Parallel Port Interface for the Mentor Robot Arm

Jacky Baltes
j.baltes@auckland.ac.nz

Weidong Xu
wxu002@student.auckland.ac.nz

Centre for Imaging Technology and Robotics
Department of Computer Science
University of Auckland
Private Bag 92019, Auckland, New Zealand

Abstract

This paper describes the design of an interface board which by emulating the bus of some popular home computers can control legacy hardware through a generic parallel port interface and serial interface. In particular, the board is currently being used to control two Mentor robot arms from a PC. We also developed firmware and a device driver for the Linux operating system.

1 Introduction

In 1999, the robotics paper offered by the computer science department at the University of Auckland was moved from the city campus to the new Centre for Imaging technology and Robotics (CITR). As part of this move, the CITR inherited two old Mentor robot arms. The robot interfaces used a 1 MHz bus to connect directly to some popular home computers available at the time of their design (ZX Spectrum, BBC micro, and VIC 20).

The computer science department had used Acorn Archimedes computers which could emulate a BBC bus. However, it was decided that it was too much work to resurrect the Acorn Archimedes computers and that instead what was desirable was to design and implement a board which supports more commonly available parallel and serial interfaces. In particular, the computer labs at the CITR mostly use PCs running Linux and the goal of this project was the design of an interface board which allows control of the robot arms from this configuration.

A modern micro controller provides a cheap flexible solution to many control problems. However, a typical micro controller does not include D-A converters. However most modern micro controllers provide special counters and timers which allow the implementation of a 1 bit D-A converter using PWM modulation. Since this method is supported by the hardware, it provides a simple and efficient method for D-A conversion. Unfortunately, most micro controllers only provide few of those special registers. In principle, any parallel output pin can be used as a 1 bit D-A converter using PWM, but this results in a high overhead on the micro controller. Therefore, it was decided to use the original motion controller board (with five D-A converters).

We designed a 68HC11F1 based micro controller board which emulates the bus of the VIC 20 computer and which supports the now standard bi-directional parallel port of a PC compatible computer and also a generic serial interface. Section 2 describes the hardware of the interface board.

Section 3 describes the communications protocol for the parallel port, since the data written by the micro is not latched. To achieve an adequate trade-off between communication speed and overhead on the PC, a master-slave communication model was used.

We also designed a device driver for the Linux OS, which provides a generic interface to robot arms with up to 6 degrees of freedom. The device driver is described in section 4.

The paper concludes with section 5. The complete schematic of the interface board is shown in the appendix.

2 Interface Board Design

This section describes the hardware of the interface board. The complete schematic is shown in the appendix.

The board uses Motorola's MC68HC11F1 micro controller. The 68HC11 is a very popular and cheap micro controller family, which we have used in a number of projects such as a local controller board for our autonomous mobile robots.

The interface board uses the extended mode of the 68HC11F1 microprocessor. In extended mode, three ports of the micro controller are used to implement the address and data bus.

A serial line driver (Max232) converts the serial port voltages from the micro controller into standard RS232 voltages. This interface can also be used to download programs into the EEPROM memory of the micro controller.

The EEPROM on the 68HC11 is limited to 512 Bytes. This is sufficient to implement the primitive communication protocol to the PC (as described in section 3) and the interface to the Mentor robot arms. To allow for future extensions, the interface board has a free socket for a 8 KByte EPROM. This allows the implementation of more sophisticated motion control and inverse kinematics algorithms on the interface itself.

U2 and U4 are low voltage indicators that are used to implement the power up reset as well as the manual reset button.

2.1 Parallel Port Interface

The parallel port is implemented using an 8-bit bidirectional buffer (74HC245) for the data bus and an 8-bit buffer for the control signals. The control signals are connected to PortA of the micro controller. The data bus is controlled via the CSIO2 signal, which is a programmable chip select signal. Any memory access to addresses $0x1F00 - 0x1FFF$ will access the buffer instead of standard memory.

