STANDARD INTERFACE BETWEEN
NC-MACHINE AND INDUSTRIAL ROBOT

BACHELOR THESIS IN
ELECTRONICS ENGINEERING
15 CREDITS, BASIC LEVEL

June, 2010

Author(s): Jose Antonio Martin and Eduardo Trujillo
Supervisor: Lars Asplund
Abstract

The aim of this thesis is the design and implementation of a standardized interface to communicate a NC machine with an industrial robot.

This interface consists of an ‘electronic box’ that include a powerful board perfect for automation projects composed of a microcontroller, serial ports, optocouplers, relays, etc... The main component of this PCB is the microcontroller. For this purpose the Atmel AT90CAN128 microcontroller is a good choice, because is perfectly suited for industrial and automotive applications and support CANopen and DeviceNet implementation.

The industrial robot’s communication between the board is performed through CAN\textsuperscript{1} bus. To communicate the interface with the NC\textsuperscript{2} machine it has eight optocouplers are used as inputs and eight relays are used as outputs. Developments of programs using C language to communicate robots and NC machines have been developed by the software that provides Atmel for 8-bit AVR applications called AVR Studio 4 + WinAVR.

\textsuperscript{1} CAN (Controller Area Network)  
\textsuperscript{2} NC (Numerical Control)
Preface

Thanks to Lars Asplund for being so patient with us and for ours funny meetings in his office. Thanks to the Ekström’s Brothers (Martin and Mikael) for help us with our theses always that we needed their help.
Table of contents

1. Introduction ..................................................................................................................... 1

2. Relevant Theory .............................................................................................................. 2
   2.1. Serial communication ............................................................................................... 2
   2.2. CAN protocol ........................................................................................................... 5
       2.2.1. Introduction ........................................................................................................ 5
   2.3. CAN working method .............................................................................................. 5
       2.3.1. CAN features ....................................................................................................... 5
   2.4. DeviceNet .................................................................................................................. 5
       2.4.1. Introduction ........................................................................................................ 5

3. History ............................................................................................................................. 6

4. Architecture .................................................................................................................... 6
   4.1.1. The Physical Layer ............................................................................................... 7
   4.1.2. The Data Link Layer ........................................................................................... 9
   4.1.3. The Network and Transports Layers .................................................................. 9
   4.1.4. The Uppers Layers ............................................................................................ 10

5. Hardware ........................................................................................................................ 12
   5.1. The schematic design ............................................................................................. 12
   5.2. Standard interface features ..................................................................................... 13
   5.3. Functional Block Diagram ..................................................................................... 14
   5.4. Microcontroller ....................................................................................................... 15
   5.5. Serial communication ............................................................................................. 17
   5.6. CAN bus interface ................................................................................................. 18
   5.7. NC-Machine communication .................................................................................. 19
   5.8. Power supply .......................................................................................................... 22
   5.9. USB ......................................................................................................................... 23
   5.10. JTAG ....................................................................................................................... 23

6. Software .......................................................................................................................... 24

7. Conclusions and future works ....................................................................................... 25
   7.1. Conclusion .............................................................................................................. 25
   7.2. Future works .......................................................................................................... 25
Chapter 1

1. Introduction

At present robotics companies are a growth industry due to advantages that provide work with robots. This work was thought to satisfy requirements of this kind of companies. This thesis project shows how the communication between NC machines and Robot through a standard interface has been made. One of the goals set is the design and develop of a prototype board. The target of this board is testing, programming and debugging programs to communicate robots and NC machines using C language.

Currently, one of the most used communication protocols for automotive, robotic and factory automation is CAN protocol. Due to this reason and also because support DeviceNet, the microcontroller that has been chosen is the AT90CAN128. This microcontroller will be described later with more detail.
Chapter 2

2. Relevant Theory

The communication with the industrial robot is performed by serial communication and the communication protocol DeviceNet. The following describes these communication protocols.

2.1. Serial communication

Serial communication is the most common form of communication between electronics devices.

There are three different transmission modes:

- Simplex: in this transmission mode the data can flow in only one direction. This transmission mode is useful when the data do not need to flow in both directions.

![Simplex transmission](image1)

Figure 1.1 – Simplex transmission
• Half duplex: is a transmission mode where the data can flow in two directions, but never at the same time.

![Half Duplex Diagram](image1)

Figure 1.2 – Half duplex transmission

• Full duplex: in this transmission mode the data flow in both directions simultaneously. This type of transmission mode consists of two simplex channels, one transmitting and one receiving.

