Data Acquisition System

Group No: D11
Mihir Mulay, 05D07022, mihir_mailme@iitb.ac.in
Rutika Muchhala, 05010025, ruitka@ee.iitb.ac.in
Shreyas Shah, 05D07010, syshah@iitb.ac.in
Supervisors A. Karandikar, D. Ghosh

Abstract

We have attempted to implement a simple 16-channel data acquisition system. This system is designed to store data in an SD-Card, as well as transfer it to a computer where is interfaces with a GUI. This system can be controlled from the GUI, and the channels can be monitored in real-time, and their configurations can be modified.

Introduction

A data acquisition system (DAS) is device used to measure, digitise, log and analyse data. It finds applications in measurement and logging of environmental and operational parameters in vehicles, storage facilities, machinery, factories, etc. It essentially consists of sensors to measure data, logger units to record this data and often has a telemetry unit for communication and exchange of data with remote units as well as a software based analysis tool.

Specifications of such systems such as the sampling rate of sensors and communication system used depends on application and the environment in which the system is to operate. Storage of data may be done on magnetic media or in semiconductor memories. Communication may be wired or wireless. Wired systems can be implemented with any among the many protocols that exist, or with combinations those protocols.

Design

We have attempted to implement a DAS that can take in data from up to a maximum of 16 sensors at a time and store it in an SD-Card, as well as transmit it to a computer for further analysis. The sensors are connected to a controller node in a multi-drop network. Data is transmitted to this central node, which logs and can also transmit it to a computer. The communication between the sensors and central node is done using the RS485 protocol, and that between the central node and computer is done using the RS232 protocol.

The sensors are 4 ADC channels on 4 separate modules. These modules consist of an ATMega16 microcontroller, and an RS485 driver, SN75HVD05. ATMega 16 was chosen because of it s multi- channel ADC as well as various noise reduction features available for sampling. The on-chip 8-channel ADC is used to sample data from 4 channels, and transmitted serially via the RS485 connection. Data is transmitted at a rate of 38.6 bps. The cumulative transmission rate from the 16 sensors to the central module has been nominally restricted by design to 1600 bps.

The central module is run with an ATMega128 microcontroller. It was chosen primarily because it has 2 USART modules. The central module also has the same RS485 driver. It was initially planned to communicate directly with the SD-Card via the on-chip SPI interface. But

owing to difficulties in writing data effectively to the SD-Card directly because of a mismatch in logic levels, we used a ready SD-Card module that takes in data serially, and implements a FAT16 file-system. Since both the USART modules of the ATMega128 are used - one for the RS485 communication and one for the RS232 communication - we have used an ATTiny2313 microcontroller as an interface between the ATMega128 and the SD-Card module and data is exchanged between the two microcontrollers in a parallel fashion.

Shown in Fig. 1 is a schematic of the design

Specifications

Components

1. ATMega128 - 1 unit
2. ATMega16 - 4 units
3. ATTiny2313 - 1unit
4. SN75HVD05 - 5 units
5. MAX232 - 1 unit
6. UMMC-100-A3 (Ready made SD-Card module from Rogue Robotics) - 1unit
7. Crystals, diodes, capacitors, resistors and LEDs.

Data Exchange Protocol

The serial data exchange between the PC and the μC as shown above involves a set of headers, data packets, and acknowledgment and error replies. This is represented by the flowchart shown in Fig. 2.

CC - Configuration header. From μC to computer, configuration data consists of the connection status of all 16 channels in general, as well as the sampling rates of all channels during initialisation. From computer to μC, configuration data consists of the read status and sampling rate adjustment for a given channel.

DD - Header for sampled data transmission

AA - Acknowledgment for successful and expected reception

EE - Error in reception. A request for re-transmission is implied.

i_count - number of channels connected and read

Serial data exchange over the RS485 network too involves a set of requests and acknowledgments. Requests are called query words (Q.W.). Responses consist of the status word (S.W.) and the data packet. The data packet consists of the header, followed by the number of bytes transmitted, and the actual bytes. Header FF corresponds to the transmitted bytes being data bytes. Header FE corresponds to transmitted bytes being status bytes.

Q.W. - |Module ID|command|

S.W. - |channel connection status|channel read status|

command - i. connect status
ii. read
iii. configure

Q.W. and S.W. consist of two nibbles. Configure command changes the read status of the channel. The complete protocol is explained by the flowchart in Fig. 3

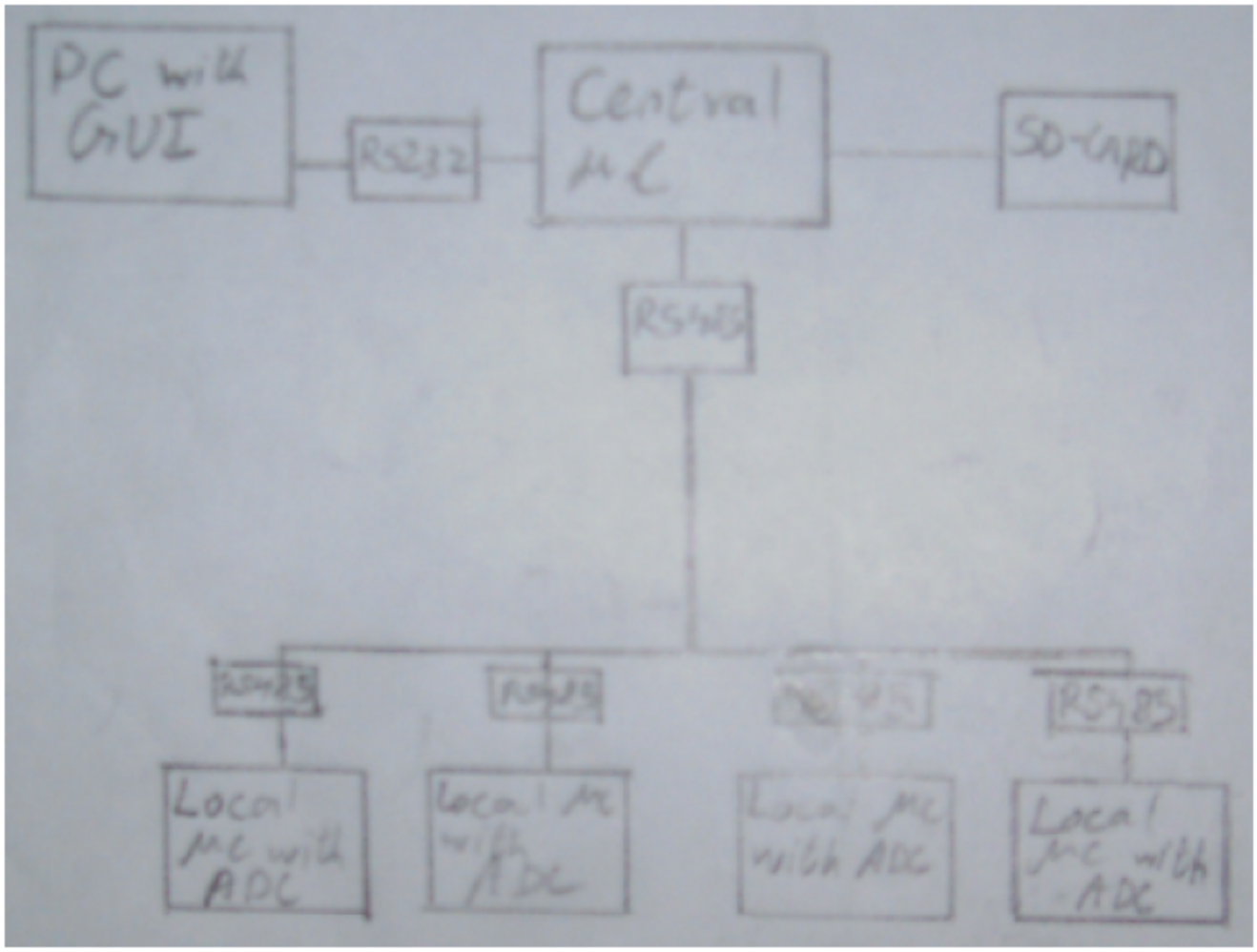
PC with GUI
RS232
Central µC
SD-CARD
RS485
RS485
RS485
RS485
Local µC with ADC
Local µC with ADC
Local µC with ADC
Local µC with ADC

Fig. 1

.

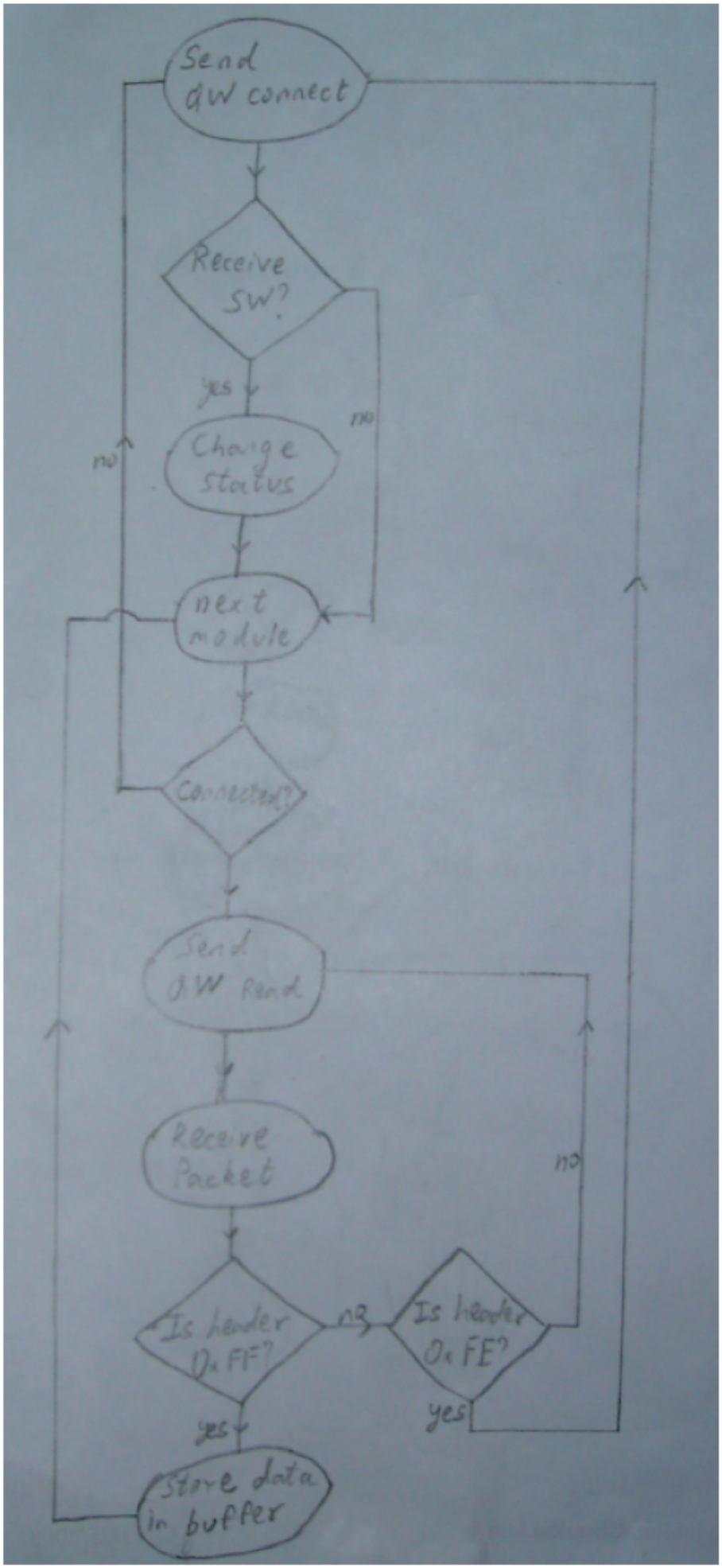
Send
qW connect
Receive
SW?
yes
no
Change
status
no
next
module
Connected?
Send
qW Read
Receive
Packet
no
Is header
0x FF?
no
Is header
0x FE?
yes
yes
Store data
in buffer