One problem is that the data at the buffer is not latched with the read/write signal. This causes a problem when sending data from the micro controller to the PC. The data on the parallel port will only be valid for exactly one clock cycle. If the PC does read the data to early or too late, it will read the incorrect data. This problem is overcome by the design of a strict master-slave communication protocol, which is described in the section 3.

2.2 Robot Arm Interface

The Mentor uses a 1 MHz bus interface with an 8 bit data bus (D0..7) and four address lines (A0..A3). In addition it uses a R/W signal, CSIO1 (Chip Select), and the clock. The interface to the robot arm is implemented using another 8 bit tri-state buffer (74HC245) which is controlled by the CSIO1 signal from the micro controller. The micro controller maps this buffer at the memory region $0x1060 - 0x17FF$. Currently, we are using a 68HC11F1 with an 8 MHz clock, which translates into a 2 MHz external clock. The robot arm, therefore, provides a clock divider (74HC74) which can be used to divide the clock by two. However, in our early testing we determined that the Mentor robot arm performed well even when using a 2 MHz clock. Therefore, the clock divider is not used in our current implementation. However, it may be necessary to use the divider when connecting to other peripheral devices.

2.3 Firmware

The interface board currently uses only the internal EEPROM of the micro controller and consists of two parts: the communication handler and the main loop.

The communication handler is entered via an interrupt whenever the PC sets nSTROBE. It receives two bytes from the PC and returns one byte. Internally the micro controller maintains two sets of values for each axis. The first block contains the desired values, that is the settings that we want the arm to be in. Since the robot arms move at a specific speed, it requires a certain amount of time before a robot arm reaches its desired position. The second block contains the actual values. The actual values should approach the desired values as time passes.

The main loop is an infinite loop. First, the desired settings for all axis are send to the micro controller. Since the desired settings are latched in the robot arm interface, this is strictly speaking not necessary. However, since the desired heading settings can not be read back by the micro, it would require additional hand shaking between the interrupt handler, which receives the data and the main loop which sends it to the robot arm interface to indicate whether a value has already been written to the robot arm interface. Code complexity is greatly simplified at negligible expense by always writing the desired positions to the robot arm interface.

In the bottom half of the loop, the actual values from the robot arm are read back and stored in the actual settings block. If the communication interrupt handler recognizes a read command, the value from the actual block is send to the PC.

3 Parallel Port Communication

The communication between the micro and the PC is fully asynchronous. The parallel port on the original IBM PC XT did not support bi-directional communication. More recently a number of enhancements have been made to the parallel port and are included in most modern PCs. In particular, they provide an enhanced mode for bi-directional communication (EPP) and a special mode for high speed communication using DMA (ECP). Since the amount of data to be transfered between the PC and the micro is small, a simple bi-directional mode (sometimes referred to as Byte Mode) is used.

In Byte mode, the parallel port provides 8 bi-directional data bits (IN-OUT) and additional control lines: nSTROBE (OUT), nAUTOFEED (OUT), nINIT (OUT), nSELECTIN (OUT), nACK (IN), BUSY (IN), PE (IN), SELECT (IN), and nERROR (IN) [1].

To reduce the load on the PC, the interface uses a master slave communication protocol, that is all communication is initiated by the PC. The timing of the communication is shown in Fig. 1. The PC first waits for the micro to clear the BUSY flag, which indicates that the micro is ready to receive data. The PC sets the STROBE signal to indicate that it is ready to transmit data to the micro. The micro acknowledges reception of the data by toggling the BUSY signal. The signal nAUTOFEED is used to synchronize communication by indicating whether the command or data byte of a command is being transmitted.

After sending two bytes, the PC reads exactly one byte from the micro. In this case, the PC sets the INIT flag. After receiving the INIT signal, the micro will write the data on the data bus. Since this write is not latched, the PC polls the ERROR flag and reads the data. The PC acknowledges receipt of the data by toggling the INIT flag.

