Plate-forme de développement de l'ATMega16 : AVR Studio - carte STK500

Le kit STK500 est une carte de développement pour les microcontrôleurs Atmel de la famille AVR. Combiné à l'environnement de développement AVR Studio, il permet de concevoir, de simuler et de tester des prototypes. Pour plus de renseignements, référez vous aux documentations.

1 STK500

1.1 Description de la carte

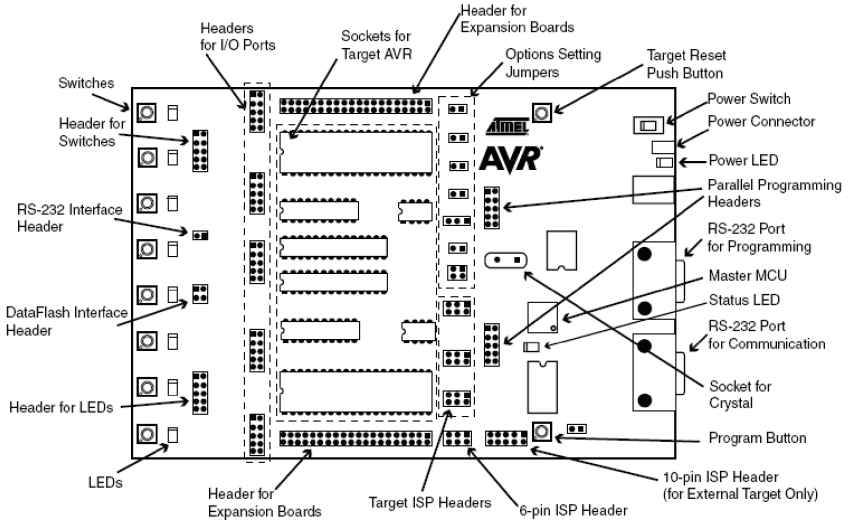


FIGURE 1 – Principaux éléments de la carte STK500.

Sur la carte STK500, les éléments importants à bien repérer sont :
- les emplacements pour les microcontrôleurs à programmer (différents emplacements pour différents types de microcontrôleurs) ;
- les connecteurs pour accéder aux ports d'entrées/sorties du microcontrôleur ;
- les séries de huit diodes et boutons-poussoirs ;
- le connecteur d'alimentation de la carte ;

- les deux connecteurs de communication série RS-232 (un pour la programmation du microcontrôleur, l'autre pour la communication classique) ;
- le connecteur à 6 broches (ISP, *In-System Programming*) permettant de connecter les signaux de programmation (du port de programmation) à l'emplacement microcontrôleur considéré.

1.2 Mise en œuvre de la carte

La mise en œuvre de la carte passe par :
- le positionnement du microcontrôleur sur son emplacement (SCKT3100A3 pour le microcontrôleur ATMega16) ;
- la mise en place du connecteur à 6 fils du connecteur ISP au connecteur SPROG correspondant à l'emplacement du microcontrôleur (se fier aux couleurs) ;
- si besoin, la connection des ports d'entrées/sorties du microcontrôleur aux diodes ou aux boutons-poussoirs ;
- la connection au PC de programmation du connecteur de programmation (liaison série) ;
- l'alimentation de la carte (12 V).

2 AVR Studio

2.1 Création d'un projet en langage C

Pour créer un projet en langage C, sélectionner :
- menu Project –> New Project
- type de projet : AVR GCC
- plate-forme : AVR Simulator
- cible : ATMega16.

2.2 Compilation et programmation de la cible

Pour programmer le microcontrôleur :
- compiler le programme (création d'un fichier .hex) ;
- se connecter à la cible (annuler si AVR Studio propose de mettre à jour la carte STK500) ;
- une fois la connection établie, dans la partie *Flash* sélectionner le fichier .hex du projet (cf. figure 5.2) ;
- le télécharger en mémoire Flash (*Program*).

2.3 Débogage d'un programme

Pour simuler l'exécution d'un programme et le déboger :
- compiler et lancer le débogueur ;
- avancer dans le programme avec :