Fig. 2

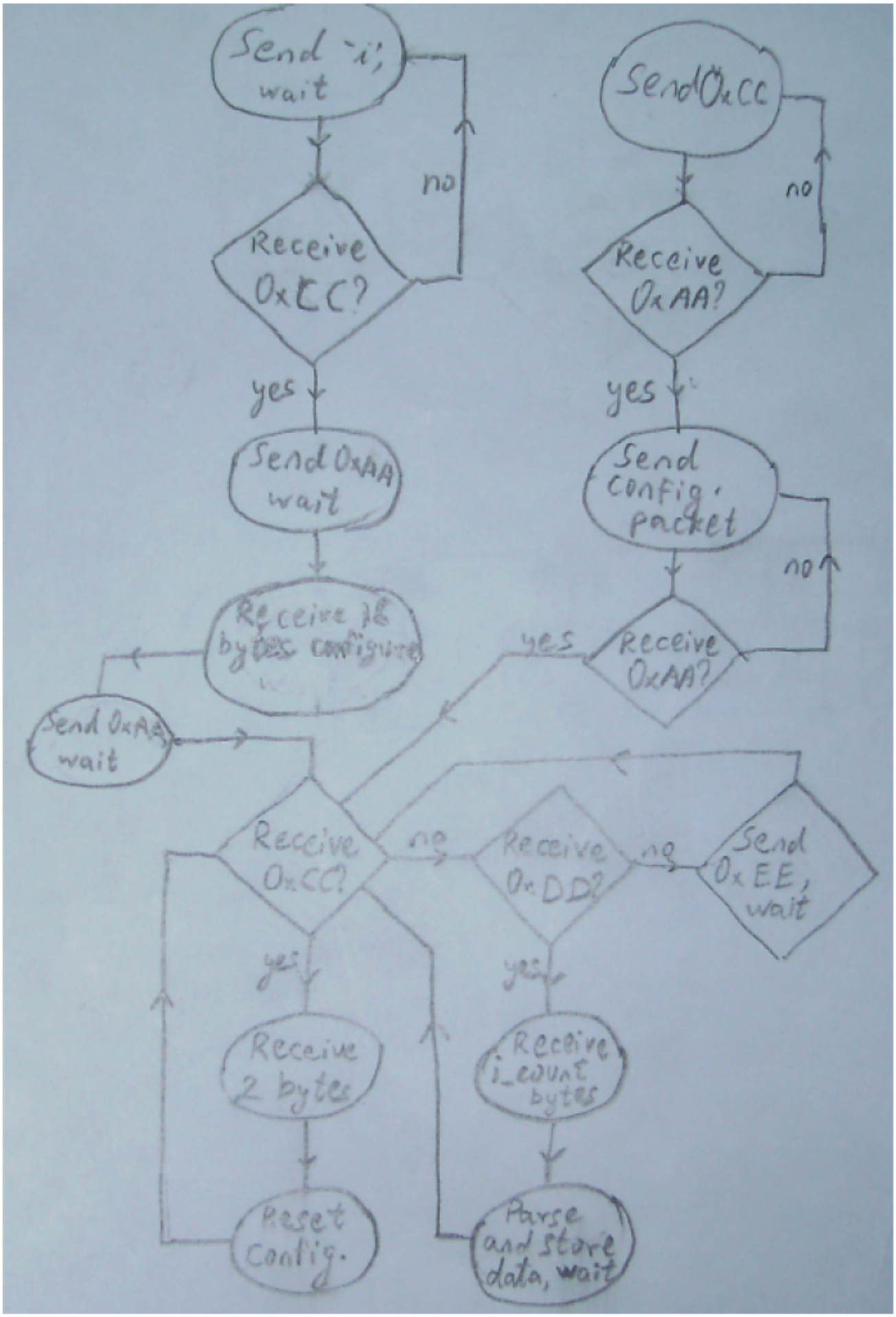
Send 'i', wait
no
Receive 0xEC?
yes
Send 0xAA wait
Receive 16 bytes configure
Send 0xAA wait
Send 0xCC
no
Receive 0xAA?
yes
Send config. packet
no
Receive 0xAA?
yes
Receive 0xCC?
no
Receive 0xDD?
no
Send 0xEE, wait
yes
Receive 2 bytes
Reset config.
yes
Receive i_count bytes
Parse and store data, wait

Fig. 3

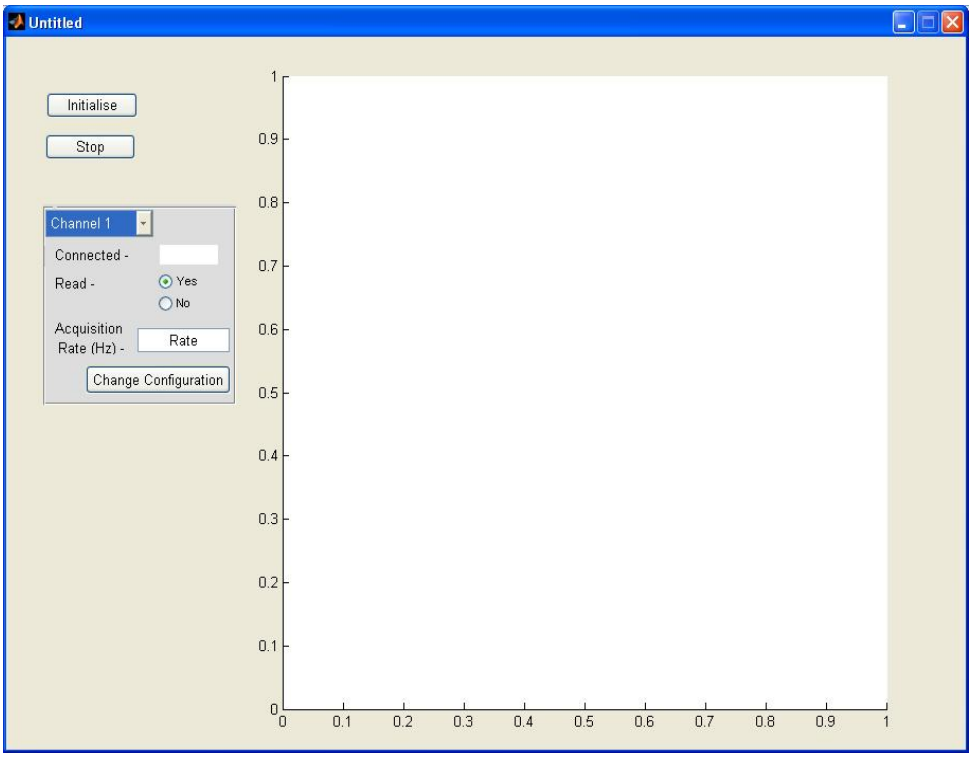

Fig. 4 Screen capture of GUI

Graphical User Interface

The GUI enables the user to monitor all the channels as well as control them from a PC. Upon initialisation, configuration data is stored in the GUI. Incoming data is parsed and stored in separate text files corresponding to different channels. All of this information can be accessed by selecting a channel number from the drop-down box. Configuration of the channel data-rate as well as the read-status of the channel i.e. whether the channel is to be read or not irrespective of whether there is a sensor connected on that channel, can be done from the GUI. The stored channel data is displayed in a graph that is updated in real-time.

Limitations and Improvements Possible

1. The biggest limitation of this system is the lack of complete flexibility with adjusting the data acquisition rates from the channels, as well as an optional adjustment of these rates for optimal utilisation of available bandwidth.
2. GUI too has limitations in that it doesn t include tools for more analysis of the data.
3. Time stamps have not been implemented. They could be implemented given more time.

Codes

We have attached the final codes that we had. They are by no means completely functional, since we lost the working codes in a laptop crash.

<u>Central module to local module -</u>

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

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

//Commands
int constatus = 0x01;
int readstatus = 0x02;
int writestatus = 0x03;
int readdata = 0x04;
int h2;

//Identities
int id1 = 0x10;
int id2 = 0x20;
int id3 = 0x30;
int id4 = 0x40;
int receive[100];
int k = 100;

void USART_Init( unsigned int baud )
{
        UCSRB |= _BV(TXCIE);
        UCSRB |= _BV(RXCIE);
        UCSRB |= _BV(RXEN);
        UCSRB |= _BV(TXEN);
        UBRRH = 0x80;
        UCSRC |= _BV(UCSZ1);
        UCSRC |= _BV(UCSZ0);
        UCSRC |= _BV(URSEL);
        UBRRH = 0x00;
        UBRRL = baud;
}

void InitADC()
{
ADMUX=(1<<REFS0);                       // For Aref=AVcc;
ADCSRA=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); //Rrescalar div factor =128
}

uint16_t ReadADC(uint8_t ch)
{
  //Select ADC Channel ch must be 0-7
  ch=ch&0b00000111;
  ADMUX|=ch;

  //Start Single conversion
  ADCSRA|=(1<<ADSC);

  //Wait for conversion to complete
  while(!(ADCSRA & (1<<ADIF)));
```

```
  //Clear ADIF by writing one to it

  ADCSRA|=(1<<ADIF);

  return(ADC);
}

void USART_Transmit( unsigned int data )
{

        while ( !(UCSRA & (_BV(UDRE))) );          /* Wait for empty transmit buffer */
        {
                UDR = data;                        /* Put data into buffer, sends the data */
   }
}

char USART_Receive( void )
{

        while ( !(UCSRA & (1<<RXC)) );          /* Wait for data to be received */
        {
//              PORTB = 0xff;
                return (UDR);                   /* Get and return received data from buffer */
        }