![Full Duplex Diagram](image2)

Figure 1.3 – Full duplex transmission

There are two methods for serial communication that ensure the correct communication:

• Synchronous: in this method the transmitter and the receiver are synchronized before the transmission begins. In addition it is requires a separate line signal line to carry a clock pulse that trigger the arrival of a new bit. The advantages of this method are high rates and less error.
Asynchronous: in this method the transmitter and receiver have independents clocks. To synchronize the sending and receiving of the data two special bits (start and stop) are added in each word. When a word is sent to the receiver in the beginning of the each word there is a special bit called bit start. The bit start triggers an internal timer in the receiver that generates the needed clock pulses. After the bit start, the bits of the word of data are transmitted. When all data word is sent, the transmitter may add bit parity. This bit parity can be used for the receiver performs error checking. The transmission is concluded when the bit stop is sent by the transmitter.

Figure 1.4 – Typical serial communication format

The standard interface uses full duplex asynchronous communication for communicate with the industrial robot.
2.2. CAN protocol

As the CAN is the backbone technology of DeviceNet the following describe a short introduction about it.

2.2.1. Introduction
The CAN is a protocol used for serial data communication that supports distributed real-time control and multiplexing for use within road vehicles and was designed for automotive applications, industrial automation and mobile machines.

2.2.2. CAN working method
The CAN protocol enables all the nodes connected to the bus to talk to one another creating a communication path between them. The designer of the system can choose if there is or not a main or central node. This protocol permits a high flexibility to implement the nodes in ways that suit the particular application.

2.2.3. CAN features
CAN Controller 2.0A & 2.0B – ISO 16845 Certified

- 15 Full Message Objects with Separate Identifier Tags and Masks.
- Transmit, Receive, Automatic Reply and Frame Buffer Receive Modes.
- 1Mbits/s Maximum Transfer Rate at 8 MHz
- Time stamping, TTC & Listening Mode (Spying or Autobaud).

2.3. DeviceNet

2.3.1. Introduction
DeviceNet is a digital multi-drop network to connect simple industrial devices (sensors and actuators) and high level devices (programmable logic controllers and
computers). It is generally used in industrial and process automation principally in the United States and now it is gaining popular in Europe, Japan and China.

2.3.2. History
DeviceNet was originally created by American company Allen-Bradley (Rockwell Automation) in 1984. In 1985 Allen-Bradley transferred its technology to Open DeviceNet Vendor Association (ODVA) with the purpose of promote its use. Now ODVA maintains the specifications, supervises advances and ensures compliance to DeviceNet standards.

2.3.3. Architecture
DeviceNet follows the Open System Interconnection (OSI) model which specifies seven layer frameworks for implementing network protocols. The seven layers are shown in the figure 1.1

![OSI Model](image)

The Physical and Data link layers of DeviceNet are defined by CAN specification. CAN was developed by Bosch in Germany for the European automobile market to replace the wiring complexity expensive for a low-cost network for cars.

In the upper layers DeviceNet uses the Common Industrial Protocol (CIP), which is specified by ODVA. The network architecture is shown in the figure 1.2
2.3.4. The Physical Layer

DeviceNet uses a trunk/line drop/line topology which means that power source and signal wiring are in a separate twisted pair bus. Using this topology, nodes can be removed and inserted while the network is on, helping to reduce production downtime. The power supplied to the network is provide
by a power source connected directly to the network, which feeds the CAN transceivers of the nodes. This power source has the following features:

- 24 Vdc
- DC output isolated from the AC input
- Current capacity compatible with the installed equipment

The maximum current rating of the trunk/line is 8 A depending on the cables types. In the figure 1.3 is shown different types of this topology.

DeviceNet supports 125 kbit/s, 250 kbit/s and 500 kbit/s the user may choose between several cable options.
2.3.5. The Data Link Layer

DeviceNet uses the CAN standard on the data layer. CAN defines four types of message frames (data, remote, overload and error), DeviceNet uses mainly only the data frame. The data frame format is shown in the figure 1.4

![Data Frame Format](image)

DeviceNet requires a minimum bandwidth to transmit and package messages CIP.

2.3.6. The Network and Transports Layers

DeviceNet is a connection-based network. Each DeviceNet node is established by either an UCMM\(^3\) or a Group 2 Unconnected Port. Furthermore, these nodes can be a client, a server or both. Typical client device’s connections produce requests and consume responses. Typical server device’s perform the opposite, consume request and produce responses.