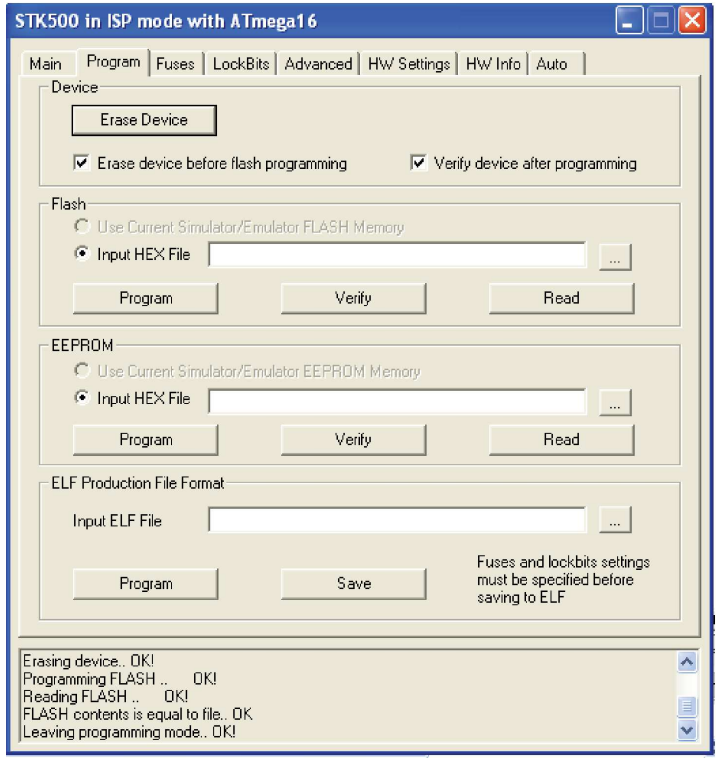


FIGURE 2 – Fenêtre de connexion à la carte STK500.

- pour avancer jusqu'au prochain point d'arrêt (*breakpoint*) ;
- pour exécuter la prochaine instruction ;
- pour rentrer dans la prochaine fonction ;
- pour sortir de la fonction ;
- pour arrêter le débogage ;

Chapitre 1

Entrées-sorties numériques

Rappels :

- Éviter de donner le même nom à des variables déclarées dans différentes fonctions.
- Veiller, pour une bonne lisibilité, à bien indenter le code.
- Tenir compte des "*warnings*" émis lors de la compilation.

La programmation en langage C des microcontrôleurs AVR (famille de microcontrôleurs Atmel) est grandement facilitée par l'utilisation de la librairie AVR Libc (http ://www.nongnu.org/avr-libc/). Celle-ci implémente une partie de la librairie standard C et fournit des fonctions classiques de la programmation des microcontrôleurs (gestion de la mémoire, des interruptions, des entrées/sorties, etc...).

1 Création d'un projet en langage C

Pour créer un projet en langage C, sélectionner :
- type de projet : AVR GCC
- plate-forme : AVR Simulator
- cible : ATMega16.

2 Compilation

Écrire un programme principal ne contenant aucune ligne de code (`void main(void){ }`). Le compiler. Lancer le débogueur et examiner le code assembleur généré par le compilateur (onglet "View", "Disassembler").

3 Entrées/sorties numériques

En langage C, l'écriture des différents registres d'entrées/sorties se fait directement en affectant la valeur souhaitée à une constante correspondant au nom du registre (ex. :

`DDRA = 0xFA`). Ces constantes sont définies par l'inclusion du fichier `io.h` de la librairie AVR Libc (`#include<avr/io.h>`).

Écrire un programme qui allumera les LEDs en fonction de l'état des boutons-poussoirs.

Écrire ensuite un programme qui fera défiler une LED allumée. Pour la temporisation, la fonction `_delay_ms()` pourra être utilisée. Celle-ci nécessite l'inclusion du fichier `util/delay.h` et la déclaration, au préalable, de la fréquence du CPU (ex. : `#define F_CPU 3686400` pour un fonctionnement à 3.6 MHz).

4 Interruptions

Programmer l'interruption INT0, afin qu'elle ré-initialise le défilement des LEDs. Relier INT0 au bouton-poussoir SW5.

Les interruptions sont gérées par le fichier `avr/interrupt.h`. Celui-ci permet de déclarer une sous-routine d'interruption grâce à la fonction `ISR(vect)`, où `vect` identifie l'interruption considérée (ex. : `ISR(INT0_vect)` pour la sous-routine associée à l'interruption INT0).

Chapitre 2

Communication série RS-232

Il s'agit dans ce TP d'établir une communication série entre le microcontrôleur ATmega16 et un PC. Il faudra ici bien différencier :

- la communication série établie entre le PC et la carte STK500 (connecteur RS232 CTRL), qui sert à programmer le microcontrôleur grâce à son port SPI ;
- la communication série qui va être mise en place par les programmes afin de faire communiquer un PC et le microcontrôleur (connecteur RS232 SPARE).

1 Configuration de la communication série

Écrire une fonction, appelée `void usart_init(unsigned int debit)`, d'initialisation de l'USART (*Universal Synchronous Asynchronous Receiver Transmitter*) qui prendra en argument le débit de la communication. Les caractéristiques de la communication seront :

- communication dans les deux sens (simultanée) autorisée ;
- pas de génération d'interruption ;
- débit de transmission : 57600 bps ;
- 8 bits de données ;
- 1 bit de stop ;
- pas de bit de parité.

Les différents registres associés à l'USART sont décrits à la fin de ce document. La configuration de la communication série est réalisée par l'intermédaire des registres UCSRB, UCSRC et UBRR (UBRRH et UBRRL). Le contenu des registres UBRR (*USART Baud Rate Register*) est donné par (cf. p.148 de la documentation ATmega16) :

$$ubrr = \frac{F_{CPU}}{16 \times debit} - 1 \tag{2.1}$$

La fréquence du CPU F_{CPU} (fréquence d'horloge du microcontrôleur) sera définie par un `#define` (ex. : `#define F_CPU 3686400` pour un fonctionnement à 3.6 MHz). La valeur des registres UBRR (en fonction de la fréquence d'horloge f_{CPU} et du débit *debit*) sera vérifiée suivant les tableaux donnés en fin de document.

2 Envoi d'un caractère

Écrire une fonction, appelée `void usart_putc(char c)`, permettant d'envoyer un caractère dont la valeur est transmise en argument. On veillera à vérifier la disponibilité du registre de données avant d'y écrire (cf. registre UCSRA).

Après avoir relié le connecteur RS232 SPARE au microcontrôleur, tester la fonction par une communication avec le PC.

3 Envoi d'une chaîne de caractères

Écrire une fonction, appelée `void usart_puts(char* s)`, permettant d'envoyer une chaîne de caractères [1].

Tester cette fonction par une communication avec le PC.

4 Réception d'un caractère

Écrire une fonction, appelée `char usart_getc(void)`, permettant de recevoir un caractère. On veillera à vérifier que la donnée a bien été transmise avant de la lire (cf. registre UCSRA).

Tester cette fonction par une communication avec le PC.

5 Communication bidirectionnelle

Écrire un programme envoyant une chaîne de caractère au PC invitant son utilisateur à saisir des caractères. Les caractères saisis, reçus par le microcontrôleur, seront ensuite ré-émis (afin de générer un "écho") et leur code ASCII sera affiché sur les LEDs.

1. Une chaîne de caractères est un tableau de `char` terminé par le caractère "\0"

Chapitre 3

Télémètre à ultrasons

Il s'agit ici de mettre en œuvre un télémètre à ultrason SRF05 (cf. documentation). Celui-ci repose sur l'émission d'une impulsion et sur la mesure du temps séparant émission et réception d'un écho.

Suivant la documentation fournie, proposez un programme permettant de mesurer la distance entre le télémètre et un obstacle. Un affichage reposant sur des diodes pourra tout d'abord être envisagé avant la transmission de la distance mesurée par la liaison série.

Chapitre 4

Joystick

Il s'agit ici de mettre en œuvre un module joystick. Les mouvements directionnels sont simplement associés à deux potentiomètres (un pour X, un pour Y). Le joystick dipose aussi d'un bouton de sélection.

Proposez un programme recevant les informations du joystick (concentrez vous tout d'abord sur une seule des directions). Un affichage reposant sur des diodes pourra être envisagé avant la transmission des informations par la liaison série.

Chapitre 5

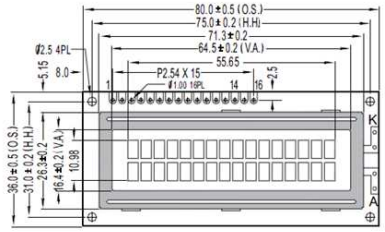
Afficheur LCD

Il s'agit ici de mettre en œuvre un afficheur à cristaux liquides (LCD).

1 Connexion

Connectez l'afficheur de la façon suivante :

- alimentation ;
- V0 (contraste) -> AREF (tension à ajuster à partie d'AVR Studio) ;
- RS –> PB0 ;
- R/W –> PB1 ;
- E –> PB2 ;
- De façon à simplifier les connexions, le port de données (DB0-DB7) –> PA0 à PD7 (PA0, PA3, PA4, PA7, ...).



PIN ASSIGNMENT		
PIN	SYMBOL	FUNCTION
1	Vss	GND
2	Vdd	Power Supply
3	V0	Contrast Adjust
4	RS	Register Select Signal
5	R/W	Data Read/Write
6	E	Enable Signal
7-14	DB0-DB7	Data Bus Line
15(A)	LED+	Power supply for BKL(+)
16(K)	LED-	Power supply for BKL(-)

FIGURE 5.1 – Brochage de l'afficheur LCD.

