By Tejal Sharma

Vellore Institute of Technology, Chennai

HDR Brachytherapy Machine  
Emulator

CERTIFICATE

This is to certify that the project titled “ HDR Brachytherapy Machine  
Emulator “ is a bonafide work of Ms. Tejal Sharma carried out under the guidance and supervision of Dr. A P Das. This report is part of the internship done by Ms. Tejal Sharma at BRIT, Vashi, Navi Mumbai.

Dr. A P Das

DGM (Design & Development)

Medical Engg Group

BRIT, Vashi,

Navi Mumbai

**Project Description**

**HDR**

**High Dose Rate(HDR) Brachytherapy or Karknidon** is a method for the treatment of cancer. Brachytherapy is a type of internal radiation therapy that treats cancer by placing radioactive materials near or inside the tumor.Brachytherapy treatments using remote controlled systems began in 1964. Early systems used cobalt-60,but the first **Iridium-192** machine was introduced in 1966. Remote Afterloaders (also known as a source delivery units or treatment units) are only used for temporary implants in a radiation oncology department or clinic*.*

Iridium-192 used in the machine is called the **Source** and is present inside a cask shielded by lead coating for radiation protection . Commands are sent from the Client to operate the HDR machine which are sent to the **Arduino Mega** for interpretation .The interpreted commands are sent to the **Arduino Nano** by the Arduino Mega microcontroller .The Arduino Nano board drives the motors and updates the encoder values based on the received commands .The Indexer is a mechanical arm that moves when certain commands are given to it. Three stepper motors are present with corresponding drivers in order to drive the Indexer, Source and Dummy. The machine has a disk-like structure with **20 slots** to atleast one of which the Indexer should be positioned in order to start the treatment process. The Dummy is sent out first for checking the path before the actual Source, from the machine in tubes. Once the path has been found out to be accurate, the Source is sent out through the tubes to the positions where the tumor is present inside the body using three insertion rods. The time of dosage depends on two factors-the size of the tumor and the strength of the source.



**Emulator**

The Emulator for the HDR has been designed using **.NET and C#** in order to facilitate the testing of commands and processes on a digital replica of the physical machine instead of the actual machine. The Emulator can be used to observe command outputs in the absence of the physical machine as well as for predictive analysis .

A program for the machine is created the IP address of which is used to connect to the Client instead of the machine. Commands are given at the Client’s side which are sent to the Server using the **TCP Protocol** which then returns the values of the encoders and sensors after processing and updation.

**Explanation of the Code for the Emulator:**

The program for the Emulator consists of three major threads:

1. **Main Thread:** Responsiblefor initializing bit values of the sensors based on the initialized encoder values and for communication between the TCP Client and the TCP Server.
2. **Emulator Thread:** Responsible for updating the values of different bits based on the inputs received from the TCP Client.
3. **Input Thread:** Responsible for taking inputs on the TCP Server side for Door Switch, Treatment Switch, Emergency Switch and Last Man Out.
4. Global static variables for sensor1,sensor2,encI,encD ,encS , dwellTime , totalTime ,errorCode are created so that they can be accessed throughout the class “Program”.encI is initialized to 1000 while other variables are by default initialized to 0.