}

void ledflash( int input)
{
        for (int i = input; i>0; i--)
                {
                        PORTB ^= _BV(PB0);
                        for (unsigned int j=0; j<32000; j++)
                        {
                                for (int k=0; k<10; k++)
                                {}
                        }
                        PORTB ^= _BV(PB0);
                        for (unsigned int j=0; j<32000; j++)
                        {
                                for (int k=0; k<10; k++)
                                {}
                        }
                }
}

void transmit_command(unsigned int id, unsigned int command)
{
```

```
	PORTB |= _BV(PB1);
	unsigned int cmd = id + command;
	USART_Transmit(cmd);
}

ISR(USART_TXC_vect)
{
	PORTB &= ~(_BV(PB1));
}

ISR(USART_RXC_vect)
{
	receive[k] = USART_Receive();
}

ISR(INT0_vect)
{
	h2=1;
}


int main()
{
	DDRB = 0xFF;			//Configure PORTB for output
	DDRA = 0xff;

	MCUCR = 0x02;
	GICR = 0x40;

	PORTB = 0x00;
	int baud = 12;
	USART_Init(baud);

	sei();

	while(k)
	{
		transmit_command(id1, constatus);
		for (int i=0; i<2500; i++)
		{}
		k--;
	}

	while(h2 == 0)
	{}

	PORTB |= _BV(PB0);

	for (int i=0; i<100; i++)
```

```
        {

                PORTA = receive[i] ;
                for (int i=0; i<1000; i++)
                {}

                PORTB |=  (_BV(PB2));
                PORTB &= ~(_BV(PB2));


                for (int i=0; i<1000; i++)
                {}

                PORTB |=  (_BV(PB3));
                PORTB &= ~(_BV(PB3));
        }

/*      transmit_command(id2, constatus);
        for (int i=0; i<250; i++)
        {}

        transmit_command(id3, constatus);
        for (int i=0; i<250; i++)
        {}

        transmit_command(id4, constatus);
        for (int i=0; i<250; i++)
        {}*/

        return 0;
}
```

Central module to SD-Card

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

int h;
int h2=0;
int buffer[100];

void USART_Init( unsigned int baud )
{
//      UCSRB |= _BV(TXCIE);
//      UCSRB |= _BV(RXCIE);
        UCSRB |= _BV(RXEN);
        UCSRB |= _BV(TXEN);
```

```
	UBRRH = 0x80;
	UCSRC |= _BV(UCSZ1);
	UCSRC |= _BV(UCSZ0);
	UCSRC |= _BV(URSEL);
	UBRRH = 0x00;
	UBRRL = baud;
}

void USART_Transmit( unsigned char data )
{

	while ( !(UCSRA & (_BV(UDRE))) );		/* Wait for empty transmit buffer */
	{
		UDR = data;				/* Put data into buffer, sends the data */
   }
}

char USART_Receive( void )
{

	while ( !(UCSRA & (1<<RXC)) );		/* Wait for data to be received */
	{
		return (UDR);				/* Get and return received data from buffer */
	}

}

void ledflash(int input)
{
	for (int i = input; i>0; i--)
		{
			PORTB ^= _BV(PB0);
			for (unsigned int j=0; j<32530; j++)
			{
				for (int k=0; k<10; k++)
				{}
			}
			PORTB ^= _BV(PB0);
			for (unsigned int j=0; j<32530; j++)
			{
				for (int k=0; k<10; k++)
				{}
			}
		}
}


void InitADC()
{
```

```
ADMUX=(1<<REFS0);                     // For Aref=AVcc;
ADCSRA=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); //Rrescalar div factor =128
}

uint16_t ReadADC(uint8_t ch)
{
  //Select ADC Channel ch must be 0-7
  ch=ch&0b00000111;
  ADMUX|=ch;

  //Start Single conversion
  ADCSRA|=(1<<ADSC);

  //Wait for conversion to complete
  while(!(ADCSRA & (1<<ADIF)));

  //Clear ADIF by writing one to it

  ADCSRA|=(1<<ADIF);

  return(ADC);
}

void Wait()
{
  uint8_t i;
  for(i=0;i<200;i++)
  {}
}



void usart_puts (char *s) {
        //  loop until *s != NULL
        while (*s) {
                USART_Transmit(*s);
                s++;
        }
}

ISR(INT0_vect)
{
        h2=1;
}



int main()
{
```

```
	DDRB = 0xFF;			//Configure PORTB for output
	DDRA = 0xff;

	PORTB = 0x00;

	uint16_t adc_result;

	//Initialize ADC
	InitADC();

	MCUCR = 0x02;
	GICR = 0x40;

	sei();


	while(h2 == 0)
	{}

	PORTB |= _BV(PB0);

	for (int i=0; i<100; i++)
	{
    buffer[i] = ReadADC(0);      // Read Analog value from channel-0
    Wait();
	}

	for (int i=0; i<100; i++)
	{

		PORTA = buffer[i] ;
		for (int i=0; i<1000; i++)
		{}

		PORTB |=  (_BV(PB1));
		PORTB &= ~(_BV(PB1));


		for (int i=0; i<1000; i++)
		{}

		PORTB |=  (_BV(PB2));
		PORTB &= ~(_BV(PB2));
	}

	return 0;
}
```

ATTiny2313 code

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

//Commands
int constatus = 0x01;
int readstatus = 0x02;
int writestatus = 0x03;
int readdata = 0x04;

//Identities
int id1 = 0x10;
int id2 = 0x20;
int id3 = 0x30;
int id4 = 0x40;

int h=0;
int h1=0;
int receive;

void USART_Init( unsigned int baud )
{
	UCSRB |= _BV(TXCIE);
	UCSRB |= _BV(RXCIE);
	UCSRB |= _BV(RXEN);
	UCSRB |= _BV(TXEN);
	UCSRC |= _BV(UCSZ1);
	UCSRC |= _BV(UCSZ0);
	UBRRH = 0x00;
	UBRRL = baud;
}

void USART_Transmit( unsigned int data )
{

	while ( !(UCSRA & (_BV(UDRE))) );		/* Wait for empty transmit buffer */
	{
		UDR = data;				/* Put data into buffer, sends the data */
   }
}

char USART_Receive( void )
{

	while ( !(UCSRA & (1<<RXC)) );		/* Wait for data to be received */
	{
//		PORTB = 0xff;
```

```
		return (UDR);			/* Get and return received data from buffer */
	}

}

void ledflash( int input)
{
	for (int i = input; i>0; i--)
		{
			PORTB ^= _BV(PB0);
			for (unsigned int j=0; j<32000; j++)
			{
				for (int k=0; k<10; k++)
				{}
			}
			PORTB ^= _BV(PB0);
			for (unsigned int j=0; j<32000; j++)
			{
				for (int k=0; k<10; k++)
				{}
			}
		}
}

void transmit_command(unsigned int id, unsigned int command)
{
	PORTB |= _BV(PB1);
	unsigned int cmd = id + command;
	USART_Transmit(cmd);
}

ISR(INT0_vect)
{
	PORTD |= _BV(PD5);
	int tran = PINB;
	USART_Transmit(tran);
}

ISR(INT1_vect)
{
	h1 = 1;
}

ISR(BADISR_vect)
{}

int main()
{
	DDRD = 0x60;
```

```
DDRB = 0x06;
PORTB = PORTB & 0xF9;

int baud = 51;
USART_Init(baud);

MCUCR = 0x0A;
GIMSK = 0xC0;


h=0;

while(h==0)
{
	int receive = USART_Receive();

	if (receive == 0x3E)
	{
		h=1;
	}
}

USART_Transmit('F');
USART_Transmit(0x0D);

int filehandle = USART_Receive();

h=0;

while(h==0)
{
	int receive = USART_Receive();

	if (receive == 0x3E)
	{
		h=1;
	}
}

USART_Transmit('O');
USART_Transmit(' ');
USART_Transmit(filehandle);
USART_Transmit(' ');
USART_Transmit('A');
USART_Transmit(' ');
USART_Transmit('/');
USART_Transmit('a');
USART_Transmit('.');
USART_Transmit('l');
```

```
USART_Transmit('o');
USART_Transmit('g');
USART_Transmit(0x0D);

h=0;

while(h==0)
{
	int receive = USART_Receive();

	if (receive == 0x3E)
	{
		h=1;
	}
}

USART_Transmit('W');
USART_Transmit(' ');
USART_Transmit(filehandle);
USART_Transmit(' ');
USART_Transmit('1');
USART_Transmit('0');
USART_Transmit('0');
USART_Transmit(0x0D);

PORTD |= _BV(PD6);
PORTD &= ~(_BV(PD6));

sei();

for (int i=0; i<100; i++)
{
	while(h1==0)
	{}
}

h=0;

while(h==0)
{
	int receive = USART_Receive();

	if (receive == 0x3E)
	{
		h=1;
	}
}

USART_Transmit('C');
```

```
	USART_Transmit(' ');
	USART_Transmit(filehandle);
	USART_Transmit(0x0D);

	h=0;

	while(h==0)
	{
		int receive = USART_Receive();