Name	Number	I/O	Interfaced with	Function
RS	1	I	MPU	Select registers. 0: Instruction register (for write) Busy flag: address counter (for read) 1: Data register (for write and read)
R/W	1	I	MPU	Select read or write. 0: Write 1: Read
E	1	I	MPU	Starts data read/write.
DB4 to DB7	4	I/O	MPU	Four high order bi-directional tristate data bus pins. Used for data transfer and receive between the MPU and the ST7066U. DB7 can be used as a busy flag.
DB0 to DB3	4	I/O	MPU	Four low order bi-directional tristate data bus pins. Used for data transfer and receive between the MPU and the ST7066U. These pins are not used during 4-bit operation.
CL1	1	O	Extension driver	Clock to latch serial data D sent to the extension driver
CL2	1	O	Extension driver	Clock to shift serial data D
M	1	O	Extension driver	Switch signal for converting the liquid crystal drive waveform to AC
D	1	O	Extension driver	Character pattern data corresponding to each segment signal
COM1 to COM16	16	O	LCD	Common signals that are not used are changed to non-selection waveform. COM9 to COM16 are non-selection waveforms at 1/8 duty factor and COM12 to COM16 are non-selection waveforms at 1/11 duty factor.
SEG1 to SEG40	40	O	LCD	Segment signals
V1 to V5	5	-	Power supply	Power supply for LCD drive V_{cc} - V5 = 10 V (Max)
V_{cc}, GND	2	-	Power supply	V_{cc}: 2.7V to 5.5V, GND: 0V
OSC1, OSC2	2		Oscillation resistor clock	When crystal oscillation is performed, a resistor must be connected externally. When the pin input is an external clock, it must be input to OSC1.

FIGURE 5.2 – Fonctions des broches de l'afficheur LCD.

2 Bus de données

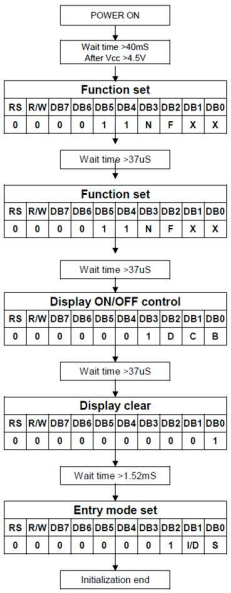
Ecrire une fonction `port_data(unsigned char octet)` qui envoie l'octet `octet` au bus de données DB0-DB7 de l'afficheur.

3 Commande du LCD

Ecrire une fonction `void lcd_command(unsigned char cmd)` qui envoie une commande au LCD. Pour ce faire, activez le bit E, puis envoyez la commande sur le bus de données. Désactivez enfin le bit E. Après chaque modification, insérez une temporisation de 5 ms.

4 Initialisation du LCD

Ecrire une fonction `void lcd_init()` qui initialise le LCD, suivant la figure 5.3.



Instruction	Instruction Code										Description	Description Time (270KHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM. and set DDRAM address to "00H" from AC	1.52 ms
Return Home	0	0	0	0	0	0	0	0	1	x	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.52 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 us
Display ON/OFF	0	0	0	0	0	0	1	D	C	B	D=1:entire display on C=1:cursor on B=1:cursor position on	37 us
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	x	x	Set cursor moving and display shift control bit, and the direction, without changing DDRAM data.	37 us
Function Set	0	0	0	0	1	DL	N	F	x	x	DL:interface data is 8/4 bits N:number of line is 2/1 F:font size is 5x11/5x8	37 us
Set CGRAM address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter	37 us
Set DDRAM address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter	37 us
Read Busy flag and address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0 us
Write data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM)	37 us
Read data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM)	37 us

FIGURE 5.3 – Séquence d'initialisation et instructions de l'afficheur LCD.

5 Envoi d'un caractère au LCD

Ecrire une fonction `void lcd_output(unsigned char data)` qui envoie le caractère `data` à afficher au LCD. Pour ce faire, activez les bits E et RS, puis envoyez le caratère sur le bus de données. Désactivez enfin le bit E. Après chaque modification, insérez une temporisation de 5 ms. Testez.

Accessing UBRRH/ UCSRC Registers

The UBRRH Register shares the same I/O location as the UCSRC Register. Therefore some special consideration must be taken when accessing this I/O location.

Write Access

When doing a write access of this I/O location, the high bit of the value written, the USART Register Select (URSEL) bit, controls which one of the two registers that will be written. If URSEL is zero during a write operation, the UBRRH value will be updated. If URSEL is one, the UCSRC setting will be updated.

The following code examples show how to access the two registers.

Assembly Code Example[1]

```
...
; Set UBRRH to 2
ldi r16,0x02
out UBRRH,r16
...
; Set the USBS and the UCSZ1 bit to one, and
; the remaining bits to zero.
ldi r16,(1<<URSEL)|(1<<USBS)|(1<<UCSZ1)
out UCSRC,r16
...
```

C Code Example[1]

```
...
/* Set UBRRH to 2 */
UBRRH = 0x02;
...
/* Set the USBS and the UCSZ1 bit to one, and */
/* the remaining bits to zero. */
UCSRC = (1<<URSEL)|(1<<USBS)|(1<<UCSZ1);
...
```

Note: 1. See “About Code Examples” on page 7.

