DEZVOLTAREA APLICATIILOR CU MICROCONTROLERUL XMC 4500

DESCRIEREA MODULUI DE GENERARE AUTOMATA A CODULUI IN MEDIULUI INTEGRAT DE DEZVOLTARE DAVE

Desfasurarea lucrarii

1. Modul de creare a unui proiect nou in DAVE- CE (Code Engine). Crearea unei aplicatii care foloseste resursele placii de evaluare **Relax 4500** (2 butoane si 2 LED-uri) si ceasul de sistem *SysClk*.
2. Setarea parametrilor pentru blocurile functionale alese din **DAVE Apps**: IO004 si SYSTM001.
3. Studiul programului exemplificat (organigrama generala, utilizarea timerelor, modul de programare a porturilor IO, analiza programului)
4. Executia programului pe placa de evaluare Relax 4500.

Teme suplimentare

1. Sa se studieze modul in care s-a creat aplicatia DMA cu **DAVE Apps (DMA003).**
2. Sa se creeze, dupa modelele prezentate, o aplicatie cu **DAVE Apps (PWM-Pulse Width Modulation),** care genereaza un semnal dreptunghiular cu factor de umplere reglabil din butoanele **Button 1** (apasarea acestui buton mareste durata pulsului) si **Button 2** (apasarea acestui buton micsoreaza durata pulsului). Perioada pulsurilor este constanta – 1 sec. Semnalul va fi vizualizat pe **LED 1**, iar **LED 2** se va aprinde/stinge cu perioada de 1 sec.
3. Sa se creeze, dupa modelele prezentate, o aplicatie cu **DAVE Apps (RTC-Real Time Control**) care implementeaza o alarma cu temporizare. La apasarea butonului **Button 1**, se masoara o durata de timp de 10 sec., dupa care se declanseaza o alarma (aprinderea **LED 1**). Daca in intervalul de 10 sec. se apasa butonul **Button 2**, atunci alarma este resetata. Starea sistemului este indicata de **LED 2** (aprins – alarma activata, stins – alarma resetata).

Descrierea aplicatiilor IO004 si SYSTM001 din DAVE Apps

Aplicatia IO004

Permite configurarea prin software a unui pin dintr-un port IO.

Structurile de date folosite de aplicatia IO004 sint:

```
typedef struct {
__IO uint32_t   OUT;
__O uint32_t    OMR;
__I  uint32_t   RESERVED1[2];
__IO uint32_t   IOCR0;
__IO uint32_t   IOCR4;
__IO uint32_t   IOCR8;
__IO uint32_t   IOCR12;
__I  uint32_t   RESERVED2[1];
__I uint32_t    IN;
__I  uint32_t   RESERVED3[6];
__IO uint32_t   PDR0;
__IO uint32_t   PDR1;
__I  uint32_t   RESERVED4[6];
__IO uint32_t   PDISC;
__I  uint32_t   RESERVED5[3];
__IO uint32_t   PPS;
 __IO uint32_t   HWSEL;
 }IO004_PORTS_TypeDef;

typedef struct IO004_HandleType
{
 uint8_t PortNr;
 uint8_t PortPin;
 IO004_PORTS_TypeDef* PortRegs;
}IO004_HandleType;

 typedef enum IO004_InputModeType
{
IO004_TRISTATE,
IO004_PULL_DOWN_DEVICE,
IO004_PULL_UP_DEVICE,
IO004_CONT_POLLING,
IO004_INV_TRISTATE,
IO004_INV_PULL_DOWN_DEVICE,
IO004_INV_PULL_UP_DEVICE,
IO004_INV_CONT_POLLING,
}IO004_InputModeType;

typedef enum IO004_OutputModeType
{
IO004_PUSHPULL =0x10,
IO004_OPENDRAIN =0x18
}IO004_OutputModeType;
```

Functiile aplicatiei IO004 sint:

void IO004_Init (void)

- initializeaza porturile IO , conform specificatiilor utilatorului.
- se apeleaza implicit in functia DAVEInit

#define IO004_ReadPin(Handle) (((Handle.PortRegs->IN) >> Handle.PortPin) & 1U)

- citeste pinul de intrare definit de Handle

```
#include <DAVE3.h>

  int main(void)
  {
    bool Value = 0;
    DAVE_Init(); // IO004_Init() is called within DAVE_Init()
    Value = IO004_ReadPin(IO004_Handle0);
    return 0;
  }
```

#define IO004_SetPin(Handle) (Handle.PortRegs->OMR) |= (1U << Handle.PortPin))

- stabileste valoarea pinului de iesire definit de Handle, in 1 logic

```
#include <DAVE3.h>

  int main(void)
  {
    DAVE_Init(); // IO004_Init() is called within DAVE_Init()
    IO004_SetPin(IO004_Handle0);
    return 0;
  }
```

#define IO004_SetOutputValue(Handle, Value) (Handle.PortRegs->OMR |= Value ? (1U << Handle.PortPin):(0x10000UL << Handle.PortPin))

- stabileste valoarea pinului de iesire definit de Handle, la valoarea Value

```
 #include <DAVE3.h>

  int main(void)
  {
    DAVE_Init(); // IO002_Init() is called within DAVE_Init()
    IO004_SetOutputValue(IO004_Handle0,1);
    return 0;
  }
```

#define IO004_ResetPin(Handle) ((Handle.PortRegs->OMR) |= (0x10000UL << Handle.PortPin))

- stabileste valoarea pinului de iesire definit de Handle, in 0 logic

```
#include <DAVE3.h>

  int main(void)
  {
    DAVE_Init(); // IO004_Init() is called within DAVE_Init()
    IO004_ResetPin(IO004_Handle0);
    return 0;
  }
```

#define IO004_TogglePin(Handle) ((Handle.PortRegs->OMR) |= (0x10001UL << Handle.PortPin))

- comuta valoarea pinului de iesire definit de Handle

```
#include <DAVE3.h>

  int main(void)
  {
    DAVE_Init(); // IO004_Init() is called within DAVE_Init()
    IO004_TogglePin(IO004_Handle0);
    return 0;
  }
```

Void IO004_DisableOutputDriver (const IO004_HandleType *Handle, IO004_InputModeType Mode)

- dezactiveaza iesirea pinului definit de Handle. Pinul este configurat ca intrare, intr-unul din modurile:

`IO004_TRISTATE` (nici un dispozitiv conectat pe pinul de intrare)
`IO004_PULL_DOWN_DEVICE` (dispozitiv pull-down conectat pe pinul de intrare)
`IO004_PULL_UP_DEVICE` (dispozitiv pull-up conectat pe pinul de intrare)
`IO004_CONT_POLLING` (Pn_OUTx esantioneaza continuu pinul de intare)
`IO004_INV_TRISTATE` (nici un dispozitiv conectat pe pinul de intrare inversor)
`IO004_INV_PULL_DOWN_DEVICE` (dispozitiv pull-down conectat pe pinul de intrare inversor)
`IO004_INV_IO004_PULL_UP_DEVICE` (dispozitiv pull-up conectat pe pinul de intrare inversor)
`IO004_INV_CONT_POLLING` (Pn_OUTx esantioneaza continuu pinul de intare inversor)

```
#include <DAVE3.h>

  int main(void)
  {
    DAVE_Init(); // IO004_Init() is called within DAVE_Init()
    IO004_DisableOutputDriver(&IO004_Handle0,IO004_PULL_UP_DEVICE);
    return 0;
  }
```

Void IO004_EnableOutputDriver (const IO004_HandleType *Handle, IO004_OutputModeType Mode)

- valideaza iesirea pinului definit de Handle. Pinul este configurat ca iesire in conformitate cu configurarea utilizatorului - Push-Pull sau Open-Drain.

```
#include <DAVE3.h>

  int main(void)
  {
    DAVE_Init(); // IO004_Init() is called within DAVE_Init()
    IO004_EnableOutputDriver(&IO004_Handle0,IO004_OPENDRAIN);
    return 0;
```

Aplicatia SYSTM001

Implementeaza timere software(maxim 32 de timere) bazate pe ceasul de sistem (*SysTick*). Timerele sint create cu functia *SYSTM001_CreateTimer()*. Fiecare timer va avea un identificator unic. Pornirea unui timer se realizeaza cu functia *SYSTM001_StartTimer()*. Timerele pot fi de doua tipuri: *One shot* si *Periodic*.

Structurile de date utilizate sint:

```
typedef enum SYSTM001_TimerStateType{
SYSTM001_STATE_RUNNING,
SYSTM001_STATE_STOPPED
}SYSTM001_TimerStateType;

typedef enum SYSTM001_TimerType
{
SYSTM001_ONE_SHOT,
SYSTM001_PERIODIC
}SYSTM001_TimerType;

typedef enum SYSTM001_ErrorCodesType
{
SYSTM001_INVALID_HANDLE_ERROR = 1,
SYSTM001_ERROR,
SYSTM001_FUNCTION_ENTRY,
SYSTM001_FUNCTION_EXIT
}SYSTM001_ErrorCodesType;

typedef void (*SYSTM001_TimerCallBackPtr)(void* ParamToCallBack);

typedef struct SYSTM001_TimerObject
{
uint32_t TimerID;
SYSTM001_TimerType TimerType;
SYSTM001_TimerStateType TimerState;
uint32_t TimerCount;
uint32_t TimerReload;
SYSTM001_TimerCallBackPtr TimerCallBack;
void* ParamToCallBack;
struct SYSTM001_TimerObject*  TimerNext;
struct SYSTM001_TimerObject*  TimerPrev;
```

```
}SYSTM001_TimerObject;
```

Functiile generate de aplicatia SYSTM001 sint:

`void  SYSTM001_Init (void)`

- initializarea aplicatiei (se face automat din DAVEInit)

`handle_t  SYSTM001_CreateTimer (uint32_t Period, SYSTM001_TimerType TimerType, SYSTM001_TimerCallBackPtr TimerCallBack, void *pCallBackArgPtr)`

- creaza un timer cu perioada *Period* (ms) si cu tipul *TimerType*.
- la expirarea timpului se va executa functia *TimerCallBack* cu parametrii indicati de pointerul *pCallBackArgPtr*.
- intoarce identificatorul timerului *Handle*

```
#include <DAVE3.h>
 static volatile bool TimerExpired;
 void my_func_a(void* Temp)
  {
    static uint32_t Count = 1;
    if(Count == 10)
    {
      TimerExpired = TRUE;
    }
    Count++;
  }
  int main(void)
  {
    handle_t TimerId;
    // ... Initializes Apps configurations ...
    DAVE_Init();
    TimerId = SYSTM001_CreateTimer(100,SYSTM001_PERIODIC,my_func_a,NULL);
    if(TimerId != 0)
    {
    //Timer is created successfully
    }
    // ... infinite loop ...
    while(1)
    {

    }

  }
```

`status_t  SYSTM001_StartTimer (handle_t Handle)`

- porneste timerul identificat prin *Handle*

```
 #include <DAVE3.h>
  static volatile bool TimerExpired;
  void my_func_a(void* Temp)
  {
```

```
    static uint32_t Count = 1;
    if(Count == 10)
    {
      TimerExpired = TRUE;
    }
    Count++;
  }
  int main(void)
  {
    handle_t TimerId;
    uint32_t Status = SYSTM001_ERROR;
    // ... Initializes Apps configurations ...
    DAVE_Init();
    TimerId = SYSTM001_CreateTimer(100,SYSTM001_PERIODIC,my_func_a,NULL);
    if(TimerId != 0)
    {
      //Timer is created successfully
        Status = SYSTM001_StartTimer(TimerId);
        if(Status == DAVEApp_SUCCESS)
        {
                 //Timer started
        }
          }
    // ... infinite loop ...
    while(1)
    {

    }
  }
```

status_t SYSTM001_StopTimer (handle_t Handle)

- opreste timerul identificat prin Handle

```
#include <DAVE3.h>
  static volatile bool TimerExpired;
  void my_func_a(void* Temp)
  {
   static uint32_t Count = 1;
   if(Count == 10)
   {
     TimerExpired = TRUE;
   }
    Count++;
  }
  int main(void)
  {
    handle_t TimerId;
   uint32_t Status = SYSTM001_ERROR;
   // ... Initializes Apps configurations ...
    DAVE_Init();
    TimerId = SYSTM001_CreateTimer(100,SYSTM001_PERIODIC,my_func_a,NULL);
    if(TimerId != 0)
    {
      //Timer is created successfully
```

```
        Status = SYSTM001_StartTimer(TimerId);
      if(Status == DAVEApp_SUCCESS)
        {
           // Wait till timer is expired
           while(TimerExpired == FALSE)
         {}
                //stop the timer
                Status = SYSTM001_StopTimer(TimerId);
            if(Status == DAVEApp_SUCCESS)
                {
              //Timer stopped
                }
       }
       // start the timer
        SYSTM001_StartTimer(TimerId);
    }
    // ... infinite loop ...
    while(1)
    {

    }

 }
```

status_t SYSTM001_DeleteTimer (handle_t Handle)

- distruge timerul identificat prin identificat prin *Handle*

```
#include <DAVE3.h>
  static volatile bool TimerExpired;
  void my_func_a(void* Temp)
  {
   static uint32_t Count = 1;
   if(Count == 10)
   {
     TimerExpired = TRUE;
   }
   Count++;
  }
  int main(void)
  {
    handle_t TimerId;
    uint32_t Status = SYSTM001_ERROR;
    // ... Initializes Apps configurations ...
    DAVE_Init();
    TimerId = SYSTM001_CreateTimer(100,SYSTM001_PERIODIC,my_func_a,NULL);
    if(TimerId != 0)
    {
      //Timer is created successfully
      Status = SYSTM001_StartTimer(TimerId);
      if(Status == DAVEApp_SUCCESS)
        {
        // Wait till timer is expired
        while(TimerExpired == FALSE)
        {}
```

```
        //stop the timer
        Status = SYSTM001_StopTimer(TimerId);
      if(Status == DAVEApp_SUCCESS)
        {
          SYSTM001_DeleteTimer(TimerId);
        }
    }
  }
  // ... infinite loop ...
  while(1)
  {

  }
}
```

uint32_t SYSTM001_GetTime (void)

- preia valoarea curenta a timerului identificat prin *Handle* (in ms)

```
#include <DAVE3.h>
 static volatile bool TimerExpired;
 void my_func_a(void* Temp)
 {
   static uint32_t Count = 1;
         if(Count == 10)
   {
     TimerExpired = TRUE;
   }
   Count++;
 }
 int main(void)
 {
         handle_t TimerId;
   uint32_t SystemTime = 0;
   uint32_t Status = SYSTM001_ERROR;
   // ... Initializes Apps configurations ...
   DAVE_Init();
   TimerId = SYSTM001_CreateTimer(100,SYSTM001_PERIODIC,my_func_a,NULL);
   if(TimerId != 0)
   {
     //Timer is created successfully
     Status = SYSTM001_StartTimer(TimerId);
     if(Status == DAVEApp_SUCCESS)
     {
       SystemTime = SYSTM001_GetTime();
     }
   }
    // ... infinite loop ...
   while(1)
   {

   }
 }
```

```
uint32_t  SYSTM001_GetSysTickCount (uint32_t Period)
```

- preia numarul de perioade de tact *SysTick* pentru timerul identificat prin *Handle*, pe durata de timp *Period* (ms)

```
#include <DAVE3.h>
  int main(void)
  {
    uint32_t SysTickCount = 0;
    DAVE_Init();
    // Get systick timer count value for 100millisec
    SysTickCount = SYSTM001_GetSysTickCount(100);
    return 0;
  }
```

Realizarea unei aplicatii bazate pe un proiect DAVE CE

Se va realiza o aplicatie (figura 1) cu urmatoarele specificatii:

- citeste starea butonului BUTTON1 si o afiseaza pe led-ul LED1, la fiecare 20 ms
- daca butonul BUTTON2 este apasat atunci led-ul LED2 isi va comuta starea de aprins/stins la fiecare 1 secunda

Pentru realizarea aplicatiei se vor utiliza 2 timere software, Timer1 si Timer2, create cu modulul SYSTM001 din DAVE Apps. Pinii de intrare – iesire vor fi controlati prin functii generate de moule de tip IO004 din DAVE Apps.

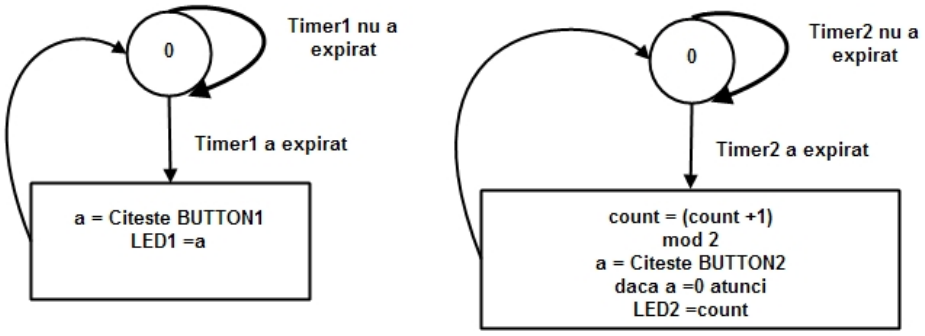
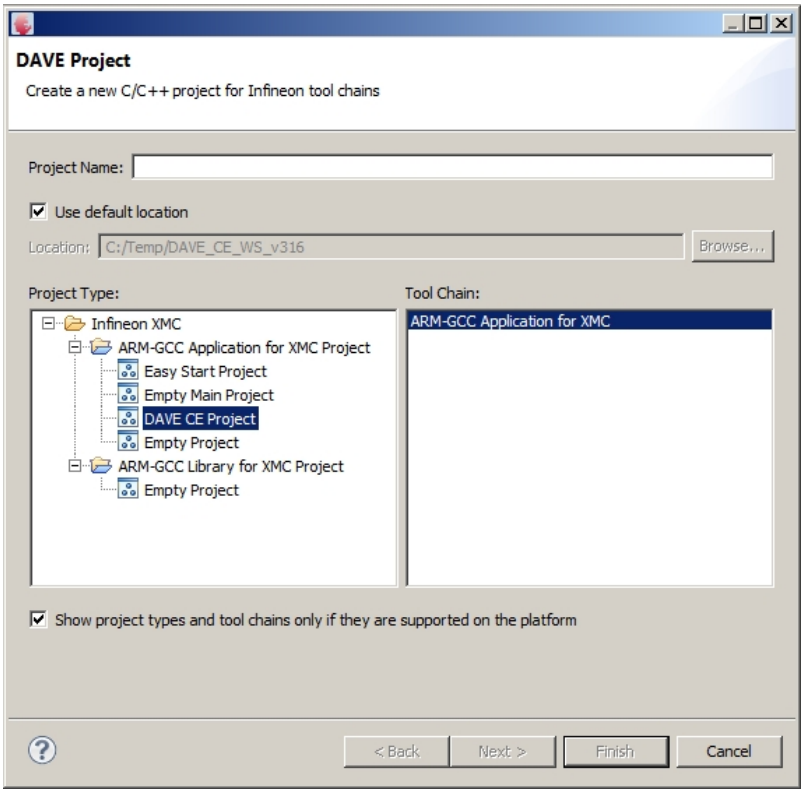


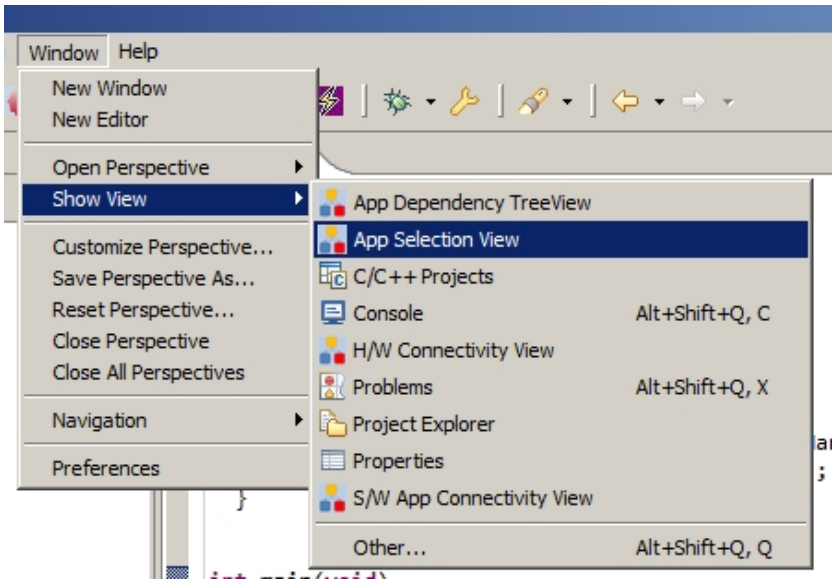
Figura 1.

Se urmeaza urmatoarele etape:

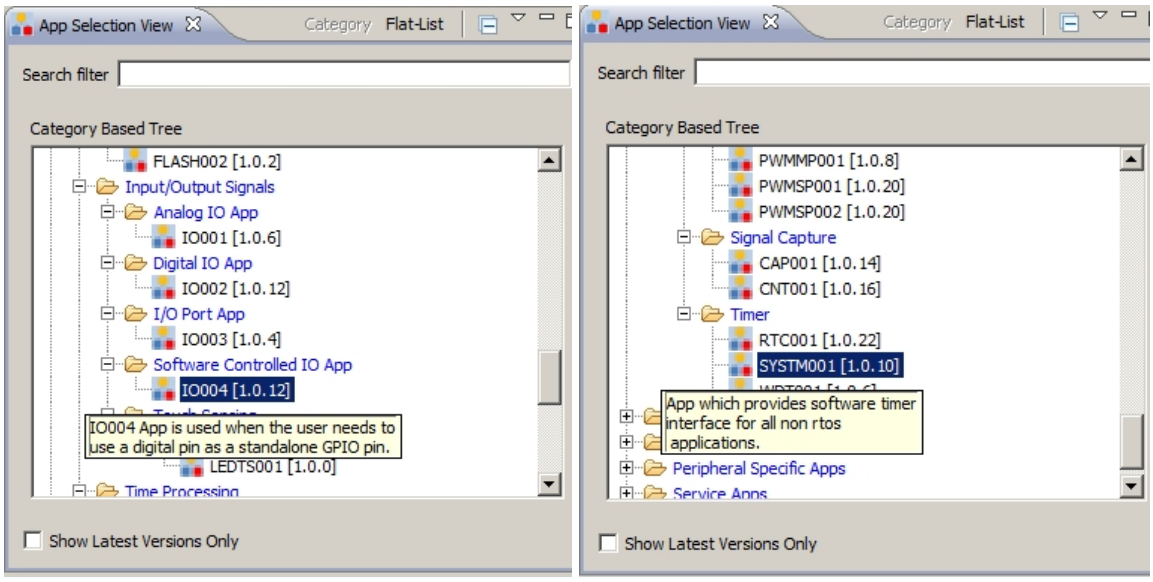
1. Se creaza un proiect nou, de tip DAVE CE



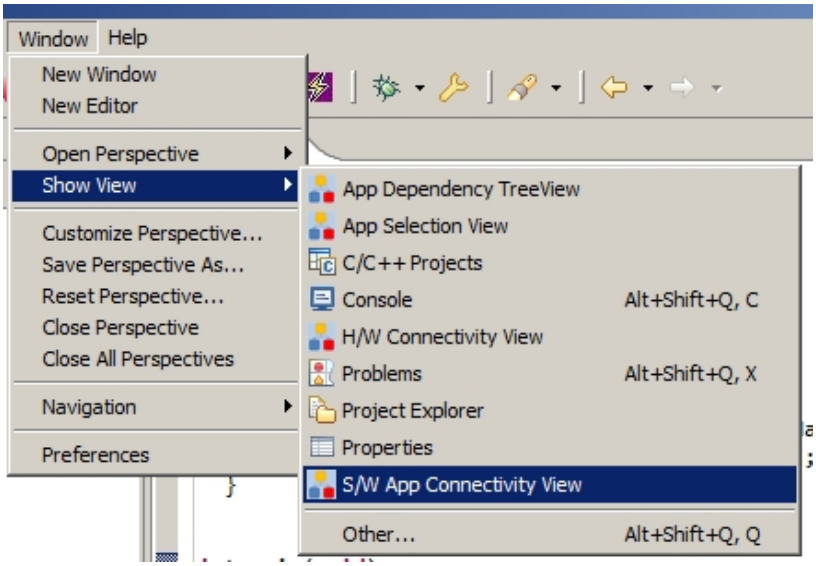
2. Se deschide fereastra de selectie a aplcatiilor DAVE Apps, pe baza carora se va genera cod automat pentru controlur resurselor microcontrolerului



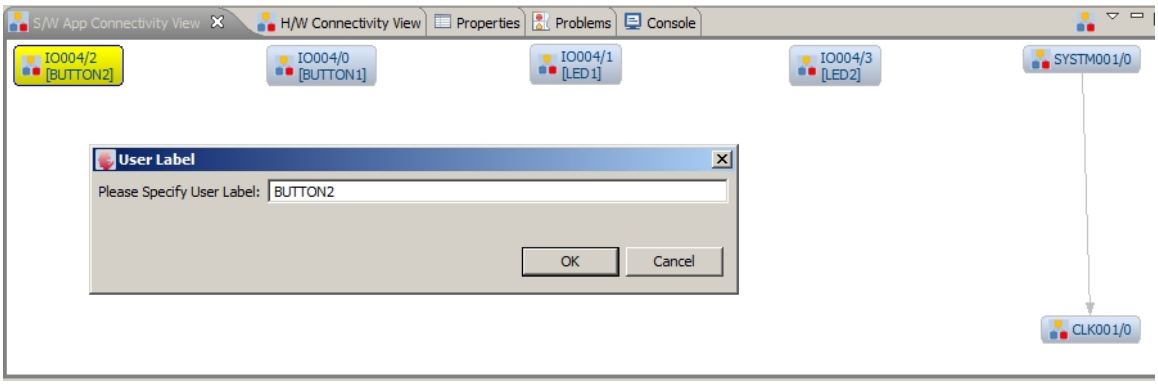
3. In fereastra App Selection View, se selecteaza (dublu click) aplicatia IO004 (de 4 ori) si aplicatia SYSTM01 (1 data)



4. Se deschide fereastra S/W App Connectivity View



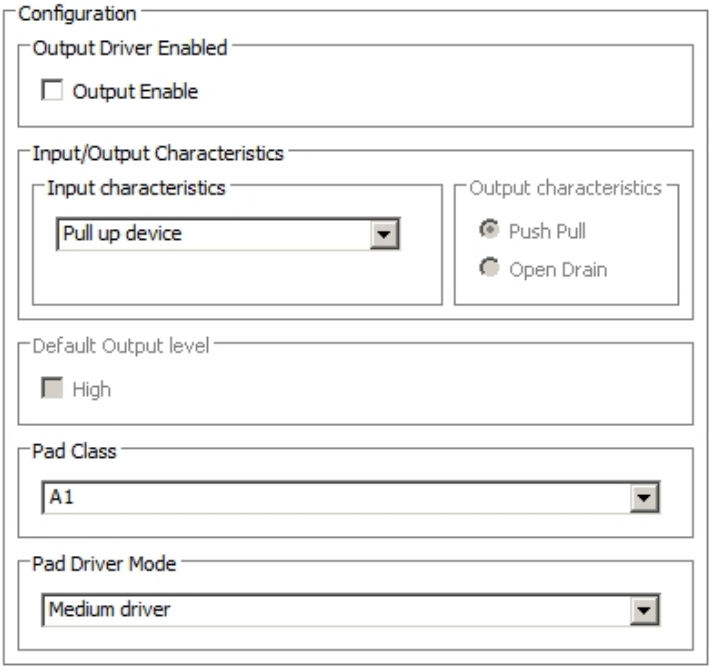
5. Se denumesc blocurile IO004 astfel:

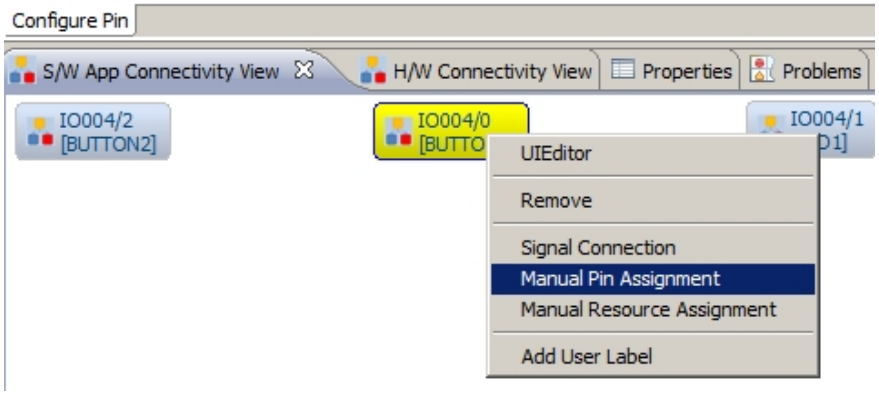


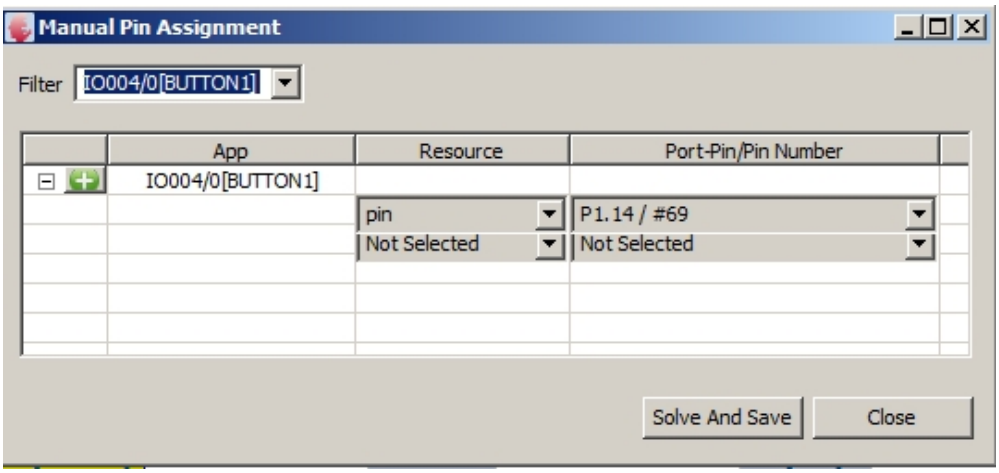
6. Se seteaza parametrii blocurilor IO004 astfel:

IO004/0 – BUTTON 1 (portul P1.14 – intrare pull-up)
IO004/2 – BUTTON 2 (portul P1.15 – intrare pull-up)
IO004/1 – LED 1 (portul P1.1 – iesire pull-push)
IO004/3 – LED 2 (portul P1.0 – iesire push-pull)
Blocul CLK001/0 nu se modifica

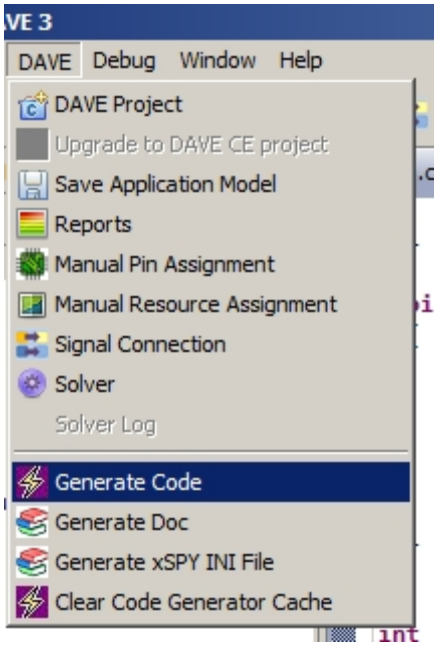
Pentru setarea parametrilor se da dublu click pe blocul care se doreste a fi configurat, iar pentru asocierea pinilor se da click dreapta si se selecteaza **Manual Pin Assignment.** Dupa fiecare setare se da comanda **Solve and save.**







7. Se genereaza codul cu comada DAVE-> Genarate Code



8. Se scrie codul utilizatorului in functiile Main si functiile asociate cu expirarea timerelor Timer1 si Timer2.

Codul este urmatorul:

```
/*
 * Main.c
 *
*/

#include <DAVE3.h>                    //Declarations from DAVE3 Code Generation
(includes SFR declaration)
```

```
uint32_t count=0;

 void Timer1(void* Temp)
  {
        bool Value = 0;
        //{asm("BKPT 255");}
        Value = IO004_ReadPin(IO004_Handle0);
        IO004_SetOutputValue(IO004_Handle1,Value);
  }

 void Timer2(void* Temp)
  {
         bool Value = 0;
        //{asm("BKPT 255");}
        count=(count+1)%2;
        Value = IO004_ReadPin(IO004_Handle2);
        if (Value==0) IO004_SetOutputValue(IO004_Handle3,count);
        else IO004_SetOutputValue(IO004_Handle3,0);
  }


int main(void)
{
//    status_t status;          // Declaration of return variable for DAVE3
APIs (toggle comment if required)

       handle_t TimerId1;
       handle_t TimerId2;

       uint32_t Status1 = SYSTM001_ERROR;
       uint32_t Status2 = SYSTM001_ERROR;

       DAVE_Init();              // Initialization of DAVE Apps

    IO004_EnableOutputDriver(&IO004_Handle1,IO004_PUSHPULL);
    IO004_EnableOutputDriver(&IO004_Handle3,IO004_PUSHPULL);

       TimerId1 = SYSTM001_CreateTimer(20,SYSTM001_PERIODIC,Timer1,NULL);
       if(TimerId1 != 0)  Status1 =SYSTM001_StartTimer(TimerId1); //Timer is
created successfully

       TimerId2 = SYSTM001_CreateTimer(1000,SYSTM001_PERIODIC,Timer2,NULL);
       if(TimerId2 != 0)  Status2 =SYSTM001_StartTimer(TimerId2); //Timer is
created successfully

       if ((Status1==DAVEApp_SUCCESS) && (Status2==DAVEApp_SUCCESS))
       {
       while(1)
       {

       }
       }
       else { asm("BKPT 255");}
```

```
return 0;

}
```

9. Se deschide o sesiune de depanare si se executa codul pe placa de evaluare **Relax 4500**.

TEMA SUPLIMENTARA

TRANSFERUL DMA
PROGRAMAREA GPDMA – XMC 4500 CU AJUTORUL DAVE CE

Descrierea blocului GPDMA al microcontrolerului XMC4500

Blocul GPDMA (General Purpose Direct Memory Access) este un controller DMA performant care realizeaza transferuri rapide (memorie-memorie si memorie-IO) cu interventia minima a procesorului.
Disopzitivele periferice ale microcontrolerului pot accesa memoria prin canale DMA.

Caracteristicile GPDMA sint:

- **Interfetele cu bus-urile**
    - 1 bus master
    - 1 bus slave
- **Canale de comunicatie**
    - 1 bloc GPDMA0 cu 8 canale
    - 1 bloc GPDMA1 cu 8 canale
    - Posibilitatea de a programa prioritatea canalelor
- **Transferuri**
    - memorie+memorie, memorie – periferice, periferice - memorie

Toate canalele DMA pot fi programate in urmatoarele moduri:

- transfer DMA declansat hardware sau software
- adrese sursa si destinatie programabile
- modificarea adresei prin incrementare sau decrementare

Canalele 0 si 1 din blocul GPDMA0 pot fi programate in modurile:

- transfer multi-bloc (ca liste inlantuite, cu auto incarcarea registrelor de programare, cu adrese continue intre blocuri)
- selectarea adreselor sursa si destinatie independent pentru fiecare bloc
- adresele sursa si destinatie un trebuei sa fie intr-o zona continua de memorie

Se pot realiza transferuri DMA de tip burst, cu ajutorul unor cozi FIFO.

Canalele DMA sint controlate independent si pot fi activate sau dezactivate. Se pot genera intreruperi la terminarea unui transfer sau la aparitia unei erori.

Schema bloc a GPDMA este prezentata in figura 2.

Se observa urmatoarele module functionale:

- interfata de cereri hardware (DMA hardware request interface - DLR)
- 12 canale DMA
- Circuit de arbitrare a cererilor DMA
- Interfetele cu bus-urile master si slave

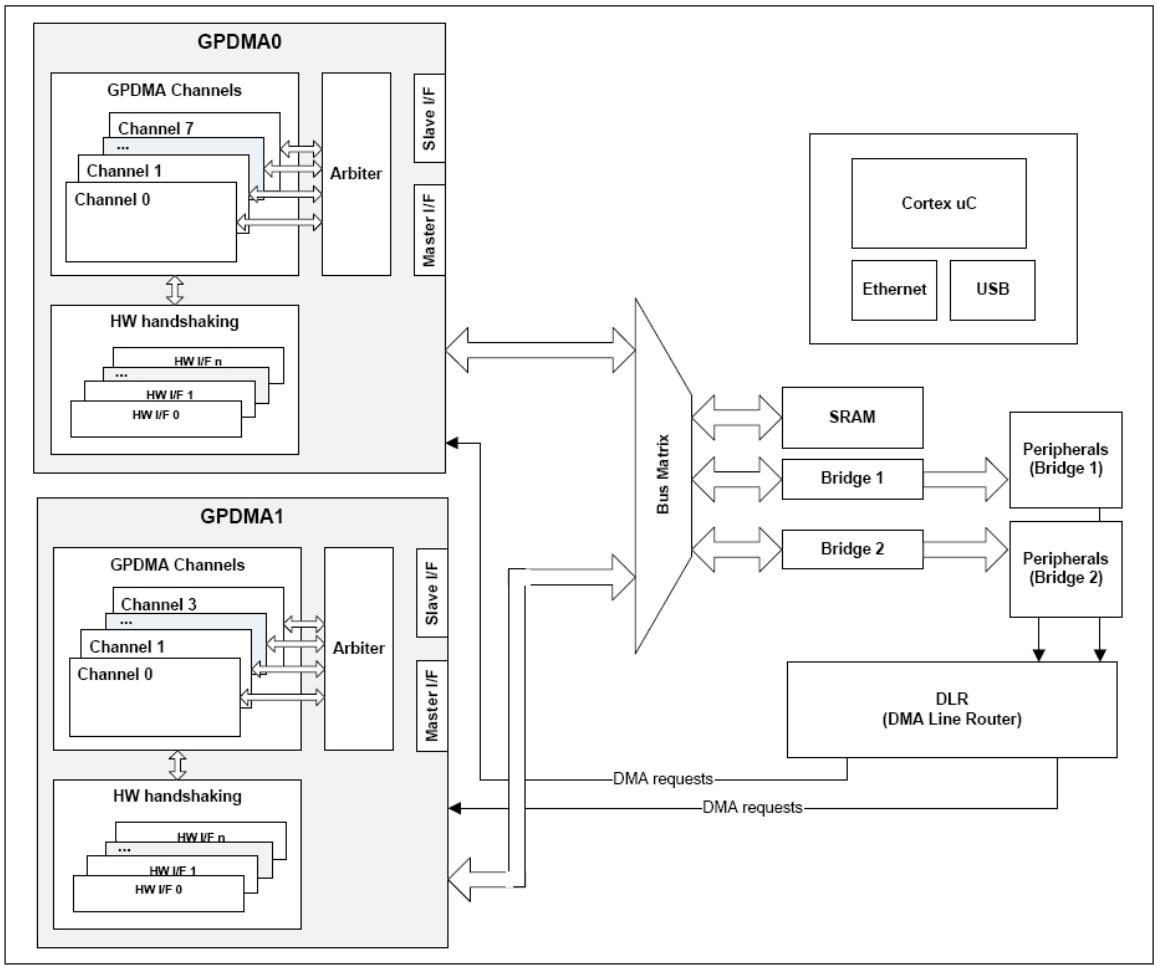


Figura 2.

Transferurile DMA pot fi initiate de catre software se va genera o intrerupere la terminarea transferului (ca in figura 3).

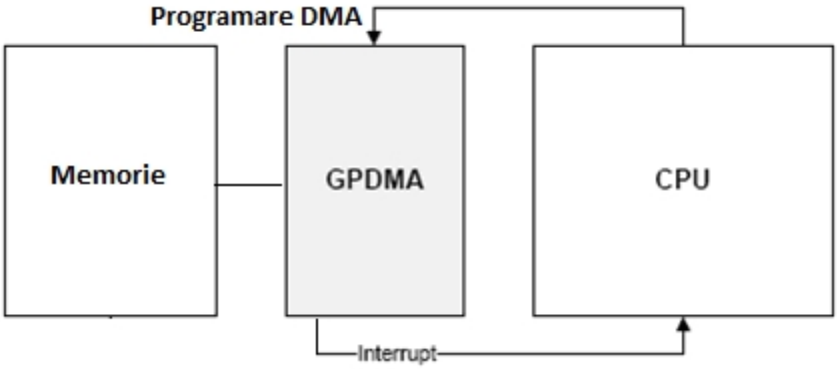


Figura 3.

Registrele GPDMA

Registrele GPDMA sint impartite in module asociate blocurilor **GPDMA0** si **GPDMA1** astfel:

Modulul
GPDMA0_CH0
GPDMA0_CH1
GPDMA0_CH2
GPDMA0_CH3
GPDMA0_CH4
GPDMA0_CH5
GPDMA0_CH6
GPDMA0_CH7
GPDMA0
GPDMA1_CH0
GPDMA1_CH1
GPDMA1_CH2
GPDMA1_CH3
GPDMA1

Exista registre asociate fiecarui canal – **GPDMAx_CHy** cu x = 0 y= 0-7 si x = 1, y = 0-3 si registre pentru tot blocul **GPDMAx** (x = 0,1).
Identificarea registrelor se face astfel: GPDMAx_CHy_nume_registru (pentru registrele de asociate canalului si de control) si GPDMAx_nume_registru (pentru registrele pentru intreruperi, *Software Handshaking*, configurare si validare si registrele *Miscellaneous*)

Registre asociate canalului

SAR	Source Address Register
DAR	Destination Address Register

Registre de control

CTLH	Control Register High
CTLL	Control Register Low
LLP	Linked List Pointer Register
SSTAT	Source Status Register
DSTAT	Destination Status Register
SSTATAR	Source Status Register
DSTATAR	Destination Status Register
CFGH	Configuration Register High
CFGL	Configuration Register Low
SGR	Source Gather Register
DSR	Destination Scatter Register

Registre pentru intreruperi

RAW* with *TFR, *BLOCK, *SRCTRAN, *DSTTRAN, *ERR	Interrupt Raw Status Registers
STATUS* with *TFR, *BLOCK, *SRCTRAN, *DSTTRAN, *ERR	Interrupt Status Registers
MASK* with *TFR, *BLOCK, *SRCTRAN, *DSTTRAN, *ERR	Interrupt Mask Registers
CLEAR* with *TFR, *BLOCK, *SRCTRAN, *DSTTRAN, *ERR	Interrupt Clear Registers
STATUSINT	Combined Interrupt Status Register

Registre pentru Software Handshaking

REQSRCREG	Source Software Transaction Request Register
REQDSTREG	Destination Software Transaction Request Register
SGLREQSRCR EG	Single Source Transaction Request Register
SGLREQDSTR EG	Single Destination Transaction Request Register
LSTSRCREG	Last Source Transaction Request Register
LSTDSTREG	Last Destination Transaction Request Register

Registre pentru configurare si validare a canalelor

DMACFGREG	Configuration Register
CHENREG	Channel Enable Register

Alte registre (Miscellaneous)

ID	GPDMA Module ID
TYPE	GPDMA Component Type
VERSION	GPDMA Component Version

Configurarea unui canal DMA

Se defineste structura de date, *ChConfig*, de tip tablou cu urmatoarele cimpuri:

Tip de date	**Denumirea cimpului**	**Semnificatie**	**Registru de comanda**
uint32_t	SrcAddress	Adresa sursa	**SAR**
uint32_t	DstAddress	Adresa destinatie	**DAR**
uint32_t	BlockSize	Dimensiunea blocului	**CTLH.BLOCK_TS**

		transferat (numarul de octeti de transferat)	
DMA_TransferFlowType *(* *DMA_MEM2PRF_DMA* *DMA_PRF2MEM_DMA* *DMA_PRF2MEM_PRF* *DMA_MEM2PRF_PRF* *)*	TransferFlow	Tipul de transfer (memorie-memorie, memorie-priferic, periferic – memorie, perifric-periferic)	**CTLL.TT_FC**
DMA_ChPriorityType	ChPriority	Prioritatea canalului	**CFGL.CH_PRIOR**
DMA_TransferWidthType (8 , 16 sau 32 de biti)	SrcTrWidth	Largimea datelor transferate (la sursa)	**CTLL.SRC_TR_WIDTH**
DMA_TransferWidthType (8 , 16 sau 32 de biti)	DstTrWidth	Largimea datelor transferate (la destinatie)	**CTLL.DST_TR_WIDTH**
DMA_BurstTransLenType	SrcMSize	Lungimea burstului transferat la sursa	**CTLL.SRC_MSIZE**
DMA_BurstTransLenType	DstMSize	Lungimea burstului transferat la destinatie	**CTLL.DEST_MSIZE**
DMA_AddrIncType	SrcInc	Modul de modificare a adresei sursa (incrementare / decrementare)	**CTLL.SINC**
DMA_AddrIncType	DstInc	Modul de modificare a adresei destinatie (incrementare / decrementare)	**CTL.DINC**

Se actualizeaza, in mod corespunzator modului de transfer ales, cimpurile registrelor de comanda).

Daca se considera ca un canal DMA este identificat prin structura *Handle* de tipul *DMA003_ChannelHandleType:*

```
typedef struct DMA003_ChannelHandleType
{
  /** Channel Configuration parameters  */
  DMA003_ChannelConfigType ChConfig;
   /** Base Address of mapped DMA Unit */
  GPDMA0_CH_TypeDef* DMAChRegs;
  GPDMA0_GLOBAL_TypeDef* DMARegs;
  /*Mapped NVIC_DMA001 App Handle */
  NVIC_DMA001_HandleType* NVICDMA_Handle;
   /** Channel allocated by Dave3  */
  uint8_t ChannelID;
  /** DMA Unit ID  */
  uint8_t DMAUnit;
  /** Channel Enabled by default */
  bool  ChannelEnabled;
  /** Source Transfer width */
  bool  HWHandshakeEnabled;
  /** Enable source transaction event */
  bool SrcTranEventEn;
  /** Enable destination transaction event */
  bool DstTranEventEn;
  /** Enable error event */
  bool ErrorEventEn;
  /** Enable block event */
  bool BlockEventEn;
  /** Enable transfer complete event */
  bool TfrEventEn;

}DMA003_ChannelHandleType;
```

atunci initializarea unui canal DMA se efectueaza cu ajutorul functiei::

DMA_Init

```
/* Set the Source and destination addresses */

DMAChRegs->SAR = (uint32_t) Handle->ChConfig.SrcAddress;
DMAChRegs->DAR = (uint32_t) Handle->ChConfig.DstAddress;
DMAChRegs->LLP = (uint32_t)0x00000000;

/* Set the Channel Configuration Parameters */

DMAChRegs->CTLL = (DMA_CTLL_DST_WIDTH(Handle->ChConfig.DstTrWidth) \
                  | DMA_CTLL_SRC_WIDTH(Handle->ChConfig.SrcTrWidth) \
                  | DMA_CTLL_DST_INC(Handle->ChConfig.DstInc)     \
                  | DMA_CTLL_SRC_INC(Handle->ChConfig.SrcInc)     \
                  | DMA_CTLL_DST_MSIZE(Handle->ChConfig.DstMSize)  \
```

```
                    | DMA_CTLL_SRC_MSIZE(Handle->ChConfig.SrcMSize)  \
                    | DMA_CTLL_FC(Handle->ChConfig.TransferFlow)    \
                    | DMA_CTLL_INT_EN );


 /* Set the block size for the DMA transfer */

DMAChRegs->CTLH =
((uint32_t)GPDMA0_CH_CTLH_BLOCK_TS_Msk & Handle->ChConfig.BlockSize);

/* Set the channel configuration */

DMAChRegs->CFGL &= ~(DMA_CFGL_CH_PRIOR_Msk);
DMAChRegs->CFGL |=
( (DMA_CFGL_CH_PRIOR(Handle->ChConfig.ChPriority)) & DMA_CFGL_CH_PRIOR_Msk );
```

Transferul DMA este pornit prin validarea canalului DMA:

DMA_Start_Transfer

```
#define DMA_CH_NUM(ch)
(uint32_t)((uint32_t)(1UL << (DMA_MAX_CHANNELS + ch)) | (uint32_t)(1UL << ch))

/* UnMask  channel interrupts */

DMARegs->MASKTFR   = DMA_CH_NUM(Channel);
DMARegs->MASKBLOCK = DMA_CH_NUM(Channel);
DMARegs->MASKERR   = DMA_CH_NUM(Channel);
DMARegs->MASKSRCTRAN = DMA_CH_NUM(Channel);
DMARegs->MASKDSTTRAN   = DMA_CH_NUM(Channel);

/* Enable the channel */

DMARegs->CHENREG = DMA_CH_NUM(Channel);
```

Utilizarea intreruperilor DMA

BLOCUL NVIC (NESTED VECTORED INTERRUPT CONTROLER)

Blocul NVIC asigura raspunsul la intreruperi pentru toate tipurile de intrerupere disponibile in microcontrolerul XCM4500. Exista 112 intreruperi posibile, prioritizate pe 64 de niveluri de intrerupere. Prioritatea unei intreruperi se poate modifica in mod dinamic.

Fiecare resursa a microcontrolerului XMC 4500 are rezervat un nivel de intrerupere (nod NVIC), in tabela vectorilor de intrerupere, astfel:

Interrupt Node assignment

Service Request	IRQ Number	Description
SCU.SR0	0	System Control
ERU0.SR0 ERU0.SR3	1...4	External Request Unit 0
ERU1.SR0 ERU1.SR3	5...8	External Request Unit 1
NC	9, 10, 11	Reserved
PMU0.SR0	12	Program Management Unit
NC	13	Reserved
VADC.C0SR0 - VADC.C0SR3	14...17	Analog to Digital Converter Common Block 0
VADC.G0SR0 - VADC.G0SR3	18...21	Analog to Digital Converter Group 0
VADC.G1SR0 - VADC.G1SR3	22...25	Analog to Digital Converter Group 1
VADC.G2SR0 - VADC.G2SR3	26...29	Analog to Digital Converter Group 2
VADC.G3SR0 - VADC.G3SR3	30...33	Analog to Digital Converter Group 3
DSD.SRM0 - DSD.SRM3	34...37	Delta Sigma Demodulator Main
DSD.SRA0 - DSD.SRA3	38...41	Delta Sigma Demodulator Auxiliary
DAC.SR0 - DAC.SR1	42, 43	Digital to Analog Converter
CCU40.SR0 - CCU40.SR3	44...47	Capture Compare Unit 4 (Module 0)
CCU41.SR0 - CCU41.SR3	48...51	Capture Compare Unit 4 (Module 1)
CCU42.SR0 - CCU42.SR3	52...55	Capture Compare Unit 4 (Module 2)
CCU43.SR0 - CCU43.SR3	56...59	Capture Compare Unit 4 (Module 3)
CCU80.SR0 - CCU80.SR3	60...63	Capture Compare Unit 8 (Module 0)
CCU81.SR0 - CCU81.SR3	64...67	Capture Compare Unit 8 (Module 1)
POSIF0.SR0 - POSIF0.SR1	68...69	Position Interface (Module 0)
POSIF1.SR0 - POSIF1.SR1	70...71	Position Interface (Module 1)
NC	72...75	Reserved
CAN.SR0 - CAN.SR7	76...83	MultiCAN
USIC0.SR0 - USIC0.SR5	84...89	Universal Serial Interface Channel (Module 0)

USIC1.SR0 - USIC1.SR5	90...95	Universal Serial Interface Channel (Module 1)
USIC2.SR0 - USIC2.SR5	96...101	Universal Serial Interface Channel (Module 2)
LEDTS0.SR0	102	LED and Touch Sense Control Unit (Module 0)
NC	103	Reserved
FCE.SR0	104	Flexible CRC Engine
GPDMA0.SR0	105	General Purpose DMA unit 0
SDMMC.SR0	106	Multi Media Card Interface
USB0.SR0	107	Universal Serial Bus
ETH0.SR0	108	Ethernet (Module 0)
NC	109	Reserved
GPDMA1.SR0	110	General Purpose DMA unit 1
NC	111	Reserved

Blocurile GPDMA au rezervate 2 niveluri de intrerupere, astfel:

Interrupt Node assignment

Service Request	IRQ Number	Description
GPDMA0.SR0	105	General Purpose DMA unit 0
GPDMA1.SR0	110	General Purpose DMA unit 1

Blocul NVIC (Nested Vectored Interrupt Controller) asigura raspunsul la intreruperile generate de GPDMA.
Se definesc urmatoarele structuri de date care identifica starea canalului DMA si modul de generare a intreruperilor:

```
typedef enum NVIC_DMA001_ChStateType{
  NVIC_DMA001_IDLE          = 0x0,
  NVIC_DMA001_SINGLE_REGION = 0x1,
  NVIC_DMA001_BURST_REGION  = 0x2
}NVIC_DMA001_ChStateType;

typedef struct
{
  uint8_t ch_num;                       /* channel number */
  NVIC_DMA001_ChStateType SrcState;     /* DMA source state*/
  NVIC_DMA001_ChStateType DstState;     /* DMA destination state*/
  uint32_t BlockCnt;                    /* count of completed blocks*/
  DMACallbackType CbListener;           /* listener function for IRQ handler*/
  uint32_t CbArg;
}NVIC_DMA001_ChInstanceType;

typedef struct{
  NVIC_DMA001_ChInstanceType Ch[DMA_MAX_CHANNELS];
```

```
  GPDMA0_GLOBAL_TypeDef* DMARegs;                    /*Pointer to DMA Unit */
}NVIC_DMA001_HandleType;
```

Nivelul de intrerupere (node interrupt) este identificat printr-o varabila *Handle* astfel:

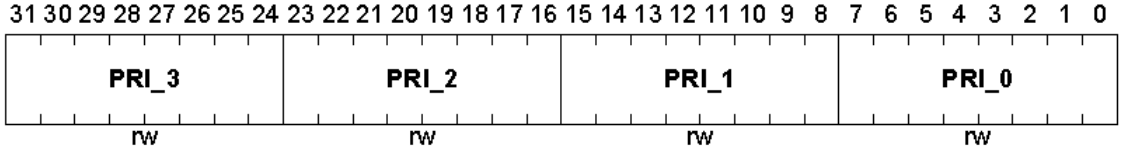
```
NVIC_DMA001_HandleType NVIC_DMA001_Handle0;
```

Initializarea intreruperilor se efectueaza prin functia (exemplificare pentru GPDMA1):

```
/* Set Interrupt Priority for NVIC 110 Node DMA Unit 1 */

NVIC_SetPriority((IRQn_Type)110,
NVIC_EncodePriority(NVIC_GetPriorityGrouping(),10U,0U));
```