		if (receive == 0x3E)
		{
			h=1;
		}
	}

	return 0;
}
```

Communication to GUI via USART

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

//Commands
int constatus = 0x01;
int readstatus = 0x02;
int writestatus = 0x03;
int readdata = 0x04;

//Identities
int id1 = 0x10;
int id2 = 0x20;
int id3 = 0x30;
int id4 = 0x40;

int h=0;
int receive;

void USART_Init( unsigned int baud )
{
	UCSRB |= _BV(TXCIE);
	UCSRB |= _BV(RXCIE);
	UCSRB |= _BV(RXEN);
	UCSRB |= _BV(TXEN);
	UBRRH = 0x80;
	UCSRC |= _BV(UCSZ1);
```

```
	UCSRC |= _BV(UCSZ0);
	UCSRC |= _BV(URSEL);
	UBRRH = 0x00;
	UBRRL = baud;
}

void USART_Transmit( unsigned int data )
{

	while ( !(UCSRA & (_BV(UDRE))) );		/* Wait for empty transmit buffer */
	{
		UDR = data;				/* Put data into buffer, sends the data */
   }
}

char USART_Receive( void )
{

	while ( !(UCSRA & (1<<RXC)) );		/* Wait for data to be received */
	{
//		PORTB = 0xff;
		return (UDR);				/* Get and return received data from buffer */
	}

}

void ledflash( int input)
{
	for (int i = input; i>0; i--)
		{
			PORTB ^= _BV(PB0);
			for (unsigned int j=0; j<32000; j++)
			{
				for (int k=0; k<10; k++)
				{}
			}
			PORTB ^= _BV(PB0);
			for (unsigned int j=0; j<32000; j++)
			{
				for (int k=0; k<10; k++)
				{}
			}
		}
}

void transmit_command(unsigned int id, unsigned int command)
{
	PORTB |= _BV(PB1);
	unsigned int cmd = id + command;
```

```
        USART_Transmit(cmd);
}

ISR(USART_TXC_vect)
{
        PORTB &= ~(_BV(PB1));
}

ISR(USART_RXC_vect)
{
        receive = USART_Receive();
        h=1;
        PORTB^=_BV(PB0);
}

void InitADC()
{
ADMUX=(1<<REFS0);                       // For Aref=AVcc;
ADCSRA=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); //Rrescalar div factor =128
}

uint16_t ReadADC(uint8_t ch)
{
  //Select ADC Channel ch must be 0-7
  ch=ch&0b00000111;
  ADMUX|=ch;

  //Start Single conversion
  ADCSRA|=(1<<ADSC);

  //Wait for conversion to complete
  while(!(ADCSRA & (1<<ADIF)));

  //Clear ADIF by writing one to it
  //Note you may be wondering why we have write one to clear it
  //This is standard way of clearing bits in io as said in datasheets.
  //The code writes '1' but it result in setting bit to '0' !!!

  ADCSRA|=(1<<ADIF);

  return(ADC);
}

void Wait()
{
  uint8_t i;
  for(i=0;i<200;i++)
  {}
}
```

```
int main()
{
	DDRB = 0xff;
	PORTB = 0x00;

	int baud = 51;
	USART_Init(baud);

	uint16_t adc_result;
	InitADC();

	int ack=0;
	int data=0x00;
	int header=0xDD;
	int sflag=0;

	sei();

	for (int i=0; i<1000; i++)
	{}

	while(h==0)
	{}


	//initialization
	while(h==1)
	{
		if (receive == 'i')
			{
				USART_Transmit(0xCC);
				h=0; //after config done exit init loop
			}
		else h=1; //redundant
	}
	h=0;


	//sending 18 bytes
	USART_Transmit(0x80);
	USART_Transmit(0x00);
	for(int j=0; j<16; j++)
	{
		USART_Transmit(0x64);
	}
```

```
	while(h==0) //wait till command received
	{}

	//wait for ack
	while(ack==0)
	{
		if(receive == 0xAA)
		{
			ack=1; //exit ack loop
		}
		else ack=0;
	}
	ack=0;


	//data transfer: packet(header, data)
	//sflag indicates packet is ready to be sent

	while(1)
{
		adc_result=ReadADC(0);		// Read Analog value from channel-0
		//packet info
		data=adc_result;
		header=0xDD;
		sflag=1;

		if(sflag==1)  //ready to send packet
		{

			int flag=1;
			while(flag==1)
			{
				h=0;
				USART_Transmit(header);
				while(h==0)
				{}

				if(receive == 0xEE);
				{
					flag=1;
				}
				if(receive == 0xAA)
				{
					h=0;
					USART_Transmit(data);

					while(h==0) //wait till command received
					{}
```

```
                                        while(ack==0) //wait for ack
                                        {
                                                if(receive == 0xAA)
                                                {
                                                        ack=1; //exit ack loop
                                                }
                                                else ack=0;
                                        }
                                        ack=0;
                                        flag=0;
                                }
                        }
                }
                Wait();
        }


        return 0;
}
```

Local Module Code

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

//Commands
int constatus = 0x01;
int readstatus = 0x02;
int writestatus = 0x03;
int readdata = 0x04;

//Identity
int id1 = 0x10;
uint16_t adc_result;

void USART_Init( unsigned int baud )
{
        UCSRB |= _BV(TXCIE);
        UCSRB |= _BV(RXCIE);
        UCSRB |= _BV(RXEN);
        UCSRB |= _BV(TXEN);
        UBRRH = 0x80;
        UCSRC |= _BV(UCSZ1);
        UCSRC |= _BV(UCSZ0);
        UCSRC |= _BV(URSEL);
        UBRRH = 0x00;
        UBRRL = baud;
```

```
}

void InitADC()
{
ADMUX=(1<<REFS0);                     // For Aref=AVcc;
ADCSRA=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); //Rrescalar div factor =128
}

uint16_t ReadADC(uint8_t ch)
{
  //Select ADC Channel ch must be 0-7
  ch=ch&0b00000111;
  ADMUX|=ch;

  //Start Single conversion
  ADCSRA|=(1<<ADSC);

  //Wait for conversion to complete
  while(!(ADCSRA & (1<<ADIF)));

  //Clear ADIF by writing one to it

  ADCSRA|=(1<<ADIF);

  return(ADC);
}

void USART_Transmit( unsigned int data )
{

        while ( !(UCSRA & (_BV(UDRE))) );          /* Wait for empty transmit buffer */
        {
                UDR = data;                          /* Put data into buffer, sends the data */
   }
}

char USART_Receive( void )
{

        while ( !(UCSRA & (1<<RXC)) );           /* Wait for data to be received */
        {
//              PORTB = 0xff;
                return (UDR);                        /* Get and return received data from buffer */
        }

}

void ledflash( int input)
{
```

```
	for (int i = input; i>0; i--)
		{
			PORTB ^= _BV(PB0);
			for (unsigned int j=0; j<32000; j++)
			{
				for (int k=0; k<10; k++)
				{}
			}
			PORTB ^= _BV(PB0);
			for (unsigned int j=0; j<32000; j++)
			{
				for (int k=0; k<10; k++)
				{}
			}
		}
}

ISR(USART_TXC_vect)
{
	PORTD &= ~(_BV(PD4));
}

ISR(USART_RXC_vect)
{
	int receive = USART_Receive();

	if (receive == constatus)
	{
		PORTD |= _BV(PD4);
		USART_Transmit(0x0a);
		ledflash(0x01);
	}

	else if (receive == readstatus)
	{
		PORTD |= _BV(PD4);
		USART_Transmit(0x02);
		ledflash(0x01);
	}

	else if (receive == writestatus)
	{
		PORTD |= _BV(PD4);
		USART_Transmit(0x03);
		ledflash(0x01);
	}

	else if (receive == readdata)
	{
```

```
                PORTD |= _BV(PD4);
                USART_Transmit(0x04);
                ledflash(0x01);
        }
}

int main()
{
        DDRB = 0xff;
        DDRD = 0x10;
        PORTB = 0x00;
        PORTD = 0x00;

        constatus = constatus + id1;
        readstatus = readstatus + id1;
        writestatus = writestatus + id1;
        readdata = readdata + id1;

        int baud = 12;
        USART_Init(baud);

        //Initialize ADC
        InitADC();

        sei();

        while(1)
        {}

        return 0;
}
```

Combined Matlab file for GUI

```
function varargout = bigger_gui_export(varargin)
% BIGGER_GUI_EXPORT M-file for bigger_gui_export.fig
%      BIGGER_GUI_EXPORT, by itself, creates a new BIGGER_GUI_EXPORT or raises the existing
%      singleton*.
%
%      H = BIGGER_GUI_EXPORT returns the handle to a new BIGGER_GUI_EXPORT or the handle
to
%      the existing singleton*.
%
%      BIGGER_GUI_EXPORT('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in BIGGER_GUI_EXPORT.M with the given input arguments.
%
%      BIGGER_GUI_EXPORT('Property','Value',...) creates a new BIGGER_GUI_EXPORT or raises
the
```

```
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before bigger_gui_export_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to bigger_gui_export_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help bigger_gui_export

% Last Modified by GUIDE v2.5 01-Dec-2008 16:55:19

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @bigger_gui_export_OpeningFcn, ...
                   'gui_OutputFcn',  @bigger_gui_export_OutputFcn, ...
                   'gui_LayoutFcn',  @bigger_gui_export_LayoutFcn, ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
end

% --- Executes just before bigger_gui_export is made visible.
function bigger_gui_export_OpeningFcn(hObject, eventdata, handles, varargin)
%handles.channel_rate = [100;200;300;400];
%handles.channel_status = [1;0;1;0];
%handles.input = [
%handles.plot_status = [0,0,0,0];
%handles.channel1_values = [1;5;34;768;54;768;33;478;23;86;83;456;342;44;542;78;12;678;545;45];
%handles.channel2_values =
[100;55;4;6768;5654;8;33;40;234;3486;83;456;342;44;542;78;12;678;545;45];
%handles.channel3_values = [1;5;34;768;54;768;33;478;23;86;83;456;342;44;542;78;12;678;545;45];
%handles.channel4_values =
[100;55;4;6768;5654;8;33;40;234;3486;83;456;342;44;542;78;12;678;545;45];
%handles.time = [1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19;20];
%handles.zeros = zeros(length(handles.time));
% This function has no output args, see OutputFcn.
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to bigger_gui_export (see VARARGIN)

%%function i_interrupt(obj, event)
%%        handles.channel_rate = fread(obj, 4)
%%        %guidata(gcf, handles);
%%end

% Choose default command line output for bigger_gui_export
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
end
% UIWAIT makes bigger_gui_export wait for user response (see UIRESUME)
% uiwait(handles.figure1);

%BytesAvailableFcn callback for serial I/O object
%function i_interrupt(obj, event)
%        handles.channel_rate = fread(obj, 4)
%        guidata(gcbo, handles);
%%
%end

% --- Outputs from this function are returned to the command line.
function varargout = bigger_gui_export_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
end

% --- Executes on button press in initialise.
function initialise_Callback(hObject, eventdata, handles)
app_struct.serial_io = serial ('COM7', 'BaudRate', 9600, 'BytesAvailableFcn', @i_interrupt,
'BytesAvailableFcnCount', 19, 'BytesAvailableFcnMode', 'byte')

%hob = hObject;
%modhandles = handles
app_struct.init = 1;
app_struct.icount = 19;
app_struct.flag = 0;
app_struct.header = [];
app_struct.return = [170];
```