4 Device Driver Implementation

In the UNIX paradigm, external devices are treated in the same way as files. For example, to send and receive data from a modem, the application program writes and reads data from a special file, which is associated with the modem hardware. These special files are called *device files*. Faster or more complex interactions (e.g., continuous synchronized capture mode of a frame grabber) between the external device and the application program can be implemented using shared memory. Since all communication between the PC and the robot arm occurs at comparatively slow speed, the robot arm device driver uses only the read and write interface.

Currently, the device driver only supports a primitive protocol: (a) setting one axis of the robot arm to a specific value (absolute motion), (b) adding or subtracting an offset to the current value of a specific axis (relative motion), and (c) reading back the current setting of a particular axis (read back).

The following table shows the syntax of these commands. All data are unsigned bytes, except for the offset value which is interpreted as a signed value.

Command	Comment
stXX,YY\n	set axis XX to YY
osXX,YY\n	set axis XX to current value + YY
rdXX,ZZ\n	read back the value of axis XX ZZ is ignored

Since all commands are assumed to be generated by an application program, the syntax of the commands is very strict and terse. Also, all commands use a fixed format, which simplifies generation of commands in the application program (e.g., fixed memory allocation).

Figure 2 shows a small example of an application program. First the application opens a file with the special name `/dev/robot0` (this corresponds to the first robot arm) for reading and appending. After executing a series of commands, the application closes the file, which makes the robot arm accessible to other programs.

5 Conclusion

The interface board was built and tested in May and June of 1999. After the first successful tests it has been used by students in the robotics paper starting from July 1999.

One problem with the communication emerged during stress test of the robot arms in the robotics lab. Currently, the aim of the robotics paper is to play a game of badminton with the two robot arms. The position of the shuttle cock is determined through binocular stereo vision from two pan and tilt cameras which are mounted on the wall and look at the playing field.

The video information is processed by two PCs with Sequence P1S frame grabbers. These two video servers capture true colour (24 bit) images with a resolution of 576 by 768 at 50 fields per second. The resulting transfer rate of 32 MB/sec puts a high load on the CPU and the PCI bus of the video server.