As the code examples illustrate, write accesses of the two registers are relatively unaffected of the sharing of I/O location.

Read Access

Doing a read access to the UBRRH or the UCSRC Register is a more complex operation. However, in most applications, it is rarely necessary to read any of these registers.

The read access is controlled by a timed sequence. Reading the I/O location once returns the UBRRH Register contents. If the register location was read in previous system clock cycle, reading the register in the current clock cycle will return the UCSRC contents. Note that the timed sequence for reading the UCSRC is an atomic operation. Interrupts must therefore be controlled (for example by disabling interrupts globally) during the read operation.

The following code example shows how to read the UCSRC Register contents.

Assembly Code Example[1]

```
USART_ReadUCSRC:
  ; Read UCSRC
  in  r16,UBRRH
  in  r16,UCSRC
  ret
```

C Code Example[1]

```
unsigned char USART_ReadUCSRC( void )
{
  unsigned char ucsrc;
  /* Read UCSRC */
  ucsrc = UBRRH;
  ucsrc = UCSRC;
  return ucsrc;
}
```

Note: 1. See “About Code Examples” on page 7.

The assembly code example returns the UCSRC value in r16.

Reading the UBRRH contents is not an atomic operation and therefore it can be read as an ordinary register, as long as the previous instruction did not access the register location.

USART Register Description

USART I/O Data Register – UDR

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDR. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR Register location. Reading the UDR Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the UDRE Flag in the UCSRA Register is set. Data written to UDR when the UDRE Flag is not set, will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxD pin.

The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, do not use read modify write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS), since these also will change the state of the FIFO.

USART Control and Status Register A – UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 – RXC: USART Receive Complete**

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the receiver is disabled, the receive buffer will be flushed and consequently the RXC bit will become zero. The RXC Flag can be used to generate a Receive Complete interrupt (see description of the RXCIE bit).

- **Bit 6 – TXC: USART Transmit Complete**

This flag bit is set when the entire frame in the transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDR). The TXC Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXC Flag can generate a Transmit Complete interrupt (see description of the TXCIE bit).

- **Bit 5 – UDRE: USART Data Register Empty**

The UDRE Flag indicates if the transmit buffer (UDR) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE Flag can generate a Data Register empty Interrupt (see description of the UDRIE bit).

UDRE is set after a reset to indicate that the transmitter is ready.

- **Bit 4 – FE: Frame Error**

This bit is set if the next character in the receive buffer had a Frame Error when received. i.e., when the first stop bit of the next character in the receive buffer is zero. This bit is valid until the receive buffer (UDR) is read. The FE bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRA.

- **Bit 3 – DOR: Data OverRun**

This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the receive Shift Register, and a new start bit is detected. This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.

- **Bit 2 – PE: Parity Error**

This bit is set if the next character in the receive buffer had a Parity Error when received and the parity checking was enabled at that point (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.

- **Bit 1 – U2X: Double the USART Transmission Speed**

This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation.

Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

- **Bit 0 – MPCM: Multi-processor Communication Mode**

This bit enables the Multi-processor Communication mode. When the MPCM bit is written to one, all the incoming frames received by the USART receiver that do not contain address information will be ignored. The transmitter is unaffected by the MPCM setting. For more detailed information see “Multi-processor Communication Mode” on page 162.

USART Control and Status Register B – UCSRB

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – RXCIE: RX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the RXC Flag. A USART Receive Complete Interrupt will be generated only if the RXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXC bit in UCSRA is set.

- **Bit 6 – TXCIE: TX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the TXC Flag. A USART Transmit Complete Interrupt will be generated only if the TXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXC bit in UCSRA is set.

- **Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable**

Writing this bit to one enables interrupt on the UDRE Flag. A Data Register Empty Interrupt will be generated only if the UDRIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDRE bit in UCSRA is set.

- **Bit 4 – RXEN: Receiver Enable**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FE, DOR, and PE Flags.

- **Bit 3 – TXEN: Transmitter Enable**

Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled. The disabling of the Transmitter (writing TXEN to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the transmit Shift Register and transmit Buffer Register do not contain data to be transmitted. When disabled, the transmitter will no longer override the TxD port.

- **Bit 2 – UCSZ2: Character Size**

The UCSZ2 bits combined with the UCSZ1:0 bit in UCSRC sets the number of data bits (Character Size) in a frame the receiver and transmitter use.

- **Bit 1 – RXB8: Receive Data Bit 8**

RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDR.

- **Bit 0 – TXB8: Transmit Data Bit 8**

TXB8 is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. Must be written before writing the low bits to UDR.

USART Control and Status Register C – UCSRC

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

The UCSRC Register shares the same I/O location as the UBRRH Register. See the "Accessing UBRRH/ UCSRC Registers" on page 163 section which describes how to access this register.

- **Bit 7 – URSEL: Register Select**

This bit selects between accessing the UCSRC or the UBRRH Register. It is read as one when reading UCSRC. The URSEL must be one when writing the UCSRC.

- **Bit 6 – UMSEL: USART Mode Select**

This bit selects between Asynchronous and Synchronous mode of operation.

Table 63. UMSEL Bit Settings

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

- **Bit 5:4 – UPM1:0: Parity Mode**

These bits enable and set type of parity generation and check. If enabled, the transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPM0 setting. If a mismatch is detected, the PE Flag in UCSRA will be set.

Table 64. UPM Bits Settings

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- **Bit 3 – USBS: Stop Bit Select**

This bit selects the number of Stop Bits to be inserted by the Transmitter. The Receiver ignores this setting.

Table 65. USBS Bit Settings

USBS	Stop Bit(s)
0	1-bit
1	2-bit

- **Bit 2:1 – UCSZ1:0: Character Size**

The UCSZ1:0 bits combined with the UCSZ2 bit in UCSRB sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

Table 66. UCSZ Bits Settings

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

- **Bit 0 – UCPOL: Clock Polarity**

This bit is used for Synchronous mode only. Write this bit to zero when Asynchronous mode is used. The UCPOL bit sets the relationship between data output change and data input sample, and the synchronous clock (XCK).

Table 67. UCPOL Bit Settings

UCPOL	Transmitted Data Changed (Output of TxD Pin)	Received Data Sampled (Input on RxD Pin)
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

USART Baud Rate Registers – UBRRL and UBRRH

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

The UBRRH Register shares the same I/O location as the UCSRC Register. See the "Accessing UBRRH/ UCSRC Registers" on page 163 section which describes how to access this register.

- **Bit 15 – URSEL: Register Select**

This bit selects between accessing the UBRRH or the UCSRC Register. It is read as zero when reading UBRRH. The URSEL must be zero when writing the UBRRH.

- **Bit 14:12 – Reserved Bits**

These bits are reserved for future use. For compatibility with future devices, these bit must be written to zero when UBRRH is written.

- **Bit 11:0 – UBRR11:0: USART Baud Rate Register**

This is a 12-bit register which contains the USART baud rate. The UBRRH contains the four most significant bits, and the UBRRL contains the 8 least significant bits of the USART baud rate. Ongoing transmissions by the transmitter and receiver will be corrupted if the baud rate is changed. Writing UBRRL will trigger an immediate update of the baud rate prescaler.

Examples of Baud Rate Setting

For standard crystal and resonator frequencies, the most commonly used baud rates for asynchronous operation can be generated by using the UBRR settings in Table 68. UBRR values which yield an actual baud rate differing less than 0.5% from the target baud rate, are bold in the table. Higher error ratings are acceptable, but the receiver will have less noise resistance when the error ratings are high, especially for large serial frames (see "Asynchronous Operational Range" on page 160). The error values are calculated using the following equation:

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right) \bullet 100\%$$

Table 68. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 1.0000 MHz				f_{osc} = 1.8432 MHz				f_{osc} = 2.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max (1)	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

1. UBRR = 0, Error = 0.0%

Table 69. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 3.6864 MHz				f_{osc} = 4.0000 MHz				f_{osc} = 7.3728 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	–	–	0	-7.8%	–	–	0	0.0%	0	-7.8%	1	-7.8%
1M	–	–	–	–	–	–	–	–	–	–	0	-7.8%
Max (1)	230.4 kbps		460.8 kbps		250 kbps		0.5 Mbps		460.8 kbps		921.6 kbps	

1. UBRR = 0, Error = 0.0%

Table 70. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 8.0000 MHz				f_{osc} = 11.0592 MHz				f_{osc} = 14.7456 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	-7.8%	3	-7.8%
1M	–	–	0	0.0%	–	–	–	–	0	-7.8%	1	-7.8%
Max (1)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

1. UBRR = 0, Error = 0.0%

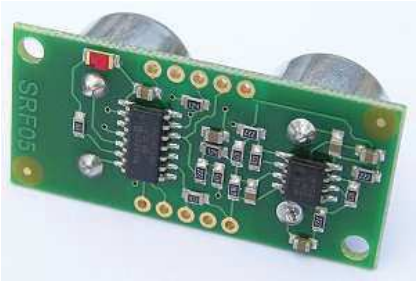
Table 71. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 16.0000 MHz				f_{osc} = 18.4320 MHz				f_{osc} = 20.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
Max (1)	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

