Design of Weave Pattern Model for the Selection Box Driving Software

Mustafa Dulger
Mechanical Engineering Department, Faculty of Engineering
University of Istanbul Cerrahpasa, Istanbul 34320, Turkey
E-mail: mdulger @istanbul.edu.tr

Abstract
A PC driven electronic selection box, simply *Selection Box,* together with its driving software, is designed to replace perforated card feed pattern system in a jacquard mechanism used on classical weaving looms. A new *Weave Pattern Model* for the *Selection Box*'s driving software is developed and presented in this paper.
The developed *Weave Pattern Model* provides access to a weave pattern and serves as an underlying data structure for the *Selection Box's* driving software. Further a feed protocol, by which holes sequence on the perforated card for a given weave pattern is determined, is automated.
The dynamic link library, *MCreator.dll*, keeping the implementation of *Weave Pattern Model* is developed in C++ language. A small client application, *HLChanger.exe*, testing the library is also developed. *HLChanger.exe*, besides testing the library, enables user change *Colour Stack* of the *Weave Pattern Model* and thus permitting the visualisation of the *Weave Pattern*. The library is available from the author on request. An example weave pattern is worked out on this client program and the result is presented at the end.

Keywords: *Weave Pattern*, *Weave Pattern Model*, Perforated Card Protocol, Perforated Card, Jacquard Mechanism, Solenoid Matrix, Loom*, Image Matrix*, *Colour Stack*, *Pattern Interface, Protocol Interface*.

1. Introductıon

A study on the modernisation of classical jacquard mechanism on weaving loom is carried out by Dülger [1]. In this study, group of *Solenoid Matrixes* forming a *Selection Box* is designed to replace perforated card feed system on a classical jacquard mechanism. For each hole in the perforated card, there is one solenoid in the *Solenoid Matrix*. Each solenoid is responsible for either opening or closing the hole assigned to it. In this context, the *Solenoid Matrix* acts as a programmable perforated card [1] [2] [3].
The driving application software package HL [1] running on an IBM-PC is developed to control and drive the *Selection Box.* A PC (Personal Computer), provided with a 32-Bit IO card, serves as a controller for the *Selection Box*. The application software package HL, converts pixel and colour information of the *Weave Pattern* into respective TTL signals at the output ports of the IO card. A certain protocol named as *five-coloured feed protocol* is used in converting *Weave Pattern* information into TTL signals. These signals are then fed into the *Selection Box*. The *Selection Box* restores states of solenoids to simulate the perforated card that should be in mesh for the current stroke of the loom.
In this manuscript, The *Weave Pattern Model* is going to be discussed in detail. The *Weave Pattern Model* is developed for the HL software package, to enable it get easy access to the *Weave Pattern.*

2. Weave Pattern Model - Overview

Block diagram of the *Weave Pattern Model* is given in Figure 1. As seen from the figure, the *Weave Pattern Model* is made up of two main components. They are the set of *Interfaces* and the *Data Model*.

2.1. Interfaces

The first interface in the model is the *Pattern Interface* and defines virtual gate functions to provide access to the elements of *Weave Pattern*. The second interface is the *Protocol Interface* and provides virtual gate functions to access perforated card protocol.

2.1.1. Pattern Interface

The *Pattern Interface* is designed to provide virtual access functions for the components of the *Weave Pattern Model*. The *Processor* object implements the *Pattern Interface*. The *Pattern Interface* has following functions;

- ReadBmp (BSTR filename);
- Convert (int *nFromTo*) ;
- ReplaceColour (DWORD *oldColour*, DWORD *newColour*)
- WriteBmp (BSTR *filename*)
- get_Bitmap (BYTE** *ppVal*);
- put_Bitmap (BYTE* *pVal*);
- GetColourArray (int *iCl*, BYTE ***ppClmArray*)
- GetRowArray (int *iRw*, BYTE ***ppRowArray*)

ReadBmp function read the pattern from a stream file. The stream file is located by the parameter *filename.* After the successful call of the function, new objects of the *Data Model* are recreated as the old ones are deleted.

Convert function builds either *Image Matrix* object from *Weave Pattern* object or vice versa depending upon the value of the parameter *nFromTo*. If the parameter *nFromTo* has the value BMPTOI = 1, the *Image Matrix* object is updated from *Weave Pattern* object and if the parameter *nFromTo* has the value ITOBMP = 0, *Weave Pattern* object is updated from *Image Matrix* object.

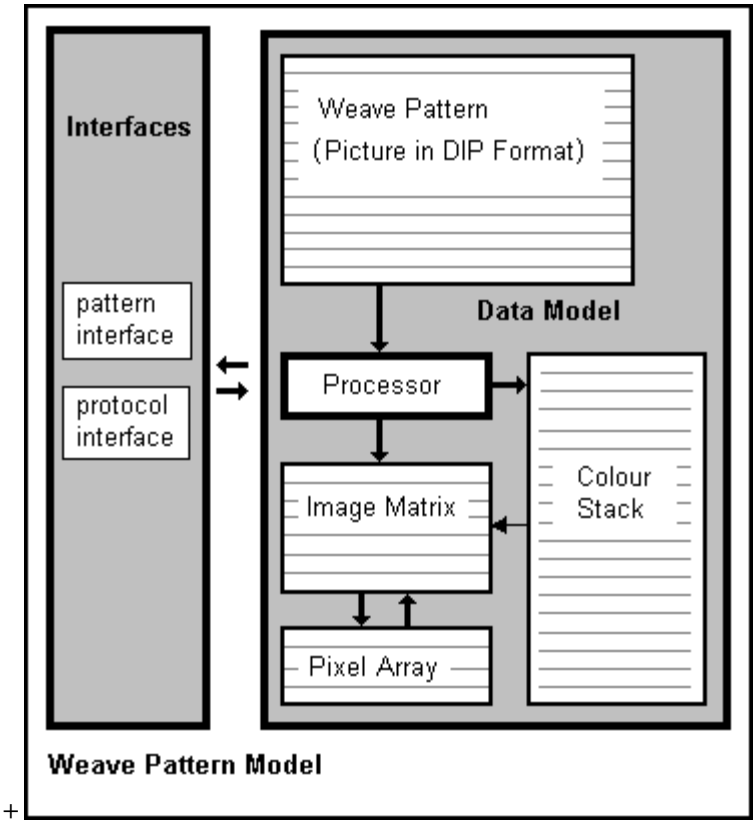

Figure 1. Weave Pattern Model

ReplaceColour function replaces the colour entity in *Colour Stack* referenced by the first parameter *oldColour* with the colour entity given by the second parameter *newColour.*

WriteBmp function stores *Weave Pattern* object into the stream file located by the parameter *filename.*

Element functions *get_Bitmap* and *put_Bitmap* provide access to the *Weave Pattern* object stored in DIB format. The parameter in the *get_Bitmap* function *ppVal* is the address of the pointer on the buffer of bytes and passed to the function. After the successful call of the function, the buffer keeps the image of the *Weave Pattern* object. The *put_Bitmap* function replaces the content of the *Weave Pattern* object with the content of the buffer which is pointed by the parameter *pVal.*

The *GetColourArray* function provides access to the colour references of the column of the *Image Matrix* indexed by the parameter *iCl.* The parameter *ppColourArray* keeps the address of the pointer on the buffer of bytes and passed to the function. After successful call of the function, the buffer keeps colour references for the indexed column of the *Image Matrix.* Similarly the *GetRowArray* function returns colour references for the row of the *Image Matrix* indexed by the parameter *iRw*. The parameter *ppRowArray* keeps the address of the returned buffer pointer.

2.1.2. Protocol Interface

In the classical jacquard mechanism data feed is done through the perforated card implementation. The commonly used perforated card layout for carpet looms has 15 rows and 84 columns as illustrated in Figure 2. This layout enables controlling of maximum 15 x 84 = 1260 needles in the loom.

Holing of the perforated card is performed according to a certain feed protocol. There are of course different feeding protocols. Discussing all of these protocols is out of the scope of this study. The wide-spread *five-coloured feed protocol* will only be explained here because it is the dominating protocol currently used in the industry.

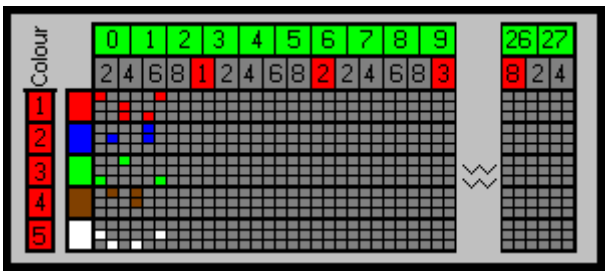

Figure 2**.** Perforated card with five-coloured feed protocol

In the *five-coloured feed protocol* there are five main rows and each of which can have three sub-rows on a perforated card. Each of the main rows corresponds to different colour in the *Weave Pattern*. The *Weave Pattern* hence have maximum of five colours.

Each column, composing of five main rows, represents three successive pixels in the *Weave Pattern.* Therefore only three of fifteen locations should be holed.

Sub-rows in a main row describe the location of pixels in order. If the successive order of pixels in *Weave Pattern* is *red*, *white* and *green*, the first sub-row of the red row, the second sub-row of the white row and the third sub-row of the green row must be holed as indicated in the first column of Figure 2.

There exist 84 columns in a single perforated card. They represent total of 84x3 = 252 successive pixels. Hundreds of perforated cards are hence required for even a simple *Weave Pattern*.

Holes sequence, *Hs*, is defined as 16 bits length word. It simulates one column in the perforated card. As there are only 15 hole locations (rows) in the column, each location is represented by one bit in the holes sequence. The first location corresponds to *Least Significant Bit*, LSB, of the holes sequence. The last location, that is the row 15 in the column corresponds to 15^{th} bit of the holes sequence. If the location is holed, the corresponding bit value in the holes sequence is 1 and if it is non-holed the bit value is 0. For example, the holes sequence for the first column in the perforated card shown in Figure 2. is indicated in binary and hexadecimal form as follow,

Hs = (0 010 000 100 000 001)b = 0x2101

In this column the colours of the first three successive pixels in the *Weave Pattern* are given in such a way that the colour of the first pixel is the first main colour, the colour of the second pixel is the fifth main colour and the colour of the third pixel is the third main colour.

The *Protocol Interface* provides gate functions implemented for the *five-coloured feed protocol*. The *Processor* object makes the implementation of the *Protocol Interface.*

The functions of the *Protocol Interface* are,

- GetRowSequence (int *nRwLine*, WORD ***ppHoles*);
- GetSequence (WORD** *ppHoles*) ;

GetRowSequence function returns the word buffer indicating the holes sequences. Each element of the buffer represents three successive pixels of the row indexed by the parameter *nRwLine*. The pointer to the address of the word buffer is given by the parameter *ppholes* and passed to the function. After the successful call of the function, the word buffer keeps the holes sequences. The size of buffer depends

upon the length of the row of the *Weave Pattern.* The last element of the word buffer keeps the value of 0xFFFF signalling the end of the buffer.
GetSequence function does the same, not for the single row but also for the entire *Weave Pattern*. The returned word buffer holds the sequences of row lines starting by the first row and ending by the last row.

2.2. Data Model
The *Data Model* keeps internal components to store and manipulate *Weave Pattern*. These components are *Weave Pattern*, *Image Matrix*, *Pixel Array, Colour Stack* and *Processor.*

2.2.1. Weave Patten
The *Weave Pattern* is a picture stored in a *Device Independent Bitmap* (DIB) format. DIB format is developed by IBM [4] and Microsoft [5] corporations. One can refer to Note 1. for more information about DIB format.

2.2.2. Image Matrix
The *Image Matrix* is the key component of the *Weave Pattern Model*. *Weave Pattern* is mapped to the *Image Matrix* in such a way that the first row of the *Weave Patten* becomes the last row in the *Image Matrix s*o that *Image Matrix* is in one to one correspondence with the weave pattern. That means, upper left pixel in the weave pattern is represented in the first row and first column of the *Image Matrix* and lower right pixel of the real weave pattern is represented in the last row and last column of the *Image Matrix.*
The *Image Matrix* keeps reference values on a colour entity in the *Colour Stack* for each pixel in the *Weave Pattern.* These reference values have only meaning if a predefined *Colour Stack* exists. The term "*reference on a colour entity"* needs a little bit more explanation. The relative position of the colour entity in the *Colour Stack* to the stack begin is defined as a reference on the colour entity. It is obvious that if the *Colour Stack* is changed the appearance of the *Weave Pattern* would also be changed although the *Image Matrix* remains unchanged.

2.2.3. Pixel Array
The *Pixel Array* is nothing else than keeping the *Image Matrix* in an array. Allocation of the *Image Matrix* entities in the *Pixel Array* is done as follow

$$\text{PixelArray}[k] = \text{ImageMatrix}[i, j] \quad (1)$$

where,

k	:	$(j\text{-}1) \times nCl + i$.
nCl	:	number of columns in the *Image Matrix*.
nRw	:	number of rows in the *Image Matrix*.
i	:	column index ranging from 1 to nCl in the *Image Matrix.*
j	:	row index ranging from 1 to nRw *in* the *Image Matrix.*

It can be concluded from equation (1) that the allocation of the rows in the *Pixel Array* starts from the first row and ends with the last row of the *Image Matrix*.
Pixel Array is constructed to achieve the compatibility with the standard DIB's *Image Array*. It requires, however, a slight modification because DIB's *Image Array* keeps the *Weave Pattern* in reverse row order. That means, it starts with the last row and ends with the first row. The correlations between *Image Array* of *Weave Pattern Model* and DIB's *Image Array* is

$$DIBImgArry[L] = ImageMatrix[i, j] \quad (2)$$

where,

L	:	$(nRw\text{-}j) \times nCl + i$

Figure 3. gives the correlation between *Image Matrix*, *Colour Stack* and *Pixel Array* schematically.

