

AVR DESIGN CONTEST 2004 – ENTRY NUMBER: A3711

The Image Capture Board (ICB) is designed to capture images at a resolution greater than 3600dpi with 8-bit grayscale depth. This board was created as the central component of a system capable of detecting counterfeit currency. Scanning at such a high resolution would allow for easy detection of micro printed characters as well as the ability to measure line widths and other dimensions.

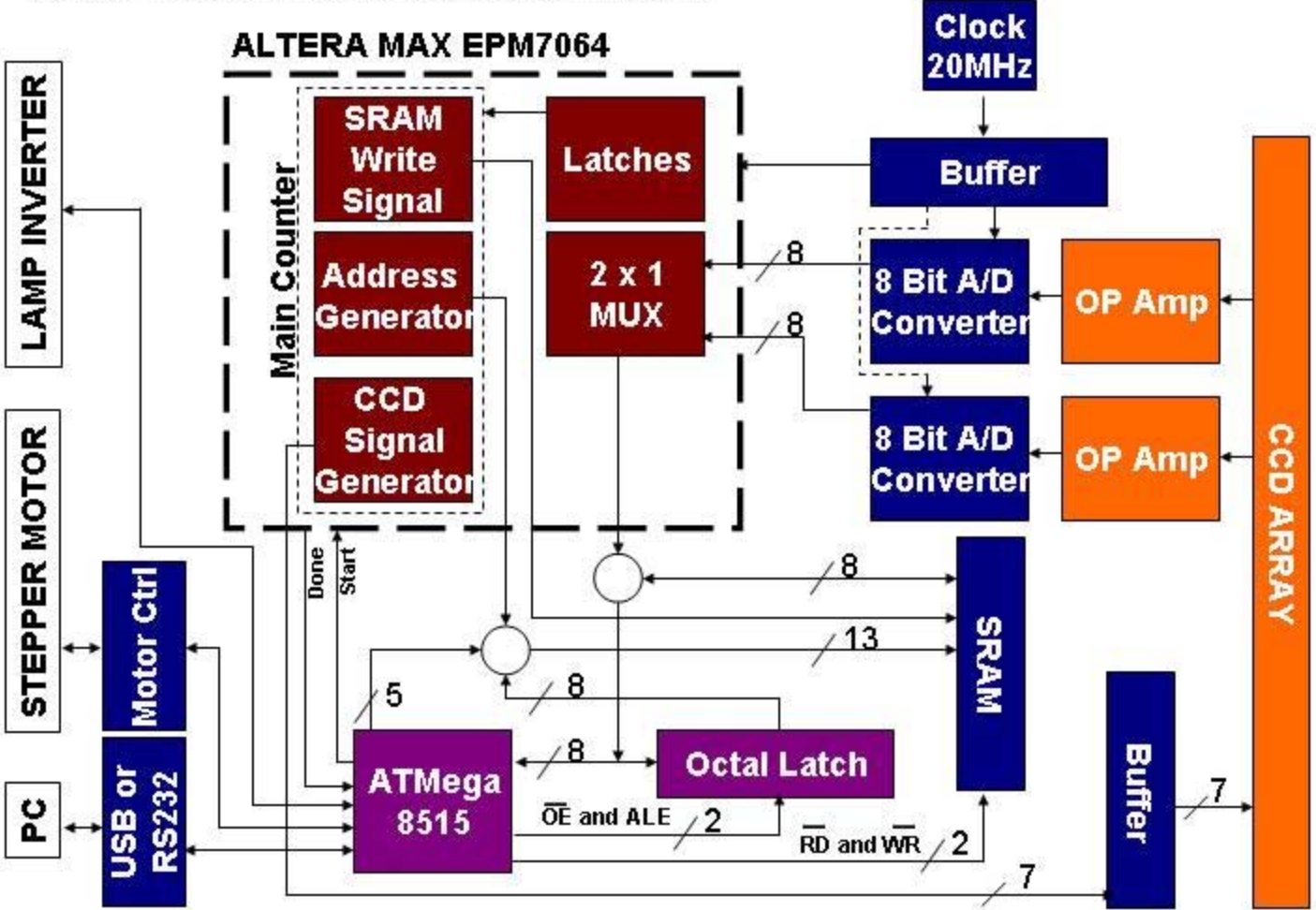
Counterfeiting is a problem in many countries, and the challenge that most governments face is how to include security features that will be both obvious to an observer, as well as difficult to replicate due to technological barriers. Many retail businesses still rely on visual inspection of bills in order to validate their authenticity. This may involve holding the bill up against the light to see a watermark, identifying color shifting ink or noticing a blurry or poorly replicated image. The main problem with this method is that the detection is left up to the observer, which is likely to yield some error.

In order to correct this problem the ICB would be used to capture an image of part of the bill, perform image filtering, and transfer the data to a PC. The PC would then be able to identify any micro print on the bill, along with validating line widths and distances to 7um. Since the ICB is able to scan at a maximum resolution of 3600+ dpi, the current micro print can be even smaller in order to make reproduction next to impossible. In this application, the ICB would be placed in an enclosure similar to those used to swipe charge cards so that each bill could be easily verified.

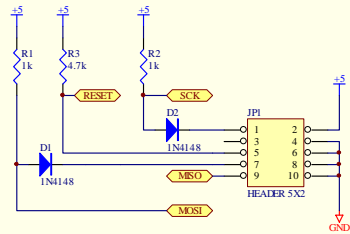
Although the original design was intended for counterfeit detection, the ICB is flexible enough to incorporate into other systems. For example, if a RGB backlight was used in place of the lamp, the scanner would be capable of capturing slides and/or negatives with 24 bit color depth.

The ICB is based on an ATmega8515 that shares external SRAM with an Altera CPLD. Using either RS232 or USB, it can transfer the image data to another external device (such as a PC) for further processing. The CCD line array is a Toshiba 1502C, and is made up of 5000 elements 7um x 7um with a center-to-center distance of 7um. In order to test the ability of the ICB, a scanner bed complete with motion controller was constructed, along with a PC application to view the raw image data.

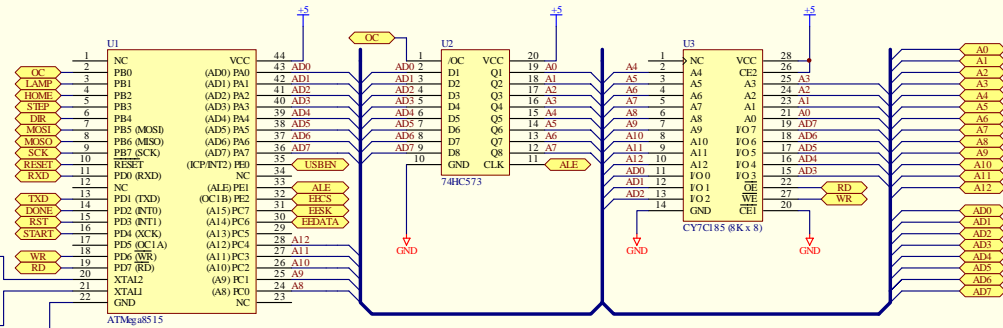
Image Capture Board (ICB) Block Diagram



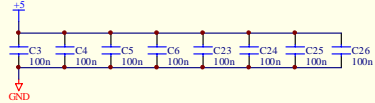
AVR Programming Connector



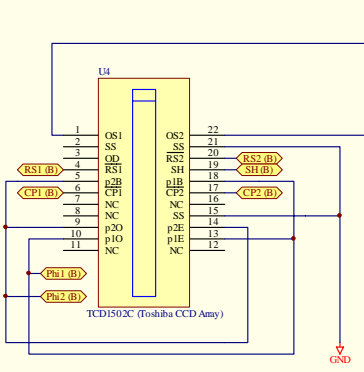
ATMega8515 8K SRAM Connection



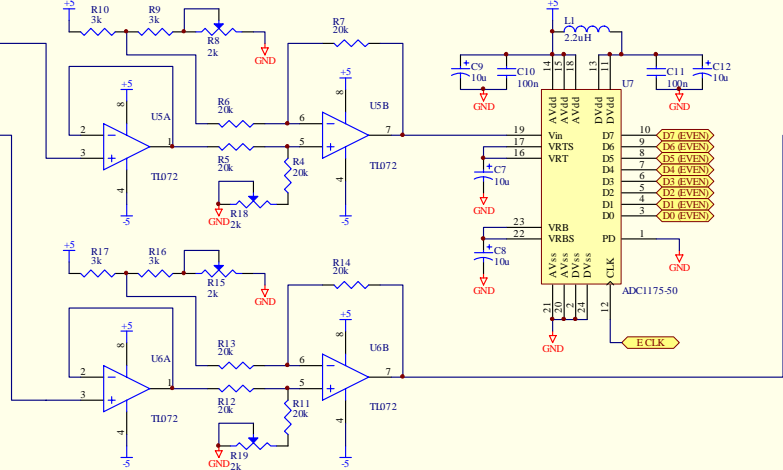
Decoupling Capacitors



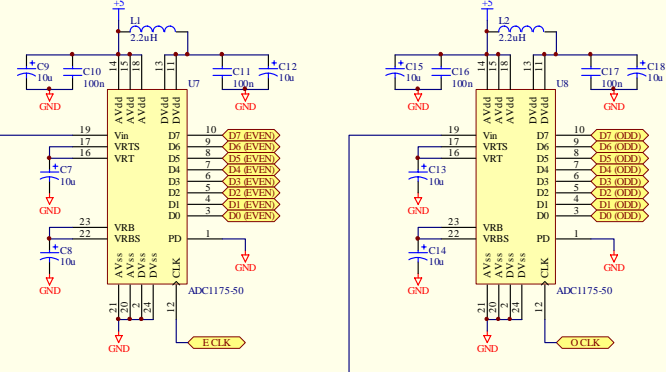
TCD1502C CCD Array



OP Amps



A/D Converters



Title		
Image Capture Board - Entry A3711		
Size	Number	Revision
B		
Date	Sheet 1 of 2	

Code Snippet from PAR Protocol

```

/*****
/* Main State Machine */
/*****
void SM_ProcessMessages(void)
{
    unsigned char ReceivedByte;

    /* Check for Data in Receive Buffer */
    if (DataInReceiveBuffer() && MsgRxBufferStatus != VALID)
    {
        ReceivedByte = ReceiveByte(); /* Store Byte */

        switch (MsgRxBufferPosition) { /* Process Byte */

            /* Identify incoming message */
            case 0:
                if (ReceivedByte == 0xA5) /* Check for
Header */
                {
                    MsgRxBufferData[MsgRxBufferPosition++] = ReceivedByte;
                    MsgRxBufferStatus = INPROGRESS;
                }
                else if (ReceivedByte == 0x06) /*
Acknowledgement */
                {
                    MsgRxBufferPosition = 0;
                    MsgRxBufferStatus = ACK;
                    MsgTxBufferStatus = EMPTY;
                    MsgTxBufferAttempts = 0;
                }
                else if (ReceivedByte == 0x15) /* Non-
Acknowledgement */
                {
                    MsgRxBufferPosition = 0;
                    MsgRxBufferStatus = NACK;
                }
                else /*
Undetermined */
                {
                    MsgRxBufferPosition = 0;
                    MsgRxBufferStatus = EMPTY;
                }
                break;

                /* If it is not an ACK/NACK, then check which
message it is */
                case 1:
                    if ((ReceivedByte & 0xF0) == 0xF0) /*
Message ID (0xFX) */
                    {
                        /* Position 1 in the buffer
records the message ID */

