest path the robot is made to move in a direction which first makes change in x equal to change in y and once equal the robot is made to move diagonally. To avoid collision a two-dimensional array is created to store the update locations of robots. The array is first initialized to zero and once a robot is added to particular location the two-dimensional array with given x and y indices is updated to one. Once the robot starts moving the array gets updated and while moving in a particular direction if the next location is already occupied then the direction is avoided, and next nearby available direction is chosen. Every packet created in Robot Ctrl is then forwarded to robot or Ctrl Center through framer using framer queue. The framer task reads the data from queue and then based on the type of data it adds header and footer to the message and forwards the packet to putBfr of oBfrPair. Once a buffer is completely read from any queue the buffer in the queue is freed to make space available for next data using free() from MemMgr module. The data is then sent out using ServiceTx()

through ISR. Again, here a semaphore keeps count of available buffers in oBfrPair and halts the transfer if there is no full buffer available.

The higher-level details of each task are given below.

Parser Task:

Module Name	PktParser.c
Purpose	To parse the incoming packet to Robot Manager
Tasks	CreatePktParserTask - interacts with Robot Manager and framer
Task Priority	6
Queues	ParserQueue- To stack the payload to robot manager Posted by parser and pended by robot manager FramerQueue- To stack the packets that are to be sent to the framer task.
Mailboxes	None
Semaphores or Mutexes	None
Data Source	Serial IO driver (Main Controller, Robots)
Data Destination	Robot manager, Framer
Special Data Structures	None
Shared Data	None

RobotManager Task:

Module Name	RobotMgr.c
Purpose	To read the data from parser and construct the message and send it to Robot Controller
Task	RobotManagerTask() - interacts with Robot Controller, Framer, Parser
Task Priority	3
Queues	Parser Queue - To read the buffers from the queue Robot Queue - To stack the messages that are sent to Robots Framer Queue – To stack the messages that are sent to Framer
Mailboxes	RobotMailbox – To send the HereIam or Stop message because these can't stay in a queue
Semaphores or Mutexes	None
Data Source	Packet Parser
Data Destination	Robot Controller, Framer
Special Data Structures	Payload
Shared Data	None

Robot Controller Task:

Module Name	RobotCtrl.c
Purpose	To read the data from RobotQueues and construct the message and send it to Framer.
Task	RobotControllerTask() - interacts with Robot Manger, Framer
Task Priority	1
Queues	Robot Queue - To read the buffer from the robot manager task Framer Queue - To stack the data to be sent to the framer task
Mailboxes	Robot Mailbox (MQueue) - To receive the Stop command or HereIam command
Semaphores or Mutexes	Mutex – used to lock the task when executing a critical section
Data Source	Robot Manager
Data Destination	Framer
Special Data Structures	Payload, Robot Locations, HereIam
Shared Data	Floor updating, Step Direction

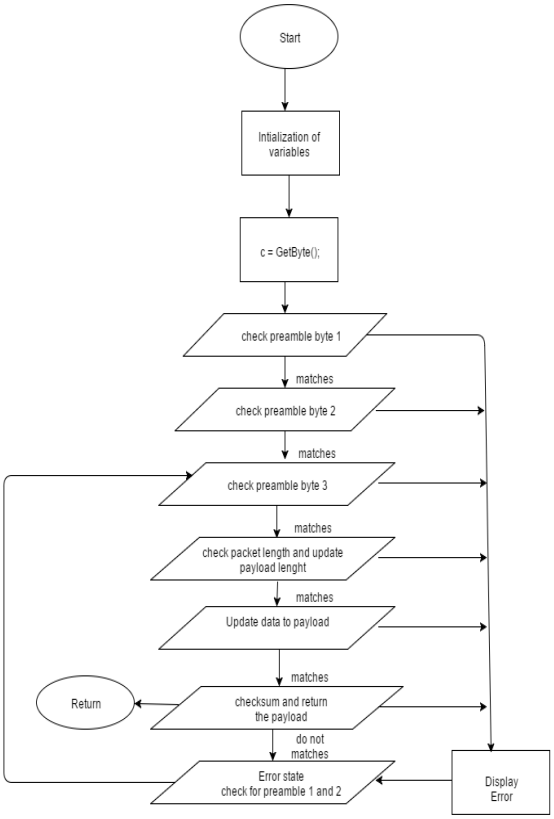
Framer Task:

