COSMIC RAY DETECTOR/ SIMULATOR

STAGE 1:

Our initial proposal was to build a cosmic ray detector using ordinary fluorescent tubes. We planned to use 2 tubelights placed one below the other, biased at a DC voltage, with wires coiled around them given high and low voltages alternately. When a muon enters a particular region between any two coils, the collisions with the vapour atoms of the tube creates an avalanche of electrons in that region, resulting in a current spike. Coincident detection of a current spike in 2 adjacent tubes confirms that the event is not terrestrial/noise.

WORK DONE:

1. The tube needs to be operated in 3 modes, namely the 'OFF', 'ON' and the 'DIM' modes. In the latter, the tube glows dimly, with just enough bias voltage to maintain a minimum level of ionisation in the tube. For this we used a half wave rectifier that steps down power and due to the reduction in the frequency of flickering, the tube glows dimly. We built a switch to operate the tube in any of these 3 modes.
2. The coils were given a DC voltage of 1000V using a step-up transformer. For better control, 220V AC input was fed to a linear voltage regulator, the output of which was fed to the transformer.

RESULT:

We blew up the transformer and switch and due to incompetency to deal with high voltages and also the time constraint, decided to switch over to the conventional muon detector using scintillators and photomultiplier tubes (PMTs).

STAGE 2:

We decided to use the scintillators and PMTs in the Radiation Lab to build a conventional coincidence detector using the Arduino programming

board. The analog outputs of 2 PMTs needed to be converted to TTL before being fed to Arduino for coincidence detection. The biggest hurdles were:

1. Difficulty to get the desired analog output from the PMT, perhaps due to malfunctioning of the PMT.
2. The width of the pulse is of the order of ~500 ns. So converting it to a TTL needs considerably fast logic.

STAGE 3:

Due to (1) above, we decided to use Co^{60}, a photon source to give us the analog signal. 2 PMTs were placed facing each other with the source in the middle. For signal processing, we used the CAMAK modules in the Radiation Lab. These contain modules for amplification of PMT signal, analog to TTL conversion and also coincidence detection between the two TTL signals thus obtained. We used the direct count obtained thus to verify the count obtained by us using the Arduino.

CIRCUIT /BLOCK DIAGRAM:

ARDUINO CODE FOR COUNTING PULSES IN INDIVIDUAL CHANNELS:

```
#include<avr/interrupt.h>
volatile int count = 0;
volatile float time = 0;
int out = 13;
int in = 3;
void setup()
{
  Serial.begin(9600);    // opens serial port, sets data rate to 9600 bps
  pinMode(in,INPUT);
  pinMode(out,OUTPUT);
  attachInterrupt(1,pulse,RISING);  //interrupt to detect pulses at pin 2
}
void loop()
{
 time=micros();
 if (time < 20000000)
 {
   Serial.println(count);
 }
 delay(1000);
}
void pulse() {count++; }
```

ARDUINO CODE FOR DETECTING COINCIDENCE:

```
#include<avr/interrupt.h>
volatile int count0 = 0;
volatile int count1 = 0;
volatile int countcoin = 0;
volatile float time = 0;
volatile float time0 = 0;
volatile float time1 = 0;
volatile float thresh = 1;
int out = 13;
int in0 = 2;
int in1 = 3;
void setup()
{
  Serial.begin(9600);    // opens serial port, sets data rate to 9600 bps
  pinMode(in0,INPUT);
  pinMode(in1,INPUT);
  pinMode(out,OUTPUT);
  attachInterrupt(0,pulse0,RISING);
  attachInterrupt(1,pulse1,RISING);
}
```

```
void pulse0()
{
 //count0++;
 time0 = micros();
}
void pulse1()
{
 //count1++;
 time1 = micros();
 /*if (time1 == time0)
  {
    countcoin++;
  }
*/
  if (((time1-time0) >= 0) && ((time1-time0) <= 1))
  {
    countcoin++;
  }
  else if (((time0-time1) >= 0) && ((time0-time1) <= 1))
  {
    countcoin++;
  }
}
void loop()
{
 //time=micros();
```

```
//if (time < 1000000)
//{
//if (countcoin != 0)
 //{

   Serial.println(countcoin);

 //}

//  Serial.print(count0);
//  Serial.print("\t");
//  Serial.println(count1);
//}
 delay(1000);
}
//Note: The different commented parts were for debugging purposes
```

RESULTS:

1. The count obtained using Arduino for individual channelsmatched with that of the module for each individual signal. We got ~400 counts per second.
2. However we got a large error in the coincidence measurement between the 2 signals. This is primarily because of the limitation in the speed of the Arduino module. The process of measuring the time-stamp of an event itself takes up many clock cycles, and thus introduces large errors in the measurement process.
3. This called for the use of an external clock of a higher frequency, which was the motivation for Stage 4.

STAGE 4: DESIGNING A COINCIDENCE DETECTOR

We tried to build a fast circuit which can detect coincidences between nanosecond pulses. An implicit assumption is that the time between 2 pulses is not of the order of the width of the pulse, in which case we would need even more modifications or perhaps an entirely different design altogether.

The basic idea is as follows:

We have two TTL signals A & B and a high frequency clock signal. A is fed to a retriggerable multivibrator whose output Am is a TTL of variable width, depending on the choice of external resistor and capacitor values. B is fed to the clock input of a toggle flip-flop, and an inverted A is used to reset its output Bt. Then we AND Am with inverted Bt and the high frequency clock, and feed the resultant output to the counter. A is also used to reset the counter. Thus effectively, the counter counts the number of clock pulses between rising edges of A and B. Then the Arduino reads the digital output pins of the counter whenever it receives a B pulse. It also displays the cumulative counts (from t=0) on the serial monitor every second.

ARDUINO CODE FOR READING OUTPUT OF COUNTER TO CHECK COINCIDENCE:

```
#include<avr/interrupt.h>

volatile int countA = 0;

volatile int countB = 0;
```

```
volatile int c0 = 0;
volatile int c1 = 0;
volatile int c2 = 0;
volatile int c3 = 0;
volatile int c4 = 0;
volatile int c5 = 0;
volatile int c6 = 0;
volatile int c7 = 0;
volatile int c8 = 0;
volatile int c9 = 0;
volatile int c10 = 0;
volatile int c11 = 0;
volatile int c = 0;
int thresh = 0;
volatile int count;
void setup()
{
  Serial.begin(9600);
  pinMode(0,INPUT);
  pinMode(1,INPUT);
  pinMode(4,INPUT);
  pinMode(5,INPUT);
  pinMode(6,INPUT);
  pinMode(7,INPUT);
  pinMode(8,INPUT);
  pinMode(9,INPUT);
  pinMode(10,INPUT);
```

```
  pinMode(11,INPUT);
  pinMode(12,INPUT);
  pinMode(13,INPUT);
  attachInterrupt(0,pA,RISING);
  attachInterrupt(1,pB,RISING);
}
void pA()
{ countA++;
}
void pB()
{
  countB++;
  c0 = digitalRead(0);
  c1 = digitalRead(1);
  c2 = digitalRead(4);
  c3 = digitalRead(5);
  c4 = digitalRead(6);
  c5 = digitalRead(7);
  c6 = digitalRead(8);
  c7 = digitalRead(9);
  c8 = digitalRead(10);
  c9 = digitalRead(11);
  c10 = digitalRead(12);
  c11 = digitalRead(13);
  count = c0 + 2*c1 + 4*c2 + 8*c3 + 16*c4 + 32*c5 + 64*c6 + 128*c7 +
256*c8 + 512*c9 + 1024*c10 + 2048*c11;
```

```
  if (c < thresh)
  {
    count++;
  }
}
void loop()
{
  Serial.println(count);
  delay(1000);
}
```

CIRCUIT DIAGRAM: