Digital I/O Module Users Guide 1.0

Introduction

This document describes how to use the Digital I/O WildCard™ Module. It provides an overview of the hardware and software for the module as well as a schematic.

The Digital I/O Module allows you to easily add configurable digital I/O to your system. The following sections guide you through the Digital I/O Module's hardware and software.

Hardware

Overview

The Digital I/O Module expands the digital I/O capabilities of the QED Board. Each Digital I/O Module provides:

- 16 channels configurable in groups of four as either inputs or outputs.
- 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.

The next sections show you how to connect the Digital I/O Module to the WildCard© Carrier Board, how to select the module address, and how to configure each line for pull-up, pull-down, or tri-state operation.

Connecting to the WildCard© Carrier Board

To connect the Digital I/O Module to the WildCard© Carrier Board, follow these simple steps:

1. Connect the WildCard© Carrier Board to the QED Board as outlined in the "WildCard© Carrier Board Users Guide".
2. With the power off, connect the Module Bus on the Digital I/O Module to Module Port 0 or Module Port 1 on the WildCard© Carrier Board. The corner mounting holes on the module should line up with the standoffs on the WildCard© carrier Board. The Module Bus on the Digital I/O Module is located opposite from the screw terminals. The module ports are shown in Figure 1 of the "WildCard© Carrier Board Users Guide". CAUTION: The WildCard© Carrier Board does not have keyed connectors. Be sure to insert the module so that all pins are connected. The WildCard© Carrier Board and the Digital I/O Module can be permanently damaged if the connection is done incorrectly.

Selecting the Module Address

Once you have connected the Digital I/O Module to the WildCard© Carrier Board, you must set the address of the module using jumper shunts across J20 and J21.

The Module Select Jumpers, labeled J20 and J21, select a 2-bit code that sets a unique address on the module port of the WildCard© Carrier Board. Each module port on the WildCard© Carrier Board accommodates up to 4 modules. Module Port 0 on the WildCard© Carrier Board provides access to modules 0-3 while Module Port 1 provides access to modules 4-7. Two modules on the same port cannot have the same address (jumper settings). Table 1 shows the possible jumper settings and the corresponding addresses.

Module Port	Module 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

Table 1: Jumper Settings and Associated Addresses

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 module 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.

Software

This section describes the software that enables you to control the Digital I/O Module. We first start with a description of how modules are addressed, then move on to how to set the direction of the I/O lines, and finally how to control the digital I/O lines.

Setting the Direction of the I/O Lines

Several bytes of memory on the QED board starting at $C000_H$ are used to communicate with the Digital I/O Module. The page used for the memory's extended address corresponds to the module address. For example, to communicate with module 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 Module 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.

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	--	--

Table 2: Organization of the Digital I/O Lines

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 module.

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

#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 Module.
{
  EXTENDED_ADDR module_addr;

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

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

```
}
```

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

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 Module.
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 QED Board.

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

#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 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.page16 = mod_num;
  module_addr.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.page16 = module_number;
  module_addr.addr16 = nibble_addr;

  nibble_status = FetchChar( module_addr );

  return( nibble_status );
}
```

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

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
```

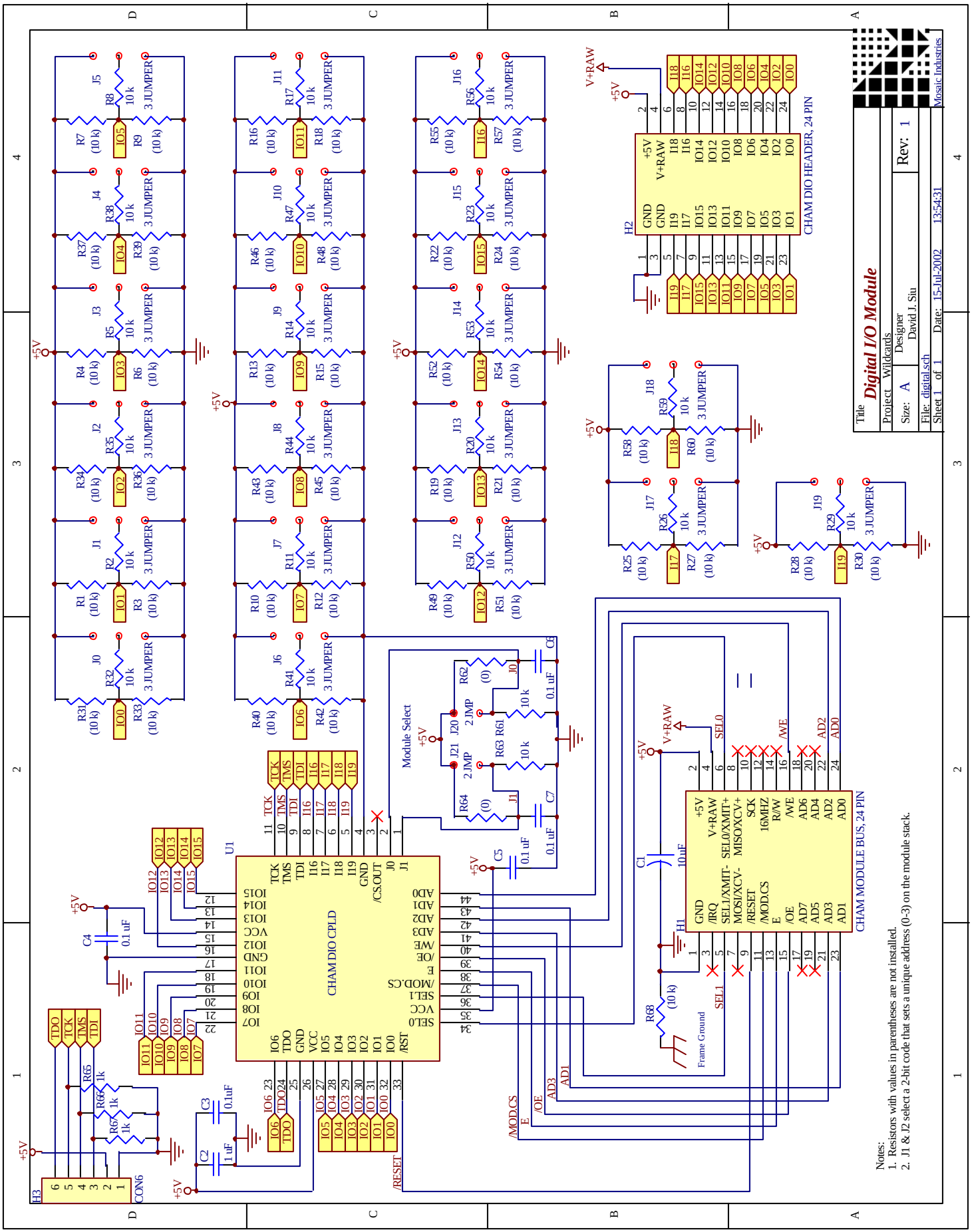
```
;

: 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 Module. All of the software routines listed in this document are also on the distribution diskette that accompanies each module.


Digital I/O Module
Project Wildcards
Designer David J. Siu
Rev: 1
Size: A
File: digital.sch
Sheet 1 of 1
Date: 15-Jul-2002
13:54:31
Mosaic Industries
CHAM DIO CPLD
CHAM MODULE BUS, 24 PIN
CHAM DIO HEADER, 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.