Module Name	Framer.c
Purpose	To read the data from FramerQueues and construct the packet and send to ControlCenter
Task	FramerTask() - Interacts with Parser, Robot Manager, Robots
Task Priority	2
Queues	Framer Queue - To read the messages from the Queue
Mailboxes	None
Semaphores or Mutexes	None
Data Source	Packet Parser, Robot Manager, Robot Controller
Data Destination	Control Center
Special Data Structures	Framer
Shared Data	None

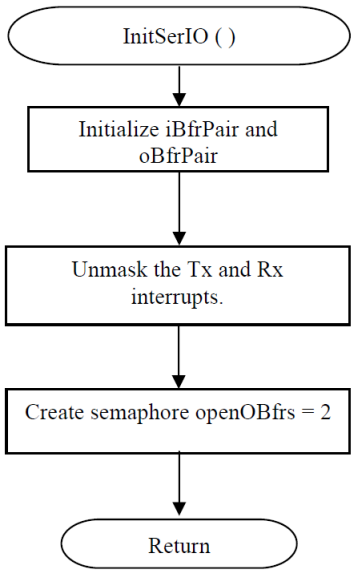
Software Description:

Flow Charts:

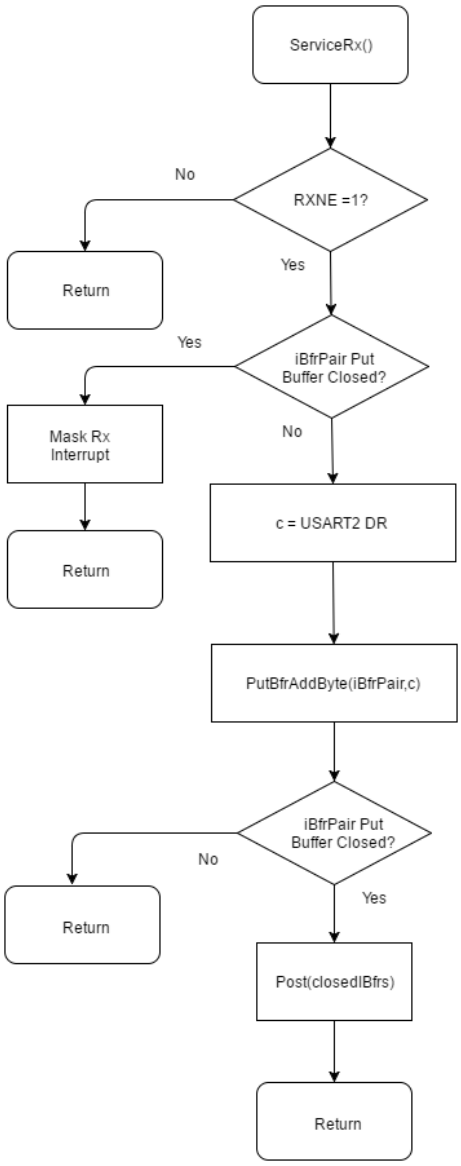
Packet Parser Finite State Machine:



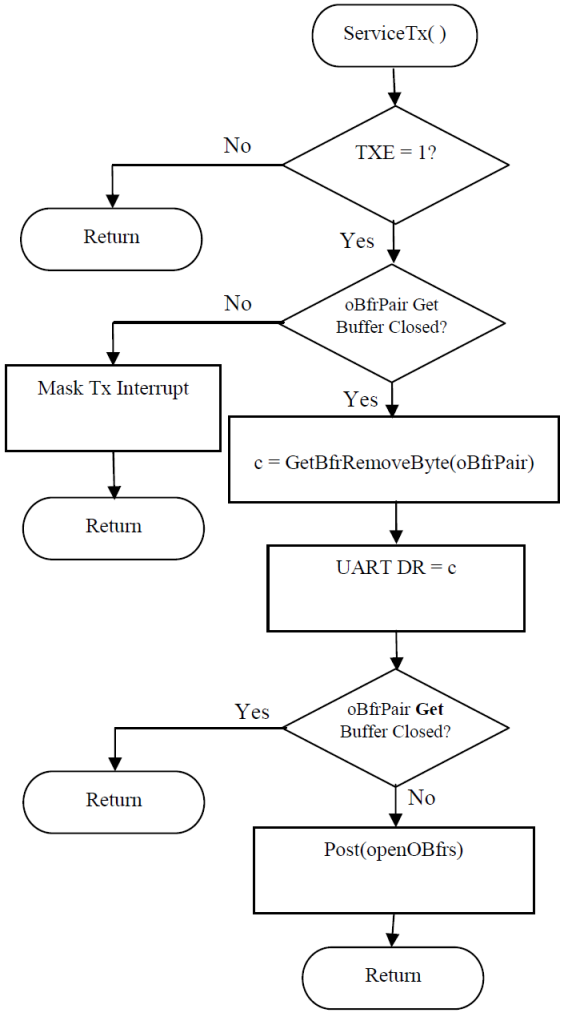
Init Ser IO:



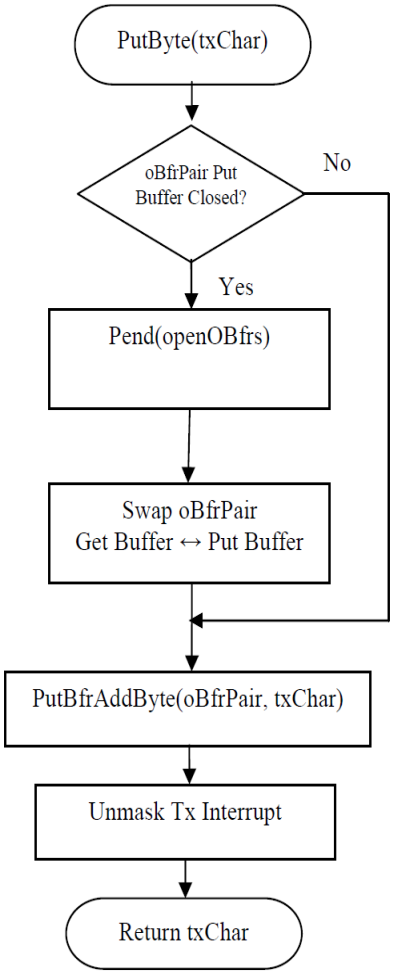
ServiceRX:



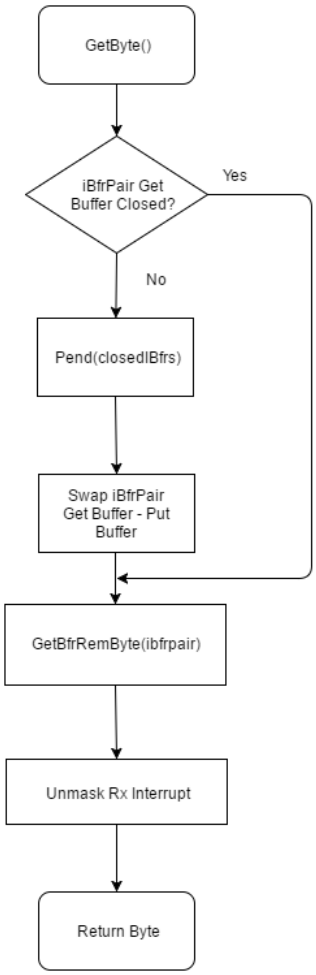
ServiceTx:



PutByte:



GetByte:



Details of each module and the function involved in those modules are discussed in coming pages.