1. UBRR = 0, Error = 0.0%

SRF05 - Ultra-Sonic Ranger

Technical Specification

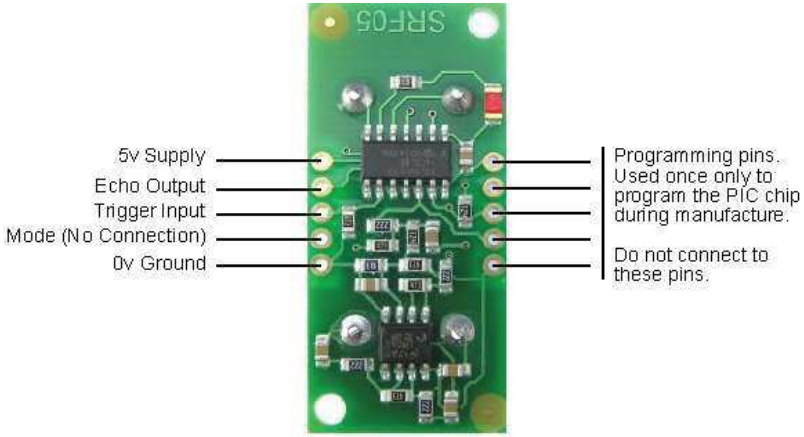


Introduction

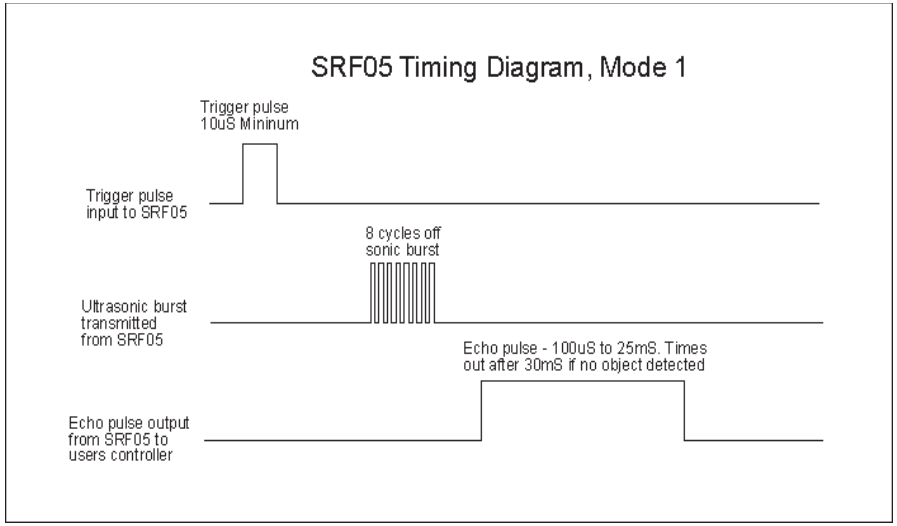
The SRF05 is an evolutionary step from the SRF04, and has been designed to increase flexibility, increase range, and to reduce costs still further. As such, the SRF05 is fully compatible with the SRF04. Range is increased from 3 meters to 4 meters. A new operating mode (tying the mode pin to ground) allows the SRF05 to use a single pin for both trigger and echo, thereby saving valuable pins on your controller. When the mode pin is left unconnected, the SRF05 operates with separate trigger and echo pins, like the SRF04. The SRF05 includes a small delay before the echo pulse to give slower controllers such as the Basic Stamp and Picaxe time to execute their pulse in commands.

Mode 1 - SRF04 compatible - Separate Trigger and Echo

This mode uses separate trigger and echo pins, and is the simplest mode to use. All code examples for the SRF04 will work for the SRF05 in this mode. To use this mode, just leave the mode pin unconnected - the SRF05 has an internal pull up resistor on this pin.

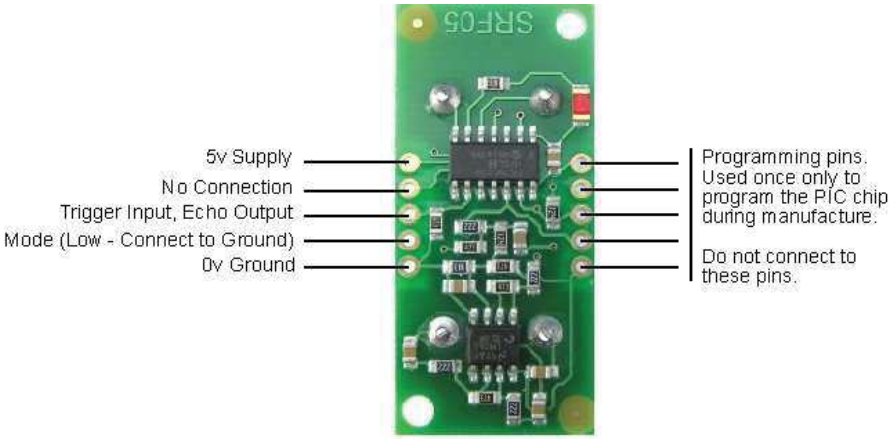


Connections for 2-pin Trigger/Echo Mode (SRF04 compatible)