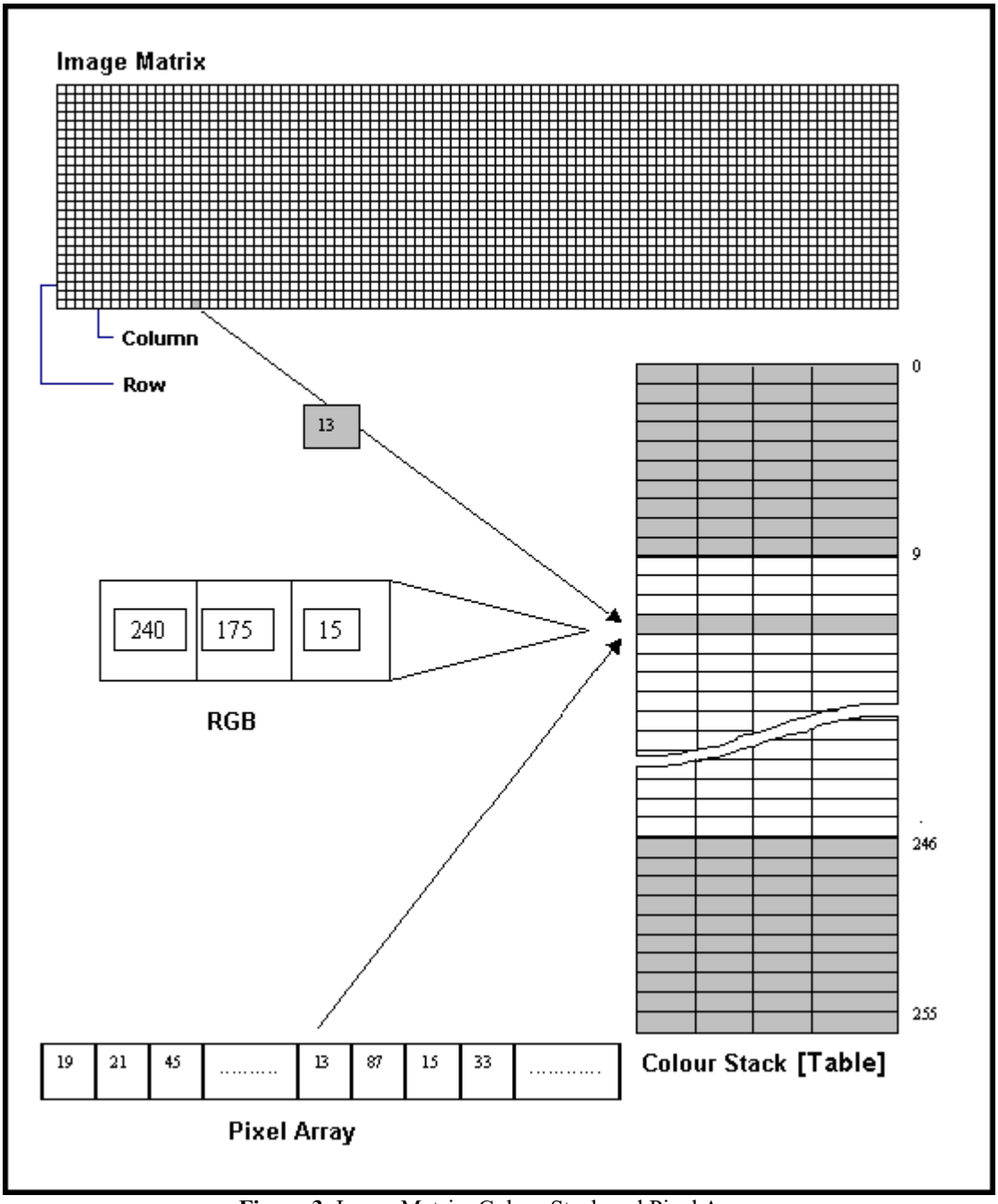

Figure 3. Image Matrix, Colour Stack and Pixel Array

2.2.4. Colour Stack

The Colour *Stack* is the proxy of the RGBQUAD array of the *Weave Pattern* (one can refer to Note 1 for more information about RGBQUAD array). The *Weave Pattern's* RGBQUAD array is hence reconstructed by the *Colour Stack* of the *Weave Pattern Model.*

Each element in the *Colour Stack* has a size of double words and represents an object of RGB colour structure. RGB stand for *Red-Green-Blue* colours. They are basic colour. Any colour, different than basic colours, can be expressed in terms of basic colours. The RGB colour structure in C++ syntax is defined as

DWORD RGB {BYTE *bRed*, BYTE *bGreen*, BYTE *bBlue*};

where,

bRed is the red component of the colour, *bGreen* is the green component of the colour and *bBlue* is the blue component of the colour.