The DeviceNet telegrams uses the 11-bit CAN identifier only to identifier each one of the messages. Theses telegrams are divided into four groups. The following figure describes these groups.

---

\(^3\) UCMM (Unconnected Message Manager)
DeviceNet incorporates automatically a mechanism for detecting nodes with duplicate address because it uses a device address inside the CAN identifier field.

### 2.3.7. The Uppers Layers

DeviceNet uses the open industrial protocol CIP in the application layer, this protocol is strictly orientated to objects and it also used in ControlNet and EtherNet/IP this means it is independent from the physical medium and from the data-link layer. The CIP has two objectives:

- Transport of I/O devices control data.
- Transport of configuration and diagnosis information of the system being controlled.

The Figure 1.5 shows the CIP protocol structure in an OSI model.
Figure 1.5 – CIP protocol structure
Chapter 3

3. Hardware

The program used for the design of the prototype board is Eagle 5.7. This program is developed by Cadsoft in Germany. The main feature of this program is that it contains three modules: schematic editor, layout editor and route manually.

The schematic editor is used for designing circuit diagrams, the layout editor allows to place the components and route the wiring of the PCB, and the last module, route manually is a tool, which is used for try to route the wiring manually.

3.1. The schematic design

The function of the standard interface is the communication between an industrial robot and a NC machine. To communicate with NC machine the interface has eight optocouplers and eight relays. The optocouplers are used as inputs and the relays that are powered by 24V are used as outputs. For the communication with the robot, it uses a serial port and CAN bus. All communications are controlled by the microcontroller AT90CAN128.
3.2. Standard interface features

- **Inputs**
  - USB
  - CAN Bus
  - Power Jack
  - JTAG

- **Outputs**
  - Relays
  - RS-232

- **Other Components**
  - Bridge Rectifiers
  - Microcontroller
  - Optocouplers
  - Fuse
3.3. Functional Block Diagram

The figure 1 shows the functional block of the standard interface.

Figure 2.1 - Hardware functional block diagram
The following describes in detail the circuit of the standard interface.

3.4. Microcontroller

The microcontroller used in the standard interface is Atmel AT90CAN128, which controls the communication between NC machine and industrial robot. The choice of this microcontroller is because it supports CAN bus that is the backbone technology of DeviceNet.

The main features of the microcontroller are:

- 128Kb Flash
- 4Kb EEPROM
- 4Kb internal SRAM
- CAN controller
- 4 Timers/Counters
- 2 UARTs
- JTAG interface
- SPI serial port
- 53 general purpose I/O lines

The complete information and how it works can be found in the datasheet of the microcontroller.
Figure 2.2 - Microcontroller
3.5. Serial communication

For serial communication the integrated circuit used is MAX232. This circuit converts signals from a RS-232 to signals TTL suitable for digital logic circuits.

Figure 2.2 – Serial communication
3.6. CAN bus interface

To carry out CAN communication the interface used the high-speed CAN transceiver MCP2551. This circuit serves as the interface between the AT90CAN128 and the physical bus.

Figure 2.3 – CAN bus interface
3.7. NC-Machine communication

- **Inputs**

The standard interface has eight inputs. These inputs have a diode bridge and optocoupler. The diode bridge converts alternating current input to direct current output and the optocoupler is used because it performs the function of adapting electrical signals and provides isolation.

![Figure 2.4 – Inputs NC-Machine](image)
• **Outputs**

The standard interface has eight relays used as outputs. To control them with the microcontroller it uses a driver that provides sufficient current and it is able to absorb the autoinduction currents that are generated by the coil.

![Figure 2.5 - Driver](image-url)
The relays are powered by 24V and they are used as outputs because they allow adapting voltage and handle higher power circuits.

Figure 2.6 – Relays
3.8. Power supply

The LM3578 is a switching regulator configured as BUK converter that provides power supply to the standard interface generating tensions of 5 and 24 volts. 24 volts are used for power supply the relays and the driver and the 5 volts the rest of the integrated circuits.

![Figure 2.7 – Power supply](image-url)
3.9. USB

To carry out the communication with the computer to program the microcontroller the standard interface uses an USB port.

![USB Circuit](image)

Figure 2.8 – USB circuit

3.10. JTAG

The standard interface also has a joint test action group (JTAG). The JTAG is used to perform operation like debugging and break pointing.

![JTAG](image)

Figure 2.9 - JTAG
Chapter 4

4. Software

Tool used for testing, programming and debugging programs using C language to communicate robots and NC machines is (AVR Studio 4 + WinAVR GCC compiler). That tool is free provided by Atmel.

AVR STUDIO 4

Is the Integrated Development Environment (IDE) for developing 8-bit AVR applications in Windows environments.

AVR Studio 4 provides a complete set of features:

- Integrated Assembler
- Integrated Simulator
- Integrates with GCC compiler plug-in
- Debugger supporting run control including source and instruction-level stepping and breakpoints, registers, memory and I/O views.
- Full programming support for standalone programmers.

Programs used to check the properly working of the standard interface device can be found in Appendix A.
5. Conclusions and future works

5.1. Conclusion

The main aim of this thesis is the design and test of a standard interface that will be used to communicate NC-Machine and industrial robot and the study of the industrial protocol DeviceNet. This standard interface have been designed and implemented successfully after a long and hard work at Mälardalen University. The debugging and programming of the microcontroller for check that IO inputs as optocouplers, serial bus and relays have been made through JTAG finally because there was some kind of trouble in the communication between USB and microcontroller maybe it was because of manually soldering.

5.2. Future works

A future development of this thesis it could be the implementation and programing of this standard interface in the DeviceNet network through the bus CAN incorporated. It also would have to check the circuit of the USB port because there are some problems with the communication.
References


#include "avr/io.h"

#include <util/delay.h>

void PORT_Init()
{
    PORTA = 0b00000000;
    DDRA = 0b00000000;  //set Optocoupler as input (Bit0:7 = 0)

    PORTB = 0b00000000;
    DDRB = 0b00000000;

    PORTC = 0b00000000;
    DDRC = 0b11111111;  //set Relay as output (Bit0:7 = 1)

    PORTD = 0b00000000;
    DDRD = 0b00000000;
}
PORTF = 0b00000000;
DDRF = 0b00000000;

}

int main()
{
    PORT_Init();
    while (1)
    {
        unsigned char i;
        // Read input from PORTA.
        // This port will be connected to the 8 optocouplers
        i = PINA;

        // Send output to PORTC.
        // This port will be connected to the 8 Relays
        PORTC = i;
        _delay_ms(10);
        _delay_ms(10);
    }
    return 1;
}
/* CPU frequency */
#define F_CPU 16000000UL

/* UART baud rate */
#define UART_BAUD 9600

#include <stdio.h>
#include <avr/io.h>
#include <util/delay.h>

/*
 static void
delay_1s(void)
{
    unsigned int i;

    for (i = 0; i < 100; i++)
        _delay_ms(10);
}
*/
void PORT_Init()
{
    PORTA = 0b00000000;
    DDRA = 0b00000000;
    
    PORTB = 0b00000000;
    DDRB = 0b00000000;
    
    PORTC = 0b00000000;
    DDRC = 0b00000000;
    
    PORTD = 0b00000000;
    DDRD = 0b00001000;  //TX set as output (Bit1 = 1)
    
    PORTE = 0b00000000;
    DDRE = 0b00000000;
    
    PORTF = 0b00000000;
    DDRF = 0b00000000;
}

void

uart_init(void)
{
#if F_CPU < 20000000UL && defined(U2X)
    UCSR1A = _BV(U2X); /* improve baud rate error by using 2x clk */
    UBRR1L = (F_CPU / (8UL * UART_BAUD)) - 1;
#else
    UBRR1L = (F_CPU / (16UL * UART_BAUD)) - 1;
#endif
    UCSR1B = _BV(TXEN) | _BV(RXEN); /* tx/rx enable */
}

unsigned char USART_send(unsigned char data){

    // wait until UDRE flag is set to logic 1
    while ((UCSR1A & (1<<UDRE1)) == 0x00){;}

    UDR1 = data; // Write character to UDR for transmission
    return data;
}

unsigned char USART1_Receive(void)
{

/* Wait for data to be received */

    while ( !(UCSR1A & (1<<RXC1)));

/* Get and return received data from buffer */

return UDR1;

}

int

main(void)

{

    //unsigned int i;
    unsigned char dato;
    PORT_Init();
    uart_init();

    while (1) {

        dato = USART1_Receive();

        // _delay_ms(10);

        //fflush(stdin);


USART_send('[');
// _delay_ms(10);
// fflush(stdin);
USART_send(dato);
// _delay_ms(10);
// fflush(stdin);
USART_send(']');

_delay_ms(10);
fflush(stdin);

// rotate right 10 times
/*
 for (i = 0; i < 10; i++)
 {
     USART_send('6');
     _delay_ms(10);
     //delay_1s();
     }*/
}

return 0;