```
app_struct.channel_view = 1;
%app_struct.axes_handle = handles.plot_window
app_struct.channel_rate = [100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100];
app_struct.channel_connect_status = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1];
app_struct.channel_read_status = [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1];
app_struct.data_handles(1, :) = [fopen('data1.txt', 'a'), fopen('data1.txt', 'r')];
app_struct.data_handles(2, :) = [fopen('data2.txt', 'a'), fopen('data2.txt', 'r')];
app_struct.data_handles(3, :) = [fopen('data3.txt', 'a'), fopen('data3.txt', 'r')];
app_struct.data_handles(4, :) = [fopen('data4.txt', 'a'), fopen('data4.txt', 'r')];
app_struct.data_handles(5, :) = [fopen('data5.txt', 'a'), fopen('data5.txt', 'r')];
app_struct.data_handles(6, :) = [fopen('data6.txt', 'a'), fopen('data6.txt', 'r')];
app_struct.data_handles(7, :) = [fopen('data7.txt', 'a'), fopen('data7.txt', 'r')];
app_struct.data_handles(8, :) = [fopen('data8.txt', 'a'), fopen('data8.txt', 'r')];
app_struct.data_handles(9, :) = [fopen('data9.txt', 'a'), fopen('data9.txt', 'r')];
app_struct.data_handles(10, :) = [fopen('data10.txt', 'a'), fopen('data10.txt', 'r')];
app_struct.data_handles(11, :) = [fopen('data11.txt', 'a'), fopen('data11.txt', 'r')];
app_struct.data_handles(12, :) = [fopen('data12.txt', 'a'), fopen('data12.txt', 'r')];
app_struct.data_handles(13, :) = [fopen('data13.txt', 'a'), fopen('data13.txt', 'r')];
app_struct.data_handles(14, :) = [fopen('data14.txt', 'a'), fopen('data14.txt', 'r')];
app_struct.data_handles(15, :) = [fopen('data15.txt', 'a'), fopen('data15.txt', 'r')];
app_struct.data_handles(16, :) = [fopen('data16.txt', 'a'), fopen('data16.txt', 'r')]
setappdata (gcbf, 'app_struct', app_struct)

fopen(app_struct.serial_io)
init = 105
fwrite(app_struct.serial_io, init)
h = gcbf
axes(handles.plot_window)
plot_handle = gca
%file_append = fopen('data.txt', 'a')
%file_read = fopen('data.txt', 'r')

function i_interrupt(obj, event)
        %modhandles.channel_rate = fread(obj, 4)
        %handles = modhandles
        app_struct = getappdata(h, 'app_struct')

        if (app_struct.init)

            a = fread(obj, [19, 1])

            c_connect_status_array(1, 1:8) = dec2bin(a(2));
            c_connect_status_array(1, 9:16) = dec2bin(a(3));
            for i = 1:16
                app_struct.channel_connect_status(1, i) = str2num(c_connect_status_array(i));
            end

            for i = 4:19
```

```
        app_struct.channel_rate(1, i) = a(i, 1);
    end

    app_struct.icount = length (find (app_struct.channel_connect_status));
    app_struct.init = 0;
    fclose(app_struct.serial_io);
    set(app_struct.serial_io,'BytesAvailableFcnCount', 1);
    fopen(app_struct.serial_io);

else

    if (~app_struct.flag)

        a = fread(obj, [1,1]);
        app_struct.header = a(1);
        if (app_struct.header==204)
            app_struct.icount = 2;
        elseif (app_struct.header==221)
            if (length (find (app_struct.channel_connect_status)) == 0);
                app_struct.icount = 1;
            else
                app_struct.icount = length (find (app_struct.channel_connect_status));
            end
        else
            app_struct.return = [238];
        end
        fclose(app_struct.serial_io);
        set(app_struct.serial_io,'BytesAvailableFcnCount', app_struct.icount);
        fopen(app_struct.serial_io);
        if ((length (find (app_struct.channel_connect_status)) == 0) || (app_struct.return(1)==238) )
            app_struct.flag = 0;
        else
            app_struct.flag = 1;
        end

    else

        a = fread(obj, [app_struct.icount, 1]);
        if (app_struct.header==204)
            c_connect_status_array(1, 1:8) = dec2bin(a(1));
            c_connect_status_array(1, 9:16) = dec2bin(a(2));
            for i = 1:16
                app_struct.channel_connect_status(1, i) = str2num(c_connect_status_array(i));
            end
        elseif (app_struct.header==221)
                %plot_read = app_struct.data_handles(app_struct.channel_view, 2)
                %fclose(app_struct.data_handles(app_struct.channel_view, 2))
                %app_struct.data_handles(app_struct.channel_view, 2) = fopen('data1.txt', 'r')
                j = 1;
```

```
                for i = 1:app_struct.icount
                    if (app_struct.channel_connect_status(i))
                        fwrite(app_struct.data_handles(i, 1), a(j));
                        j = j+1;
                    end
                end
                %handles
                %modhandles = guihandles(gcbf)
                %%Plot updates%%
                %modhandles = gcf
                %app_struct.axes_handle = modhandles.plot_window
                %gca
                %axes(gca)
                plot_data = fread(app_struct.data_handles(app_struct.channel_view, 2))
                fseek (app_struct.data_handles(app_struct.channel_view, 2), 0, 'bof')
                plot(plot_handle, plot_data);
                %guidata(gcf, modhandles)
                %%
            end

            fclose(app_struct.serial_io);
            set(app_struct.serial_io,'BytesAvailableFcnCount', 1);
            fopen(app_struct.serial_io);
            app_struct.flag = 0;

        end

    end

    %setappdata (gcbf, 'input', a)
    %setappdata (gcbf, 'file', file)
    b = [app_struct.return]
    app_struct.return = [170];
    fwrite(obj, b)

%       if (header==204)
%           c_array(1, 1:8) = dec2bin(a(2))
%           c_array(1, 9:16) = dec2bin(a(3))
%           for i = 1:16
%               app_struct.channel_status(1, i) = str2num(c_array(i))
%           end
%           icount = 17;
%       end
%
%       if (a(1)==221)
%           j = 2;
%               for i = 1:16
%                   if (app_struct.channel_status(i))
%                   fwrite(app_struct.data_handles(i, 1), a(j));
```

```
%                 j = j+1;
%                 end
%             end
%         end

%         set (app_struct.serial_io, 'BytesAvailableFcnCount', icount)
        setappdata (h, 'app_struct', app_struct)

        %fwrite(file_append, a)
        %c = fread(file_read)
%%          guidata(gcbo, handles);
%%
end

handles = modhandles;
guidata(hObject, handles)

% hObject    handle to initialise (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end

%function plot_update()
%
%h = gcbf;
%app_struct = getappdata (h, 'app_struct')
%plot_y = fread(app_struct.data_handles(app_struct.channel_view, 2));
%
%end

% --- Executes during object creation, after setting all properties.
function channel_select_CreateFcn(hObject, eventdata, handles)
% hObject    handle to channel_select (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
end

% --- Executes on selection change in channel_select.
function channel_select_Callback(hObject, eventdata, handles)
h = gcbf
app_struct = getappdata (h, 'app_struct')
```

```
val = get(hObject,'Value');
        switch val
            case 1
                if (app_struct.channel_connect_status(1))
                    set(handles.channel_connect_status, 'string', 'Yes');
                else
                    set(handles.channel_connect_status, 'string', 'No');
                end
                if (app_struct.channel_read_status(1))
                    set(handles.channel_read_status_yes, 'Value', 1.0);
                    set(handles.channel_read_status_no, 'Value', 0.0);
                else
                    set(handles.channel_read_status_yes, 'Value', 0.0);
                    set(handles.channel_read_status_no, 'Value', 1.0);
                end
                set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(1)));
                app_struct.channel_view = 1;
                setappdata (h, 'app_struct', app_struct)
                guidata(hObject,handles)
            case 2
                if (app_struct.channel_connect_status(2))
                    set(handles.channel_connect_status, 'string', 'Yes');
                else
                    set(handles.channel_connect_status, 'string', 'No');
                end
                if (app_struct.channel_read_status(2))
                    set(handles.channel_read_status_yes, 'Value', 1.0);
                    set(handles.channel_read_status_no, 'Value', 0.0);
                else
                    set(handles.channel_read_status_yes, 'Value', 0.0);
                    set(handles.channel_read_status_no, 'Value', 1.0);
                end
                set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(2)));
                app_struct.channel_view = 2;
                setappdata (h, 'app_struct', app_struct)
                guidata(hObject,handles)
            case 3
                if (app_struct.channel_connect_status(3))
                    set(handles.channel_connect_status, 'string', 'Yes');
                else
                    set(handles.channel_connect_status, 'string', 'No');
                end
                if (app_struct.channel_read_status(3))
                    set(handles.channel_read_status_yes, 'Value', 1.0);
                    set(handles.channel_read_status_no, 'Value', 0.0);
                else
                    set(handles.channel_read_status_yes, 'Value', 0.0);
                    set(handles.channel_read_status_no, 'Value', 1.0);
                end
```

```
    set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(3)));
    app_struct.channel_view = 3;
    setappdata (h, 'app_struct', app_struct)
    guidata(hObject,handles)
case 4
    if (app_struct.channel_connect_status(4))
       set(handles.channel_connect_status, 'string', 'Yes');
    else
       set(handles.channel_connect_status, 'string', 'No');
    end
    if (app_struct.channel_read_status(4))
       set(handles.channel_read_status_yes, 'Value', 1.0);
       set(handles.channel_read_status_no, 'Value', 0.0);
    else
       set(handles.channel_read_status_yes, 'Value', 0.0);
       set(handles.channel_read_status_no, 'Value', 1.0);
    end
    set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(4)));
    app_struct.channel_view = 4;
    setappdata (h, 'app_struct', app_struct)
    guidata(hObject,handles)
case 5
    if (app_struct.channel_connect_status(5))
       set(handles.channel_connect_status, 'string', 'Yes');
    else
       set(handles.channel_connect_status, 'string', 'No');
    end
    if (app_struct.channel_read_status(5))
       set(handles.channel_read_status_yes, 'Value', 1.0);
       set(handles.channel_read_status_no, 'Value', 0.0);
    else
       set(handles.channel_read_status_yes, 'Value', 0.0);
       set(handles.channel_read_status_no, 'Value', 1.0);
    end
    set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(5)));
    app_struct.channel_view = 5;
    setappdata (h, 'app_struct', app_struct)
    guidata(hObject,handles)
case 6
    if (app_struct.channel_connect_status(6))
       set(handles.channel_connect_status, 'string', 'Yes');
    else
       set(handles.channel_connect_status, 'string', 'No');
    end
    if (app_struct.channel_read_status(6))
       set(handles.channel_read_status_yes, 'Value', 1.0);
       set(handles.channel_read_status_no, 'Value', 0.0);
    else
       set(handles.channel_read_status_yes, 'Value', 0.0);
```

```
        set(handles.channel_read_status_no, 'Value', 1.0);
    end
    set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(6)));
    app_struct.channel_view = 6;
    setappdata (h, 'app_struct', app_struct)
    guidata(hObject,handles)
case 7
    if (app_struct.channel_connect_status(7))
        set(handles.channel_connect_status, 'string', 'Yes');
    else
        set(handles.channel_connect_status, 'string', 'No');
    end
    if (app_struct.channel_read_status(7))
        set(handles.channel_read_status_yes, 'Value', 1.0);
        set(handles.channel_read_status_no, 'Value', 0.0);
    else
        set(handles.channel_read_status_yes, 'Value', 0.0);
        set(handles.channel_read_status_no, 'Value', 1.0);
    end
    set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(7)));
    app_struct.channel_view = 7;
    setappdata (h, 'app_struct', app_struct)
    guidata(hObject,handles)
case 8
    if (app_struct.channel_connect_status(8))
        set(handles.channel_connect_status, 'string', 'Yes');
    else
        set(handles.channel_connect_status, 'string', 'No');
    end
    if (app_struct.channel_read_status(8))
        set(handles.channel_read_status_yes, 'Value', 1.0);
        set(handles.channel_read_status_no, 'Value', 0.0);
    else
        set(handles.channel_read_status_yes, 'Value', 0.0);
        set(handles.channel_read_status_no, 'Value', 1.0);
    end
    set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(8)));
    app_struct.channel_view = 8;
    setappdata (h, 'app_struct', app_struct)
    guidata(hObject,handles)
case 9
    if (app_struct.channel_connect_status(9))
        set(handles.channel_connect_status, 'string', 'Yes');
    else
        set(handles.channel_connect_status, 'string', 'No');
    end
    if (app_struct.channel_read_status(9))
        set(handles.channel_read_status_yes, 'Value', 1.0);
        set(handles.channel_read_status_no, 'Value', 0.0);
```

```
    else
        set(handles.channel_read_status_yes, 'Value', 0.0);
        set(handles.channel_read_status_no, 'Value', 1.0);
    end
    set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(9)));
    app_struct.channel_view = 9;
    setappdata (h, 'app_struct', app_struct)
    guidata(hObject,handles)
case 10
    if (app_struct.channel_connect_status(10))
        set(handles.channel_connect_status, 'string', 'Yes');
    else
        set(handles.channel_connect_status, 'string', 'No');
    end
    if (app_struct.channel_read_status(10))
        set(handles.channel_read_status_yes, 'Value', 1.0);
        set(handles.channel_read_status_no, 'Value', 0.0);
    else
        set(handles.channel_read_status_yes, 'Value', 0.0);
        set(handles.channel_read_status_no, 'Value', 1.0);
    end
    set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(10)));
    app_struct.channel_view = 10;
    setappdata (h, 'app_struct', app_struct)
    guidata(hObject,handles)
case 11
    if (app_struct.channel_connect_status(11))
        set(handles.channel_connect_status, 'string', 'Yes');
    else
        set(handles.channel_connect_status, 'string', 'No');
    end
    if (app_struct.channel_read_status(11))
        set(handles.channel_read_status_yes, 'Value', 1.0);
        set(handles.channel_read_status_no, 'Value', 0.0);
    else
        set(handles.channel_read_status_yes, 'Value', 0.0);
        set(handles.channel_read_status_no, 'Value', 1.0);
    end
    set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(11)));
    app_struct.channel_view = 11;
    setappdata (h, 'app_struct', app_struct)
    guidata(hObject,handles)
case 12
    if (app_struct.channel_connect_status(12))
        set(handles.channel_connect_status, 'string', 'Yes');
    else
        set(handles.channel_connect_status, 'string', 'No');
    end
    if (app_struct.channel_read_status(12))
```

```
        set(handles.channel_read_status_yes, 'Value', 1.0);
        set(handles.channel_read_status_no, 'Value', 0.0);
    else
        set(handles.channel_read_status_yes, 'Value', 0.0);
        set(handles.channel_read_status_no, 'Value', 1.0);
    end
    set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(12)));
    app_struct.channel_view = 12;
    setappdata (h, 'app_struct', app_struct)
    guidata(hObject,handles)
case 13
    if (app_struct.channel_connect_status(13))
        set(handles.channel_connect_status, 'string', 'Yes');
    else
        set(handles.channel_connect_status, 'string', 'No');
    end
    if (app_struct.channel_read_status(13))
        set(handles.channel_read_status_yes, 'Value', 1.0);
        set(handles.channel_read_status_no, 'Value', 0.0);
    else
        set(handles.channel_read_status_yes, 'Value', 0.0);
        set(handles.channel_read_status_no, 'Value', 1.0);
    end
    set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(13)));
    app_struct.channel_view = 13;
    setappdata (h, 'app_struct', app_struct)
    guidata(hObject,handles)
case 14
    if (app_struct.channel_connect_status(14))
        set(handles.channel_connect_status, 'string', 'Yes');
    else
        set(handles.channel_connect_status, 'string', 'No');
    end
    if (app_struct.channel_read_status(14))
        set(handles.channel_read_status_yes, 'Value', 1.0);
        set(handles.channel_read_status_no, 'Value', 0.0);
    else
        set(handles.channel_read_status_yes, 'Value', 0.0);
        set(handles.channel_read_status_no, 'Value', 1.0);
    end
    set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(14)));
    app_struct.channel_view = 14;
    setappdata (h, 'app_struct', app_struct)
    guidata(hObject,handles)
case 15
    if (app_struct.channel_connect_status(15))
        set(handles.channel_connect_status, 'string', 'Yes');
    else
        set(handles.channel_connect_status, 'string', 'No');
```

```
            end
            if (app_struct.channel_read_status(15))
               set(handles.channel_read_status_yes, 'Value', 1.0);
               set(handles.channel_read_status_no, 'Value', 0.0);
            else
               set(handles.channel_read_status_yes, 'Value', 0.0);
               set(handles.channel_read_status_no, 'Value', 1.0);
            end
            set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(15)));
            app_struct.channel_view = 15;
            setappdata (h, 'app_struct', app_struct)
            guidata(hObject,handles)
         case 16
            if (app_struct.channel_connect_status(16))
               set(handles.channel_connect_status, 'string', 'Yes');
            else
               set(handles.channel_connect_status, 'string', 'No');
            end
            if (app_struct.channel_read_status(16))
               set(handles.channel_read_status_yes, 'Value', 1.0);
               set(handles.channel_read_status_no, 'Value', 0.0);
            else
               set(handles.channel_read_status_yes, 'Value', 0.0);
               set(handles.channel_read_status_no, 'Value', 1.0);
            end
            set(handles.channel_acquisition_rate, 'String', num2str(app_struct.channel_rate(16)));
            app_struct.channel_view = 16;
            setappdata (h, 'app_struct', app_struct)
            guidata(hObject,handles)

      end

setappdata (h, 'app_struct', app_struct)
% hObject    handle to channel_select (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns channel_select contents as cell array
%        contents{get(hObject,'Value')} returns selected item from channel_select
end

% --- Executes on button press in change_config.
function change_config_Callback(hObject, eventdata, handles)
h = gcbf
app_struct = getappdata (h, 'app_struct')
rate = str2num(get(handles.channel_acquisition_rate, 'String'))
channel = get(handles.channel_select, 'Value')
read = get(handles.channel_read_status_yes, 'Value')
out = [204, 16*(channel-1)+app_struct.channel_read_status(1, channel), rate]
```

```
fwrite(app_struct.serial_io, out)
app_struct.channel_rate(1, channel) = rate;
app_struct.channel_read_status(1, channel) = read;
guidata(hObject, handles)
setappdata (h, 'app_struct', app_struct)
% hObject    handle to change_config (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end

% --- Executes on button press in refresh_plot.
function refresh_plot_Callback(hObject, eventdata, handles)
 if (get(handles.checkbox_channel1,'Value') == get(hObject,'Max'))
    handles.plot_status(1,1) = 1
 else
    handles.plot_status(1,1) = 0
 end
 if (get(handles.checkbox_channel2,'Value') == get(hObject,'Max'))
    handles.plot_status(1,2) = 1
 else
    handles.plot_status(1,2) = 0
 end
 if (get(handles.checkbox_channel3,'Value') == get(hObject,'Max'))
    handles.plot_status(1,3) = 1
 else
    handles.plot_status(1,3) = 0
 end
 if (get(handles.checkbox_channel4,'Value') == get(hObject,'Max'))
    handles.plot_status(1,4) = 1
 else
    handles.plot_status(1,4) = 0
 end
 handles.plot_on = find(handles.plot_status);
 axes(handles.plot_window)
 if (handles.plot_status(1,1))
    plot(handles.input(:,1), 'r')
    hold on;
 else
    plot(handles.time, handles.zeros)
    hold on;
 end
 if (handles.plot_status(1,2))
    plot(handles.time, handles.channel2_values, 'g')
    hold on;
 else
    plot(handles.time, handles.zeros)
    hold on;
 end
 if (handles.plot_status(1,3))
```

```
    plot(handles.time, handles.channel3_values, 'b')
    hold on;
 else
    plot(handles.time, handles.zeros)
    hold on;
 end
 if (handles.plot_status(1,4))
    plot(handles.time, handles.channel4_values, 'k')
    hold off;
 else
    plot(handles.time, handles.zeros)
    hold off;
 end
% hObject    handle to refresh_plot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end

% --- Executes on button press in stop.
function stop_Callback(hObject, eventdata, handles)

h = gcbf
app_struct = getappdata (h, 'app_struct')

fclose(app_struct.serial_io)

for i = 1:16
   for j = 1:2
      fclose(app_struct.data_handles(i,j));
   end
end

setappdata (gcbf, 'app_struct', app_struct)

guidata(hObject, handles)
% hObject    handle to stop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end



% --- Creates and returns a handle to the GUI figure.
function h1 = bigger_gui_export_LayoutFcn(policy)
% policy - create a new figure or use a singleton. 'new' or 'reuse'.

persistent hsingleton;
if strcmpi(policy, 'reuse') & ishandle(hsingleton)
   h1 = hsingleton;
```

```
    return;
end
load bigger_gui_export.mat


appdata = [];
appdata.GUIDEOptions = struct(...
    'active_h', [], ...
    'taginfo', struct(...
    'figure', 2, ...
    'pushbutton', 6, ...
    'frame', 6, ...
    'popupmenu', 3, ...
    'text', 6, ...
    'edit', 2, ...
    'axes', 2, ...
    'checkbox', 5, ...
    'radiobutton', 3, ...
    'uipanel', 4), ...
    'override', 0, ...
    'release', 13, ...
    'resize', 'none', ...
    'accessibility', 'callback', ...
    'mfile', 1, ...
    'callbacks', 1, ...
    'singleton', 1, ...
    'syscolorfig', 1, ...
    'lastSavedFile', 'C:\Documents and Settings\user\Desktop\matlab_d11\bigger_gui_export.m', ...
    'blocking', 0, ...
    'lastFilename', 'C:\Documents and Settings\d11\Desktop\matlab_d11\bigger_gui.fig');
appdata.lastValidTag = 'figure1';
appdata.GUIDELayoutEditor = [];

h1 = figure(...
'Units','characters',...
'PaperUnits',get(0,'defaultfigurePaperUnits'),...
'Color',[0.925490196078431 0.913725490196078 0.847058823529412],...
'Colormap',[0 0 0.5625;0 0 0.625;0 0 0.6875;0 0 0.75;0 0 0.8125;0 0 0.875;0 0 0.9375;0 0 1;0 0.0625
1;0 0.125 1;0 0.1875 1;0 0.25 1;0 0.3125 1;0 0.375 1;0 0.4375 1;0 0.5 1;0 0.5625 1;0 0.625 1;0 0.6875
1;0 0.75 1;0 0.8125 1;0 0.875 1;0 0.9375 1;0 1 1;0.0625 1 1;0.125 1 0.9375;0.1875 1 0.875;0.25 1
0.8125;0.3125 1 0.75;0.375 1 0.6875;0.4375 1 0.625;0.5 1 0.5625;0.5625 1 0.5;0.625 1 0.4375;0.6875
1 0.375;0.75 1 0.3125;0.8125 1 0.25;0.875 1 0.1875;0.9375 1 0.125;1 1 0.0625;1 1 0;1 0.9375 0;1
0.875 0;1 0.8125 0;1 0.75 0;1 0.6875 0;1 0.625 0;1 0.5625 0;1 0.5 0;1 0.4375 0;1 0.375 0;1 0.3125 0;1
0.25 0;1 0.1875 0;1 0.125 0;1 0.0625 0;1 0 0;0.9375 0 0;0.875 0 0;0.8125 0 0;0.75 0 0;0.6875 0 0;0.625
0 0;0.5625 0 0],...
'IntegerHandle','off',...
'InvertHardcopy',get(0,'defaultfigureInvertHardcopy'),...
'MenuBar','none',...
'Name','Untitled',...
```

```
'NumberTitle','off',...
'PaperPosition',get(0,'defaultfigurePaperPosition'),...
'PaperSize',[20.98404194812 29.67743169791],...
'PaperType',get(0,'defaultfigurePaperType'),...
'Position',[103.8 11.5384615384615 180.8 49.9230769230769],...
'Renderer',get(0,'defaultfigureRenderer'),...
'RendererMode','manual',...
'Resize','off',...
'ButtonDownFcn','bigger_gui_export("figure1_ButtonDownFcn",gcbo,[],guidata(gcbo))',...
'HandleVisibility','callback',...
'Tag','figure1',...
'UserData',[],...
'Behavior',get(0,'defaultfigureBehavior'),...
'Visible','on',...
'CreateFcn', {@local_CreateFcn, '', appdata} );

appdata = [];
appdata.lastValidTag = 'initialise';

h2 = uicontrol(...
'Parent',h1,...
'Units','normalized',...
'Callback','bigger_gui_export("initialise_Callback",gcbo,[],guidata(gcbo))',...
'FontSize',10,...
'ListboxTop',0,...
'Position',[0.042168991733396 0.885886177888811 0.0942360475754803 0.0355987055016181],...
'String','Initialise',...
'Tag','initialise',...
'Behavior',get(0,'defaultuicontrolBehavior'),...
'CreateFcn', {@local_CreateFcn, '', appdata} );

appdata = [];
appdata.lastValidTag = 'change_config';

h3 = uicontrol(...
'Parent',h1,...
'Units','normalized',...
'Callback','bigger_gui_export("change_config_Callback",gcbo,[],guidata(gcbo))',...
'FontSize',10,...
'ListboxTop',0,...
'Position',[0.0829646017699115 0.499229583975347 0.151548672566372 0.0400616332819723],...
'String','Change Configuration',...
'CreateFcn', {@local_CreateFcn, 'bigger_gui_export("change_config_CreateFcn",gcbo,
[],guidata(gcbo))', appdata} ,...
'Tag','change_config',...
'Behavior',get(0,'defaultuicontrolBehavior'));

appdata = [];
appdata.lastValidTag = 'channel_connect_status';
```

```
h4 = uicontrol(...
'Parent',h1,...
'Units','normalized',...
'BackgroundColor',[1 1 1],...
'FontSize',10,...
'ListboxTop',0,...
'Position',[0.160398230088496 0.681047765793529 0.0608407079646018 0.0277349768875193],...
'String',' ',...
'Style','text',...
'Tag','channel_connect_status',...
'Behavior',get(0,'defaultuicontrolBehavior'),...
'CreateFcn', {@local_CreateFcn, '', appdata} );

appdata = [];
appdata.lastValidTag = 'channel_acquisition_rate_label';

h5 = uicontrol(...
'Parent',h1,...
'Units','normalized',...
'BackgroundColor',[0.868 0.868 0.868],...
'FontSize',10,...
'ListboxTop',0,...
'Position',[0.0439988178999101 0.553794485915599 0.0878316559926807 0.0501618122977346],...
'String',{  'Acquisition '; 'Rate (Hz) -' },...
'Style','text',...
'Tag','channel_acquisition_rate_label',...
'Behavior',get(0,'defaultuicontrolBehavior'),...
'CreateFcn', {@local_CreateFcn, '', appdata} );

appdata = [];
appdata.lastValidTag = 'channel_acquisition_rate';

h6 = uicontrol(...
'Parent',h1,...
'Units','normalized',...
'BackgroundColor',[1 1 1],...
'Callback','bigger_gui_export("channel_acquisition_rate_Callback",gcbo,[],guidata(gcbo))',...
'FontSize',10,...
'ListboxTop',0,...
'Position',[0.136405039308876 0.55733989558245 0.0960658737419946 0.0339805825242718],...
'String','Rate',...
'Style','edit',...
'CreateFcn', {@local_CreateFcn, 'bigger_gui_export("channel_acquisition_rate_CreateFcn",gcbo,
[],guidata(gcbo))', appdata} ,...
'Tag','channel_acquisition_rate',...
'Behavior',get(0,'defaultuicontrolBehavior'));

appdata = [];
```

```
appdata.lastValidTag = 'plot_window';

h7 = axes(...
'Parent',h1,...
'Position',[0.288299300988258 0.0574072134875163 0.629377431906615 0.886731391585761],...
'CameraPosition',[0.5 0.5 9.16025403784439],...
'CameraPositionMode',get(0,'defaultaxesCameraPositionMode'),...
'Color',get(0,'defaultaxesColor'),...
'ColorOrder',get(0,'defaultaxesColorOrder'),...
'XColor',get(0,'defaultaxesXColor'),...
'YColor',get(0,'defaultaxesYColor'),...
'ZColor',get(0,'defaultaxesZColor'),...
'Tag','plot_window',...
'Behavior',get(0,'defaultaxesBehavior'),...
'CreateFcn', {@local_CreateFcn, '', appdata} );

appdata = [];
appdata.SerializedAnnotationV7 = struct(...
   'LegendInformation', struct(...
   'IconDisplayStyle', 'on'));

h8 = get(h7,'title');

set(h8,...
'Parent',h7,...
'Color',[0 0 0],...
'HorizontalAlignment','center',...
'Position',[0.5 1.01128472222222 1.00005459937205],...
'VerticalAlignment','bottom',...
'HandleVisibility','off',...
'Behavior',struct(),...
'CreateFcn', {@local_CreateFcn, '', appdata} );

appdata = [];
appdata.SerializedAnnotationV7 = struct(...
   'LegendInformation', struct(...
   'IconDisplayStyle', 'on'));

h9 = get(h7,'xlabel');

set(h9,...
'Parent',h7,...
'Color',[0 0 0],...
'HorizontalAlignment','center',...
'Position',[0.498242530755712 -0.0407986111111112 1.00005459937205],...
'VerticalAlignment','cap',...
'HandleVisibility','off',...
'Behavior',struct(),...
'CreateFcn', {@local_CreateFcn, '', appdata} );
```

```
appdata = [];
appdata.SerializedAnnotationV7 = struct(...
   'LegendInformation', struct(...
   'IconDisplayStyle', 'on'));

h10 = get(h7,'ylabel');

set(h10,...
'Parent',h7,...
'Color',[0 0 0],...
'HorizontalAlignment','center',...
'Position',[-0.0500878734622144 0.497395833333333 1.00005459937205],...
'Rotation',90,...
'VerticalAlignment','bottom',...
'HandleVisibility','off',...
'Behavior',struct(),...
'CreateFcn', {@local_CreateFcn, '', appdata} );

appdata = [];
appdata.SerializedAnnotationV7 = struct(...
   'LegendInformation', struct(...
   'IconDisplayStyle', 'on'));

h11 = get(h7,'zlabel');

set(h11,...
'Parent',h7,...
'Color',[0 0 0],...
'HorizontalAlignment','right',...
'Position',[-0.459578207381371 1.05989583333333 1.00005459937205],...
'HandleVisibility','off',...
'Behavior',struct(),...
'Visible','off',...
'CreateFcn', {@local_CreateFcn, '', appdata} );

appdata = [];
appdata.lastValidTag = 'stop';

h12 = uicontrol(...
'Parent',h1,...
'Units','normalized',...
'Callback','bigger_gui_export(''stop_Callback'',gcbo,[],guidata(gcbo))',...
'FontSize',10,...
'ListboxTop',0,...
'Position',[0.040929203539823 0.827426810477658 0.0940265486725664 0.0354391371340524],...
'String','Stop',...
'Tag','stop',...
'Behavior',get(0,'defaultuicontrolBehavior'),...
```

```
'CreateFcn', {@local_CreateFcn, '', appdata} );

appdata = [];
appdata.lastValidTag = 'channel_info';

h13 = uibuttongroup(...
'Parent',h1,...
'Units','characters',...
'Title','',...
'BackgroundColor',[0.866666666666667 0.866666666666667 0.866666666666667],...
'Position',[7 24.2307692307692 36.4 14.0769230769231],...
'Clipping','off',...
'Tag','channel_info',...
'Behavior',struct(),...
'SelectedObject',[],...
'SelectionChangeFcn',[],...
'OldSelectedObject',[],...
'CreateFcn', {@local_CreateFcn, '', appdata} );

appdata = [];
appdata.lastValidTag = 'channel_read_status_yes';

h14 = uicontrol(...
'Parent',h13,...
'Units','characters',...
'BackgroundColor',[0.866666666666667 0.866666666666667 0.866666666666667],...
'Callback',mat{1},...
'CData',[],...
'Position',[21.6 7.84615384615385 11.2 1.61538461538462],...
'String','Yes',...
'Style','radiobutton',...
'Value',1,...
'Tag','channel_read_status_yes',...
'UserData',[],...
'Behavior',get(0,'defaultuicontrolBehavior'),...
'CreateFcn', {@local_CreateFcn, '', appdata} );

appdata = [];
appdata.lastValidTag = 'channel_read_status_no';

h15 = uicontrol(...
'Parent',h13,...
'Units','characters',...
'BackgroundColor',[0.866666666666667 0.866666666666667 0.866666666666667],...
'Callback',mat{2},...
'Position',[21.6 6.23076923076923 11.2 1.84615384615385],...
'String','No',...
'Style','radiobutton',...
'Tag','channel_read_status_no',...
```