RGB colour structure is widely supported by nowadays computer technology. By combining basic colours in different intensities, it is possible to get huge amount of different colours. Basic colours have an intensity range of 0-250. Hereby number 0 indicates no intensity and number 250 indicates full intensity. For example {255,0,0} correspond pure red, {0,255,0} pure green and {0,0,250} pure blue.
Each *Weave Pattern* has its own *Colour Stack*. One must not make much consideration about *Colour Stack* as it is integral part of DIB format.

2.2.5. Processor
The *Processor* is the operational component of the *Weave Pattern Model.* It simply converts *Weave Pattern* in DIB format into the *Image Matrix* and creates corresponding *Colour Stack*. In this conversion only the DIB image array of the *Weave Pattern* is converted to the *Image Matrix* of the *Weave Pattern Model*. This conversion can be carried out in reverse direction as well. That means if the *Image Matrix* and its *Colour Stack* exist, the *Processor* rebuilds the corresponding *Weave Pattern.*

3. Discussion & Conclusion
The *Weave Pattern Model* developed in this study provides full access to the *Weave Pattern*. The *Weave Pattern,* in which the picture is kept in *Device Independent Bitmap* format, is modelled as a matrix of colour references and resulting matrix is named as *Image Matrix*. Two interfaces are designed to provide full access to the *Weave Pattern Model*.
The entire implementation for the *Weave Pattern Model* is carried out in programming language C++ for the Microsoft Windows platform. A dynamic link library module named *MCreator.dll*, is designed for this purpose. The Weave Pattern Model is designed as COM objects (Component Object Model) [6] [7]. Any client application linking with this library has an access to the *Weave Pattern Model*. The library is entirely used by the *Selection Box* Driving Software *HL*, and demonstrates full success in all functions.
The header file for the *Weave Pattern Model* is given in Note 2. All declarations related with the *Weave Pattern Model* are given in this header file.
In order to test the library, a small test program *HLChanger.exe* is developed. In this test program, the *Weave Pattern* is visualized. The *Colour Stack* in the *Weave Pattern* has five different colours. The user can pick any of these colours in the *Colour Stack* and replace it with another colour. The *Weave Pattern* is redrawn by using the changed *Colour Stack.* Colour change is performed by using the *Pattern Interface.* Snapshots showing the same *Weave Pattern* with different *Colour Stack* are taken from the test program *HLChanger.exe* and given through Figure 6. and Figure 7. in Note 3.

As a result following points can be drawn.

- A new *Weave Pattern Model* is designed.
- The *Weave Pattern Model* incorporates a *Weave Pattern* stored in *Device Independent Bitmap* (DIB) format.
- Keeping the *Weave Pattern* in DIB format extremely eases *Weave Pattern* generation because there are plenty of bitmap editors available.
- Two interfaces are designed providing full access to the *Weave Pattern Model.*
- The *Weave Pattern Model* is fully implemented in C++ and a Dynamic Link Library having the COM object of *Weave Pattern Model* is developed.
- *Pattern Model* provides full functionality for the *Selection Box* driving software, *HL.*

Symbols & Abbreviations

HL	:	Selection Box Driving Software
HLChanger	:	Client Test Program testing *Weave Pattern Model*
IBM-PC	:	IBM Personal Computer
IO	:	Input / Output
IO32	:	32 Bit Input / Output Card
HL	:	Selection Box Driving Program
TTL	:	Transistor-Transistor-Logic
DIB	:	Device Independent Bitmap
OS/2	:	Operating System 2 by IBM for IBM-PC
i,	:	Colour index
j	:	Row index
iCl	:	Colour index

iRw	:	Row index
nCl	:	Number of colours in the *Image Matrix*
nRw	:	Number of rows in the *Image Matrix*
ImageMatrix[- ,-]	:	Two dimensional *Image Matrix*
PixelArray[]	:	One dimensional *Pixel Array*
DIBImgArry[]	:	One dimensional DIB image array
BYTE	:	8-Bit unsigned character
WORD	:	2-BYTE unsigned character
DWORD	:	Double WORD character
int	:	integer
ITOBMP	:	Enumerator, image to bitmap (= 0)
BMPTOI	:	Enumerator, bitmap to image (= 1)
BSTR	:	B-String (system allocated character string)
Filename	:	file name parameter of type BSTR
nFromTo	:	convert parameter of type integer
oldColour	:	colour parameter of type DWORD
newColour	:	colour parameter of type DWORD
ppClmArray	:	address of pointer to colour array
ppRowArray	:	address of pointer to row array
LSB	:	Least Significant Bit
Hs	:	Array of holes sequence in WORD