**sensor1=0**

Bit Position:| 7 | 6 | 5 | 4 |

Description:| IndexerLock | Dummy Home | Dummy Overshoot | Indexer Home |

| 3 | 2 | 1 | 0 |

| AdapterSensor | Source Home | Source Overshoot | Source Out |

**sensor2=0**

Bit Position:| 7 | 6 | 5 | 4 |

Description:| Treatment Switch | Emergency Switch | Last Man Out | Door Switch |

| 3 | 2 | 1 | 0 |

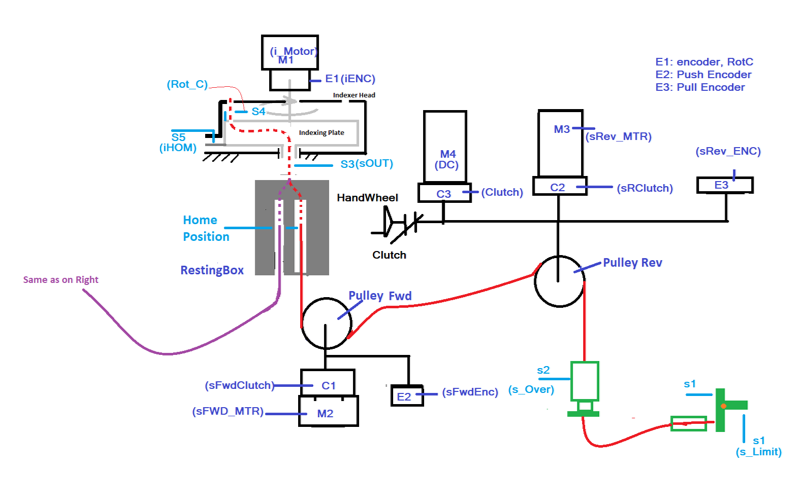
| Indexer Calibrated | Mains On | CommandInProgress | Moving |

1. A global static constant array named “indexerVals” is created to store the values of all the different indexed locations.
2. **Main Thread**
3. **Socket Creation**:
   * 1. An endpoint is created combining the Port Number and IP Address of the TCP Server Machine.
     2. A TcpListener is created to enable the TCP Server to listen to any incoming requests from potential TCP Clients.
     3. The TcpListener then binds with the endpoint to enable communication.
     4. Once a TCP Client’s request is encountered ,the TCP Server accepts the same and a handler is created in order to handle the communication process with the TCP Client.
     5. The handler is used to get the common stream through which the TCP Server and TCP Client would communicate with each other.
     6. A buffer is created in order to store the data (that would be sent as bytes) sent by the TCP Client on the common stream.
     7. The data stored in the buffer would then be converted to the string datatype for interpretation.
4. **Communication:**
   1. During the process of communication, the TCP Server reads data from the buffer sent by the TCP Client.
   2. First calls the HB function.

**HB(Heart Beat)** -Sends current values of the sensors and encoders.

* 1. Then the Emulator thread is created for the interpretation of the commands sent by the TCP Client on the common stream and then subsequently the bit values are changed based on the updated encoder values.

1. **Emulator Thread**
   * 1. First the format of the received data is checked using Regular Expressions to confirm that the received data is a command. If a command is confirmed further checks are made.
     2. Each command starts by first setting the 1st and 0th bit of sensor2 i.e “**CommandInProgress**” and “**Moving**” bits and on command completion the same bits are cleared to indicate that the command is completed.
     3. Any other command given to the Emulator while a command is already in progress i.e when the **CommandInProgress** and **Moving** bits are set ,is simply ignored, except the HB command.
     4. Hence while starting the execution of any command two conditions are checked:
        1. CommandInProgress bit value
        2. Treatment Switch ,Emergency Switch and Door Switch bit values. Commands cannot be executed if the Switches are not in the required state. After each iteration of the command as well, the switch values are checked before the continuation of execution.
     5. The Dummy and Source cannot be moved out unless the indexer is present at one of the indexed 20 slot locations .If one still tries to move them before the indexer is positioned properly ,*error 205* is sent to the Client indicating that the adapter/indexer is not present at the indexed location.
     6. **Source Out** **bit**(bit 0) of sensor1 is set when any one of the Source or Dummy reaches the Sout position of *1585 encoder counts*.
     7. **Source Overshoot bit** is set when the Source reaches *21,240 encoder* *counts i.e 4248 motor pulses*.
     8. **Dummy Overshoot** is set when the Dummy reaches -*21,240 encoder counts i.e 4248 motor pulses.*



