Getting Started with TAMS GPIO interfaces

GPIO is a highly flexible interface that can be configured to meet a variety of user needs. This flexibility does present challenges however, due to the number of configurable parameters and the unlimited degree of freedom in designing the hardware at the user end.

The interfaces are capable of either handshake operation, or a simple read bits/set bits mode of operation. In compatibility mode, there are 16 input and 16 output lines, plus two status inputs and two control outputs. In enhanced mode, the 16 input lines can be used in a bi-directional mode (handshake or non-handshake), and the 16 output lines can be set to arbitrary patterns out in a non-handshake mode. Generally the handshake operation will be used with high-speed data transfer, the non-handshake mode will be used with mechanical handlers, used to control relays or lights, monitoring switches, etc.

The DOUT lines are high-voltage, high-current (30 volts, 40 ma) open collector TTL , with optional pullups to 5 volts provided. The DIN lines are restricted to TTL levels, both in compatibility and enhanced mode.

For output operation, there is a PCTL delay parameter that can be set to allow settling time on the DOUT lines before the handshake proceeds. This is frequently needed when driving long cables.

For input operation, the user circuitry is responsible for delaying the handshake after driving data to the DIN lines of the interface, if needed.

Non-Handshake Example

This is the easiest mode to configure, since no circuitry need be used with the handshake lines. Sample C code is given here. This uses SICL calls, which can be used in either the Windows or Linux environment when the appropriate I/O libraries and interface drivers are loaded on the system.

```
/*
       Sample program demonstrating non-handshake operation for
       input and output with GPIO cards.

       Requires an interface configured with SICL name of 'gpio'
*/


#include <sicl.h>
#include <stdio.h>
#include <stdlib.h>

main(argc,argv)  {
   INST id;
   int i;
   unsigned long data;


   id=iopen("gpio");
   if(id==0) {
          printf("Error opening 'gpio' device file\n");
          exit(1);
   }


   igpiosetwidth(id, 16);             // set 16 bit operation

   for (i=0;i<65536;i++)  igpioctrl(id,I_GPIO_DATA,i);  // cycle through all patterns
   igpioctrl(id,I_GPIO_DATA,0x1234);   // leave distinctive pattern on output bus
   igpiostat(id, I_GPIO_DATA, &data);  // read data pattern on input bus
   printf("DIN data = %X hex\n",data);

}
```