```
'Behavior',get(0,'defaultuicontrolBehavior'),...
'CreateFcn', {@local_CreateFcn, '', appdata} );

appdata = [];
appdata.lastValidTag = 'channel_read_status_label';

h16 = uicontrol(...
'Parent',h13,...
'Units','normalized',...
'BackgroundColor',[0.868 0.868 0.868],...
'FontSize',10,...
'ListboxTop',0,...
'Position',[0.0164835164835166 0.551912568306011 0.291208791208791 0.0983606557377049],...
'String','Read -',...
'Style','text',...
'Tag','channel_read_status_label',...
'Behavior',get(0,'defaultuicontrolBehavior'),...
'CreateFcn', {@local_CreateFcn, '', appdata} );

appdata = [];
appdata.lastValidTag = 'channel_select';

h17 = uicontrol(...
'Parent',h13,...
'Units','normalized',...
'BackgroundColor',[1 1 1],...
'Callback','bigger_gui_export(''channel_select_Callback'',gcbo,[],guidata(gcbo))',...
'FontSize',10,...
'ListboxTop',0,...
'Position',[0.010989010989011 0.814207650273224 0.56043956043956 0.158469945355191],...
'String',{  'Channel 1'; 'Channel 2'; 'Channel 3'; 'Channel 4'; 'Channel 5'; 'Channel 6'; 'Channel 7';
'Channel 8'; 'Channel 9'; 'Channel 10'; 'Channel 11'; 'Channel 12'; 'Channel 13'; 'Channel 14'; 'Channel
15'; 'Channel 16' },...
'Style','popupmenu',...
'Value',1,...
'CreateFcn', {@local_CreateFcn, 'bigger_gui_export(''channel_select_CreateFcn'',gcbo,
[],guidata(gcbo))', appdata} ,...
'Tag','channel_select',...
'Behavior',get(0,'defaultuicontrolBehavior'));

appdata = [];
appdata.lastValidTag = 'channel_connect_status_label';

h18 = uicontrol(...
'Parent',h13,...
'Units','normalized',...
'BackgroundColor',[0.868 0.868 0.868],...
'FontSize',10,...
'ListboxTop',0,...
```

```
'Position',[0.00561797752808989 0.69060773480663 0.48314606741573 0.0994475138121547],...
'String','Connected -',...
'Style','text',...
'Tag','channel_connect_status_label',...
'Behavior',get(0,'defaultuicontrolBehavior'),...
'CreateFcn', {@local_CreateFcn, '', appdata} );


hsingleton = h1;


% --- Set application data first then calling the CreateFcn.
function local_CreateFcn(hObject, eventdata, createfcn, appdata)

if ~isempty(appdata)
   names = fieldnames(appdata);
   for i=1:length(names)
       name = char(names(i));
       setappdata(hObject, name, getfield(appdata,name));
   end
end

if ~isempty(createfcn)
   eval(createfcn);
end


% --- Handles default GUIDE GUI creation and callback dispatch
function varargout = gui_mainfcn(gui_State, varargin)


%   GUI_MAINFCN provides these command line APIs for dealing with GUIs
%
%     BIGGER_GUI_EXPORT, by itself, creates a new BIGGER_GUI_EXPORT or raises the existing
%     singleton*.
%
%     H = BIGGER_GUI_EXPORT returns the handle to a new BIGGER_GUI_EXPORT or the handle
to
%     the existing singleton*.
%
%     BIGGER_GUI_EXPORT('CALLBACK',hObject,eventData,handles,...) calls the local
%     function named CALLBACK in BIGGER_GUI_EXPORT.M with the given input arguments.
%
%     BIGGER_GUI_EXPORT('Property','Value',...) creates a new BIGGER_GUI_EXPORT or raises
the
%     existing singleton*.  Starting from the left, property value pairs are
%     applied to the GUI before untitled_OpeningFunction gets called.  An
%     unrecognized property name or invalid value makes property application
%     stop.  All inputs are passed to untitled_OpeningFcn via varargin.
```

```
%
%     *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%     instance to run (singleton)".

%   Copyright 1984-2004 The MathWorks, Inc.
%   $Revision: 1.4.6.8 $ $Date: 2004/04/15 00:06:57 $

gui_StateFields =  {'gui_Name'
                    'gui_Singleton'
                    'gui_OpeningFcn'
                    'gui_OutputFcn'
                    'gui_LayoutFcn'
                    'gui_Callback'};
gui_Mfile = '';
for i=1:length(gui_StateFields)
   if ~isfield(gui_State, gui_StateFields{i})
      error('Could not find field %s in the gui_State struct in GUI M-file %s', gui_StateFields{i},
gui_Mfile);
   elseif isequal(gui_StateFields{i}, 'gui_Name')
      gui_Mfile = [gui_State.(gui_StateFields{i}), '.m'];
   end
end

numargin = length(varargin);

if numargin == 0
   % BIGGER_GUI_EXPORT
   % create the GUI
   gui_Create = 1;
elseif isequal(ishandle(varargin{1}), 1) && ispc && iscom(varargin{1}) &&
isequal(varargin{1},gcbo)
   % BIGGER_GUI_EXPORT(ACTIVEX,...)
   vin{1} = gui_State.gui_Name;
   vin{2} = [get(varargin{1}.Peer, 'Tag'), '_', varargin{end}];
   vin{3} = varargin{1};
   vin{4} = varargin{end-1};
   vin{5} = guidata(varargin{1}.Peer);
   feval(vin{:});
   return;
elseif ischar(varargin{1}) && numargin>1 && isequal(ishandle(varargin{2}), 1)
   % BIGGER_GUI_EXPORT('CALLBACK',hObject,eventData,handles,...)
   gui_Create = 0;
else
   % BIGGER_GUI_EXPORT(...)
   % create the GUI and hand varargin to the openingfcn
   gui_Create = 1;
end

if gui_Create == 0
```

```
    varargin{1} = gui_State.gui_Callback;
    if nargout
        [varargout{1:nargout}] = feval(varargin{:});
    else
        feval(varargin{:});
    end
else
    if gui_State.gui_Singleton
        gui_SingletonOpt = 'reuse';
    else
        gui_SingletonOpt = 'new';
    end

    % Open fig file with stored settings.  Note: This executes all component
    % specific CreateFunctions with an empty HANDLES structure.

    % Do feval on layout code in m-file if it exists
    if ~isempty(gui_State.gui_LayoutFcn)
        gui_hFigure = feval(gui_State.gui_LayoutFcn, gui_SingletonOpt);
        % openfig (called by local_openfig below) does this for guis without
        % the LayoutFcn. Be sure to do it here so guis show up on screen.
        movegui(gui_hFigure,'onscreen')
    else
        gui_hFigure = local_openfig(gui_State.gui_Name, gui_SingletonOpt);
        % If the figure has InGUIInitialization it was not completely created
        % on the last pass.  Delete this handle and try again.
        if isappdata(gui_hFigure, 'InGUIInitialization')
            delete(gui_hFigure);
            gui_hFigure = local_openfig(gui_State.gui_Name, gui_SingletonOpt);
        end
    end

    % Set flag to indicate starting GUI initialization
    setappdata(gui_hFigure,'InGUIInitialization',1);

    % Fetch GUIDE Application options
    gui_Options = getappdata(gui_hFigure,'GUIDEOptions');

    if ~isappdata(gui_hFigure,'GUIOnScreen')
        % Adjust background color
        if gui_Options.syscolorfig
            set(gui_hFigure,'Color', get(0,'DefaultUicontrolBackgroundColor'));
        end

        % Generate HANDLES structure and store with GUIDATA
        guidata(gui_hFigure, guihandles(gui_hFigure));
    end

    % If user specified 'Visible','off' in p/v pairs, don't make the figure
```

```
% visible.
gui_MakeVisible = 1;
for ind=1:2:length(varargin)
   if length(varargin) == ind
      break;
   end
   len1 = min(length('visible'),length(varargin{ind}));
   len2 = min(length('off'),length(varargin{ind+1}));
   if ischar(varargin{ind}) && ischar(varargin{ind+1}) && ...
         strncmpi(varargin{ind},'visible',len1) && len2 > 1
      if strncmpi(varargin{ind+1},'off',len2)
         gui_MakeVisible = 0;
      elseif strncmpi(varargin{ind+1},'on',len2)
         gui_MakeVisible = 1;
      end
   end
end

% Check for figure param value pairs
for index=1:2:length(varargin)
   if length(varargin) == index
      break;
   end
   try set(gui_hFigure, varargin{index}, varargin{index+1}), catch break, end
end

% If handle visibility is set to 'callback', turn it on until finished
% with OpeningFcn
gui_HandleVisibility = get(gui_hFigure,'HandleVisibility');
if strcmp(gui_HandleVisibility, 'callback')
   set(gui_hFigure,'HandleVisibility', 'on');
end

feval(gui_State.gui_OpeningFcn, gui_hFigure, [], guidata(gui_hFigure), varargin{:});

if ishandle(gui_hFigure)
   % Update handle visibility
   set(gui_hFigure,'HandleVisibility', gui_HandleVisibility);

   % Make figure visible
   if gui_MakeVisible
      set(gui_hFigure, 'Visible', 'on')
      if gui_Options.singleton
         setappdata(gui_hFigure,'GUIOnScreen', 1);
      end
   end

   % Done with GUI initialization
   rmappdata(gui_hFigure,'InGUIInitialization');
```

```
    end

    % If handle visibility is set to 'callback', turn it on until finished with
    % OutputFcn
    if ishandle(gui_hFigure)
        gui_HandleVisibility = get(gui_hFigure,'HandleVisibility');
        if strcmp(gui_HandleVisibility, 'callback')
            set(gui_hFigure,'HandleVisibility', 'on');
        end
        gui_Handles = guidata(gui_hFigure);
    else
        gui_Handles = [];
    end

    if nargout
        [varargout{1:nargout}] = feval(gui_State.gui_OutputFcn, gui_hFigure, [], gui_Handles);
    else
        feval(gui_State.gui_OutputFcn, gui_hFigure, [], gui_Handles);
    end

    if ishandle(gui_hFigure)
        set(gui_hFigure,'HandleVisibility', gui_HandleVisibility);
    end
end

function gui_hFigure = local_openfig(name, singleton)

% openfig with three arguments was new from R13. Try to call that first, if
% failed, try the old openfig.
try
    gui_hFigure = openfig(name, singleton, 'auto');
catch
    % OPENFIG did not accept 3rd input argument until R13,
    % toggle default figure visible to prevent the figure
    % from showing up too soon.
    gui_OldDefaultVisible = get(0,'defaultFigureVisible');
    set(0,'defaultFigureVisible','off');
    gui_hFigure = openfig(name, singleton);
    set(0,'defaultFigureVisible',gui_OldDefaultVisible);
end
```