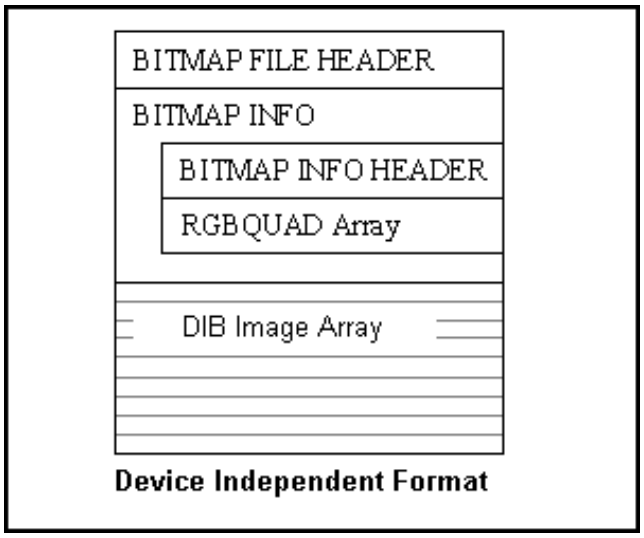

Figure 4. Device Independent Bitmap Format Structure

Notes

Note 1: Device Independent Bitmap Format

DIB has been defined by IBM and Microsoft corporations. Only a brief explanation of the DIB format is done here for the sake of the completeness of the article.

The DIB can exist in two different formats; the *Windows* format and the *OS/2* format. Because the OS/2 format is rarely used, the DIB in the Windows format will be employed.

The DIB stored in a file consists of four structures, as shown in Figure 4. The first structure is the BITMAP FILE HEADER structure. It is optional and indicates whether the DIB is persistent. The other structures are obligatory. BITMAP INFO structure provides information about, how the DIB' Image Array is to be interpreted. It also provides complete colour stack when DIB has one.

The BITMAPFILEHEADER structure is used only when the bitmap is read or stored to disk. When a DIB is manipulated in memory, the BITMAPFILEHEADER structure is often discarded. The remaining parts of the DIB structure follow the same format whether they are located in a memory or in a disk file. The BITMAPINFO structure contains a BITMAPINFOHEADER and zero or more colour stack values for pixels stored in the bitmap. BITMAPINFOHEADER contains information about the dimensions, colour format, and compression for the bitmap.
RGBQUAD Array is the colour table of the bitmap. Each element of the colour table is four byte long and holds a colour entity defined in RGB system.
DIB Image Array is an array of pixel information. Every pixel in the bitmap is represented by a pointer to the colour entity in the colour table. Each element in the image array hence contains a value pointing to the colour entity entries. If, for example, the element in the array has a value of 32, the corresponding pixel in the bitmap will use the colour found in colour table entry number 32. The main characteristic of DIB image array is that, allocation of pixel in bitmap takes place in reverse row order as for the true picture.

Note 2: Header File for the *Weave Pattern Model* Class

```
// MPocessor.h : Declaration of com class CMPocessor
#ifndef __MPOCESSOR_H_
#define __MPOCESSOR_H_
#include "resource.h"
#include "Imatrix.h"
// CMPocessor
class ATL_NO_VTABLE CMProcessor :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CMProcessor, &CLSID_MProcessor>,
    public IDispatchImpl<IProtocol, &IID_IProtocol, &LIBID_MCREATORLib>,
    public IDispatchImpl<IPattern, &IID_IPattern, &LIBID_MCREATORLib>
{
public:
    CMProcessor();
    ~CMProcessor();
DECLARE_REGISTRY_RESOURCEID(IDR_MPROCESSOR)
DECLARE_PROTECT_FINAL_CONSTRUCT()
BEGIN_COM_MAP(CMProcessor)
    COM_INTERFACE_ENTRY(IProtocol)
    COM_INTERFACE_ENTRY(IPattern)
END_COM_MAP()
public:
    // IPattern
    STDMETHOD(GetSequence)(/*[out, retval]*/ WORD** ppHoles);
    STDMETHOD(GetRowSequence)(/*[in]*/ int nRowLine, /*[out, retval]*/ WORD ** ppHoles);
    // IMPocessor
    STDMETHOD(GetRowArray)(/*[in]*/ int nRow, /*[out, retval]*/  BYTE ** ppRowArray);
    STDMETHOD(GetColumnArray)(/*[in]*/ int nClm, /*[out, retval]*/  BYTE ** ppClmArray);
    STDMETHOD(get_Bitmap)(/*[out, retval]*/ BYTE * *ppVal);
    STDMETHOD(put_Bitmap)(/*[in]*/ BYTE * pVal);
    STDMETHOD(WriteI)(/*[in]*/ BSTR * filename);
    STDMETHOD(Convert)(/*[in]*/ int nFromTo);
    STDMETHOD(WriteBmp)(/*[in]*/ BSTR filename);
    STDMETHOD(ReadBmp)(/*[in]*/  BSTR  filename);
    STDMETHOD(ReplaceColor)(/*[in]*/ DWORD oldColor, /*[in]*/ DWORD newColor);
private:
    CDIBitmap*      m_pbitmap;
    CImatrix*       m_pImatrix;
    TColorArray*    m_pColorArray;
    BYTE*           m_pRow;
    BYTE*           m_pClm;
    WORD*           m_pRowSequence;
    WORD*           m_pSequence;
    void _iToBmp();
    void _bmpToI();
    BYTE * GetMotiveLineArray(BYTE * pMotiveLineArray, int nLineIndex);
    WORD ThreeByteToWord(BYTE b1, BYTE b2, BYTE b3);
    void WordToThreeByte(WORD& wrd, BYTE& b1, BYTE& b2, BYTE& b3);
};
#endif //__MPOCESSOR_H_
```