* + 1. **EI(Erase Indexer)-** Reponsible forerasing the encoder value of the Indexer and clearing all the bits that were set by Indexer movements. Also sets Indexer Home bit(sensor1 bit 4).
    2. **ES(Erase Source)-** Reponsible forerasing the encoder value of the Source and resetting all the bits that were set by Source movements. Also sets Source Home bit(sensor1 bit 2).
    3. **ED(Erase Dummy) -** Reponsible forerasing the encoder value of the Dummy and all the bits that were set by Dummy movements. Also sets Dummy Home bit(sensor1 bit 6).
    4. **MSFn(Move Source Forward by n motor pulses)-** Responsible for moving the Source out .After moving out it is checked if Sout position is crossed or reached. If this condition is satisfied i.e the value of encS is greater than or equal to 1585,**Source Out** bit is set and the Dummy cannot be moved as already a wire is present at the Sout sensor .If one still tries to move the Dummy, *error 500* is sent to the Client indicating the presence of other wire at the Sout sensor. If while moving forward the **Source Overshoot** sensor is hit i.e the Source encoder reaches the value of 21,240, the Source Overshoot bit(bit 1) of sensor1 is cleared. If one tries to move the wire further out, *error 209* is sent to the Client indicating that the Overshoot sensor is crossed and the target movement is not achieved as the entire wire is out of the machine .Following this the value of encS is set to 21,240 which is the maximum feasible movement.
    5. **MSRn(Move Source Reverse by n motor pulses)-** Responsible formoving the Source back into the machine. If the Source is already at home and one tries to give the MSR command, *error 208* is sent to the Client indicating that the home sensor is hit while moving the Source in and target movement is not achieved and the value of encS remains 0 .While moving back, if the value of encS becomes less than 21,240, the **Source Overshoot** bit(bit 1) of sensor1 is set indicating that the Source wire is now inside the machine.If the value still becomes less than 1585 ,the **Sout** bit is cleared indicating no wire is present at the Sout position.
    6. **MDFn(Move Dummy Forward by n motor pulses)-** Responsible for moving the Dummy out .After moving out it is checked if Sout position is crossed or reached. If this condition is satisfied i.e the value of encD is less than or equal to -1585(negative sign indicates that the Dummy encoder is connected in an orientation such that its value becomes negative as the dummy moves out i.e when value is in the range -1585 to -21,240),**Source Out** bit is set and the Source cannot be moved as already a wire is present at the Sout sensor .If one still tries to move the Source, *error 500* is sent to the Client indicating the presence of other wire at the Sout sensor. If while moving forward the **Dummy Overshoot** sensor is hit i.e the Dummy encoder reaches the value of -21,240, the Dummy Overshoot bit(bit 1) of sensor1 is cleared. If one tries to move the wire further out, *error 204* is sent to the Client indicating that the Overshoot sensor has been crossed and the target movement is not achieved as the entire wire is out of the machine. Following this the value of encD is set to -21,240 which is the maximum feasible movement.
    7. **MDRn(Move Dummy Reverse by n motor pulses)-** Responsible formoving the Dummy back into the machine. If the Dummy is already at home and one tries to give the MDR command, *error 203* is sent to the Client indicating that the home sensor is hit while moving the Dummy in and target movement is not achieved and the value of encD remains 0. While moving back, if the value of encD becomes greater than -21,240 (negative sign indicates that the Dummy encoder is connected in an orientation such that its value becomes negative as the dummy moves out i.e in the range -21,239 to 0), the **Dummy Overshoot** bit(bit 1) of sensor1 is set indicating that the Dummy wire is now inside the machine. If the value still becomes greater than -1585 , the **Sout** bit is cleared indicating no wire is present at the Sout position.
    8. **MIFn(Move Indexer Forward by n motor pulses)-** Responsible for moving the Indexer in the forward(counter-clockwise) direction .If while moving forward the indexed location values are encountered(within a range of 10 encoder counts i.e indexed position-5 <= encI <= indexed position+5) the Rot\_C /AdapterSensor bit (sensor1 bit 3) is set for the period during which encI is within the specified range. Only when the Rot\_C bit is set, Source and Dummy movements can be achieved. Otherwise these commands would be simply ignored.