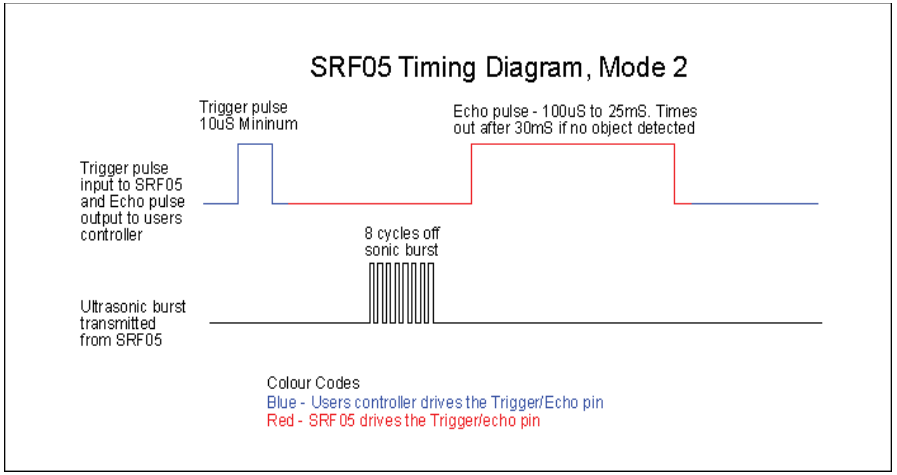


Mode 2 - Single pin for both Trigger and Echo

This mode uses a single pin for both Trigger and Echo signals, and is designed to save valuable pins on embedded controllers. To use this mode, connect the mode pin to the 0v Ground pin. The echo signal will appear on the same pin as the trigger signal. The SRF05 will not raise the echo line until 700uS after the end of the trigger signal. You have that long to turn the trigger pin around and make it an input and to have your pulse measuring code ready. The PULSIN command found on many popular controllers does this automatically.



Connections for single pin Trigger/Echo Mode



To use mode 2 with the Basic Stamp BS2, you simply use PULSOUT and PULSIN on the same pin, like this:

```
SRF05 PIN 15              ' use any pin for both trigger and echo
Range VAR Word            ' define the 16 bit range variable

SRF05 = 0                 ' start with pin low
PULSOUT SRF05, 5          ' issue 10uS trigger pulse (5 x 2uS)
PULSIN SRF05, 1, Range    ' measure echo time
Range = Range/29          ' convert to cm (divide by 74 for inches)
```

Calculating the Distance

The SRF05 Timing diagrams are shown above for each mode. You only need to supply a short 10uS pulse to the trigger input to start the ranging. The SRF05 will send out an 8 cycle burst of ultrasound at 40khz and raise its echo line high (or trigger line in mode 2). It then listens for an echo, and as soon as it detects one it lowers the echo line again. The echo line is therefore a pulse whose width is proportional to the distance to the object. By timing the pulse it is possible to calculate the range in inches/centimeters or anything else. If nothing is detected then the SRF05 will lower its echo line anyway after about 30mS.

The SRF04 provides an echo pulse proportional to distance. If the width of the pulse is measured in uS, then dividing by 58 will give you the distance in cm, or dividing by 148 will give the distance in inches. uS/58=cm or uS/148=inches.

The SRF05 can be triggered as fast as every 50mS, or 20 times each second. You should wait 50ms before the next trigger, even if the SRF05 detects a close object and the echo pulse is shorter. This is to ensure the ultrasonic "beep" has faded away and will not cause a false echo on the next ranging.

The other set of 5 pins

The 5 pins marked "programming pins" are used once only during manufacture to program the Flash memory on the PIC16F630 chip. The PIC16F630's programming pins are also used for other functions on the SRF05, so make sure you don't connect anything to these pins, or you will disrupt the modules operation.

Changing beam pattern and beam width

You can't! This is a question which crops up regularly, however there is no easy way to reduce or change the beam width that I'm aware of. The beam pattern of the SRF05 is conical with the width of the beam being a function of the surface area of the transducers and is fixed. The beam pattern of the transducers used on the SRF05, taken from the manufacturers data sheet, is shown below.