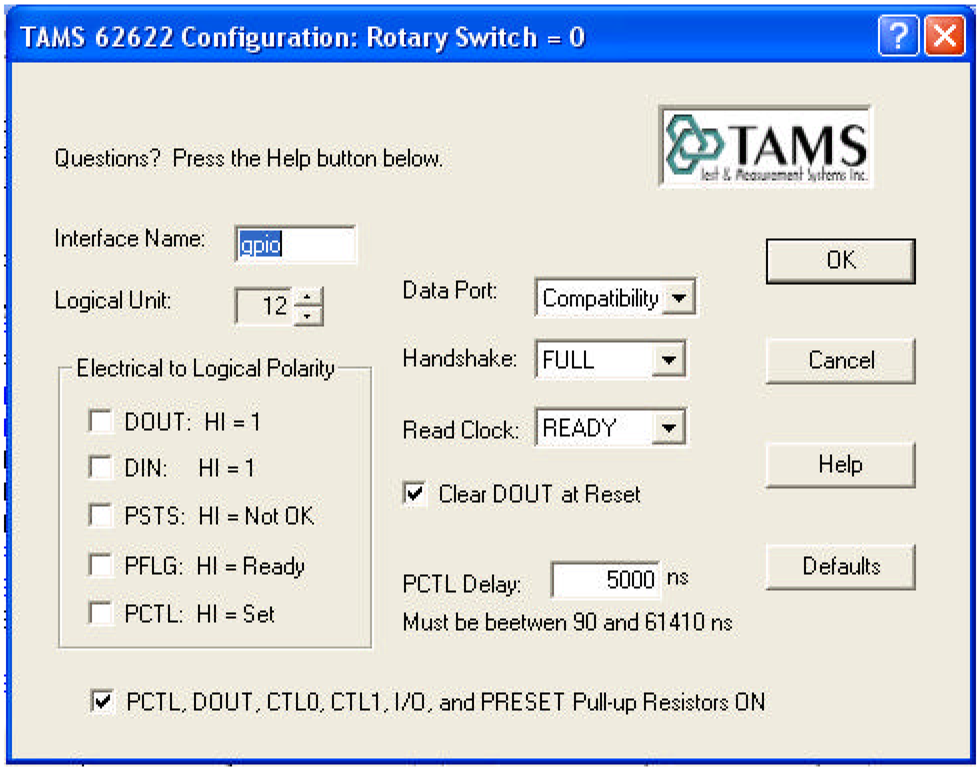
Handshake Example

Handshake operation requires more work to create appropriate user circuitry. Both inputs and outputs can be handshaken, the circuitry must monitor the I/O line if both are being used. The handshake is a two-wire interlocked handshake, which allows the user circuitry to pace the transfer as needed.

For purposes of this document, assume that none of the optional inversions of control or data lines are used. Turn on the optional pullup resistors. We'll use the default data polarity, where a logical '1' is a low level.

For read operations (into the gpio interface), the Ready (RDY) clock option will be used. This latches the data internally at the second edge of PFLG, which is the end of the handshake sequence. PSTS (Peripheral Status) is another input to the gpio interface which can be used to hold off a handshake. It is configured here to be High=OK, and the pullup resistor pulls up the level so that handshakes can proceed. Five microseconds is used as a PCTL delay, to allow substantial setting time on outputs.

The configuration on Windows should look like this:


TAMS 62622 Configuration: Rotary Switch = 0

Questions? Press the Help button below.

Interface Name: gpio

Logical Unit: 12

Electrical to Logical Polarity
- [ ] DOUT: HI = 1
- [ ] DIN: HI = 1
- [ ] PSTS: HI = Not OK
- [ ] PFLG: HI = Ready
- [ ] PCTL: HI = Set