Figure 5. Screen copy of the Header File for the *Weave Pattern Model* Class

Note 3: Weave Patterns

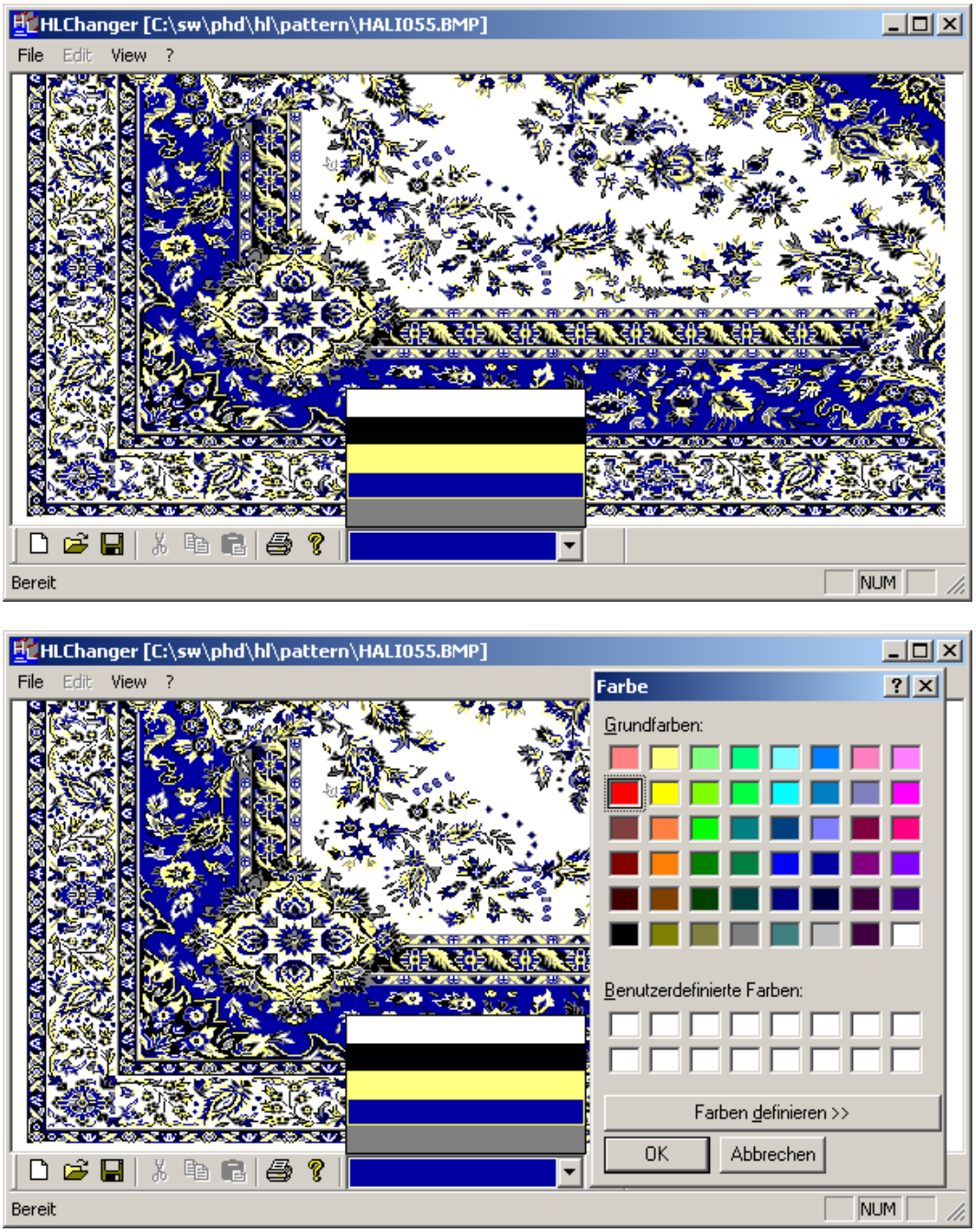

Figure 6. Weave Pattern (example 1)