```

```

MsgRxBufferData[MsgRxBufferPosition++] = ReceivedByte;
    }
    else
    {
        MsgRxBufferPosition = 0;
        MsgRxBufferStatus = EMPTY;
    }
    break;

    /* Store the number of data bytes in the message
*/
    case 2:
        /* Message too large, purge message */
        if (ReceivedByte >= MSG_BUFFER_SIZE)
        {
            MsgRxBufferPosition = 0;
            MsgRxBufferStatus = EMPTY;
        }
        else
        {
            /* Position 2 in the buffer
records the # of bytes */
MsgRxBufferData[MsgRxBufferPosition++] = ReceivedByte;
        }

        break;

        /* Get the remaining Bytes */

    default:
        MsgRxBufferData[MsgRxBufferPosition++] =
ReceivedByte;

        /* LRC is the last byte */
        if (MsgRxBufferPosition ==
(MsgRxBufferData[2] + (unsigned char)4) )
        {
            if
(! (CalculateLRC((MsgRxBufferData[2] +
MsgRxBufferData)))
                (unsigned char)4),
ACK */
                {
                    TransmitByte (0x06); /* Send
MsgRxBufferStatus = VALID;
                }
            else
            {
                TransmitByte (0x15); /* Send
NACK */
                MsgRxBufferStatus = EMPTY;
            }
            MsgRxBufferPosition = 0;
        }
    }
}

```

```

                break;
            }
        }
    }

/*****
/* External Function Interface */
*****/
unsigned char SM_CheckForValidMessage(void)
{
    RetransmissionHandler();
    SM_ProcessMessages();

    /* Check if there is a valid message */
    if (MsgRxBufferStatus == VALID) return TRUE;
    else return FALSE;
}

/*****
/* Retransmission Handler */
*****/
void RetransmissionHandler (void)
{
    if ((MsgTxBufferStatus & WAITFORACK) == WAITFORACK)
    {
        /* If timer expires, retry up to MSG_TX_RETRIES times */
        if (!(GetTimer(TMR_MESSAGE_RETRY)))
        {
            if (++MsgTxBufferAttempts <= MSG_TX_RETRIES)
                TransmitMessage();
            else
            {
                MsgTxBufferStatus = EMPTY;
                MsgTxBufferAttempts = 0;
            }
        }
    }
}

/*****
/* Send the scanned line data, unbuffered */
*****/
void SendScannedLine (unsigned char group, unsigned char oddeven)
{
    /* Special case message, where data is unbuffered */
    /* Will allow message to be quite long */
    unsigned char LRC = 0;
    unsigned char *CCDBuffer;
    unsigned long count;
    unsigned long start, end;

    /* Calculate the start and end position of the group */
    /* Total of 2500 even and odd pixels */
    start = ((unsigned long)group) * 250;

```

```

end = start + 250;

/* Set start location of buffer */
CCDBuffer = (unsigned char *)CCD_BUFFER_START;
CCDBuffer += start;

/* If the request is for odd pixels, shift the SRAM start
location */
if (oddeven) CCDBuffer += 0x1000;

LRC ^= 0xA5; /* Header
*/
LRC ^= MSG_SCANNER_SEND_SCAN_LINE; /* Message ID
*/
LRC ^= 0xFA; /* Data = 250
*/

TransmitByte (0xA5); /* Prepare header for message */
TransmitByte (MSG_SCANNER_SEND_SCAN_LINE);
TransmitByte (0xFA);

SRAMControl(ENABLE); /* Enable External SRAM */
for (count=0;count<(end-start);count++)
{
    LRC ^= *CCDBuffer; /* Send Message */
    TransmitByte (*CCDBuffer);
    CCDBuffer++;
}
SRAMControl(DISABLE); /* Disable External SRAM */

TransmitByte (LRC);
}

```