MODULE SerIODriver.C:

SerIODriver DATA TYPE AND FUNCTIONS

PURPOSE	PROTOTYPE	PARAMETERS	RETURN VALUE
Initialize the RS232 I/O driver by initializing both iBfrPair and oBfrPair. Unmask the Tx and the Rx. Also, intialize the two semaphores openObfrs and closedIBfrs.	void InitSerIO(void)	None	None
If the oBfrPair put buffer is not closed, write one byte into the buffer, unmask the Tx interrupt, and return txChar as the return value. If, the buffer is closed pend on the semaphore openOBfrs and then swap the buffers.	CPU_INT16S PutByte(CPU_INT16S c)	The byte that is to be transmitted	txchar
If TXE = 0, just return. Otherwise, if the get buffer is not closed, mask the Tx interrupt and return. If TXE = 1 and the get buffer is closed, remove the next byte from the get buffer and output it to the Tx. If the get buffer is now open, post the semaphore openOBfrs.	void ServiceTx(void)	Buffer address	None
If iBfrPair get buffer is closed read the byte from the buffer or pend on the closediBfr semaphore	CPU_INT16S GetByte(void)	None	Byte from the buffer
If RXNE = 0 add the byte to the transmit data register and post the semophore openObfrs if the buffer is empty If RXNE = 1 return.	void ServiceRx(void)	None	None

MODULE BUFFER.C:

BUFFER DATA TYPE AND FUNCTIONS

```
typedef struct
{
volatile CPU_BOOLEAN closed; /* -- True if buffer has data ready to process */
CPU_INT16U size; /* -- The capacity of the buffer in bytes */
CPU_INT16U putIndex; /* -- The position where the next byte is added */
CPU_INT16U getIndex; /* -- The position of the next byte to remove */
CPU_INT08U *buffer; /* -- The address of the buffer data space */
} Buffer;
```

PURPOSE	PROTOTYPE	PARAMETERS	RETURN VALUE
Initialize a buffer: record the size set putIndex and getIndex to zero and mark the buffer open.	**void** BfrInit(Buffer *bfr, CPU_INT08U *bfrSpace, CPU_INT16U size);	buffer address, address of the buffer data space, buffer capacity in bytes	None
Reset the buffer: set putIndex and getIndex to zero and mark the buffer open.	**void** BfrReset(Buffer *bfr)	buffer address	None
Test whether or not a buffer is closed.	CPU_BOOLEAN BfrClosed(Buffer *bfr);	buffer address	TRUE if closed; otherwise FALSE
Mark the buffer closed.	**void** BfrClose(Buffer *bfr);	buffer address	None
Mark the buffer open	**void** BfrOpen(Buffer *bfr);	buffer address	**None**
Test whether or not a buffer is full.	CPU_BOOLEAN BfrFull(Buffer *bfr);	buffer address	TRUE if full,otherwise FALSE
Add a byte to a buffer at position “putIndex” and increment “putIndex” by 1. If the buffer becomes full, mark it closed.	CPU_INT16S BfrAddByte(Buffer *bfr, CPU_INT16S theByte);	buffer address byte to be added	The byte added, unless the buffer was full. If the buffer was full, return -1.
Return the byte from position “getIndex” or return -1 if the buffer is empty.	CPU_INT16S BfrNextByte(Buffer *bfr);	buffer address	The byte from position “getIndex” unless the buffer isempty. If the buffer is empty, return -1.
Return the byte from position “getIndex” and increment “getIndex” by 1. If the buffer becomes empty, mark it open.	CPU_INT16S BfrRemoveByte(Buffer *bfr);	buffer address	The byte from position “getIndex” unless the buffer is empty. If the buffer is empty, return -1.

