Digital I/O Wildcard Users Guide

Introduction

This document describes how to use the Digital I/O Wildcard™, providing an overview of the hardware and software and a schematic. The Digital I/O Wildcard allows you to easily add up to 20 lines of digital I/O to your system. It can be used to expand the I/O of any of Mosaic's controller products. The following sections guide you through the Digital I/O Wildcard's hardware and software.

Table 1-1. Digital I/O Wildcard Specifications

Digital I/O Wildcard Specifications		
Channels		
	Configurable Channels:	16, configurable as input or output in groups of 4
	Fixed Input Channels:	4
Inputs		
	Input Voltage Range:	0 – 5 V (-0.5 to 5.5 absolute max)
	Input Low Voltage:	< 0.80 V
	Input High Voltage:	> 2.0 V
	Input Leakage Current:	± 10 μA
	Input Capacitance:	10 pF
Outputs		
	Output Voltage Range:	0 – 5 V (-0.5 to 5.5 absolute max)
	Output Low Voltage:	< 0.5 V at 24 mA (i.e., 12Ω typ., 21Ω worst case)
	Output High Voltage:	>3.5 V typ., 5.0 V pulled up, >2.4 V at -4.0 mA
	Output Current:	24 mA sink, 4 mA source
	Max Output Current:	100 mA on any pin
	Max Internal Power:	250 mW all pins together
Pull up/down		
	Optional Pull-up/down :	10 KΩ, jumper selectable pull up/down

Hardware

Overview

The Digital I/O Wildcard expands the digital I/O capabilities of Mosaic embedded controllers. Each Digital I/O Wildcard provides 16 channels configurable in groups of four as either inputs or outputs plus 4 additional digital input channels. Each line is configurable for pull up, pull down, or tri-state operation. Outputs sink up to 24 mA or source up to 4 mA continuously.

The next sections show you how to hook up the Digital I/O Wildcard to your controller, how to select the Wildcard address, and how to configure each line for pull-up, pull-down, or tri-state operation.

Connecting To the Mosaic Controller

To connect the Digital I/O Wildcard to a Wildcard-ready Mosaic controller or to a Wildcard Carrier Board, follow these simple steps:

With the power off, connect the Wildcard Bus on the Digital I/O Wildcard to Wildcard Port 0 or Wildcard Port 1 on your controller or the Wildcard Carrier Board (These may also be called *Module* Port 0 or 1). If you are using a Wildcard Carrier Board, connect it to the QED Board as outlined in the "Wildcard Carrier Board Users Guide". The corner mounting holes on the Wildcard should line up with the standoffs on your controller.

CAUTION: The Wildcard bus does not have keyed connectors. Be sure to insert the Wildcard so that all pins are connected. The Mosaic controllers and the Digital I/O Wildcard can be permanently damaged if the connection is done incorrectly.

Selecting the Wildcard Address

Once you have connected the Wildcard you must set the address of the module using jumper shunts across J20 and J21. These Wildcard Select Jumpers select a 2-bit code that sets a unique address on the Wildcard port of the Mosaic controller. Each Wildcard port on the controller board accommodates up to 4 Wildcards. Wildcard Port 0 on the Mosaic controller provides access to Wildcards addressed at locations 0-3 while Wildcard Port 1 provides access to Wildcards 4-7. Two Wildcards on the same port cannot have the same address (jumper settings). Table 1-2 shows the possible jumper settings and the corresponding addresses.

Table 1-2: Jumper Settings and Associated Addresses

Wildcard Port	Wildcard Address	Installed Jumper Shunts
0	0	None
	1	J20
	2	J21
	3	J20 and J21
1	4	None
	5	J20
	6	J21
	7	J20 and J21

Configuring the Digital I/O Lines

You may configure each I/O line for pull-up, pull-down, or tri-state operation. This allows you to set the appropriate level of each I/O line between power-up and software initialization.

