Terence Henriod

HW 11

CPE 301: Microprocessors System Design

November 14, 2013

1. Design an ATmega2560 microcontroller (Harvard architecture) system memory address decoding circuit which has a single 8Kx8 RAM chip starting at address 0x8000 and an ADC0820 Analog to Digital Converter starting at address 0xC000 so that they are both mapped as external DATA memory. Use ABSOLUTE decoding with no fold-back for the external DATA memory. Show your work and include the following items:
   1. MEMORY MAP

|  |  |
| --- | --- |
|  | ^^^ FFFF |
|  |  |
|  |  |
| ADC 0820 vvv | VVV C000 |
|  |  |
|  |  |
| RAM ^^^ | End: 9FFF |
| vvv | Start:8000 |
|  | ^^^ 7FFF (Halfway point) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | vvv 0 |

* 1. MEMORY ADDRESS TABLE

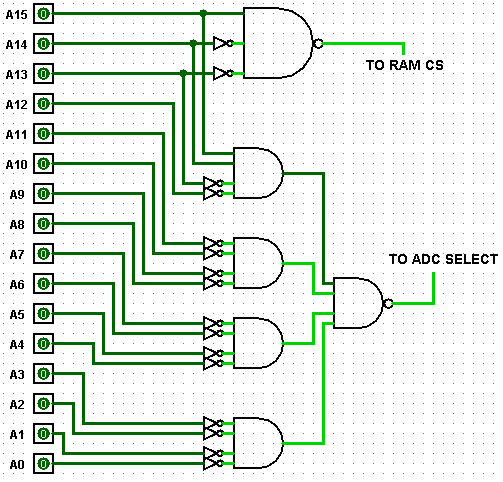
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Chip/  Device | Hex Addresses | | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
| RAM | Start: | 0x8000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| End: | 0x9FFF | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ADC | Start: | 0xC000 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| End: | 0xC000 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* 1. BOOLEAN EQUATIONS for each chip select signal

RAM =

ADC =

* 1. Full schematic logic diagram of your DECODING CIRCUITS



1. Using AVR C, write a function which will first send a start of conversion signal to the ADC0820 Analog to Digital converter (as defined in question 1) to start a conversion, then loop until the end of conversion signal goes active, and then read a single digital value from the ADC0820 when it is ready. The function will return the unsigned char value read from the ADC0820. The ADC0820 active low INT\* (EOC) signal is connected to the ATmega PortC bit 0 GPIO pin. NOTES: a. Assume the GPIO pins are predefined and initialized for the proper I/O directions with the available Global pointers: volatile unsigned char \*pinC = (unsigned char \*) 0x0026; volatile unsigned char \*portC = (unsigned char \*) 0x0028; b. You must define any other necessary pointers to the ADC0820. c. A high-to-low-to-high pulse on the ADC0820 WR\* pin (SOC) will start a conversion as shown on the ADC0820 timing diagram. When the end of conversion (EOC) occurs the ADC0820 INT\* pin will go low. To read the digital value you need to simply read external data memory at the address defined in question 1.

unsigned char pollADC0820()

{

// variables

unsigned char\* theADC = (unsigned char\*) 0xC000;

// note: there is no SCR for the 0820, unlike the ADC built into Arduino Mega

// start the conversion

\*theADC = 1; // writing anything will pulse the ADC0820 and start the conversion

// wait for the end of conversion bit

while( ( \*pinC & 0x01 ) == 0x01 ) {}; // EOC is active low in this case so we need to

// wait for a low signal in bit 0

// return the data held by the ADC

Return \*theADC;

}

1. The Raspberry Pi single board computer (SBC) is built around the ARM11 processor. The Pi’s microcontroller is a Broadcom BCM2835 built around this ARM CPU. This microcontroller contains what is referred to as a “mini” UART. Define the necessary pointers and write the following four C language functions for the mini UART. Consult the BCM2835 User’s Manual for the necessary information.
   1. Pointers

volatile unsigned char\* serialData = (unsigned char\*) 0x7E215040; // data register

volatile unsigned char\* dataFormat = (unsigned char\*) 0x7E21504C; // for char size

volatile unsigned char\* serialStatus = (unsigned char\*) 0x7E215054; // for TBE and RDA

volatile unsigned char\* serialControl = (unsigned char\*) 0x7E215060; // to enable UART

volatile unsigned short\* baudRegister = (unsigned short\*) 0x7E215068; // baud rate reg

* 1. sinit(baud)

void sinit( const int baud )

{

// variables

unsigned long int systemFreq = 250000000; // seems to be default system clock speed

unsigned short baudRegVal = 0; // short for 16 bit type

// according to documentation, interrupts are disabled by default.

// set the data size to 8 bits

\*dataFormat |= 0x01; // sets data size format to 8 bits

// turn on the mini-UART

serialControl |= 0x03; // Tx enable and Rx enable are in bits 1:0;

// calculate the baud rate

baudRegVal = (short)( ( systemFreq / ( 8 \* baud ) ) – 1 );

// set the baud rate

baudRegister = baudRegVal;

// no return - void

}

* 1. kbhit(void)

unsigned char kbhit()

{

// return the truth of received data waiting

return ( \*serialStatus & 0x01 ); // data available flag in bit 0

}

* 1. getchar(void)

unsigned char getchar()

{

// return the received data waiting to be read

Return \*serialData;

}

* 1. putchar(char cname)

void putchar( const char cname )

{

// wait for previous data to finish transmitting

while( ( \*status & 0x20 ) == 0 ) {}; // Transmit Buffer Empty flag in bit 5

// send the data

\*serialData = cname;

// no return - void

}