MODULE BfrPair.C:

BfrPair DATA TYPE AND FUNCTIONS

typedef struct
{
CPU_INT08U putBfrNum; /* -- The index of the put buffer */
Buffer buffers[NumBfrs]; /* -- The 2 buffers */
} BfrPair

PURPOSE	PROTOTYPE	PARAMETERS	RETURN VALUE
Initialize both buffers of the buffer pair.	void BfrPairInit(BfrPair *bfrPair, CPU_INT08U *bfr0Space, CPU_INT08U *bfr1Space, CPU_INT16U size);	buffer pair address address of buffer 0 space address of buffer 1 space buffer capacity in bytes	None
Reset the put buffer.	void PutBfrReset(BfrPair *bfrPair);	buffer pair address	None
Obtain the address of the get buffer's buffer data space.	CPU_INT08U *GetBfrAddr(BfrPair *bfrPair);	buffer pair address	The address of the get buffer's buffer data space
Test whether the put buffer is closed,	CPU_BOOLEAN PutBfrClosed(BfrPair *bfrPair);	buffer pair address	TRUE if the put buffer is closed, otherwise FALSE
Test whether the get buffer is closed,	CPU_BOOLEAN GetBfrClosed(BfrPair *bfrPair);	buffer pair address	TRUE if the get buffer is closed, otherwise FALSE
Mark the put buffer closed.	**void** ClosePutBfr(BfrPair *bfrPair);	buffer pair address	None
Mark the get buffer open.	**void** OpenGetBfr (BfrPair *bfrPair);	buffer pair address	None
Add a byte to the put buffer at position "putIndex" and increment "putIndex" by 1. If the buffer becomes full, mark it closed.	CPU_INT16S PutBfrAddByte(BfrPair *bfrPair, CPU_INT16S byte);	buffer pair address the byte to be added	The byte added, unless the buffer was full. If the buffer was full, return -1.
Return the byte from position "getIndex" of the get buffer or return -1 if the get buffer is empty.	CPU_INT16S GetBfrNextByte(BfrPair *bfrPair);	buffer pair address	The byte from position "getIndex" of the get buffer unless the get buffer is empty. If the buffer is empty, return -1.
Return the byte from position "getIndex" iand increase the get buffer's "getIndex" by 1. If the buffer becomes empty, markit open.	CPU_INT16S GetBfrRemByte(BfrPair *bfrPair);	buffer pair address	The byte from position "getIndex" of the get buffer unless the get buffer is empty. If the get buffer is empty, return -1.

PURPOSE	PROTOTYPE	PARAMETERS	RETURN VALUE
Test whether a buffer pair is ready to be swapped. It is ready if the put buffer is closed and the get buffer is open,	CPU_BOOLEAN BfrPairSwappable(BfrPair *bfrPair);	buffer pair address	TRUE if ready to swap, otherwise FALSE
Swap the put buffer and the get buffer and reset the put buffer.	void BfrPairSwap(BfrPair *bfrPair);	buffer pair address	None

MODULE PktParser.C:

PktParser DATA TYPE AND FUNCTIONS

typedef struct

```
{
  CPU_INT08S  payloadLen; //length of the payload
  CPU_INT08U  msgdata[1]; //Remaing data from packet
}PktBfr;
```

PURPOSE	PROTOTYPE	PARAMETERS	RETURN VALUE
Creates and initializes the Parser task.	CPU_VOID CreateParserTask(CPU_VOID);	None	None
Allocate a buffer, Extract the payload and keep in the buffer and post it to the parser queue. Implements a finite state machine to receive the packets correctly	CPU_VOID ParsePkt(CPU_VOID *data)	Address of data that can be passed while creating a task	None
Allocate a buffer, add the error number and post it to the framer queue	void ErrorState(CPU_INT08U ErrorNumber)	buffer pair address	The address of the get buffer's buffer data space

MODULE Framer.C:

Framer DATA TYPE AND FUNCTIONS

typedef struct

```
{
  CPU_INT08U Msg; //can be error number or can be direction of the robot
  CPU_INT08U Pkt_type; //packet type that is to be sent
  CPU_INT08U CurrentRobot; //robot address
}Framer;
```

PURPOSE	PROTOTYPE	PARAMETERS	RETURN VALUE
Create and Initialize the Framer task.	CPU_VOID CreateFramerTask(CPU_VOID)	None	None
Task function for Framer task.	CPU_VOID FramerTask(CPU_VOID *data)	pointer to task data	None
Creates the packet as defined in the output packet structure and stores it in a buffer.	void Framedata(CPU_INT08U Robot,CPU_INT08U Pkt_type,CPU_INT08U Msg_code)	Robot – robot address Pkt_type – Type of packet Msg_code - Message	None

MODULE PBUFFER.C:

PBUFFER DATA TYPE AND FUNCTIONS

```
typedef struct
{
        CPU_INT08U  *in;          // Points to space to add next byte
        CPU_INT08U  *out;          // Points to next byte to remove
        CPU_INT08U  bfr[PBfrSize];    // The buffer storage area
}PBuffer;
```

PURPOSE	PROTOTYPE	PARAMETERS	RETURN VALUE
Test if the buffer is empty	CPU_BOOLEAN Empty(PBuffer *theBfr)	buffer address	true if the buffer is empty false otherwise
Add a byte to the buffer at the position "in," and increment "in."	CPU_INT16S AddByte(PBuffer *theBfr, CPU_INT16S theByte)	buffer address, the byte that is to be added	Returns the byte that is added
Remove the byte at position"out" from the buffer and increment "out.".	CPU_INT16S RemoveByte(PBuffer *theBfr)	buffer address	Returns the byte that is removed from the buffer
Initialize a buffer. Reset "in" and "out" to the beginning of the buffer.	void InitBfr(PBuffer *theBfr)	buffer address	None

MODULE MemMgr.C:

MemMgr DATA TYPE AND FUNCTIONS

PURPOSE	PROTOTYPE	PARAMETERS	RETURN VALUE
Initialize the memory manager. Create a buffer available semaphore and memory for the buffer	void InitMemMgr(void)	None	None
Allocate a buffer from the buffer pool - block if none available.	PBuffer *Allocate(void)	None	Returns the address of the buffer that is allocated
Return a buffer to the pool.	void Free(PBuffer *bfr)	Buffer address	None

MODULE Project.C:

Project DATA TYPE AND FUNCTIONS

PURPOSE	PROTOTYPE	PARAMETERS	RETURN VALUE
Main function which initializes the OS and run the user application(init task)	CPU_INT32S main (CPU_VOID)	None	Successful completion
Creates all the tasks and start running	static CPU_VOID Init (CPU_VOID *data)	Pointer to data that can be passed while creating the task	Returns the address of the buffer that is allocated

MODULE RobotMgr.C:

RobotMgr DATA TYPE AND FUNCTIONS

```
typedef struct /*payload structure datatype*/
{
 CPU_INT08U  payloadLen;
 CPU_INT08U  dstAddr;
 CPU_INT08U  srcAddr;
 CPU_INT08U  msgType;
 union
 {
  struct
  {
   CPU_INT08U RobotAddress;
   struct
   {
    CPU_INT08U X;
    CPU_INT08U Y;
   }Pos[10];
  }msgcmd;
 }cmdExt;
}payload;
```

PURPOSE	PROTOTYPE	PARAMETERS	RETURN VALUE
Creates and Initializes the Robot Manager Task	void CreateRobotManagerTask(void)	None	None
Robot Manager task's function that is to be executed	void RobotManagerTask(void *data)	Address of the data that can be shared when task creation	None
To create a ACk message that can be understood by framer.	void AckCommand(CPU_INT08U Type)	Type of the message that is to be acknowledged	None
To create a message taht can be understood by Robots.	void makemsg(PBuffer *bfr,CPU_INT08U CurrentRobot)	Buffer address, Robot address	None
To create a stop message that is posted to mailbox.	void stop(CPU_INT08U Robot)	Robot address	None
To create a HereIam message that is posted to mailbox.	void HereIam(CPU_INT08U Robot,CPU_INT08U x,CPU_INT08U y)	Robot address, X coordinate, Y coordinate	None

MODULE RobotCtrl.C:

MemMgr DATA TYPE AND FUNCTIONS

typedef struct
{
  CPU_INT08U RobotAddress;
  CPU_INT08U xLocation;
  CPU_INT08U yLocation;
  CPU_INT08U Nextx;
  CPU_INT08U Nexty;
  CPU_BOOLEAN STOP;
  CPU_INT08U Retry;
}RobotLocations;

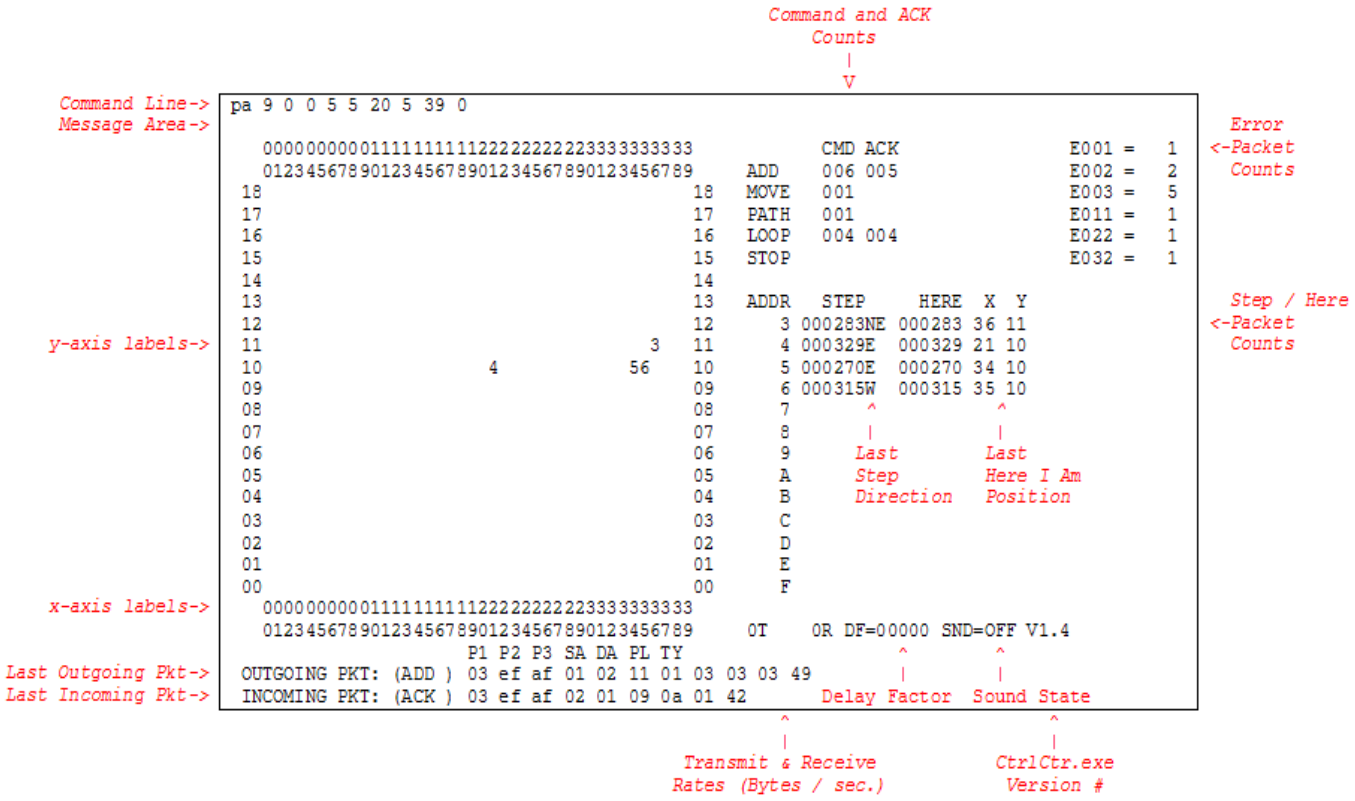
typedef struct
{
  CPU_INT08U Addr;
  CPU_INT08U RAddr;
  CPU_INT08U x;
  CPU_INT08U y;
}Here; //for the Hereiam and stop message