There are twenty three-post jumpers on the Digital I/O Wildcard labeled J0-J19. The labels are located next to each jumper post. The jumpers are used to configure the twenty digital I/O lines for pull-up, pull-down, or tri-state operation. To configure a digital line for pull-up operation, simply install a jumper shunt across the two jumper posts above the + sign located next to the jumper label. To configure a digital line for pull-down operation, install the jumper shunt across the two pins above the – sign. By not installing a jumper shunt, the line is configured for tri-state operation.

Once you have connected and configured all of the hardware, you can use the software drivers to read or set the digital lines.

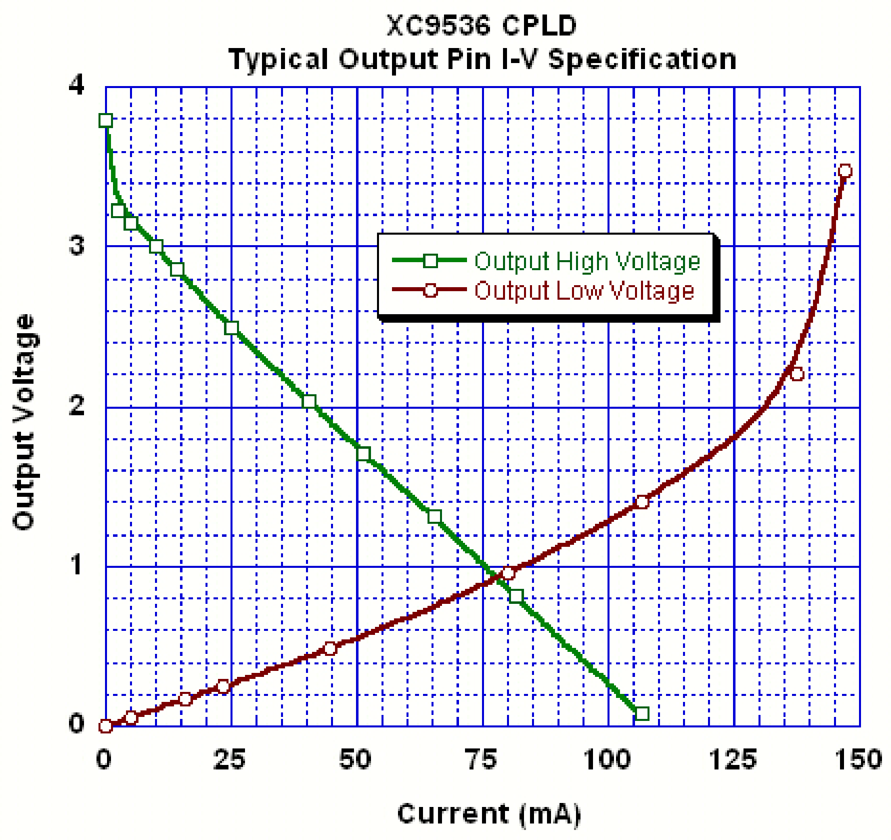
Current Capability of the Digital Output Lines

The inputs and outputs are all provided by a single device, a Xilinx CPLD, XC9536.

The digital outputs provide TTL-compatible levels for driving logic devices. But you can also control devices, such as relays and light emitting diodes, that require more current. You can either provide current from a digital output high, or you can sink current into a digital output low. Generally speaking, the output low is capable of sinking much more current than the output high can source, and using it results in lower power dissipation. So if you need substantial current, you should turn on the external device by sinking current from it using an output low.

As an output is loaded, its voltage level, V_{OL} or V_{OH}, rises or falls with the current. It is often useful to know just how much to expect the V_{OL} and V_{OH} levels to degrade with current. For currents of less than 100 mA the voltage change is approximately linear with current; that is, it can be modeled as a voltage source of either zero volts (for an output low) or 3.3 volts (for an output high) and an equivalent series resistance. At greater output currents the resistance becomes nonlinear, but you shouldn't ever source or sink more than 100 mA.

The following figure shows the degradation of output voltage levels with increasing source or sink current for a typical device.



An output high is typically 3.8 V, but rises to 5 V when the pull-up jumper is installed. As current is drawn by an external device the voltage drops, appearing as a 3.3 volt source and a 30 Ω resistance. That is,

$$V_{OH} = 3.3\ V - 30\ \Omega * I_{source}$$

The output impedance for an output low is considerably less, allowing it to sink more current than an output high can source. With no current, the output low is zero volts, and it increases with current as a 12 Ω resistance (12 Ω typically, 21 Ω worst case), up to a limit of approximately 100 mA:

$$V_{OL} = 12\ \Omega * I_{sink}$$

The above graph and equations can be used to choose component values for particular circuits. For example if we wish to use an output low to drive a light-emitting diode we would place the LED in series with a resistor and connect them between an output pin and a +5V supply. The resistor limits the current through the LED and into the output low. From the LED data sheet we note that its forward voltage at a current of 10 mA is specified to be 2.2 V. What should the resistor value be? We calculate it as,

$$R = (5V-2.2V - V_{OL}) / 10\ mA$$

Consulting the V_{OL} vs I curve for the output pin we find that at 10 mA $V_{OL} = 0.12V$. We therefore need a resistance near 270 ohms.

Protecting the Input and Output Pins

These output pins are very useful because they can directly drive a wide range of devices. Nevertheless, any circuitry connected to them should take care to:

- Prevent excessive voltage levels at the pin;
- Prevent excessive currents; and,
- Prevent excessive power in the Wildcard.

We'll address each of these concerns in turn.

Preventing Excessive Voltages: Excessive voltages are prevented by ensuring that voltages of less than a diode drop below ground (-0.5 V) or greater than a diode drop above 5V (5.5 V) are never applied to an output pin. For some applications, particularly when driving inductive loads such as relays, you may need to provide Schottkey diode clamps between the pin and +5V and between the pin and ground. If an output pin is directly connected to a voltage source below ground or above 5V the Wildcard will be destroyed. Whenever possible, it's a good idea to drive external devices through a current limiting resistor (say 100 Ω).

Preventing Excessive Currents: The current into or out of any output pin on the Wildcard should also be limited to prevent damage. These pins can withstand brief source or sink currents of up to 100 mA at room temperature, but you should never allow currents greater than 100 mA. Load circuitry that requires significant current to be sourced or sunk by the digital output should include external resistors to ensure that an absolute maximum rating of 100 mA on a single output pin is never exceeded.

Preventing Excessive Power: Never cause more than 250 mW of I/O pin power to be dissipated on the Wildcard. The total power allowed is the sum of the power dissipated by each pin. For output highs that power is the product of the source current and the difference between +5V and V_{OH}. For output lows that power is the product of the sinked current and V_{OL}. A limitation of the total power to 250 mW is actually quite generous. Note that from the above figure, when 25 mA is sunk into an output low the power contributed is only 7.5 mW (for a typical device, max of 12.5 mW for any device). Consequently, you can continuously sink 25 mA into all the output pins simultaneously.

Connecting to the Field Header

All connections to the Wildcard should be made throught the Field Header, H2. This right-angle header provides the 20 I/O lines as well as power and ground connections.

Table 1-3: Digital I/O Field Header H2

Signal	Pins		Signal
GND	– 1	2 –	+5V
GND	– 3	4 –	V+RAW (5.5-26V)
Input 19	– 5	6 –	Input 18
Input 17	– 7	8 –	Input 16
Input/Output 15	– 9	10 –	Input/Output 14
Input/Output 13	– 11	12 –	Input/Output 12
Input/Output 11	– 13	14 –	Input/Output 10
Input/Output 9	– 15	16 –	Input/Output 8
Input/Output 7	– 17	18 –	Input/Output 6
Input/Output 5	– 19	20 –	Input/Output 4
Input/Output 3	– 21	22 –	Input/Output 2
Input/Output 1	– 23	24 –	Input/Output 0

Software

This section describes the software that enables you to control the Digital I/O Wildcard. Briefly, the Wildcard is addressed by addressing specific memory locations with a page address corresponding to the Wildcard's physical address, the direction (as inputs or outputs) of each nibble of the I/O is configured by writing to a specific location, inputs are read by reading a memory location and outputs are written by writing to the same location.

Setting the Direction of the I/O Lines

Several bytes of memory starting at C000$_H$ are used to communicate with the Digital I/O Wildcard. The page used for the memory's extended address corresponds to the Wildcard address. For example, to communicate with Wildcard 1 on the Wildcard Carrier Board, use the 6 byte memory block starting at address C000$_H$ on page 1.

The 20 digital I/O lines on the Digital I/O Wildcard are organized into five 4-channel groups or nibbles. The five nibbles are accessed using addresses C000$_H$ to C004$_H$. The four digital I/O lines (digital inputs 15-19) at C004$_H$ are read-only inputs. I/O lines 0-15 are configured using a direction register at C005$_H$. Each bit of the least significant 4-bit nibble of the direction register controls the direction of four digital I/O lines. Table 2 summarizes the organization of the digital I/O lines.

Table 2: Organization of the Digital I/O Lines

Address	Name	Digital Lines	Direction
C000	Nibble 0	0 - 3	Inputs/Outputs
C001	Nibble 1	4 - 7	Inputs/Outputs
C002	Nibble 2	8 - 11	Inputs/Outputs
C003	Nibble 3	12 - 15	Inputs/Outputs
C004	Nibble 4	16 - 19	Inputs
C005	Direction Register	--	--

In setting the direction, a one in a bit position in C005 causes the corresponding nibble of I/O to be an output and a zero sets it as an input. The least significant bit of C005 controls the direction of the lowest four output lines (those whose values are controlled by C000), the next bit of C005 controls lines 4-7 (whose values are set by C001) and so on. The upper nibbles of C000 through C005 do nothing.

For example, for a Wildcard addressed at location 3, from the Forth terminal you could set the direction of digital lines 0 through 7 to outputs and 8 through 15 as inputs by executing,

```
    HEX
    03 C005 03 C! \ Sets bits 0 and 1 to one
                  \ and bits 2 and 3 to zero at address C005 on page 03
```

Setting bits 0 and 1 to one configures the lower two nibbles, that is, lines 0 through 7, as outputs. And setting bits 2 and 3 to zero configures the next two nibbles, lines 8 through 15, as inputs.

The output lines would immediately assume the values provided by the contents of addresses C000 through C001 (so you might want to initialize them first!). You could send alternate output highs and lows to lines 0 through 7 by executing,

```
    0A C000 03 C!
    0A C001 03 C!
```

At each of the nibble locations there are separate read and write registers – you can read only from the read register and you can write only to the write register. For example, whatever the value you read from C000, the values writtten to lines 0 through 3 when the direction is set to output will be the last nibble written to C000. Also, whether the pins are configured as inputs or outputs, reading from C000 will always return the actual pin values, and not necessarily what you last wrote to C000.

On power up, all of the digital I/O lines are initialized as inputs. Initializing the direction of the digital I/O lines is similar to setting the direction of Port A or Port C. The following C and FORTH code presents an example routine to set the direction of the I/O lines on the Digital I/O Wildcard.

```
// C Code to initialize the Digital I/O Wildcard

#include <allqed.h>                       // Include QED header files

#define DIRECTION_REGISTER    0xC005

#define NIBBLE_0              1           // Lines 0-3
#define NIBBLE_1              2           // Lines 4-7
#define NIBBLE_2              4           // Lines 8-11
#define NIBBLE_3              8           // Lines 12-15

#define OUTPUT                1
#define INPUT                 0

void Init_IO_Direction ( uchar module_number, uchar nibble, uchar direction )
```

```
// Valid module numbers are 0-7.  Valid nibbles are NIBBLE_0 to NIBBLE_3
// Valid directions are INPUT or OUTPUT.
// ---------------------------------------------------------------------------
// The module number depends on the module select jumpers.  See Table 1 for
// the jumper settings and associated addresses.
// ---------------------------------------------------------------------------
// No error checking is done on the input parameters!
// ---------------------------------------------------------------------------
// This routine initializes the direction of a nibble of I/O lines on the
// Digital I/O Wildcard.
{
  EXTENDED_ADDR module_addr;

  module_addr.sixteen_bit.page16 = module_number;
  module_addr.sixteen_bit.addr16 = DIRECTION_REGISTER;

  if(direction)
  {
    SetBits( nibble, module_addr.addr32 );      // set nibble as output
  }
  else
  {
    ClearBits( nibble, module_addr.addr32 );    // set nibble as input
  }
}
```

```
\ Forth Code to initialize the Digital I/O Wildcard

HEX

4 USE.PAGE          \ Initialize the memory map.
15 WIDTH !          \ Avoid non-unique names.
ANEW DIO.CODE       \ Forget marker for easy re-loading.

C005  CONSTANT    DIRECTION_REGISTER

1     CONSTANT    NIBBLE_0          \ Lines 0-3
2     CONSTANT    NIBBLE_1          \ Lines 4-7
4     CONSTANT    NIBBLE_2          \ Lines 8-11
8     CONSTANT    NIBBLE_3          \ Lines 12-15

1     CONSTANT    OUTPUT
0     CONSTANT    INPUT

: Init_IO_Direction ( byte1\u\byte2 -- )
\ byte1 = module number, byte2 = nibble, byte3 = direction
\ Valid module numbers are 0-7.  Valid nibbles are NIBBLE_0 to NIBBLE_3
\ Valid directions are INPUT or OUTPUT.
\ ---------------------------------------------------------------------------
\ The module number depends on the module select jumpers.  See Table 1 for
\ the jumper settings and associated addresses.
\ ---------------------------------------------------------------------------
\ No error checking is done on the input parameters!
\ ---------------------------------------------------------------------------
\ This routine initializes the direction of a nibble of I/O lines on the
\ Digital I/O Wildcard.
```

```
locals{ &direction &nibble &module }

&direction
IF
  &nibble DIRECTION_REGISTER &module SET.BITS   \ set nibble as output
ELSE
  &nibble DIRECTION_REGISTER &module CLEAR.BITS \ set nibble as input
ENDIF
;
```

Controlling the I/O Lines

Once you have set the direction of the I/O lines, you can read and set the I/O lines like the other digital I/O ports on the Mosaic Controller.

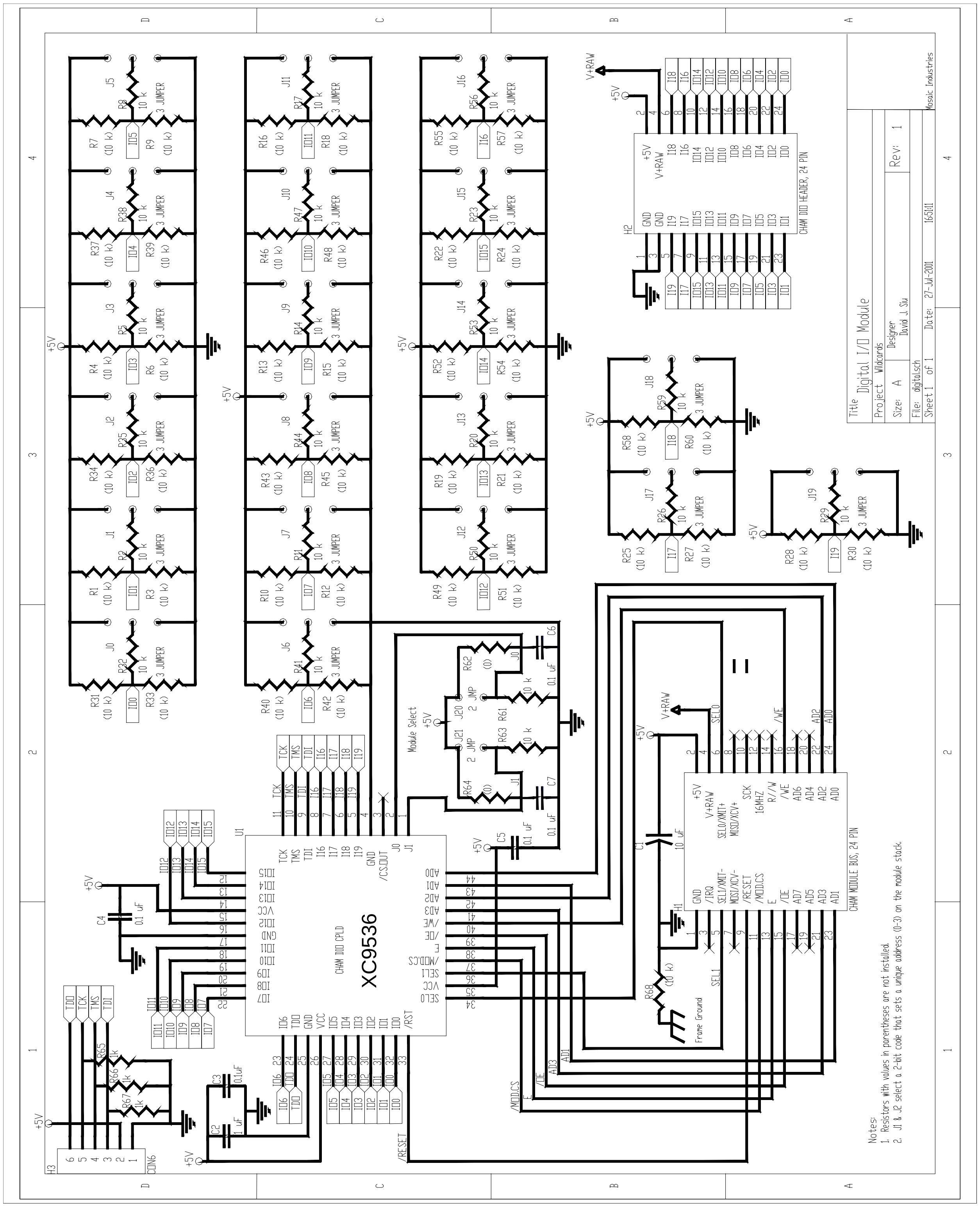
```
// C Code to control the Digital I/O Wildcard

#define OUTPUT_HIGH                1
#define OUTPUT_LOW                 0

#define NIBBLE_0_ADDR              0xC000  // Lines 0-3.
#define NIBBLE_1_ADDR              0xC001  // Lines 4-7.
#define NIBBLE_2_ADDR              0xC002  // Lines 8-11.
#define NIBBLE_3_ADDR              0xC003  // Lines 12-15.
#define NIBBLE_4_ADDR              0xC004  // Lines 16-19.  Inputs only

#define LINE_0                     1
#define LINE_1                     2
#define LINE_2                     4
#define LINE_3                     8
#define LINE_4                     1
#define LINE_5                     2
#define LINE_6                     4
#define LINE_7                     8
#define LINE_8                     1
#define LINE_9                     2
#define LINE_10                    4
#define LINE_11                    8
#define LINE_12                    1
#define LINE_13                    2
#define LINE_14                    4
#define LINE_15                    8

void Control_DIO ( uchar mod_num, uint nibble_addr, uchar line, uchar state )
// Sets I/O line of specified nibble to the appropriate state (high or low).
// Valid module (ie Wildcard) numbers are 0-7.
// Valid nibble addresses are NIBBLE_0_ADDR to NIBBLE_3_ADDR.
// Valid lines are LINE_0 to LINE_15
// Valid states are OUTPUT_HIGH or OUTPUT_LOW
{
  EXTENDED_ADDR module_addr;

  module_addr.sixteen_bit.page16 = mod_num;
  module_addr.sixteen_bit.addr16 = nibble_addr;
  if(state) // set line high
  {
    SetBits( line, module_addr.addr32 );
  }
  else      // set line low
  {
    ClearBits ( line, module_addr.addr32 );
  }
}

uchar Read_Nibble ( uchar module_number, uint nibble_addr )
// Reads the current state of the Digital I/O nibble.
// Valid module numbers are 0-7.
// Valid nibble addresses are NIBBLE_0_ADDR to NIBBLE_4_ADDR.
// Returns an unsigned character whose least significant nibble represents
// the four I/O lines.  For example, if nibble 1 is read and a 1 is returned
// (0001 in binary), then line 4 is high and lines 5-7 are low.  If 12 is
// returned (1100 in binary) after reading nibble 3, then lines 12 and 13 are
// low and lines 14 and 15 are high.  The four most significant bits of the
// returned byte do not matter.
{
  EXTENDED_ADDR module_addr;
  uchar nibble_status;

  module_addr.sixteen_bit.page16 = module_number;
  module_addr.sixteen_bit.addr16 = nibble_addr;

  nibble_status = FetchChar( module_addr.addr32 );

  return( nibble_status );
}
```

```
\ Forth Code to control the Digital I/O Wildcard

HEX

1           CONSTANT OUTPUT_HIGH
0           CONSTANT OUTPUT_LOW

C000  CONSTANT NIBBLE_0_ADDR        \ Lines 0-3.
C001  CONSTANT NIBBLE_1_ADDR        \ Lines 4-7.
C002  CONSTANT NIBBLE_2_ADDR        \ Lines 8-11.
C003  CONSTANT NIBBLE_3_ADDR        \ Lines 12-15.
C004  CONSTANT NIBBLE_4_ADDR        \ Lines 16-19.  Inputs only.

1           CONSTANT LINE_0
2           CONSTANT LINE_1
4           CONSTANT LINE_2
8           CONSTANT LINE_3
1           CONSTANT LINE_4
2           CONSTANT LINE_5
4           CONSTANT LINE_6
8           CONSTANT LINE_7
1           CONSTANT LINE_8
2           CONSTANT LINE_9
```

```
4           CONSTANT LINE_10
8           CONSTANT LINE_11
1           CONSTANT LINE_12
2           CONSTANT LINE_13
4           CONSTANT LINE_14
8           CONSTANT LINE_15

: Control_DIO ( byte1\u\byte2\byte3 -- )
\ Sets I/O line of specified nibble to the appropriate state (high or low).
\ byte1 = module number, u = nibble address, byte2 = line, byte3 = state.
\ Valid module numbers are 0-7.
\ Valid nibble addresses are NIBBLE_0_ADDR to NIBBLE_3_ADDR.
\ Valid lines are LINE_0 to LINE_15
\ Valid states are OUTPUT_HIGH or OUTPUT_LOW
locals{ &state &line &nibble_addr &module }

  &state
  IF                          \ set line high
    &line &nibble_addr &module SET.BITS
  ELSE                        \ set line low
    &line &nibble_addr &module CLEAR.BITS
  ENDIF
;

: Read_Nibble ( byte1\u –- byte2 | byte1 = module number, u = nibble addr )
\ Reads the current state of the Digital I/O nibble.
\ Valid module numbers are 0-7.
\ Valid nibble addresses are NIBBLE_0_ADDR to NIBBLE_4_ADDR.
\ Returns an unsigned character whose least significant nibble represents
\ the four I/O lines.  For example, if nibble 1 is read and a 1 is returned
\ (0001 in binary), then line 4 is high and lines 5-7 are low.  If 12 is
\ returned (1100 in binary) after reading nibble 3, then lines 12 and 13 are
\ low and lines 14 and 15 are high.  The four most significant bits of the
\ returned byte do not matter.
locals{ &nibble_addr &module }

  &nibble_addr &module C@
;
```

Conclusion

Now you are ready to start using your Digital I/O Wildcard. All of the software routines listed in this document are also on the distribution diskette that accompanies each Wildcard.


Digital I/O Module
Project Wildcards
Designer David J. Siu
Size: A
Rev: 1
File: digital.sch
Sheet 1 of 1
Date: 27-Jul-2001
Mosaic Industries
XC9536
CHAM DIO CPLD
CHAM DIO HEADER, 24 PIN
CHAM MODULE BUS, 24 PIN
Module Select
Frame Ground
Notes:
1. Resistors with values in parentheses are not installed.
2. J1 & J2 select a 2-bit code that sets a unique address (0-3) on the module stack.