**R32V2020 Peripheral Memory Map**

**2019-08-29**

|  |  |
| --- | --- |
| **Address Range** | **Description** |
| x0000-x07FF (2KB) | VGA Display |
| x0800-x0FFF (2KB) | PS/2 Keyboard |
| x1000-x17FF (2KB) | SD Card |
| x1800-x1FFF (2KB) | UART (6850 ACIA) |
| x2000-x27FF (2KB) | Pushbutton Switches |
| x2800-x2FFF (2KB) | Individual LEDs |
| x3000-x37FF (2KB) | Seven Segment Display (4 or 8 digits) |
| x3800-x3FFF (2KB) | Timers |
| x4000-x47FF (2KB) | Music/Note |
| x4800-x4FFF (2KB) | LED Ring |
| x5000-x57FF (2KB) | I/O Latch |
| x5800-x5FFF (2KB) | External I2C Address |
| x6000-x67FF (2KB) | SPI Address |
| x6800-x6FFF (2KB) | EEPROM I2C Address |
|  |  |
|  |  |
|  |  |

**Board Specific Data**

* [Land Boards RETRO-EP4 Board](http://land-boards.com/blwiki/index.php?title=RETRO-EP4CE6)
* [Land Boards EP2C5-DB Board](http://land-boards.com/blwiki/index.php?title=EP2C5-DB)
* [ZrTech v2.00 EP4CE6 Board](http://land-boards.com/blwiki/index.php?title=ZrTech_V2_EP4CE6_Cyclone_IV_FPGA_EP4CE6E22C8N_Development_Board_USB_V2)
* [A-C4E6 Board](http://land-boards.com/blwiki/index.php?title=A-C4E6_Cyclone_IV_FPGA_EP4CE6E22C8N_Development_Board)
* [A-C4E10 Board](http://land-boards.com/blwiki/index.php?title=A-C4E10_Cyclone_IV_FPGA_EP4CE10E22C8N_Development_Board)
* [A-ESTF V2 EP4CE22 Board](http://land-boards.com/blwiki/index.php?title=A-ESTF_V2_EP4CE22_Board)

**Features Matrix**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Board** | **VGA (r/g/b)** | **Pushbutton/ DIP Sw** | **LEDs** | **7 Seg Disp** | **Speaker/ Buzzer** | **EEPROM** |
| RETRO-EP4 | 2/2/2 | 2/0 | 4 | None | None |  |
| EP2C5-DB | 2/2/2 | None | 1 | None | None |  |
| ZrTech 2.00 | 5/6/5 | 4 |  | 4-digit | Speaker |  |
| A4-CE6 | 1/1/1 | 4 | 12 – Ring | 8-digit | Buzzer | AT24C04 |
| A4-CE6 | 1/1/1 | 4 | 12 - Ring | 8-digit | Buzzer | AT24C04 |
| A-ESTF | 5/6/5 | 3x3 PBs(K1-9) 4 PBs 8 DIP | 8x8 matrix | 8-digit | Not sure | AT24C04 |
|  |  |  |  |  |  |  |

**ANSI (VGA) Display**

* Character display
* UART style interface
  + Address offset = 0 – Status
  + Address offset = 1 - Data
* 80 rows, 25 lines
* Support ANSI escape sequences from Grant Searle’s Multicomp
  + <http://searle.hostei.com/grant/Multicomp/index.html#ANSICodes>
* 6 bits of color (64 colors) supported
  + Normal/bold colors
* Color support varies by FPGA card (wiring to VGA resistor networks)
  + RETRO-EP4
    - 6 bits color (2/2/2)
  + A4-CE6, A4-CE10
    - 3 bits color (1/1/1)
  + ZR Tech
    - Hardware has 16 bits color (5/6/5)
    - R32V2020 only supports 6 bits

**PS/2 Keyboard (Latched and unlatched)**

There are two sets of PS/2 interfaces. They both connect to the same PS/2 connector. Either can be used by software. Or both can be used (with some care).

The Latched interface is the “normal” interface that would be used by most programs. When a key is pressed the interface indicates that a new key was pressed with a data valid signal. When the key is read the interface (data valid) gets automatically cleared.

The “polled” interface can be used by programs such as games and piano keyboard simulators where it is useful to know if a button is continually pressed. This can be used to emulate joysticks or piano keyboards where the duration of the key press controls the duration of movement or sound.

|  |  |
| --- | --- |
| **Address** | **Function** |
| x0800 | Latched Keyboard Data |
| x0801 | Latched Keyboard Status |
| x0802 | Polled Keyboard Data |
| x0803 | Polled Keyboard Status |

**Latched Keyboard Data**

* This interface latches up PS/2 keyboard key press value and strobe
* Includes PS/2 to ASCII conversion table
* PS/2 Keyboard connector is present on all hardware
* D0..d7 = ASCII keyboard data
* D8..d31 = 0

**Latched Keyboard Status**

* D0 - Data valid
  + 1 = Data valid
    - Cleared by read of Keyboard Data
  + 0 – No data present
* D1..D31 = 0

**Polled Keyboard Data**

* Useful for applications like piano keyboard where press/release of key is used
* Includes PS/2 to ASCII conversion table
* PS/2 Keyboard connector is present on all hardware
* D0..d7 = ASCII keyboard data
* D8..d31 = 0

**Polled Keyboard Status**

* D0 – Key pressed
  + 1 = Key is being pressed
  + 0 – Key is not being pressed
* D1..D31 = 0

**SD Card**

This design uses the SPI interface and supports "standard capacity" (SDSC) and "high capacity" (SDHC) cards.

**Address Register**

|  |  |  |
| --- | --- | --- |
| **Address Offset** | **Register Name** | **Read/Write** |
| 0 | SDDATA | read/write data |
| 1 | SDSTATUS | read |
| 1 | SDCONTROL | write |
| 2 | SDLBA0 | write-only |
| 3 | SDLBA1 | write-only |
| 4 | SDLBA2 | write-only (only bits 6:0 are valid) |

For both SDSC and SDHC (high capacity) cards, the block size is 512bytes (9-bit value) and the SDLBA registers select the block number. SDLBA2 is most significant, SDLBA0 is least significant.

For SDSC, the read/write address parameter is a 512-byte aligned byte address. ie, it has 9 low address bits explicitly set to 0. 23 of the 24 programmable address bits select the 512-byte block. This gives an address capacity of 2^23 \* 512 = 4GB .. BUT maximum SDSC capacity is 2GByte.

The SDLBA registers are used like this:

31 30 29 28.27 26 25 24.23 22 21 20.19 18 17 16.15 14 13 12.11 10 09 08.07 06 05 04.03 02 01 00  
+------- SDLBA2 -------+-------- SDLBA1 -------+------ SDLBA0 ------+ 0 0 0 0 0 0 0 0 0

For SDHC cards, the read/write address parameter is the ordinal number of 512-byte block ie, the 9 low address bits are implicity 0. The 24 programmable address bits select the 512-byte block. This gives an address capacity of 2^24 \* 512 = 8GByte. SDHC can be upto 32GByte but this design can only access the low 8GByte (could add SDLBA3 to get the extra address lines if required).

The SDLBA registers are used like this:

31 30 29 28.27 26 25 24.23 22 21 20.19 18 17 16.15 14 13 12.11 10 09 08.07 06 05 04.03 02 01 00  
 0 0 0 0 0 0 0 0+------- SDLBA2 --------+-------- SDLBA1 -------+-------- SDLBA0 -------+

The end result of all this is that the addressing looks the same for SDSC and SDHC cards.

**SDSTATUS (Read Only)**

|  |  |
| --- | --- |
| **Bit** | **Description** |
| b7 | Write Data Byte can be accepted |
| b6 | Read Data Byte available |
| b5 | Block Busy |
| b4 | Init Busy |
| b3 | Unused. Read 0 |
| b2 | Unused. Read 0 |
| b1 | Unused. Read 0 |
| b0 | Unused. Read 0 |

**SDCONTROL (Write Only)**

|  |  |
| --- | --- |
| **Bit** | **Description** |
| b7 | 0 = Read Block 1 = Write Block |
| b6 | N/A |
| b5 | N/A |
| b4 | N/A |
| b3 | N/A |
| b2 | N/A |
| b1 | N/A |
| b0 | N/A |

*To read a 512-byte block from the SDCARD:*

* Wait until SDSTATUS=0x80 (ensures previous cmd has completed)
* Write SDLBA0, SDLBA1 SDLBA2 to select block index to read from
* Write 0 to SDCONTROL to issue read command
* Loop 512 times:
  + Wait until SDSTATUS=0xE0 (read byte ready, block busy)
  + Read byte from SDDATA

*To write a 512-byte block to the SDCARD:*

* Wait until SDSTATUS=0x80 (ensures previous cmd has completed)
* Write SDLBA0, SDLBA1 SDLBA2 to select block index to write to
* Write 1 to SDCONTROL to issue write command
* Loop 512 times:
  + Wait until SDSTATUS=0xA0 (block busy)
  + Write byte to SDDATA

At HW level each data transfer is 515 bytes: a start byte, 512 data bytes, 2 CRC bytes. CRC need not be valid in SPI mode, \*except\* for CMD0.

SDCARD specification can be downloaded from

https://www.sdcard.org/downloads/pls/

All you need is the "Part 1 Physical Layer Simplified Specification"

**UART (6850 ACIA)**

* Programmed via the 6850 ACIA interface
  + Control/Status Register
  + Read/Write data register
* A4-CE6, A4-CE10, ZrTech boards
  + Connects to USB to Serial interface
* RETRO-EP4 board
  + Connects to on-board FTDI Interface adapter
* EP2C5-DB board
  + 4-pin RX/TX/RTS/GND connector
    - Attach via serial cable to FTDI USB-TTL adapter

**Pushbutton Switches/DIP switches**

* Not all FPGA cards have pushbuttons and/or DIP switches
* Pushbutton switches and DIP switches are packed into 16-bit read value
  + D0-D2 = Pushbuttons
  + D3 = 0
  + D4-D11 = DIP switches
* (3) Pushbutton switches are de-bounced in hardware
* DIP Switches are not de-bounced in hardware
* Value of bits
  + 0 = Button pressed
  + 1 = Button not pressed
* Reset pushbutton can’t be read
  + Reset function activates when button is released
* A4-CE6, A4-CE10 boards
  + One pushbutton (K5) resets the CPU
  + Three readable pushbuttons
  + 8 position DIP switch
* RETRO-EP4 board
  + Reset and nCONFIG (not readable)
  + No pushbutton or DIP switches
* ZrTech board
  + Reset and 3 general purpose pushbutton switches

**Individual LEDs**

* Not all FPGA cards have LEDs

**Seven Segment Display**

* 32-bit data register

**Timers**

* Four Timers
* 32-bit data
* Read-Only
* Addresses

|  |  |
| --- | --- |
| Address | Timer |
| X3800 | Elapsed Time Counter   * FPGA clocks * Typically 50 MHz ticks (20 nS) |
| X3801 | MicroSeconds Counter |
| X3802 | Milliseconds Counter |
| X3803 | CPU Instruction Counter |

* Determine time by reading counter and then re-reading counter.

**Music/Note**