PURPOSE	PROTOTYPE	PARAMETERS	RETURN VALUE
Initialize the robot control tasks.	Void CreateRobot(RobotLocations Robots)	A data of type RpbptLocations	None
Robot Manager task's function that is to be executed	void RobotTask(void *data)	Robot Locations data is sent while creating a task	None
Take a decision on where to step and send the step packet to framer	Void StepRobot(CPU_INT08U Robot,CPU_INT08U x,CPU_INT08U y)	Robot address, X location, Y location	None
Take a decision on which direction to step	CPU_INT08U Step(CPU_INT08U cx,CPU_INT08U cy,CPU_INT08S X,CPU_INT08S Y)	current X location, current Y location, Future X location, Future Y location	Direction in which the robot moves
Update the robot's data in the floor datastructure if it is a hereIam message and Stop the robot if it is a Stop message	void HereIAM(CPU_INT08U CurrentRobot)	Robots address	None

Control Center User Manual:

The control center is running on a PC which sends and receives the packets using the serial port. The commands are taken from the user and sent to the Robot Control Center which are decoded and sent back to the PC. The robots are implemented as a simulation on the PC where the incoming packet is decoded a shown on the screen as can be interpreted by a robot.

CtrlCtr.exe V1.4 Display Screen Snapshot



Shown below are the list of commands that can be sent to the Robot Control System from the user. A robot's log is provided so that it will be easy to read the incoming packets and debug the application.

COMMAND	DESCRIPTION	COMMAND PACKET TYPE
AD *addr x y*	Add a new robot whose address is *addr* at position (*x*, *y*).	Add robot
MV *addr x y*	Move robot *addr* to position (*x*, *y*).	Move robot
PA *addr* x_1 y_1 x_2 y_2 x_3 y_3 …	Make robot *addr* follow the path from its current position to up to 10 stops at positions (x_1, y_1), (x_2, y_2), (x_3, y_3), …	Follow Path
LP *addr* x_1 y_1 x_2 y_2 x_3 y_3 …	Make robot *addr* repeatedly follow the path from its current position to up to 10 stops at positions (x_1, y_1), (x_2, y_2), (x_3, y_3), … (x_1, y_1), (x_2, y_2), (x_3, y_3), …(x_1, y_1), (x_2, y_2), (x_3, y_3), … The robot will continually trace out a looping path through the specified stop positions. Note that the loop does not include the starting position of the robot at the time the command is issued (unless the first stop is the initial position.	Loop
ST addr	Make robot *addr* stop looping.	Stop Looping
OF *filename*	Open and execute command file *filename*.cmd (Up to 20 levels of nesting).	None
FE 1 *n*	Force a preamble byte 1 error once per *n* command packets transmitted to the STM32F107.	None
FE 2 *n*	Force a preamble byte 2 error once per *n* command packets transmitted to the STM32F107.	None
FE 3 *n*	Force a preamble byte 3 error once per *n* command packets transmitted to the STM32F107.	None
FE 4 *n*	Force a checksum error once per *n* command packets transmitted to the STM32F107.	None
FE 5 *n*	Force a packet length error once per *n* command packets transmitted to the STM32F107.	None
FE 6 *n*	Force a message type error once per *n* command packets transmitted to the STM32F107.	None
FE	Stop forcing errors.	None
QU	Quit from CtrlCtr.exe.	None
DF n	Set the delay factor to "n." Increasing the delay factor slows the rate that Robot.exe transmits bytes on the RS232 link.	None
RE	Send a Reset packet to the microcontroller, forcing a reboot.	Reset
--	Comment prefix	None
WA	Await a key press before continuing script execution.	None
SO	Toggle sound on and off.	None

Conclusion: The robot control system is successfully implemented on STM32F107 development board and a PC. While implementing the project learned about UART communication, Device Drivers, Double Buffered IO, Preemptive Multitasking , Scheduling and Synchronization concepts.