Data Port: Compatibility

Handshake: FULL

Read Clock: READY

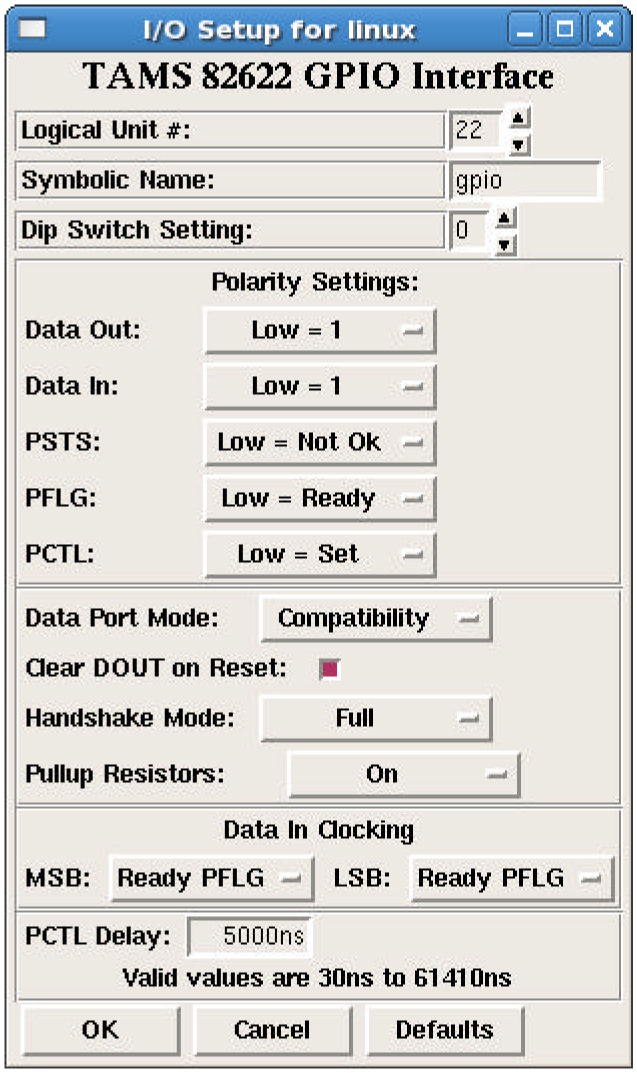
- [✓] Clear DOUT at Reset

PCTL Delay: 5000 ns

Must be beetwen 90 and 61410 ns

- [✓] PCTL, DOUT, CTL0, CTL1, I/O, and PRESET Pull-up Resistors ON

On a Linux system, it looks like:



I/O Setup for linux

TAMS 82622 GPIO Interface

Logical Unit #:	22
Symbolic Name:	gpio
Dip Switch Setting:	0

Polarity Settings:

Data Out:	Low = 1
Data In:	Low = 1
PSTS:	Low = Not Ok
PFLG:	Low = Ready
PCTL:	Low = Set

Data Port Mode:	Compatibility
Clear DOUT on Reset:	■
Handshake Mode:	Full
Pullup Resistors:	On

Data In Clocking

MSB: Ready PFLG LSB: Ready PFLG

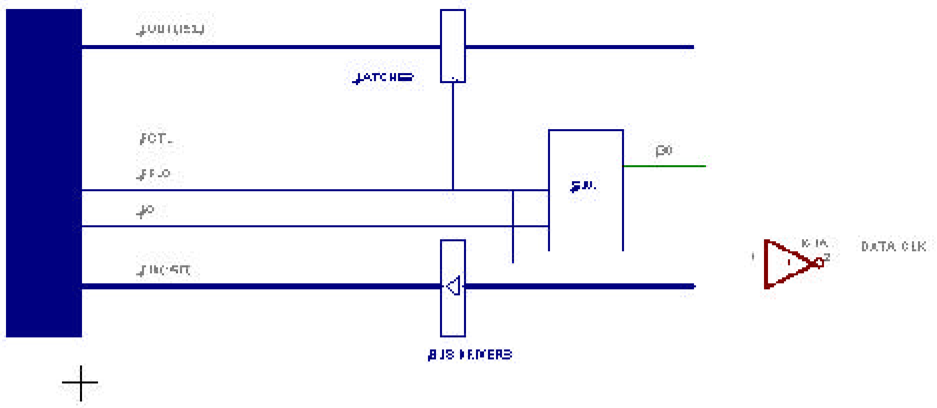
PCTL Delay: 5000ns

Valid values are 30ns to 61410ns

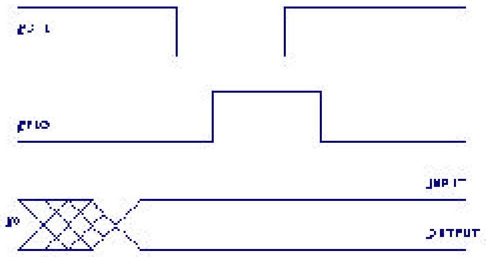
The corresponding entry in /etc/opt/sicl/hwconfig.cf is:

```
# TAMS 82622 GPIO Interface
22 gpio t82622 0 0 0b100000 0x10 0x22 5000ns
```

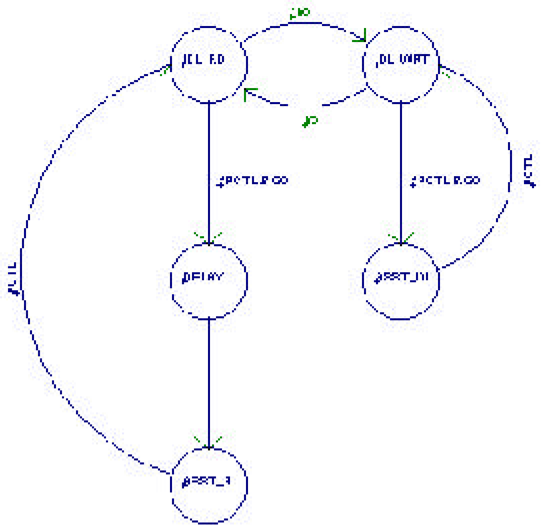
In this example, a state machine with 3 inputs and 1 output is used. The inputs are the PCTL handshake line, the I/O line, and GO from the user system. Output is PFLG, the handshake line from the external circuitry. Its rising edge could be used to latch data into the circuitry from a write operation, or its falling edge could clock external circuitry to drive the next data on the bus. GO is a signal from the external circuitry, indicating the hardware can accept, or source, the next piece of data.



DOUT(15:0)
LATCHES
PCTL
RO
DIN(15:0)
BUS DRIVERS
SM
DATA CLK



PCTL
INPUT
OUTPUT



IDLE_RD
IDLE_WRT
DELAY
ASRT_R
ASRT_W
PCTL & GO
PCTL & GO
PCTL
PCTL

The state transition diagram for the state machine is shown above, and a listing of next state information and output is shown here:

Current state	Next State	Outputs

IDL_RD	if *PCTL & GO then DELAY
	else if *I/O then IDL_WRT
	else IDL_RD	

IDL_WRT	if *PCTL & GO then ASST_W
	else if I/O then IDL_RD
	else IDL_WRT	

DELAY	ASST_R	

ASST_R	if PCTL then IDL_RD
	else ASST_R
		PFLG

ASST_W	if PCTL then IDL_WRT
	else ASST_W
		PFLG

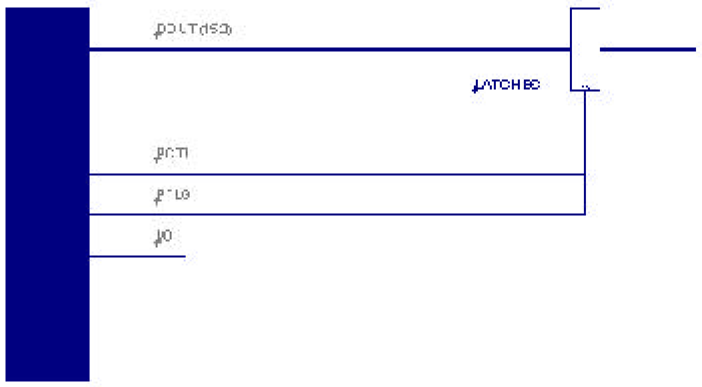
A program appropriate for this example is:

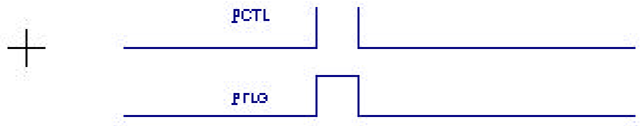
```
/*
       Sample program demonstrating handshake operation for
       input and output with GPIO cards.

       Requires an interface configured with SICL name of 'gpio'
        Invert polarity of PCTL, not PFLG.
*/

#include <sicl.h>
#include <stdio.h>
#include <stdlib.h>

main(argc,argv)  {
   INST id;
   int i;
   int reason;
   unsigned long data, actual;
   unsigned char buf[4];

   id=iopen("gpio");
   if(id==0) {
          printf("Error opening 'gpio' device file\n");
          exit(1);
   }

   igpiosetwidth(id, 16);             // set 16 bit operation

   for (i=0;i<65536;i++)  {
      buf[0] = i;
      buf[1] = i/256;
      iwrite(id,buf, 2, 0, &actual); // cycle through all patterns
   }

   buf[0] = 0x34;
   buf[1] = 0x12;
   buf[2] = 0x78;
   buf[3] = 0x56;
   iwrite(id,buf, 4, 0, &actual); // write "1234h", then "5678h"
   if(actual != 4) printf("Error, didn't write 4 bytes\n");

   iread(id, buf, 2, &reason ,  &actual);   // read data pattern on input bus
   printf("DIN data = %X hex\n",256*buf[1] + buf[0]);
}
```

Simplified Handshake Example

As a simpler example, assume the data transfer is output-only, and the external circuitry can keep up with the max data rates.