* + 1. **MIRn(Move Indexer Reverse by n motor pulses)-** Responsible for moving the Indexer in the reverse(clockwise) direction .While moving backward if the slot location values are encountered(within a range of 10 encoder counts i.e indexed position-5 <= encI <= indexed position+5) the Rot\_C /AdapterSensor bit (sensor1 bit 3) is set for the period during which encI value is within the specified range. Only when the Rot\_C bit is set, Source and Dummy movements can be achieved. Otherwise these commands would be simply ignored. If encI value becomes negative, 1,00,000 is added to the negative value indicating that the Indexer has now reached the greater indexed positions(as the Indexer rotates under the disk).
    2. **OI(Homing command for Indexer)-** Responsible for sending the Indexer Home i.e making value of encI equal to 0.While moving back home, it clears the Rot\_C bit indicating that the Indexer is not present at any indexed position and at encI=0 sets the Indexer Home bit(sensor1 bit 4).
    3. **OS(Homing command for Source)-** Responsible for sending the Source Home i.e making value of encS equal to 0.While moving back home, it clears all the bits that were set by Source movements and at encS=0 sets the Source Home bit(sensor1 bit 2).
    4. **OD(Homing command for Dummy)-** Responsible for sending the Indexer Home i.e making value of encD equal to 0. While moving back home, it clears all the bits that were set by Dummy movements and at encD=0 sets the Dummy Home bit(sensor1 bit 6).
    5. **R(errorCode Reset)-** Responsible for clearing the errorCode value.
    6. **Pn(Sending Indexer to nth encoder position)-** Responsible for sending the Indexer at the specified encoder count position i.e making the value of encI equal to n .During the movement, if the indexed location values are encountered(within a range of 10 encoder counts i.e indexed position-5 <= encI <= indexed position+5) the Rot\_C /AdapterSensor bit (sensor1 bit 3) is set while encI is within the specified range.
    7. **Sn(Sending Source to nth encoder position)-** Responsible for sending the Source at the specified encoder count position i.e making the value of encS equal to n .During the movement, if Sout position(1585 encoder counts) and Source Overshoot(21,240) values are crossed, their respective bits are updated.
    8. **Dn(Sending Dummy to nth encoder position)-** Responsible for sending the Dummy at the specified encoder count position i.e making the value of encD equal to n .During the movement if Sout position(-1585 encoder counts) and Dummy Overshoot(-21,240) values are crossed, their respective bits are updated.
    9. **Wn(Wait for n seconds)-** Responsible for sending the process in the **sleep** state for n seconds.
    10. **I(Indexer Calibration)-** Responsible for calibrating the Indexer and sending the Indexer to Home(setting Indexer Home bit).

1. **Input Thread**
   1. Always remains active to take inputs from the TCP Server so that the switches can be controlled at any point during the communication process.
   2. The switches are of 4 types:
      1. **Door Switch(sensor2 bit 4)**-Initially set to 0 i.e the door is closed. There exists two commands to control this switch:
         * 1. **DOOR -**Responsible for setting the Door Switch.
           2. **DOORR-** Responsible for clearing the Door Switch.
2. **Last Man Out(sensor2 bit 5)-**Initially set to 1 i.e no one (except the patient) is present inside the room .There exists two commands to control this switch:
3. **LMO-** Responsible for setting the Last Man Out Switch.
4. **LMOR-** Responsible for clearing the Last Man Out Switch.
5. **Emergency Switch(sensor2 bit 6)-**Initially set to 0 i.e emergency is not there .There exists two commands to control this switch:
6. **EMG-** Responsible for setting the Emergency Switch.
7. **EMGR-** Responsible for clearing the Emergency Switch.
8. **Treatment Switch(sensor2 bit 7)-**Initially set to 1 i.e the treatment key has been inserted and the treatment can be started. There exists two commands to control this switch:
9. **TRT-**Responsible for setting the Treatment Switch.
10. **TRTR-** Responsible for clearing the Treatment Switch.
    1. If any switch bit is changed, ongoing command gets hauled but encoder values do not get erased. No other command will be processed till the switch bits are not reverted back except the HB command.
    2. After the switch bits are reverted, the hauled process does not continue its execution because it was completely terminated by the bits getting changed. To continue execution the command needs to be given again.