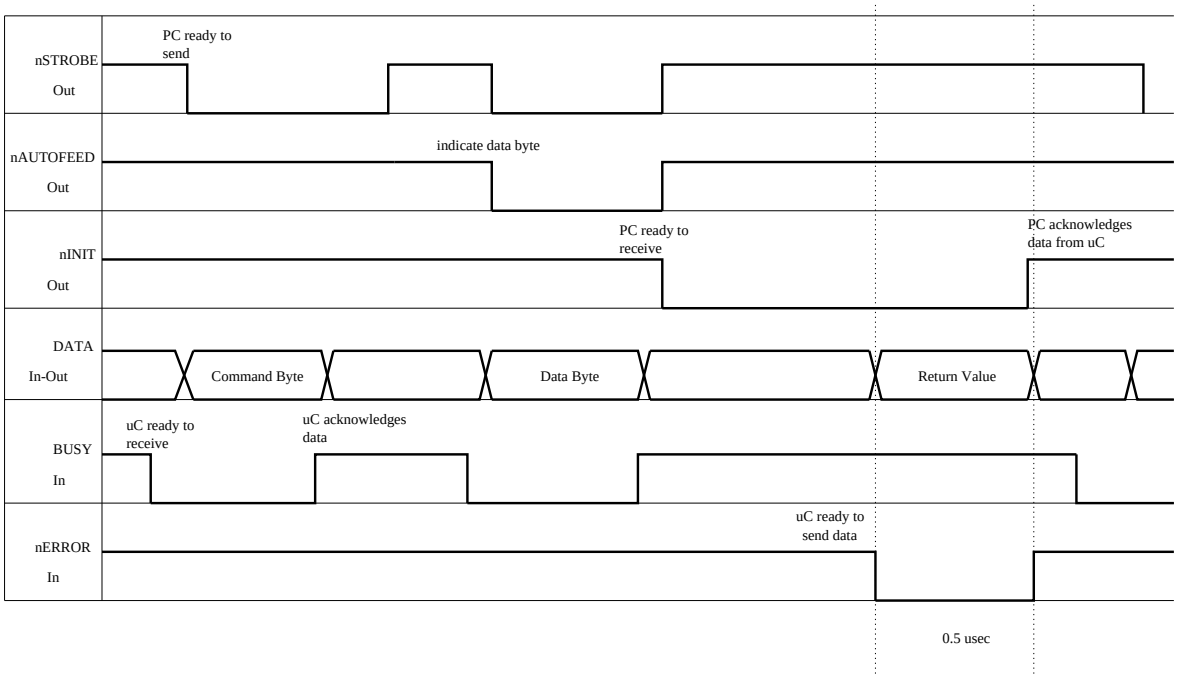


Figure 1: Timing of the Parallel Port Communication between the PC and the Micro-controller

```
#include <stdio.h>

int main(int argc, char *argv[]) {
  FILE * fp;
  int axis, value;

  fp = fopen("/dev/robot0","rw+");

  /* Set axis 1 to value 80 */
  axis = 1; value = 0x80;
  fprintf(fp,"st%02X,%02X\n",axis,value);
  fflush(fp);

  /* Send read back command for axis 2 */
  axis=2;
  fprintf(fp,"rd%02X,00\n",axis);
  fflush(fp);
  /* read back the value of axis 2 */
  fread(&value,1,1,fp);

  fclose(fp);
  return 0;
}
```

Figure 2: Schematic of the Robot Arm Interface

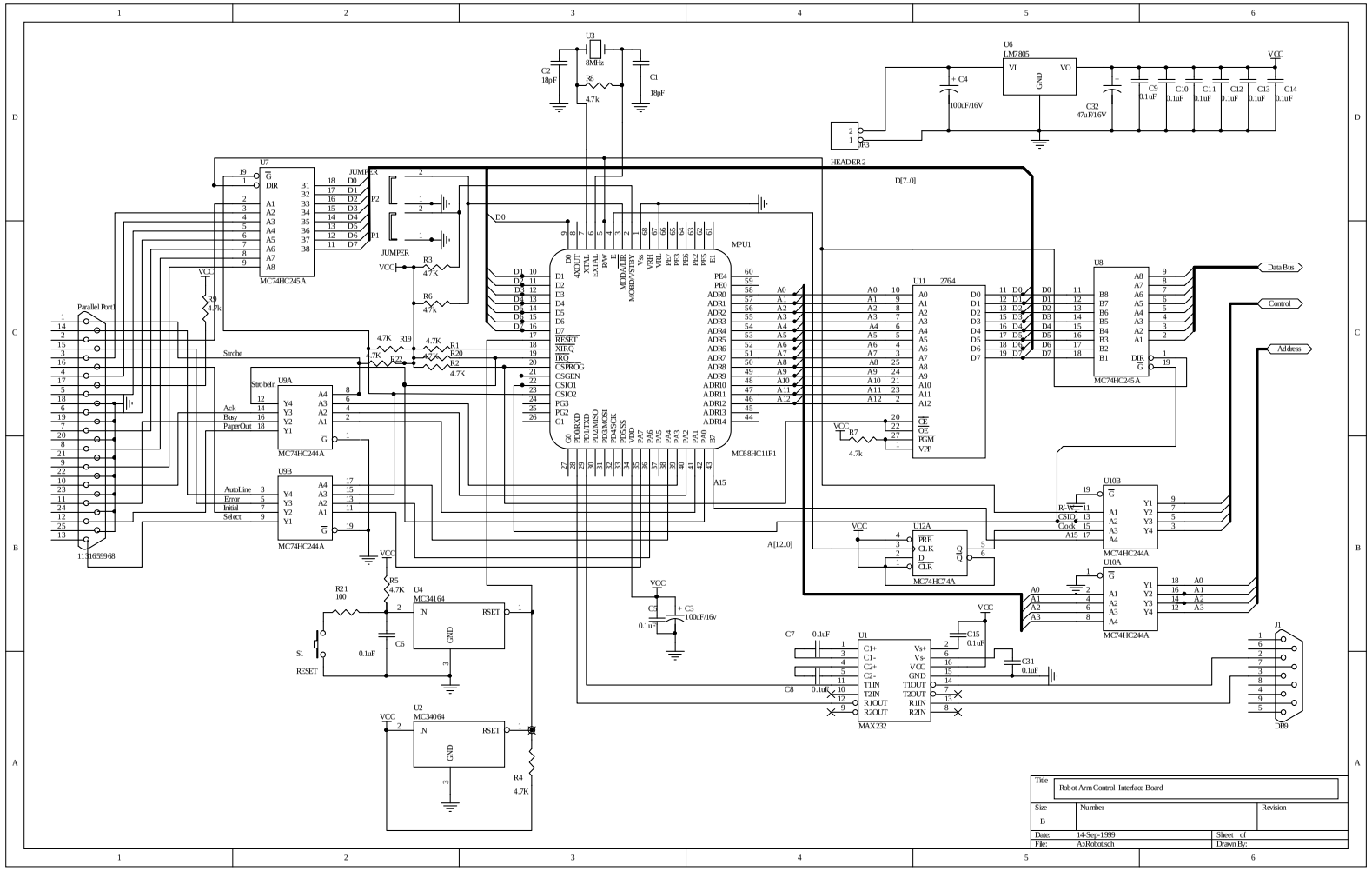
Under these conditions it is impossible to control the robot arm while simultaneously running the video server, since as described in section 3 the data from the micro to the PC is not latched. Communication errors occur when the higher priority frame grabber device interrupts the robot arm device driver.