PFLG can be driven from the PCTL line, and the required inversion can be implemented with checking the PCTL inversion box. Settling time on the bus is controlled by the PCTL delay, which can be used to slow down the transfer even further if desired. The I/O line does not need to be monitored, because all transfers are output only. A short, positive going pulse appears on the handshake lines that can be used to latch data locally.





This example could be expanded to make a simple test hood for testing purposes by just connecting the DOUT lines to the corresponding DIN lines, and removing the latches if desired. No settling time is needed for reading the data since it will have settled long before a read operation can follow a write. The internal pullup resistors on the interface card are entirely adequate for providing good signal quality in this situation.

Enhanced Mode Example

Enhanced mode can be used when there is a need to maximize the number of output lines available. The lines labeled DIN become a bi-directional bus with handshake capability, and the DOUT lines can be set to arbitrary patterns in a non-handshake mode.

The program also demonstrates setting CTL bits and reading STI bits, which is done independently of any handshake. Enhance mode should be set in the GUI's, and appropriate external hardware provided to implement a bi-directional bus on the DIN lines. Do not drive data onto that bus from the external circuitry when the gpio interface is driving it with an output operation.

```
/*
       Sample program demonstrating handshake operation for
       input and output with GPIO cards in Enhanced Mode.

       Also includes setting CONTROL bits, reading STATUS bits.

       Requires an interface configured with SICL name of 'gpio'.
       Also requires external hardware to implement the handshake.
*/


#include <sicl.h>
#include <stdio.h>
#include <stdlib.h>

main(argc,argv)  {
   INST id;
   int i,aa;
   int reason;
   unsigned long  data, actual;
   unsigned char buf[4];

   id=iopen("gpio");
   if(id==0) {
          printf("Error opening 'gpio' device file\n");
          exit(1);
   }


   igpiosetwidth(id, 16);            // set 16 bit operation

   for (i=0;i<65536;i++)  {
      buf[0] = i;
      buf[1] = i/256;
      iwrite(id,buf, 2, 0, &actual); // cycle through all patterns on DIN lines
   }

   buf[0] = 0x34;
   buf[1] = 0x12;
   buf[2] = 0x78;
   buf[3] = 0x56;
   iwrite(id,buf, 4, 0, &actual); // write "1234h", then "5678h" on DIN lines
   if(actual != 4) printf("Error, didn't write 4 bytes\n");

   iread(id, buf, 2, &reason ,  &actual);   // read data pattern on DIN lines
   if(actual != 2) printf("Error, didn't read 2 bytes\n");
   printf("DIN data = %X hex\n",256*buf[1] + buf[0]);

   aa=igpioctrl(id, I_GPIO_AUX, 0x9876);    // set pattern on DOUT (AUX) lines
   if(aa) printf("Setting AUX lines failed\n");
   aa=igpioctrl(id, I_GPIO_CTRL,  I_GPIO_CTRL_CTL0);     // set CTL0, clear CTL1
   if(aa) printf("Setting CTL lines failed\n");
   aa=igpiostat(id, I_GPIO_STAT,  &data);                // read STI1, STI0
   if(aa) printf("Reading STAT lines failed\n");
   printf("STI1 = %2i\n",  (data & I_GPIO_STAT_STI1) >> 1);  // display STI1
   printf("STI0 = %2i\n",  data & I_GPIO_STAT_STI0);         // display STI0

}
```

Electrical Considerations

The DIN lines and other input lines are TTL inputs, biased to a high level when not connected to external circuitry. Care must be taken to insure the input voltages do not exceed normal TTL operating limits.

The DOUT lines are open-collector high voltage/high current (30 volts, 40 ma) outputs. There are weak pullups available to 5 volts, but these only provide 1 ma per line. The x2622 product has socketed resistors that allow lower-value resistors to be used, to increase the pullup current if needed. See the product manual for details. External pullup resistors may also be used.

If driving long cables, the drive capability of the circuitry becomes important for maintaining adequate signal integrity and good performance. See “How to drive long GPIO cables”, available in the Support section of the TAMS website.