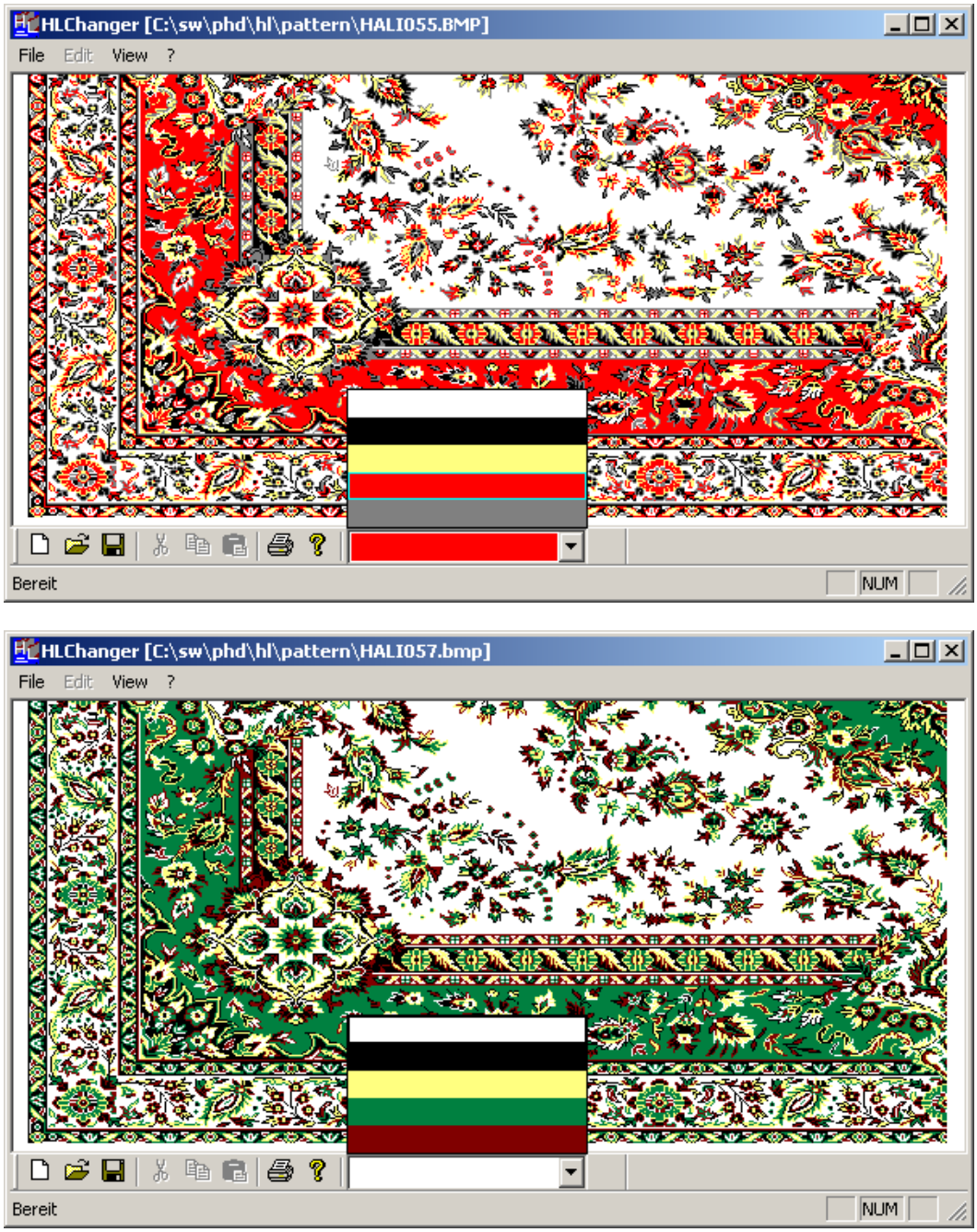

Figure 7. Weave Pattern (example 2)

References

[1] Dülger, M. (2003). "*Developing a Software Package for Electronic Jacquard System of Classical Textile Machines",* Ph.D. Thesis, University of Gaziantep.

[2] Dülger, M. (June 2009). "*Preliminaries on Latching Low Level Physical Signals from a parallel port of a PC."* Int. Eng. J. Research & Development, Vol.1, No.2.

[3] Dülger, M. (June 2009). "*Design of a PC Driven Selection Box for a Jacquard Mechanism on a Conventional Carpet Weaving Loom."* Int. Eng. J. Research & Development, Vol.1, No.2.

[4] IBM - International Business Machines Co., http://www.ibm.com

[5] Microsoft, http://www.microsoft.com

[6] MSDN - "Microsoft Developer Network Library" http://msdn.microsoft.com/en-us/library/default.aspx

[7] Rogerson, D. (January 1997) "*Inside Com*", Microsoft Press.