Therefore, we are currently modifying the board to use a 74HC574 (Octal D-Flipflop with 3 State Outputs) bi-directional latch.

Currently, the device driver only implements a low level interface to the robot arm. We are working on extending the device driver to support queuing of commands and includes inverse kinematics routines.

References

[1] Warp Nine Engineering. Ieee 1284-1994 standard. http://www.fapo.com/1284int.htm, September 1999.


U3
8MHz
C2
18pF
R8
4.7k
C1
18pF
U6
LM7805
VI
VO
GND
+ C4
100uF/16V
C32
47uF/16V
C9
0.1uF
C10
0.1uF
C11
0.1uF
C12
0.1uF
C13
0.1uF
C14
0.1uF
VCC
JP3
HEADER 2
D[7..0]
U7
MC74HC245A
JUMPER
JUMPER
R3
4.7K
R6
4.7K
R9
4.7k
MPU1
MC68HC11F1
Parallel Port1
1113659968
Strobe
R19
4.7K
R1
4.7K
R20
4.7K
R22
4.7K
R2
4.7K
U9A
U9B
MC74HC244A
Strobeln
Ack
Busy
PaperOut
Select
AutoLine
Error
Initial
U11
2764
R7
4.7k
U8
MC74HC245A
Data Bus
Control
Address
U10B
U10A
MC74HC244A
U12A
MC74HC74A
A[12..0]
A15
R21
100
R5
4.7K
U4
MC34164
C6
0.1uF
S1
RESET
C5
0.1uF
C3
100uF/16v
U2
MC34064
R4
4.7K
C7
0.1uF
C8
0.1uF
U1
MAX232
C15
0.1uF
C31
0.1uF
J1
DB9
Title
Robot Arm Control Interface Board
Size
B
Number
Revision
Date
14-Sep-1999
Sheet of
File:
A:\Robot.sch
Drawn By: