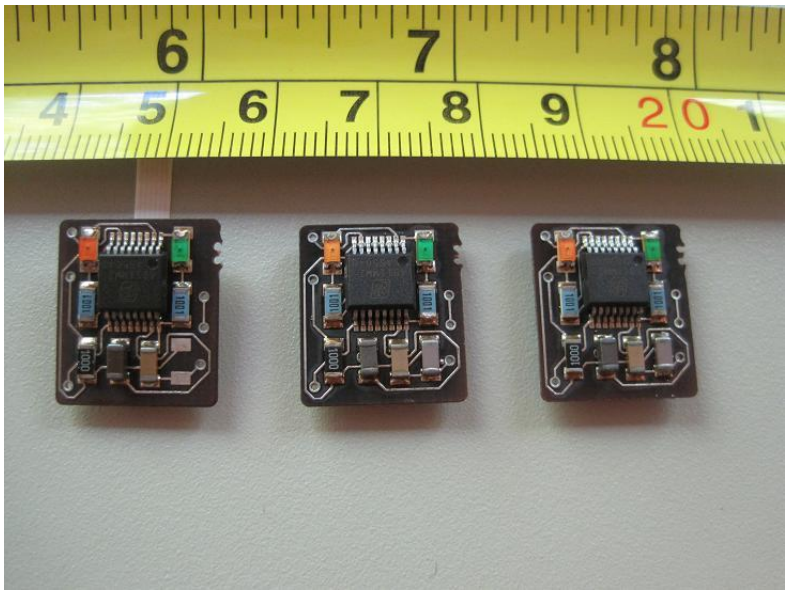
ROBOT HAND CONTROL

IMPLEMENTATION OF SERIAL INTERFACE

Thesis work of Toru Kizaki

4/10/2010



Supervisors : PhD. Johan Tegin, KTH Machine Design, Mechatronics Lab.
Professor. Jan Wikander KTH Machine Design, Mechatronics Lab.

Acknowledgement

I appreciate the great support and help of Johan Tegin during this thesis work. Due to his advises my work had been done well. Beside this project work, he also gave me various advises for the life in Stockholm.

I thank Afifa Lahatulain for daily cooperation and sharing various informations.

I thank Bengt Eriksson for updating Matlab and dSPACE software to latest ones.

I thank Mikael Hellgren for helping when the electric power supply of the lab was stopped.

I thank Staffan Qvarnström for introducing of the soldering machine and ordering various components.

I thank William Sandqvist for teaching me regarding PIC programming.

In the end, I thank all the people who helped and supported me.

07-10-2010 Toru Kizaki

Contents

1 Introduction

1.1 Motivation

In these days, various domestic robots are studied. Among the various parts of the domestic robot, hand is one of the parts which have been studied and developed by many researchers. Mainly the research regarding robot hand is focusing on how to control hand versatilely. The example is the work in the Columbia University. Matei has been worked on the method of hand controlling. In order to make a dimensionality of the grasping as small as possible, he proposed the grasping method using "Eigengrasps" [1].

In this research work, the authors focused on how to make human-like-hand which can be applied to nursing care robots. For nursing care robots, the versatility, lightness and low-cost are required. Each finger of the hand is moved by a tendon, and the tendon is pulled by the motor embedded in palm. Because of this, the hand acquired higher versatility than the hand which has motors at each finger joint. And the lightness has been achieved by using DuraForm® PA plastic [2]. Low-cost is also achieved by using mass-produced electrical or mechanical components.

1.2 Specification of the system of the current hand

Figure 1.1 and Figure 1.2 show the current hands. Figure 1.3 shows inside of the hand. Each finger is driven by the geared motor via tendon. The rotational angle of the motor is obtained by the magnetic encoder placed vertically to the central axis of the motor shaft. The angular position data had been transferred via PWM interface from the magnetic encoder to the dSPACE© which is the interface board embedded in the computer.

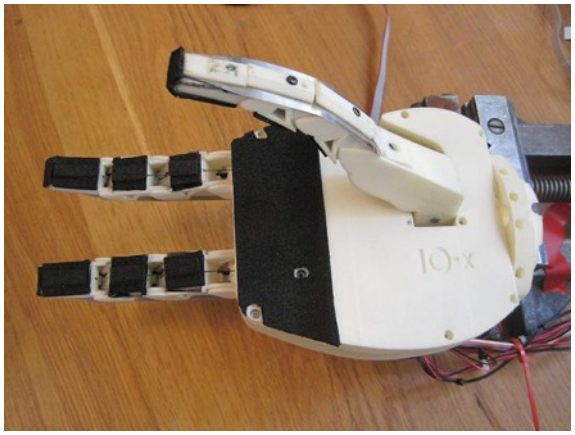


Figure 1.1 Previous hand prototype

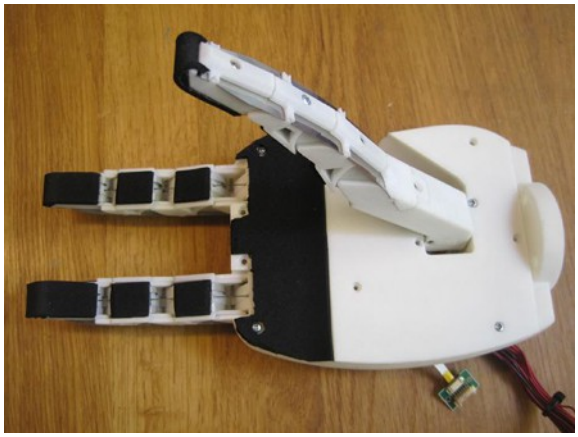


Figure 1.2 New hand prototype [3]

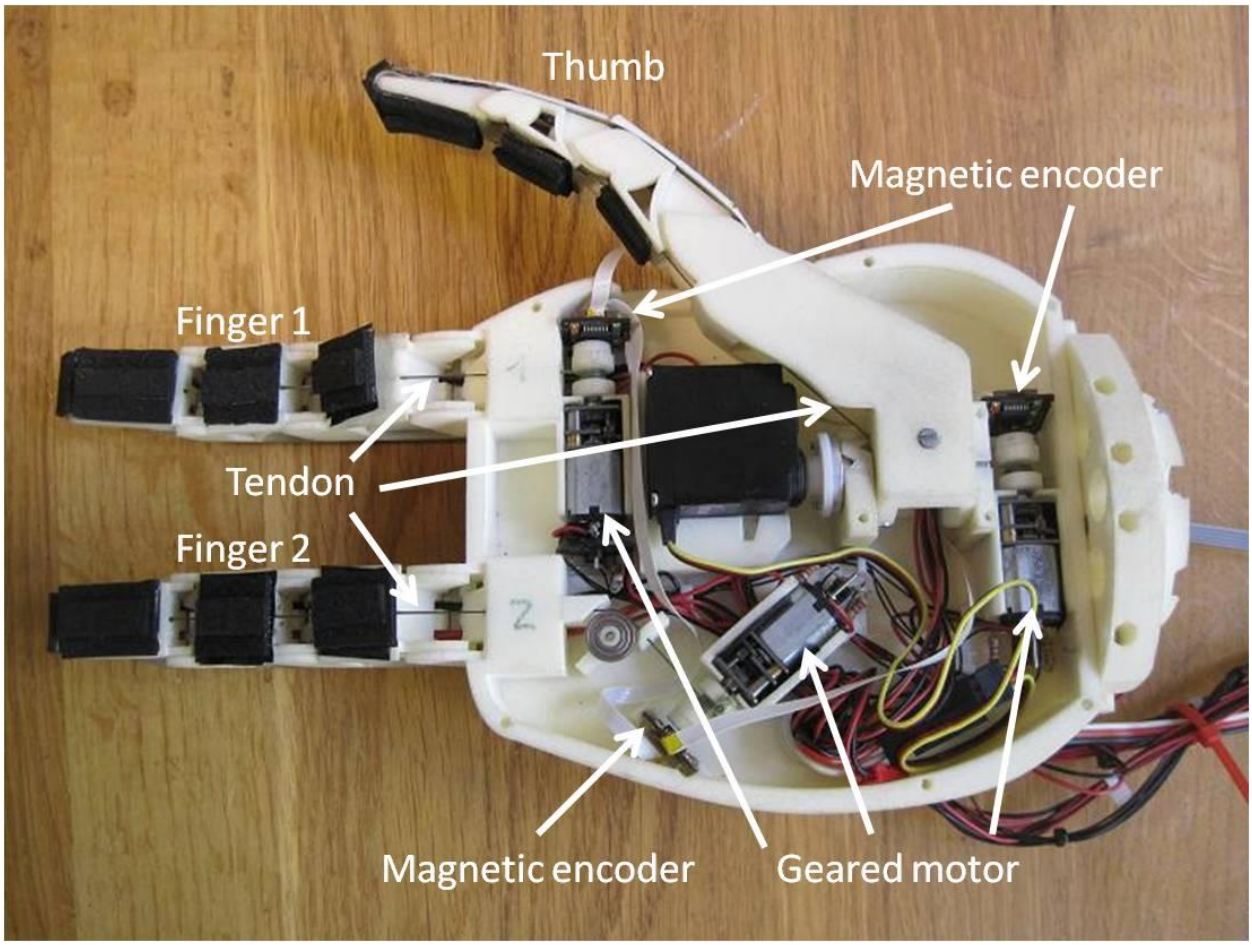


Figure 1.3 Inside of the previous hand

1.3 Problem regarding the interface between the magnetic encoder and the computer.

In the previous work, the fault of the PWM interface was revealed, therefore there is a "Dead Band" at dSPACE© which means the area of the duty cycle below 0.05 and above 0.96. The PWM signal whose duty cycle is in dead band is not reliable because of the jumping or fluctuation of the signal.

In order to make hand work smoothly, avoidance of data fluctuation is desired. The cause of fluctuation is the disability of dSPACE© in receiving PWM signal of near 1 or 0 duty cycle, therefore of dead band. So, PWM interface was needed to be replaced by the serial interface [4].

2 Objectives

The objective of this work is to implement the serial interface for the transferring of the angular position data from the magnetic encoders to the dSPACE©.

More concretely, the serial interface is implemented by using a micro controller as a relay device in between the magnetic encoders and dSPACE©. The data is transferred via synchronous serial interface from the magnetic encoders to a micro controller, and via asynchronous serial interface from a micro controller to the dSPACE©.

3 Communicating components

3.1 AS5040

AS5040[1] is a contactless magnetic rotary encoder. The chip measures the angular position of the two-pole magnet rotating over the center of the chip. The typical arrangement is shown in Figure 3.1. The pin configuration is described in Figure 3.2.

The resolution of the angle is 0.35 degree[2]. This data is output in both a PWM signal and a serial bit stream. The duration of the PWM signal is 1 ms, therefore the sampling frequency is 1000 Hz. The maximum read-out frequency of a serial bit stream is 1 MHz. One cycle of the data consists of 16 bits, so the sampling rate of the angular position data is 62.5 kHz. AS5040 has the Daisy Chain mode in which the angular position data from multiple AS5040 devices can be transferred through one serial line [5].

3.2 dSPACE©

dSPACE© is an interface board placed in between a computer and other devices. The board which is used in this work is "DS1104 R&D Controller Board". This board has A/D, D/A converters, 20 bit digital I/O, serial interface[3], and PWM ports.

"DS1104 R&D Controller Board" is a real-time hardware, and with Real-Time Interface (RTI) the code is generated from Simulink models via Real-Time Workshop[6].

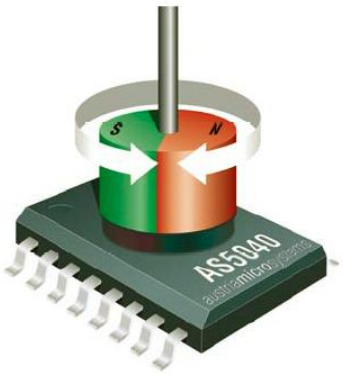


Figure 3.1 Magnetic encoder "AS5040" with magnet [5]

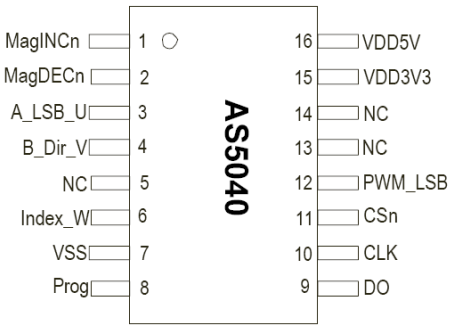


Figure 3.2 Pin configuration of AS5040 [5]

[1] AS5040 is produced by "austriamicrosystems AG.".

[2] 1024 positions per revolution.

[3] dSPACE© has RS232, RS422, RS485 ports.

Figure 3.3 dSPACE© DS1104 R&D Controller Board [6]

4 The problem regarding direct communication

In the previous work, the angular position data from the AS5040 magnetic encoders are transferred by using PWM signal. However, as referred in 1.3, dSPACE© has the "Dead Band" that means the band of the duty cycle, less than 0.05 and more than 0.96. In this "Dead Band" fluctuation or jumping of the received data occurs.

So, it is proposed to use serial interface instead of using PWM. Basically, serial interface has no "Dead Band". AS5040 has synchronous serial communication function, however dSPACE© has no synchronous serial interface. Synchronous serial interface is a kind of interface requiring clock signal for communication. So, direct communication between AS5040 and dSPACE© is not feasible.

On the other hand, the dSPACE© has several asynchronous serial interfaces such as RS232. That is a kind of interface requiring no clock signal.

There are three options of the measure to solve this problem.

1. To change AS5040 to another component which has the asynchronous serial interface.
2. To change dSPACE© to another device which has the synchronous serial interface.
3. To insert a relay component which translate synchronous serial data from AS5040 into asynchronous serial data for dSPACE©.

In order to choose one of these three options, the decision matrix was made. This matrix is shown in Table 4.1. There are five criteria. The most important criteria are "Reliabilty of the Communication" and "Speed of the Communication".

"Using relay component" is inferior to other two options in Reliability.

In Speed, “changing dSPACE©” is the best, the score of “Using relay component” is the lowest because the delay of the time occurs while translating.
In Complexity, “Changing dSPACE©” is the worst because changing dSPACE© means rewriting the whole code for the controlling of the hand.
In Cost, “Using relay component” is the best. Translation does not require complicated system, so a kind of small micro controller will be enough for this purpose.
In Adaptability, Option 3 is the best. The relay component will be easily adapted to some changes of both AS5040 and dSPACE©.

By summing up each scores multiplied by Weight, it is found that “Using relay component” is the most appropriate.
So, It is decided to use the relay component to convert the synchronous serial data from AS5040 into the asynchronous serial data for dSPACE©.

Table 4.1 Decision matrix to decide how to implement the serial interface

	Reliability of the Communication	Speed of the Communication	Complexity	Cost	Adaptability for Future Improvement	Sum
Changing AS5040	8	6	6	3	7	**220**
Changing dSPACE©	8	10	1	1	1	**195**
Using relay component	6	4	7	9	9	**225**
Weight	10	10	5	5	5	

5 The relay component

5.1 Reqirement of the relay component

The requirements for the relay component is

1. Implementation of the translation of the data is feasible.
2. The speed of transferring of the data is 1 ms per an angular position data.

In order to fulfill Requirement 1, the relay component has to be a programmable component such as a micro controller. For Requirement 2, the frequency of the clock signal of the component must be high enough to transfer a data within 1 ms. However, exact time is not clear until when the system is implemented.

5.2 Selection of the relay component

A microcontroller is suitable for the relay component. They vary from 8-bit one to 32-bit one. However, the translation of the data is comparatively simple, so the simple component is suitable for the purpose.

PIC[4] is a programmable microcontroller produced by Microchip Technology Inc. It has simplicity and there is an integrated developing editor for this chip.

In the PIC series, PIC16F690 has the function of the serial interface[5] and the maximum frequency of the clock is 20 MHz[6]. This chip fulfills the requirement written in 5.1. PIC16F690 was determined to be used for the translation of the data from AS5040 to dSPACE©.

5.3 PIC16F690

PIC16F690 is a 20-Pin Flash-Based 8-bit microcontroller which is produced by Microchip Technology Inc. PIC16F690 is shown in Figure 5.1. The configuration of the pins is shown in Figure 5.2. The frequency of the internal clock is 8 MHz, and the frequency can be raised up to 20 MHz by using external clock source. This chip has the function of EUSART[7], therefore serial interface can be implemented on this chip easily [7].

---

[4] Peripheral Interface Controller

[5] EUSART

[6] Requiring external oscillator such as crystal oscillator

[7] Enhanced Universal Synchronous Asynchronous Transmitter

This component can be programmed by using computer. The MPLAB IDE is the integrated development editor distributed by Microchip Technology Inc. The interface between PIC16F690 and a computer is provided by PICkit2. The code written in MPLAB IDE is translated to machine language and transferred to PIC16F690 through the PICkit2.

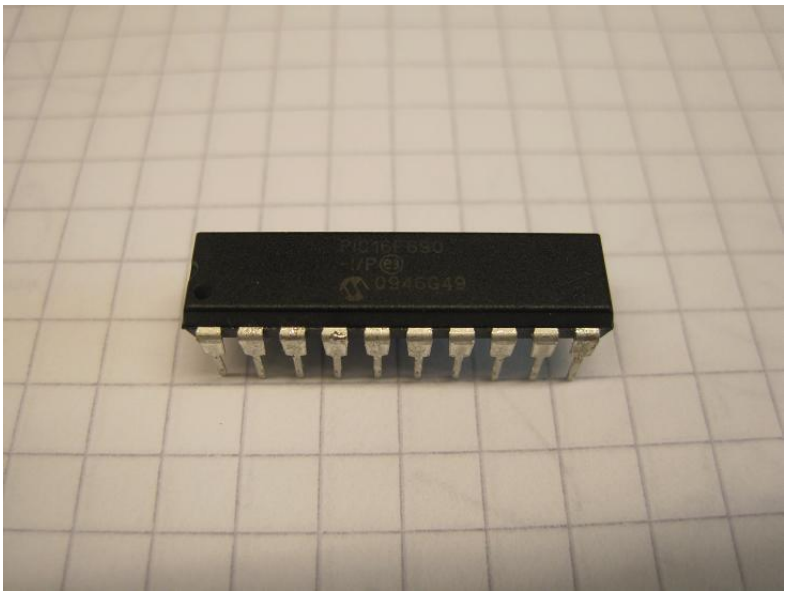


Figure 5.1 PIC16F690

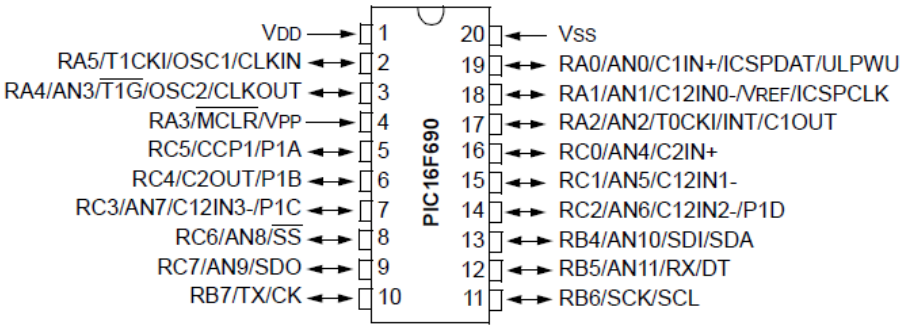


Figure 5.2 Pin configuration of PIC16F690 [7]

6 The serial interface

6.1 Communication between AS5040 and PIC16F690

6.1.1 The hardware and the software for the development

The hardware for the development consists of three components, the AS5040 on the testing board, and the geared motor[8] with the magnet attached on the top of the shaft and the PIC16F690 on thePICkit2 demo board. The geared motor is fixed on AS5040 testing board by using a jig, so that the magnet on the motor can always stay within the X-Y-Z tolerance limits of the AS5040. PICkit2 is the interface device between a computer and a PIC, and used when PIC is programmed. Figure 6.1 shows the geared motor, Figure 6.2 shows the AS5040 on the testing board, Figure 6.3 shows the geared motor and AS5040 testing board connected by the jig. Figure 6.4 shows the PIC16F690 on the PIKkit2 demo board. The simplified schematic diagram is shown in Figure 6.5.

The PIC16F690 was coded in C. The C code was written on MPLAB IDE v8.53, and MPLAB IDE v8.53 is the Integrated Developing Editor distributed for free by Microchip Technology Inc. The compiler of the C code was CC5X distributed by B Knudsen Data, Norway. Those software which was used for coding are summarized in Table. 6.1

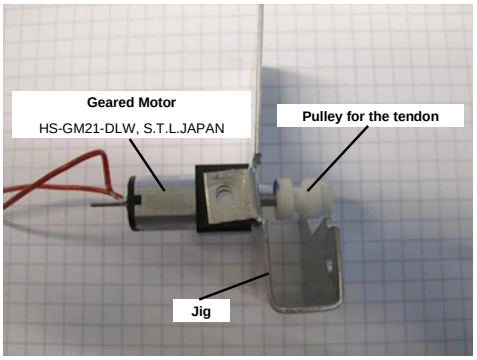


Figure 6.1 Geared motor

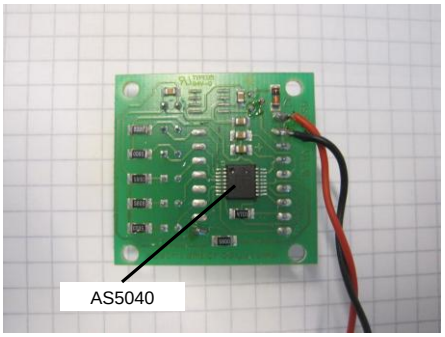


Figure 6.2 AS5040 on the testing board

[8] HS-GM21-DLW, S.T.L Japan

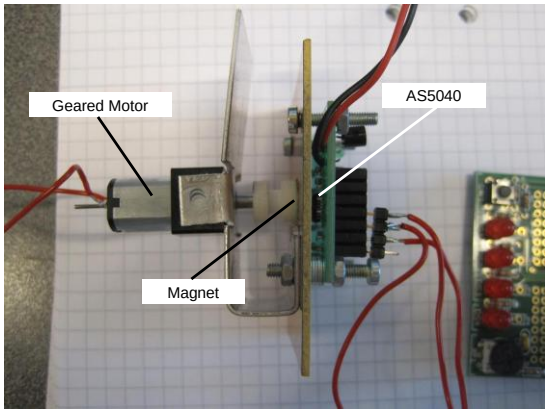


Figure 6.3 Geared motor and AS5040 testing board

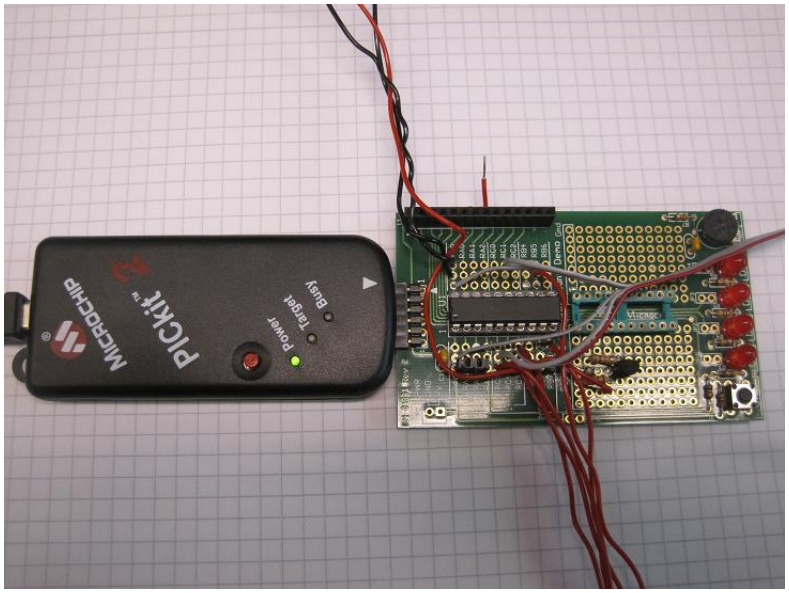


Figure 6.4 PIC16F690 on PICkit2 demo board

Table. 6.1 The software used for coding

The name of the software	summary	Distributor
MPLAB IDE v8.53	Integrated Developing Editor	Microchip Technology Inc, USA
CC5X	C compiler	B Knudsen Data, Norway

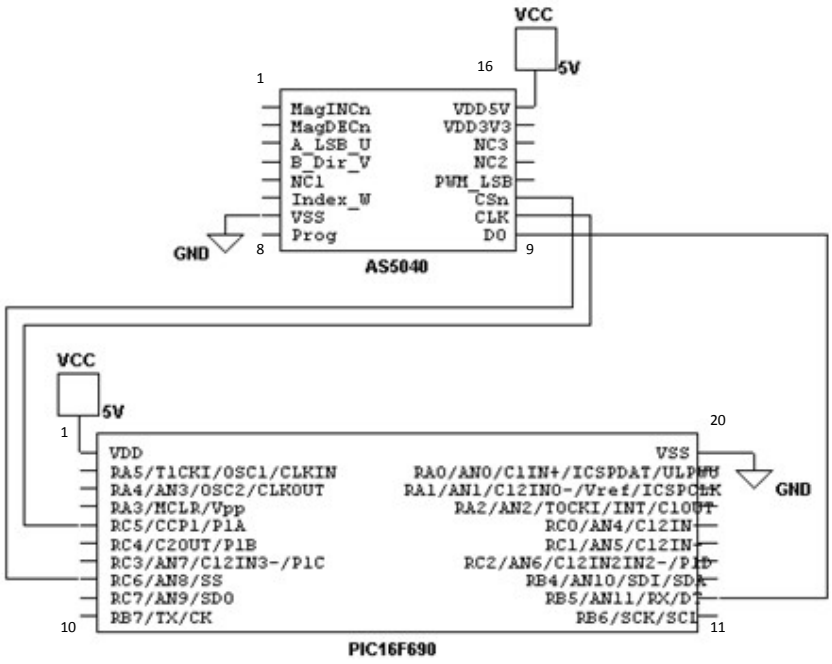


Figure 6.5 Schematic diagram of the connection between AS5040 and PIC16F690

6.1.2 The Structure and the Flow of the data

Magnetic encoder AS5040 has the Synchronous Serial Interface (SSI), and transmits a data by 16 bits. 16bits consist of 10 bits of the angular position data, and 6bits of the status bits. Figure 6.6 describes the schematic of the SSI of the AS5040. The resolution of the angular position data is shown below.

$$\frac{360}{2^{10}} \cong 0.35deg \qquad (1)$$

The SSI of the AS5040 is synchronized with the external clock signal, therefore the speed of the transmission depends on the frequency of the external clock signal. The maximum tolerable frequency of the external clock signal is 1 MHz. At the end of the one cycle of the transmission a pulse signal is also required at CSn pin of the AS5040.

The three lines are necessary for the transmission. One is the line for the data, and one is for the clock signal, the last one is for the CSn signal.

PIC16F690 can deal with only 8bits data, so the 16bits from the AS5040 must be split into two 8bits data in the PIC16F690. Therefore, the receptions at the PIC16F690 are done

twice for one cycle of the data from the AS5040. Figure 6.8 describes the schematic of this flow of the data.

6.1.3 Implementation of the SSI on PIC16F690

The algorithm of the reception mainly consists of transmitting the clock pulse, and reading the one bit of the data. This algorithm is described in Figure 6.7 as a flowchart. This algorithm is implemented in the function "serial_rec_mag (char* serial_upper, char* serial_lower)".

Whole source code is attached on Appendix.

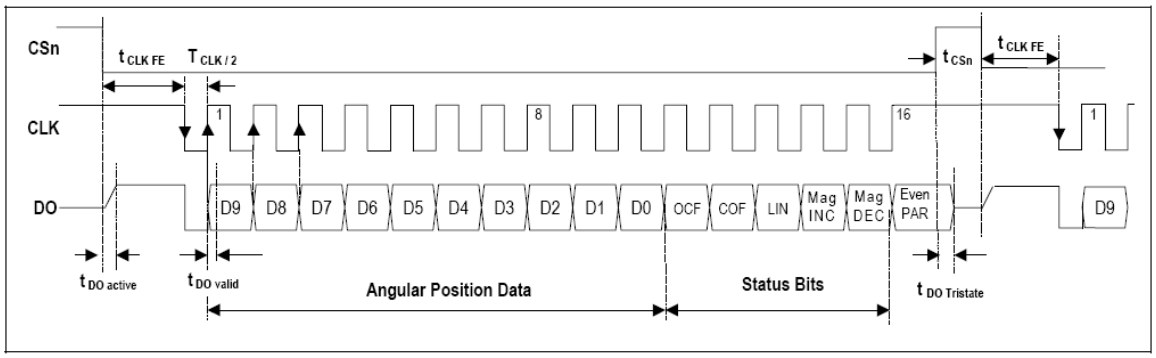


Figure 6.6 SSI with absolute angular position data

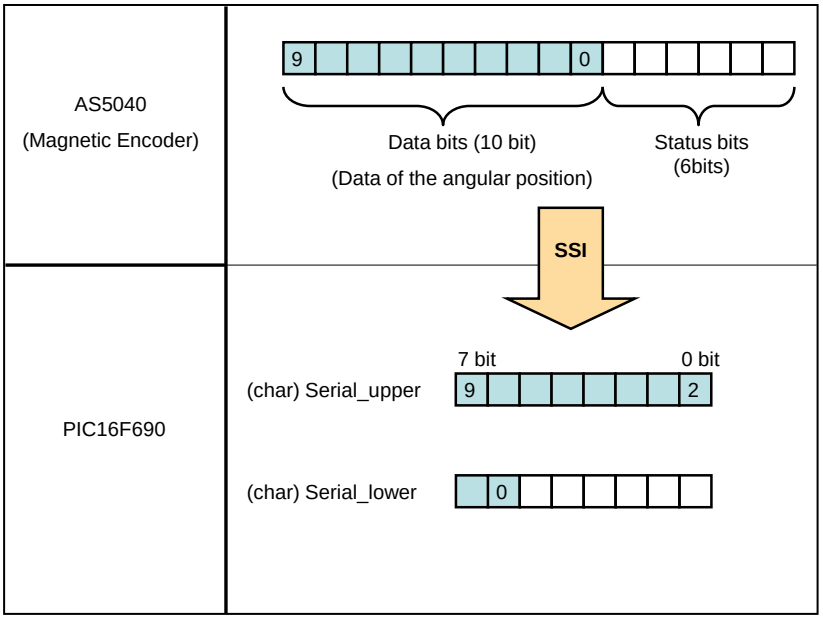


Figure 6.8 Schematic of the data flow

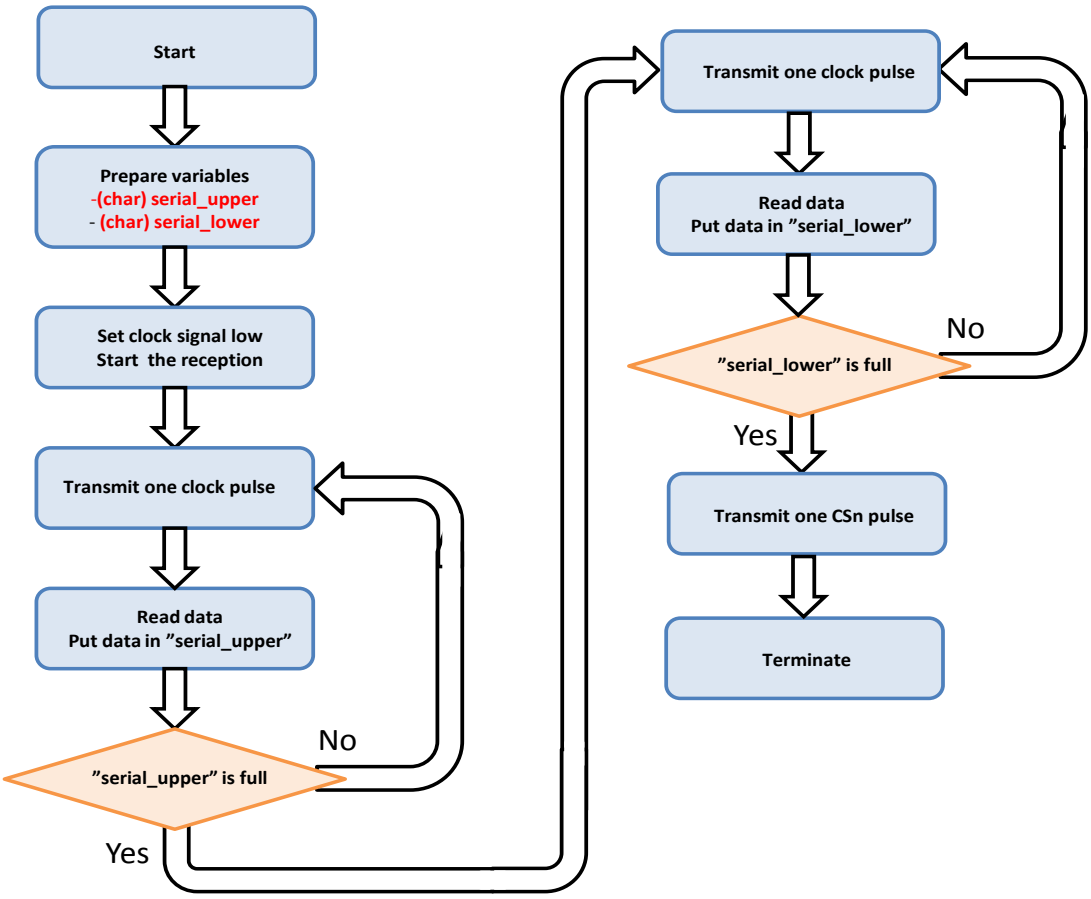


Figure 6.7 Flowchart of the SSI reception

6.2 Communication between PIC16F690 and dSPACE©

6.2.1 The hardware and the software for the development

The hardware is PIC16F690 and dSPACE©. The dSPACE© has a port for the asynchronous serial interface (RS232) and PIC16F690 also has the function for the asynchronous serial interface called EUSART. So, the RS232 communication between these two devices is feasible.

The connection between PIC16F690 and dSPACE© is shown in the simplified schematic diagram in Figure 6.9. There is an inverter circuit between PIC16F690 and dSPACE©. The reason of that is explained in 6.2.2.

The software is MPLAB IDE v8.53, CC5X, MATLAB Simulink, and dSPACE Control Desk. MPLAB IDE v8.53 and CC5X were used in coding of the RS232 transmission of the PIC16F690. MATLAB Simulink was used in implementing the function of the RS232 reception of dSPACE©. dSPACE Control Desk was used as a graphical user interface of the dSPACE©. Those software were summarized in Table 6.1

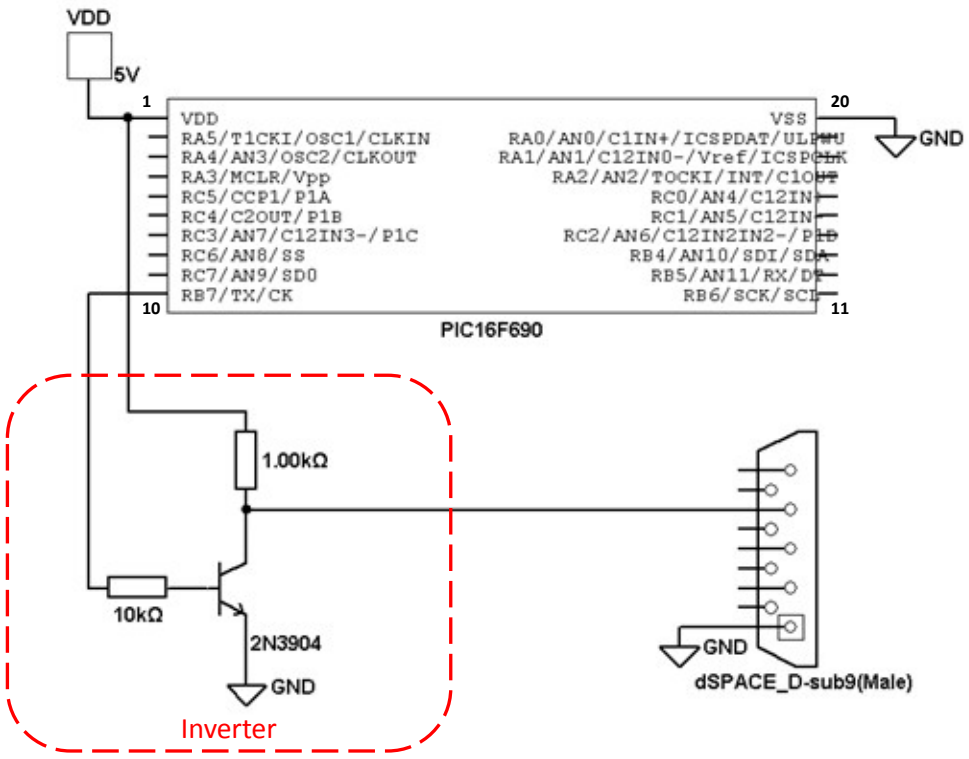


Figure 6.9 The schematic diagram of the connection between PIC16F690 and dSPACE©

Table 6.1 The software

The name of the software	Summary	Distributor
MPLAB IDE v8.53	Integrated Developing Editor	Microchip Technology Inc, USA
CC5X	C compiler	B Knudsen Data, Norway
MATLAB Simulink	Tool for Model-Based designing of real time system	The MathWorks. Inc, USA
dSPACE Control Desk	GUI of the dSPACE	dSPACE GmbH, Germany

6.2.2 The structure and the flow of the data

RS232 is one type of the asynchronous serial interface, and usually uses three lines, one for transmission, one for reception, the last one for sharing ground level. However, in this developed system, transferring of the data is one-way traffic, from PIC16F690 to dSPACE©, so only two lines are necessary. Currently, D-sub9 connector is usually used for RS232 communication. The configuration of the pins of the D-sub9 connector is shown in Figure 6.10.

The asynchronous serial interface does not require the clock signal. Instead of using clock, the timing of transmitting or receiving is set in both devices as the baud rate. Therefore, the baud rate set in both devices need to be the same value as that of each other, otherwise the communication would fail. The unit of the baud rate is "baud".

In addition, the start bit and the stop bit are used to synchronize each other. Usually start bit is LOW "0", and the stop bit is HIGH "1" in RS232 communication. However, in dSPACE©, the definition of the start bit and stop bit are different from that. The definition of Start bit and Stop bit are unchangeable by using codes in both devices. That is the reason of inverting the logic of the data in between PIC16F690 and dSPACE© by using inverter circuit.

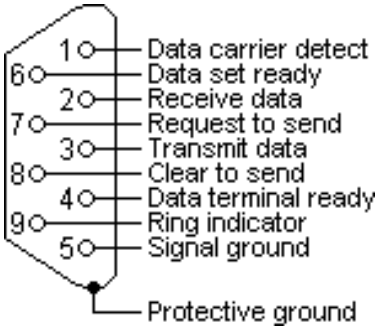


Figure 6.10 D-sub9 (Male) pinout [8]

As the characteristic of RS232 communication, the logic of the transmitted data is inverted. For example, the data "11000011" will be transmitted after inversion, then received as "00111100".

EUSART function of the PIC16F690 could do serial communication almost automatically with only a few initializations. Figure 6.11 describes the transmission sequence of the PIC16F690 by using EUSART. After

putting the data in the 8bit register called TXREG, that data is automatically transmitted by PIC16F690.

The data from the AS5040 is 16bits, and stored in two 8 bits variables in PIC16F690. So, two transmissions are required for one transmission of 16bits data. The data flow is shown in Figure 6.12.

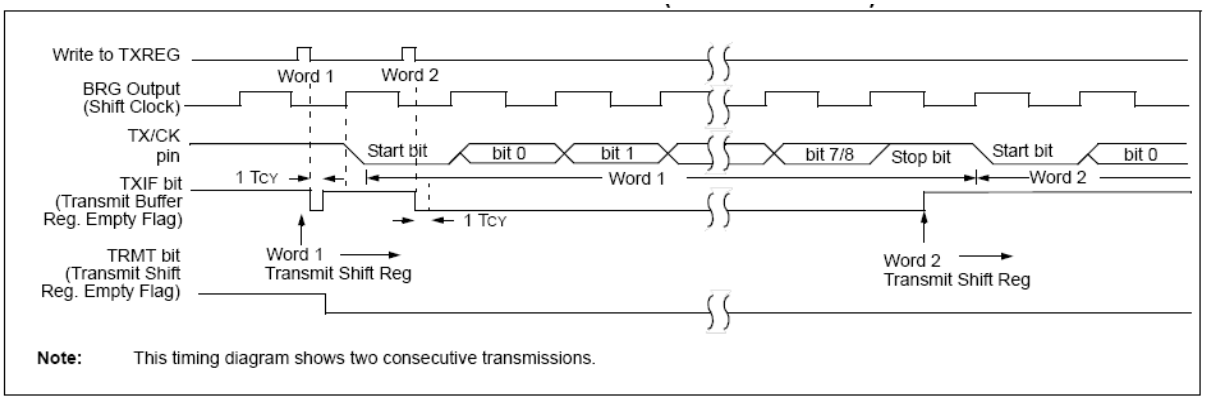


Figure 6.11 Sequence of the asynchronous serial transmission of EUSART of the PIC16F690

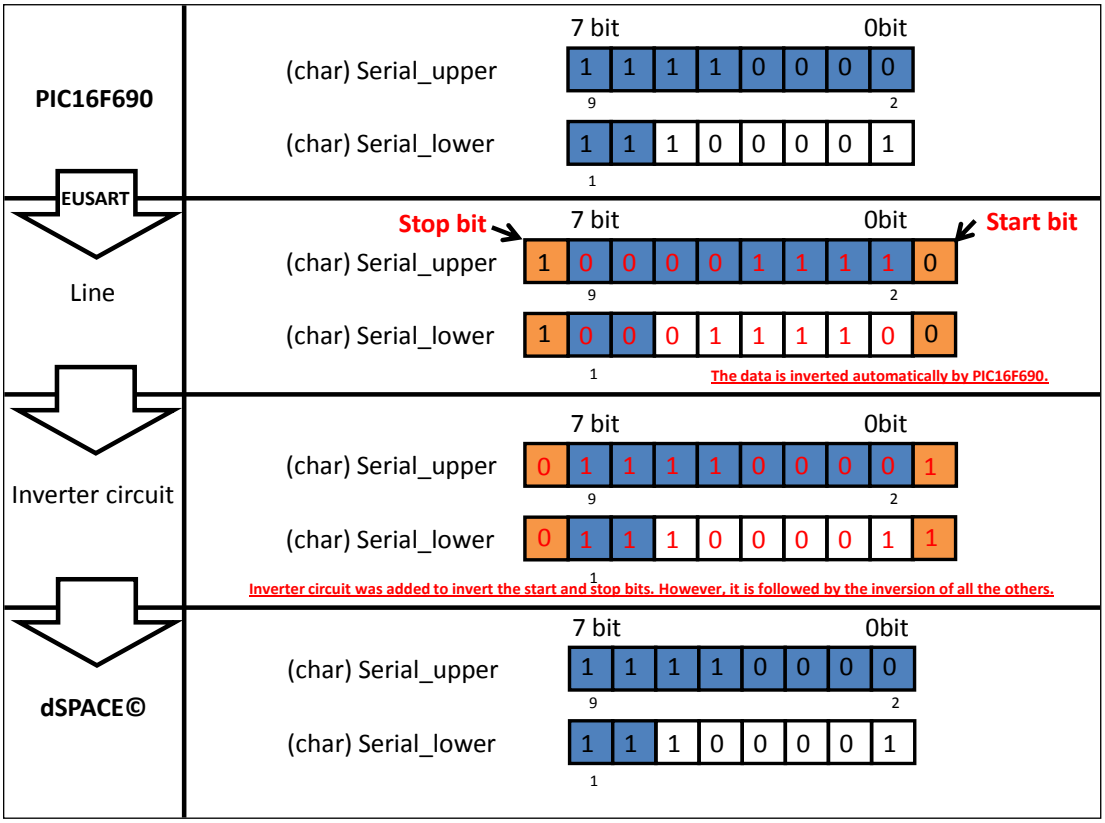


Figure 6.12 The data flow between PIC16F690 and dSPACE©

6.2.3 Implementation of the asynchronous serial interface on both PIC16F690 and dSPACE

The code for the serial reception was also written in C.

The algorithm is implemented in the function “serial_tran_rs232”. In this algorithm, first, the configuration is initialized. Setting of the baud rate is included in the initialization. The baud rate was set 55556 baud in this experiment, and to set this value, the corresponding value must be set in BRGH and SPBRGH and SPBRG register. Second, the data is transmitted. Transmission is started immediately after the data has been put in TXREG register. The flowchart of this algorithm is shown in Figure 6.14.

The RS232 reception of dSPACE© was coded by using MATLAB Simulink. The model for the RS232 reception has already been implemented in the library of the Simulink. Although the implementation of the RS232 reception is simple if this model is used, the initialization such as the setting of the baud rate is still required. Especially, the baud rate must be set to the same value as that of the configuration of the PIC16F690, therefore 55556 baud.

After the reception, the split data was combined again, this is the finish of the data transferring from the AS5040(magnetic encoder) to the dSPACE©. This algorithm of the RS232 reception is shown as a flowchart in Figure 6.13.

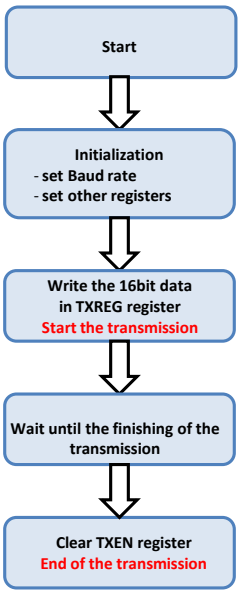
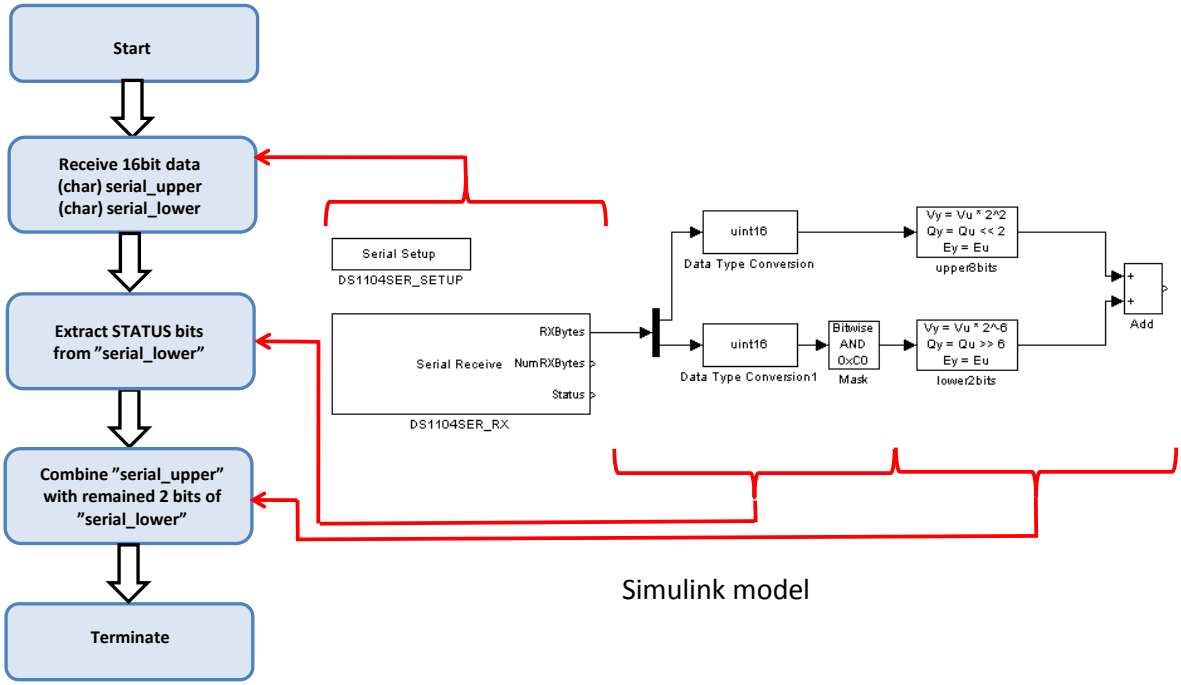


Figure 6.14 Flowchart of the RS232 transmission of the PIC16F690



Simulink model

Figure 6.13 Flowchart of the RS232 reception of the dSPACE© with the Simulink model

6.3 Data transferring from three magnetic encoders

6.3.1 Daisy Chain mode

The current hand has three fingers. Each finger has a magnetic encoder. In order to obtain rotational angle of the motor in each finger, the implementation of the reception from three magnetic encoders is necessary.

AS5040 has "Daisy Chain Mode". By using this mode, the serial reception from three magnetic encoders with one data line becomes feasible, and the data is transferred one after the other. How to connect AS5040 and MCU (PIC16F690) is shown in Figure 6.15. The schematic of data transfer is shown in Figure 6.16.

Beside the implementation of the Daisy Chain Mode, It is also necessary to re-design the circuit board for the AS5040, because the previous one was made only for PWM transmission.

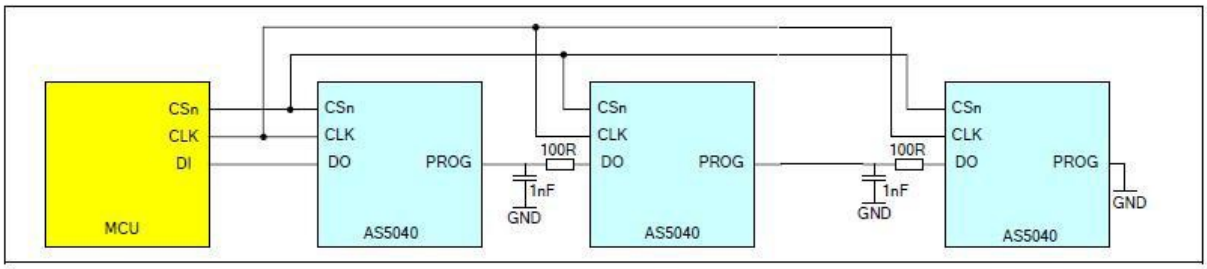


Figure 6.15 How to connect AS5040 and MCU

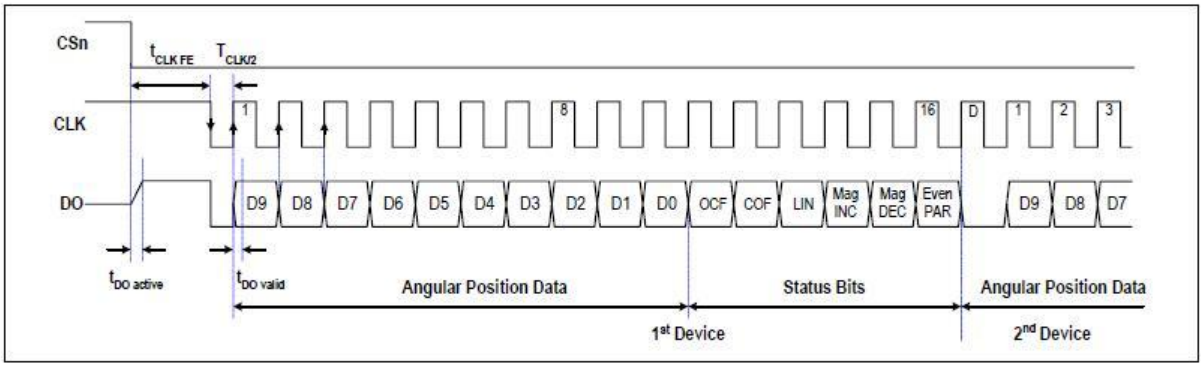


Figure 6.16 Schematic of Daisy Chain mode

6.3.2 Data transferring in correct order

The angular position data from one encoder is sent to dSPACE© in two 8-bit data. In case of the reception from three encoders, six 8-bit data is sent from encoder to dSPACE©. Data is transferred one after the other, so the confusion of the receiving order may occur. To solve this problem, sending control byte such as “00000000” at first is one option. However, this control byte has to be unique and have not to be the same value as any other data bytes. So it was necessary to make the flag-exclusive bit in each byte. So, a flag bit which indicate the order of the data was attached to each byte.

Figure 6.17 describes how the flag bit is attached to each data. First, the bits of the data are shifted rightward by a bit, then the flag is attached to the 7th bit of the data. The lapping bits which had been 0th bit before bit shifting are collected and put into “0bit_of_each_data” variable. Therefore, 7 bytes are transmitted in each cycle.

As a first step of the reception sequence at dSPACE©, the data which has ´1´ as the flag bit is searched for. After the data with ´1´ flag has been received, following 6 bytes are received. Confusion of the reception order was avoided by using this flag bit.

After dSPACE© has received all the 7 bytes from PIC, the data is decoded.

Figure 6.17 Flag attachment at PIC16F690

6.3.3 Implementation into the KTHand

In order to implement this daisy chain system into KTHand, new circuit board for AS5040 was designed. The schematic diagram of the circuit is shown in Figure 6.18, and its foot print is shown in Figure 6.19 and Figure 6.20.

Made circuit board is shown in Figure 6.21, and the boards embedded into the hand are shown in Figure 6.22, Figure 6.23 and Figure 6.24.

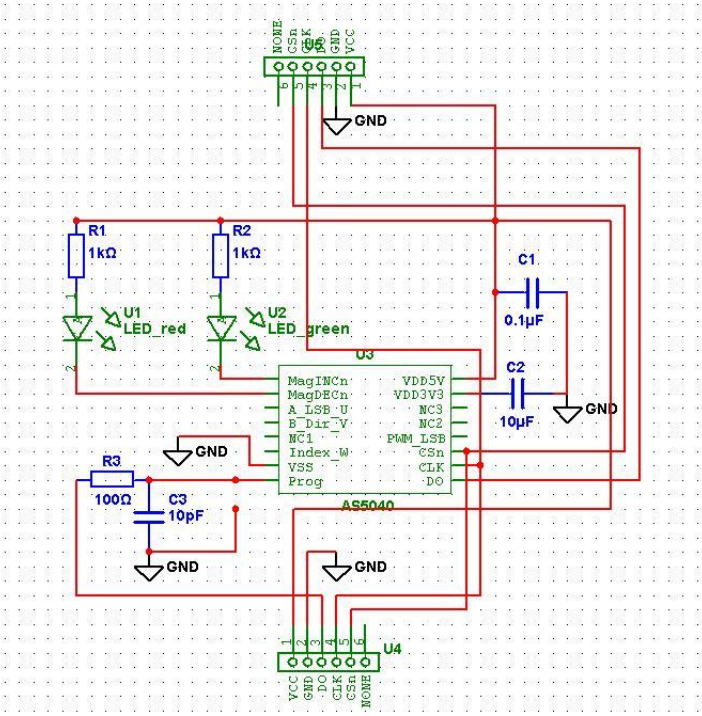


Figure 6.18 Schematic diagram of the new board

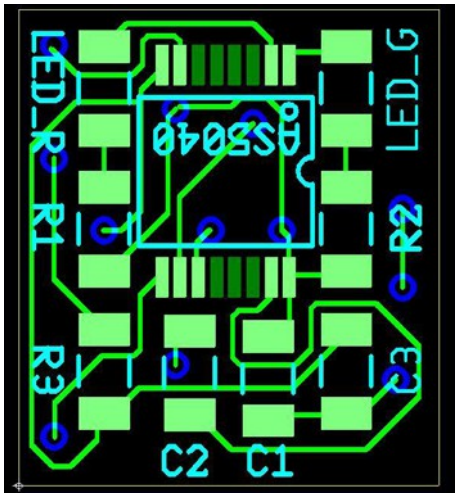


Figure 6.19 circuit board top

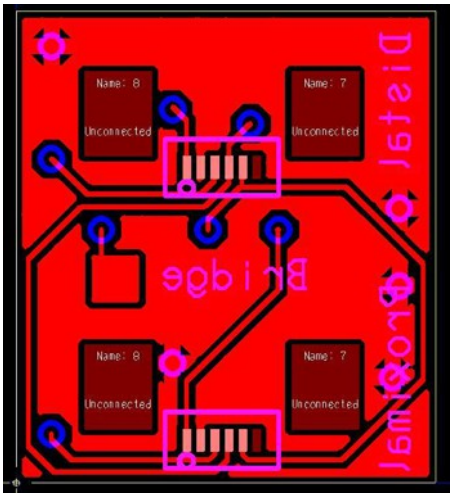


Figure 6.20 circuit board bottom

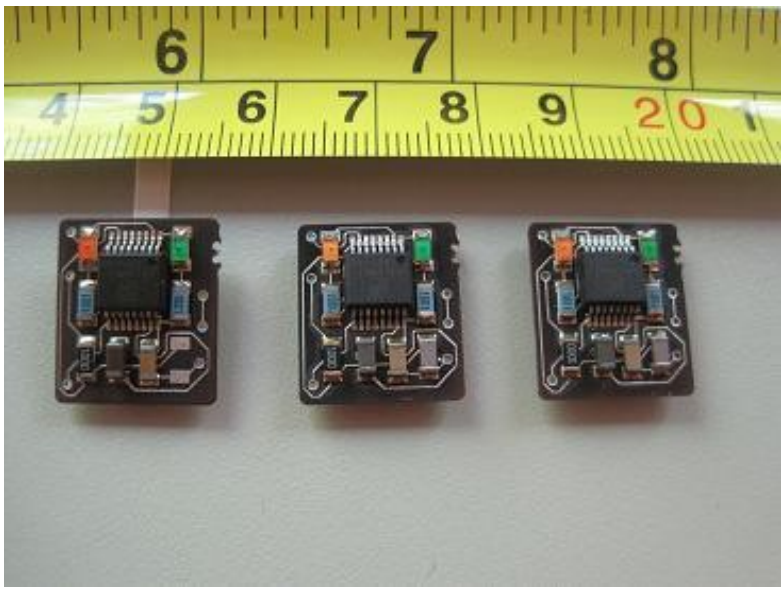


Figure 6.21 AS5040 circuit board

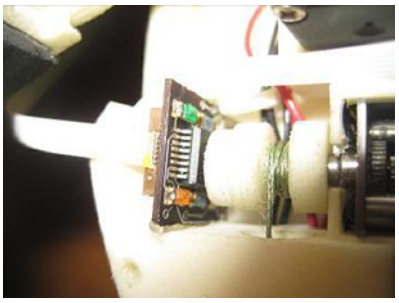


Figure 6.22 Encoder for finger1

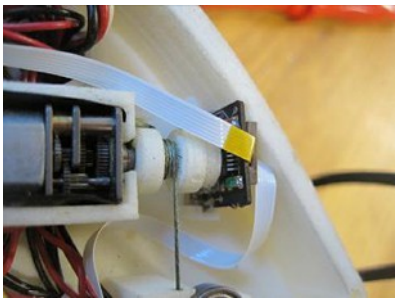


Figure 6.23 Encoder for finger2

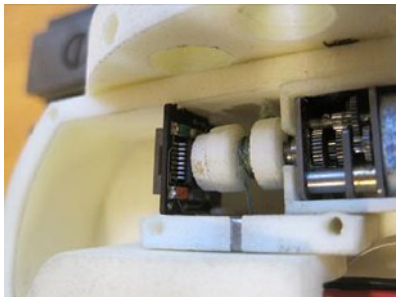


Figure 6.24 Encoder for Thumb

6.4 Performance of the system

6.4.1 The dead band

It is known that there is a dead band of the angular position in using PWM signal (below 0.05 and above 0.96 duty cycle). In the dead band, the data fluctuates and jumps, so the reliability of the received data is extremely low.

Basically, when the serial interface is used, there cannot be any dead band. To confirm this, the author conducted an experiment to acquire the data near 0 degree, 180 degree and 360 degree. The data acquisition was conducted for 15 second for each angular position. The result is shown in Figure 6.25. The angular position data is represented by integer of 0-1024, because they are transferred in 10 bits. The mean and the standard deviation at each angle were shown in Table 6.2Table 6.2. Standard deviation of "0 degree" and "359 degree" are as small as that of "174 degree". So, the data fluctuation in dead band was avoided.

6.4.2 The sampling time

The data is transferred from AS5040 to dSPACE© through PIC. In using PWM signal, the signal period is 1025 μs. Therefore, the sampling frequency of the dSPACE© was 975.6 ≈ 1000 Hz.

As discussed in Chapter 2, the sampling time should be the same in using serial interface than in using PWM.

The bottleneck of the sampling time is PIC16F690. In order to estimate the sampling time, the assembly code generated by the CC5X compiler during the compiling of the C codes was used. The length of the time which it takes each operation of the assembly language is already known. The total time can be estimated by summing up the time of all the operations in the codes.

エラー! 参照元が見つかりません。 shows the estimated time and measured time in using 8 MHz as an OSC of the PIC16F690 and 55556 baud as the baud rate. The measurement of the time was conducted by using oscilloscope. In the table the times for each function of the C codes are shown, and the whole time is also shown.

In the column of "time for one AS5040", the error of the estimated time is 15.8 %. So, the estimation is valid.

In the column of "time for three AS5040", the estimated time is 2098 μs, therefore the sampling frequency will be 477 Hz. It is just half of the frequency in using PWM (1000 Hz). Actual time was also 2000 ms. Actually, the hand has worked correctly with this sampling

frequency. However the improvement of the sampling frequency has remained as a futurework.

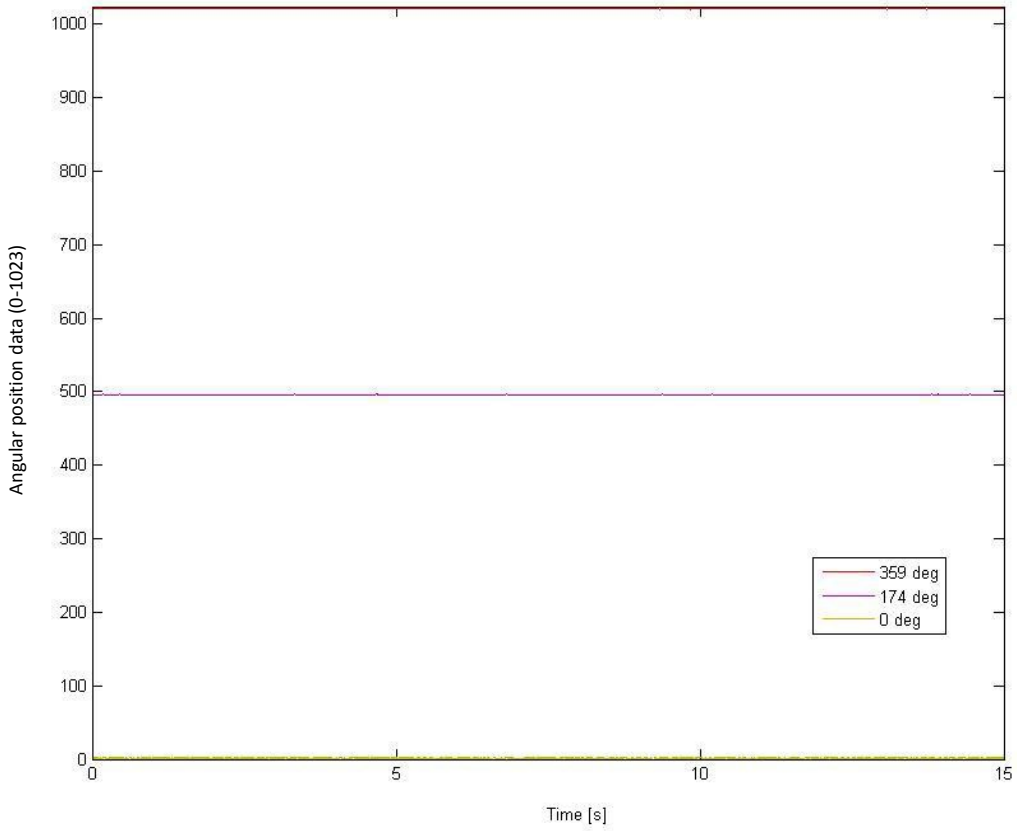


Figure 6.25 Check of the dead band

Table 6.2 Mean and Standard Deviation of each angle

	"0 degree"	"174 degree"	"359 degree"
Number of sample	28419	58592	58811
Mean	1.231 (0.4327 degree)	496.0 (174.4 degree)	1021 (358.9 degree)
Standard Deviation	0.4449 (0.1564 degree)	0.1938 (0.06813 degree)	0.1699 (0.05973 degree)

Table 6.3 Estimation and measurement of the sampling time

	"main" (in "while" loop)	"serial_rec_mag"	"serial_tran_rs232" (baud rate 55556)	time for one AS5040	time for three AS5040
Estimated time (µs)	16.5	183	360	950	2098
Measured time (µs)				800	2000

7 Conclusions

The serial interface between magnetic encoder (AS5040) and dSPACE© was implemented. The micro controller (PIC16F690) was used as a relay component and put in between AS5040 and dSPACE©. The interface between magnetic encoder and micro controller is synchronous serial interface (SSI), and the one between micro controller and dSPACE© is asynchronous serial interface (RS232). The data from three encoders are transferred one after the other via “daisy chain mode”.

By the implementation of this serial interface, the dead band has been vanished, and the sampling frequency has been 500 Hz.

8 Appendix

8.1.1 Whole C code for PIC16F690

```
/* Pins connection
pin 1                   :Vdd
pin 2                   :NONE
pin 3                   :NONE
pin 4                   :NONE
pin 5 (PORTC.5)         :Clock for AS5040
pin 6                   :NONE
pin 7                   :NONE
pin 8 (PORTC.6)         :CSn for AS5040
pin 9                   :NONE
pin 10                  :Data for dSPACE (RS232)
pin 11                  :NONE
pin 12 (PORTB.5)        :Data from AS5040 (SSI)
pin 13                  :NONE
pin 14                  :NONE
pin 15                  :NONE
pin 16                  :NONE
pin 17                  :NONE
pin 18                  :NONE
pin 19                  :NONE
pin 20                  :GND
*/
#define NUM_ENC 3 //Number of fingers. Up to 3 fingers. For more, add new variable "zerobit_of_each_data_2"

void clock(){
        PORTC.5 = 1;
        PORTC.5 = 0;
}
void csn(){
        PORTC.6 = 1;
```

```
        PORTC.6 = 0;
}
/*"serial_rec_mag" is a function to receive angular position data from a magnetic encoder (AS5040) by using
synchronous serial interface (SSI).
"char* serial_upper": the higher 8 bits of the data from AS5040.
"char* serial_lower": the lower 8 bits of the data from AS5040.
"int fing_num": the ordering number of the finger whose angular position data is treated now.
*/
void serial_rec_mag(char* serial_upper, char* serial_lower, int fing_num){
                PORTC.6 = 0;
                PORTC.5 = 0;
                clock();
                //receive first bit. put received bit into "serial_upper".
                if(PORTB.5 == 1){(*serial_upper) = ((*serial_upper)+1) << 1;}
                else {*serial_upper = (*serial_upper) << 1;
                }
                //reception of first bit end.
                clock();
                //receive second bit.
                if(PORTB.5 == 1){*serial_upper = ((*serial_upper)+1) << 1;}
                else {*serial_upper = (*serial_upper) << 1;
                }
                //reception of second bit end.
                clock();
                //receive third bit.
                if(PORTB.5 == 1){*serial_upper = ((*serial_upper)+1) << 1;}
                else {*serial_upper = (*serial_upper) << 1;
                }
                //reception of third bit end.
                clock();
                //receive 4th bit.
                if(PORTB.5 == 1){*serial_upper = ((*serial_upper)+1) << 1;}
                else {*serial_upper = (*serial_upper) << 1;
```

```
}
//reception of 4th bit end.
clock();
//receive 5th bit.
if(PORTB.5 == 1){*serial_upper= ((*serial_upper)+1) << 1;}
else {*serial_upper = (*serial_upper) << 1;
}
//reception of 5th bit end.
clock();
//receive 6th bit.
if(PORTB.5 == 1){*serial_upper = ((*serial_upper)+1) << 1;}
else {*serial_upper = (*serial_upper) << 1;
}
//receive 6th bit end.
clock();
//receive 7th bit.
if(PORTB.5 == 1){*serial_upper = ((*serial_upper)+1) << 1;}
else {*serial_upper = (*serial_upper) << 1;
}
//reception of 7th bit end.
clock();
//receive 8th bit.
if(PORTB.5 == 1)*serial_upper = ((*serial_upper)+1);
//reception of 8th bit end.
clock();
//receive 9th bit. And put it into "serial_lower".
if(PORTB.5 == 1){*serial_lower = ((*serial_lower)+1) << 1;}
else {*serial_lower = (*serial_lower) << 1;
}
//reception of 9th bit end.
clock();
//receive 10th bit.
if(PORTB.5 == 1){*serial_lower = ((*serial_lower)+1) << 1;}
```

```
else {*serial_lower = (*serial_lower) << 1;
}
//reception of 10th bit end.
clock();
//receive 11th bit.
if(PORTB.5 == 1){*serial_lower = ((*serial_lower)+1) << 1;}
else {*serial_lower = (*serial_lower) << 1;
}
//reception of 11th bit end.
clock();
//receive 12th bit.
if(PORTB.5 == 1){*serial_lower = ((*serial_lower)+1) << 1;}
else {*serial_lower = (*serial_lower) << 1;
}
//reception of 12th bit end.
clock();
//receive 13th bit.
if(PORTB.5 == 1){*serial_lower = ((*serial_lower)+1) << 1;}
else {*serial_lower = (*serial_lower) << 1;
}
//reception of 13th bit end.
clock();
//receive 14th bit.
if(PORTB.5 == 1){*serial_lower = ((*serial_lower)+1) << 1;}
else {*serial_lower = (*serial_lower) << 1;
}
//reception of 14th bit end.
clock();
//receive 15th bit.
if(PORTB.5 == 1){*serial_lower = ((*serial_lower)+1) << 1;}
else {*serial_lower = (*serial_lower) << 1;
}
//reception of 15th bit end.
```

```
                PORTC.5 = 1;
                PORTC.5 = 1;
                //receive last bit.
                if(PORTB.5 == 1)*serial_lower = ((*serial_lower)+1);
                //reception of last bit end.
                if(fing_num != (NUM_ENC -1)){ //treating final clock
                        PORTC.5 = 0;
                        PORTC.5 = 1;
                }else{
                        csn(); //This csn signal means the end of the reception.
                }
}
/*"attach_header" is a function to attach a flag at the 7th bit of each data.
"char* serial_upper": the higher 8 bits of the data from AS5040.
"char* serial_lower": the lower 8 bits of the data from AS5040.
"char* zerobit_of_each_data": the 0th bit of each data before data shifting in "attach_header" function.
"int fing_num": the ordering number of the finger whose angular position data is treated now.
*/
void attach_header(char* serial_upper, char* serial_lower, char* zerobit_of_each_data, int fing_num){
        //Shift bits of each data rightward by a bit. Then, put lapped bit by shifting into "zerobit_of_each_data"
variable.
        if((*serial_upper & 0b00000001) == 0b00000001){
                *zerobit_of_each_data = *zerobit_of_each_data << 1;
                *zerobit_of_each_data = *zerobit_of_each_data + 0b00000001;
        } else {
                *zerobit_of_each_data = *zerobit_of_each_data << 1;
        }
        if((*serial_lower & 0b00000001) == 0b00000001){
                *zerobit_of_each_data = *zerobit_of_each_data << 1;
                *zerobit_of_each_data = *zerobit_of_each_data + 0b00000001;
        } else {
                *zerobit_of_each_data = *zerobit_of_each_data << 1;
        }
```

```
		*serial_upper = *serial_upper >> 1;
		*serial_lower = *serial_lower >> 1;
		//Shifting end.
		if(fing_num == 0){
			*serial_upper = *serial_upper + 0b10000000;//the first byte´s flag is set ´1´
		}
}

/* "serial_tran_rs232" is a function to transmit angular position data to dSPACE.
"char* serial_upper": the higher 8 bits of the data from AS5040.
"char* serial_lower": the lower 8 bits of the data from AS5040.
"char zerobit_of_each_data": the 0th bit of each data before data shifting in "attach_header" function.
"int fing_num": the ordering number of the finger whose angular position data is treated now.
*/
void serial_tran_rs232(char* serial_upper, char* serial_lower, char zerobit_of_each_data, int fing_num){
		int i;
		//Initialize the asynchronous serial interface.
		if(fing_num == 0){
			//set baud rate.
			BRG16 = 0;
			BRGH = 1;
			SPBRGH = 0;
			SPBRG = 8;
			SYNC = 0;
			//set baud rate end.
			SPEN = 1;
			TXEN = 1; // This means the start of the asynchronous serial transmisssion.
		}
		//Write transmitting data into TXREG register.
		TXREG = *serial_upper;
		TXREG = *serial_lower;
		//Write transmitting data into TXREG register end.
		//Wait until transmission has finished
```

```
        while(1){
            if(TRMT==0){
                while(1){
                    if(TRMT==1){
                    break;
                    }
                }
                break;
            }
        }
        //Waiting until transmission has finished, end.

        if(fing_num == (NUM_ENC-1)){
            TXREG = zerobit_of_each_data; // At last, "zerobit_of_each_data" is transmitted.
            // Wait until the transmission has finished.
            while(1){
                if(TRMT==0){
                    while(1){
                        if(TRMT==1){
                        break;
                        }
                    }
                    break;
                }
            }
            //Waiting end.
            TXEN = 0; // This means the stop of the asynchronous serial transmission.
        }
}

void main() {
        char serial_upper[NUM_ENC];
        char serial_lower[NUM_ENC];
```

```
	char zerobit_of_each_data; //the gathered value of "0 bit" of each variable.
	int i;

	//Initial pin configuration. ´0´ means corresponding pin is for output, and `1´ means corresponding pin is for input.
	TRISC = 0b00000000;
	TRISB = 0b00100000;
	//Initial pin configuration end.
	//Set AN1-11 pins digital I/O.
	ANSEL = 0;
	ANSELH = 0; //set AN1-11 pins to degital I/O. As a default configuration, these pins are set to analog I/O. Without this setting, serial communication will never be available.
	//Set AN1-11 pins digital I/O end.
	//Set clock frequency 8 MHz
	OSCCON.6 = 1;
	OSCCON.5 = 1;
	OSCCON.4 = 1;
	OSCCON.0 = 1;
	//Set clock frequency 8 MHz end
	PORTC.5 = 1; //set "clock" for AS5040 HIGH

	while (1){
		/*initialization of the variables*/
		for(i=0; i<NUM_ENC; i++){
			serial_upper[i] = 0;
			serial_lower[i] = 0;
		}
		zerobit_of_each_data = 0;
		/*initialization of the variables end*/

		for(i=0; i<NUM_ENC; i++){
			serial_rec_mag(&serial_upper[i] , &serial_lower[i], i); //receive data from magnetic encoder by using synchronos serial interface.
```

```
                }
                for(i=0; i<NUM_ENC; i++){
                        attach_header(&serial_upper[i], &serial_lower[i], &zerobit_of_each_data, i); // header
is attached to avoid order confusion of data.
                }
                for(i=0; i<NUM_ENC; i++){
                        serial_tran_rs232(&serial_upper[i], &serial_lower[i], zerobit_of_each_data, i); //send
data to dSPACE by using RS232 interface.
                }
        }
}
```

8.1.2 Simulink model for the serial reception at dSPACE©

The whole Simulink model is shown in Figure 8.1. Each cycle of the angular position data from three magnetic encoders contains 7 bytes. The transferring of each byte is done one after the other. So, the confusion of receiving order may occur. Therefore, the detection of the first byte of each cycle is necessary.

That is why the flag bit was attached to the 7th bit of each byte at PIC16F690. At dSPACE©, this flag bit is checked at first. After finding a byte with the flag bit of ´1´, the following 6 bytes are received. Then the data is decoded, therefore the flag bit is extracted and the data is converted into angular position data.

8.1.3 Schematic diagram of electrical circuit

Schematic diagram is shown in Figure 8.2. Three encoders connected by in daisy chain order. The angular position data is transferred one after the other. The cable used to connect encoders were FFC[9]. Because of this the space saving in the hand has been achieved.

[9] Flexible Flat Cable

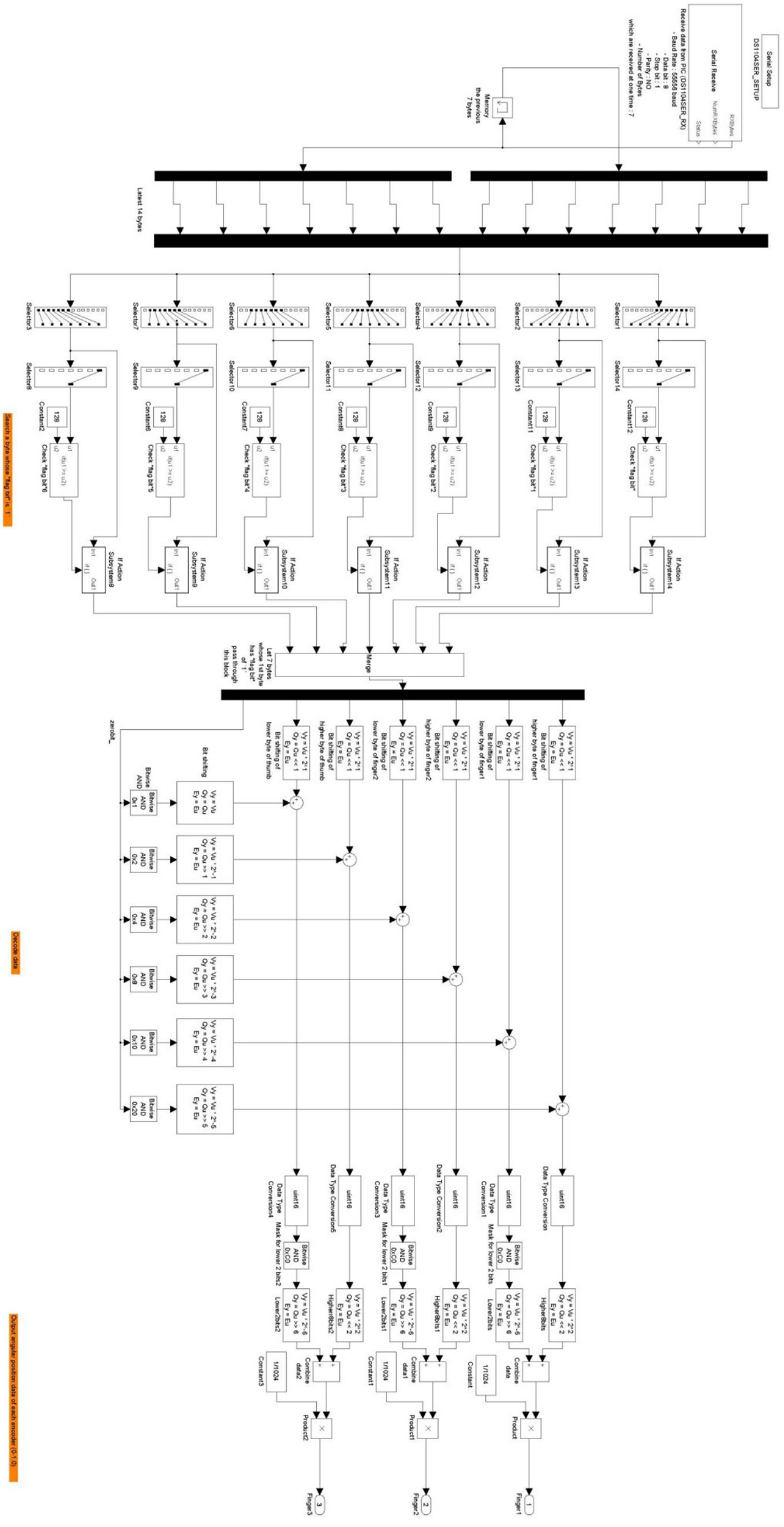


Figure 8.1 Simulink model

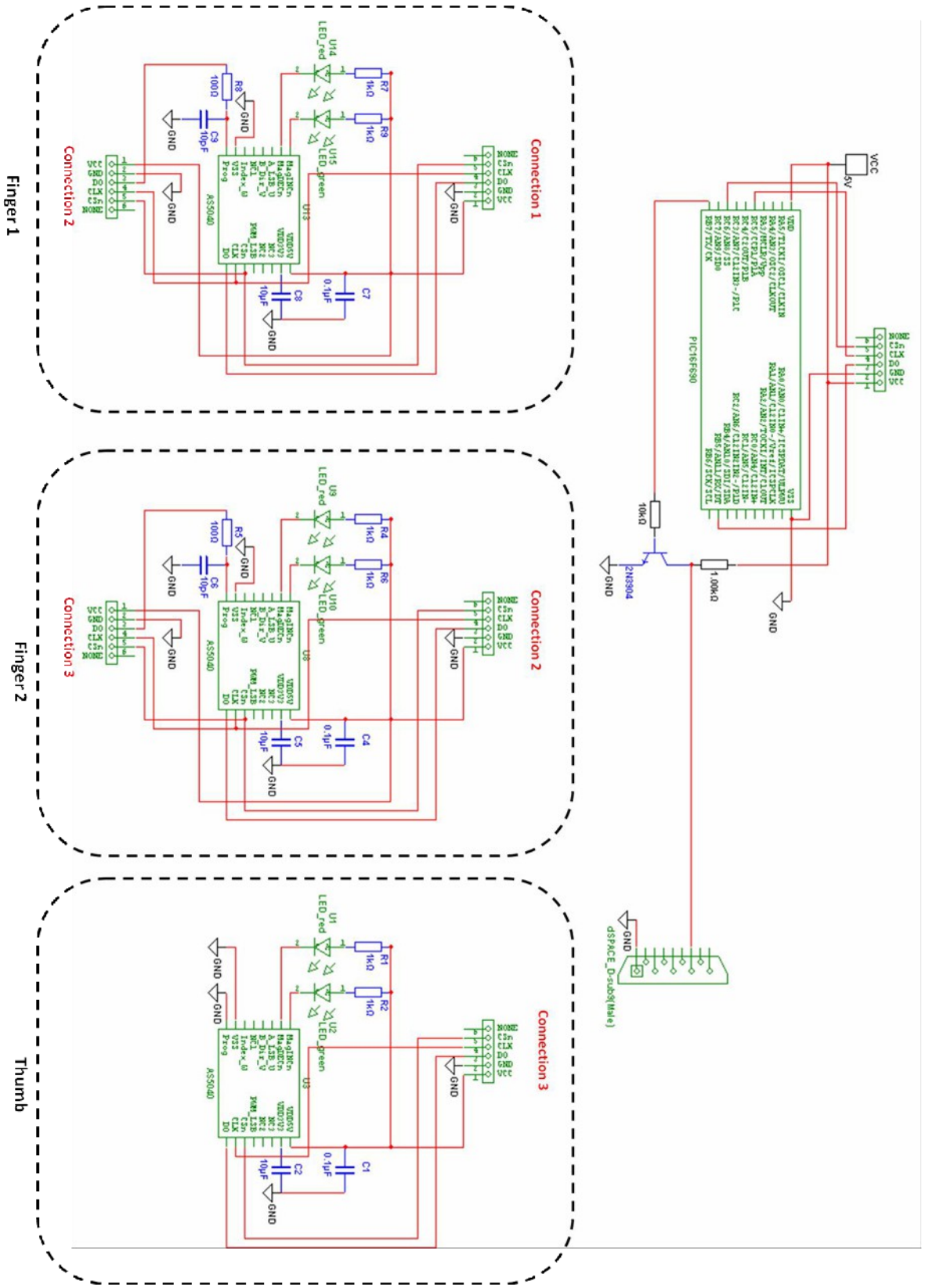


Figure 8.2 Schematic diagram of the electrical circuit

8.1.4 How to use PIC

Programming

1, Compile the code.

- "Project" → "Build"

2, Confirm the compile finished successfully.

3, Stop supplying power to PIC.

- "Programmer" → "Hold in Reset"

4, Send compiled code to PIC.

- "Programmer" → "Program"

5, Supply power to PIC

- "Programmer" → "Release from Reset"

Supplying power to PIC(Through PICkit2)

1, Open the "MPLAB IDE" software.

2, Supply power to PIC

- "Programmer" → "Release from Reset"

MPLAB IDE tutorial

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&redirects=mplab

C compiler "CC5X"

http://www.bknd.com/cc5x/

PIC16F690(Data Sheet)

http://ww1.microchip.com/downloads/en/DeviceDoc/41262C.pdf

9 Reference

[1] Matei Ciocarlie, Corey Goldfeder, Peter Allen, *"Dexterous Grasping via Eigengrasps: A Low-dimensional Approach to a High-complexity Problem"*, 2007.

[2] Specification of the DuraForm® PA plastic, http://www.nrri.umn.edu/NLTC/DS-DuraForm_PA_plastic_1106.pdf-

[3] Siim Viilup, *"ROBOT HAND REDESIGN-Fingertip force sensing implementation and assembling simplification"*, 2010, Student research project, KTH Machine Design.

[4] ERALP KARAKULLUKCU, *"Synchronous Serial Interface Of AS5040 Magnetic Encoder"*, 2010, Student research project, KTH Machine Design.

[5] AS5040 datasheet, http://pdf1.alldatasheet.com/datasheet-pdf/view/216354/AMSCO/AS5040.html

[6] dSPACE©, http://www.dspaceinc.com/en/inc/home.cfm?

[7] PIC16F690 datasheet, http://ww1.microchip.com/downloads/en/DeviceDoc/41262C.pdf

[8] Pin configuration of "RS232", http://www.lammertbies.nl/comm/cable/RS-232.html