NVIC_IPRx (x=0-27)
Interrupt Priority Register x

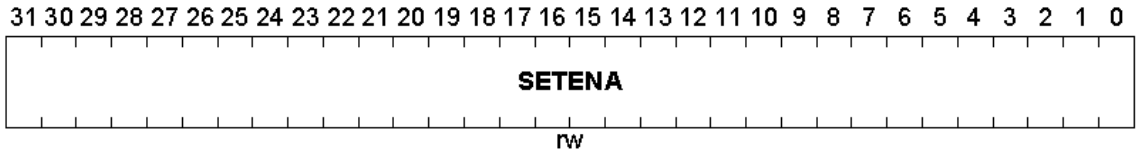


Field	Bits	Type	Description
PRI_0	[7:0]	rw	**Priority value 0**
PRI_1	[15:8]	rw	**Priority value 1**
PRI_2	[23:16]	rw	**Priority value 2**
PRI_3	[31:24]	rw	**Priority value 3** The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits[7:n] of each field, bits[n-1:0] read as zero and ignore writes.

```
#define __NVIC_PRIO_BITS    6
NVIC->IP[(uint32_t)(IRQn)] = ((priority << (8 - __NVIC_PRIO_BITS)) & 0xff);


/* Enable Interrupt */
NVIC_EnableIRQ((IRQn_Type)110);
```

NVIC_ISERx (x=0-3)
Interrupt Set-enable Register x



Field	Bits	Type	Description
SETENA	[31:0]	rw	**Interrupt set-enable bits** 0_B Write: no effect Read: interrupt disabled 1_B Write: enable interrupt Read: interrupt enabled

```
NVIC->ISER[(uint32_t)((int32_t)IRQn) >> 5] = (uint32_t)(1 <<
((uint32_t)((int32_t)IRQn) & (uint32_t)0x1F));
```

```
/* Reset the DMA1 Unit*/
RESET001_DeassertReset(PER2_DMA1);

/* Enable the DMA1 Unit*/
GPDMA1->DMACFGREG |= (uint32_t) GPDMA1_DMACFGREG_DMA_EN_Msk;
```

Asocierea dintre nivelul de intrerupere si rutina de servire este facuta cu functia:

```
void DMA003_SetListener(
  const DMA003_ChannelHandleType* Handle,
  DMACallbackType userFunction,
  uint32_t CbArgs);
```

apelata cu parametrii:

```
DMA003_SetListener(&DMA003_Handle0,DMA_Interrupt,0);
```

care apeleaza functia:

```
void NVIC_DMA001_RegisterCallback
(
  NVIC_DMA001_HandleType* Handle,
  uint8_t ch_index,
  DMACallbackType userFunction,
  uint32_t CbArgs
 )
```

cu parametrii:

```
NVIC_DMA001_RegisterCallback (Handle->NVICDMA_Handle,
             Handle->ChannelID,userFunction,CbArgs); // Handle pentru DMA
```

care contine liniile:

```
Handle->Ch[ch_index].CbListener = userFunction; // Handle pentru intrerupere
Handle->Ch[ch_index].CbArg = CbArgs;
```

Figura 4 ilustreaza modul in care se genereaza intrerupereile GPDMA.

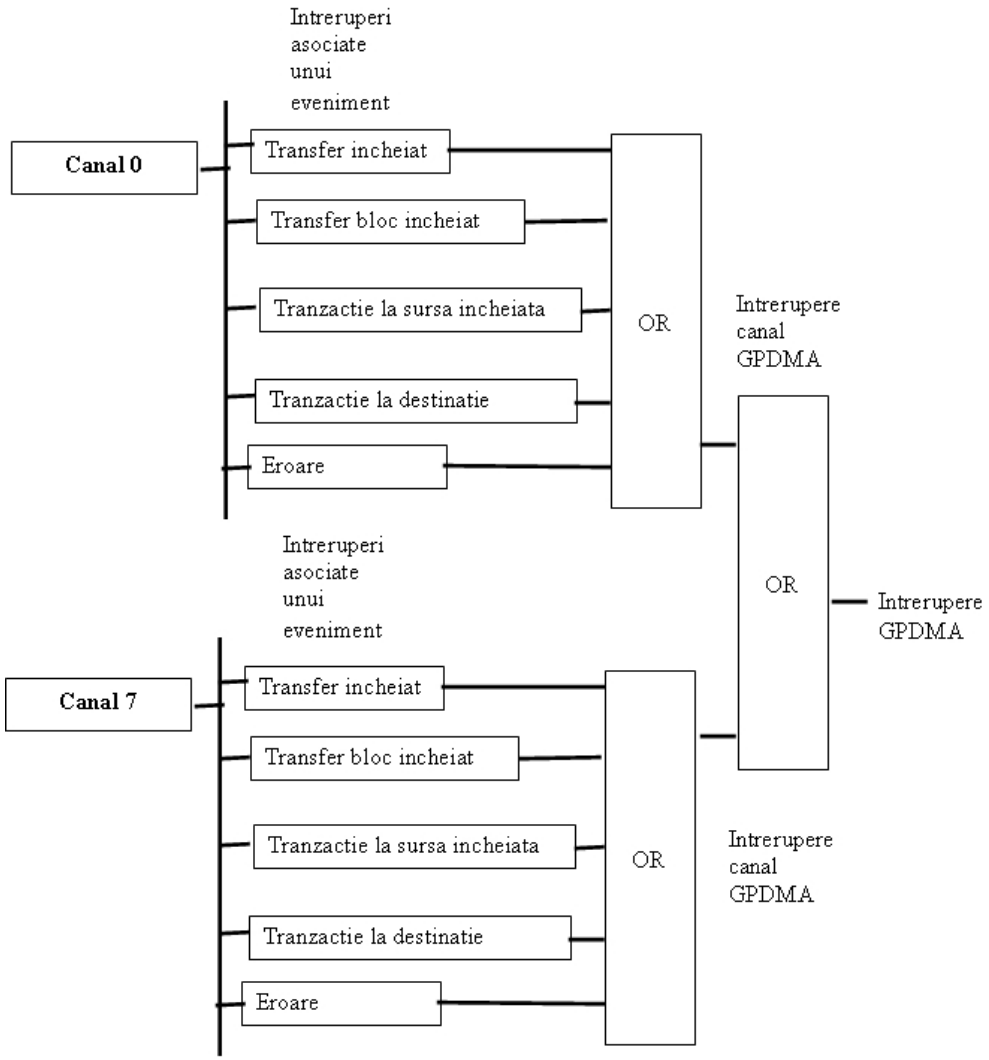


Figura 4.

Se poate asocia cite o rutina de intrerupere pentru fiecare canal DMA. Se genereaza o singura intrerupere pentru modulul GPDMA (SAU logic intre intreruperile de pe fiecare canal). Rutina de servire a intreruperii globale GPDMA (handler-ul de intreruperi) va testa daca au aparut intrreruperi pe fiecare canal si acestea se vor servi pe rind fiecare efectuind un salt la rutina de servire asociata canalului propriu. Pentru aceasta se va utiliza structura de date `NVIC_DMA001_HandleType`.

Rutina de servire a intreruperii pe un canal este de tipul:

```
void DMA_Interrupt(DMA_IRQType IRQType,uint32_t CbArg);
```

cu parametrul *DMA-IRQType,* care indica motivul generarii intreruperii, definit astfel:

```
typedef enum DMA_IRQType
{
  DMA_IRQ_NONE       = 0x00,     /** no interrupts*/
  DMA_IRQ_TFR        = 0x01,     /** transfer complete*/
  DMA_IRQ_BLOCK      = 0x02,     /** block transfer complete*/
  DMA_IRQ_SRCTRAN    = 0x04,     /** source transaction complete*/
  DMA_IRQ_DSTTRAN    = 0x08,     /** destination transaction complete*/
  DMA_IRQ_ERR        = 0x10,     /** error*/
  DMA_IRQ_ALL        = 0x1f      /** all interrupts*/
}DMA_IRQType;
```

In acest mod rutina de servire a intreruperii poate sa faca direrenta intre diferitele situatii pentru care s-a generat intreruperea pentru un canal (este o singura intrerupere generate pentru unn canal).

Utilizarea DMA din mediul DAVE CE

Se creaza un proiect nou de tip DAVE CE in care se incarac aplicatia DMA003 ca in figura 5:

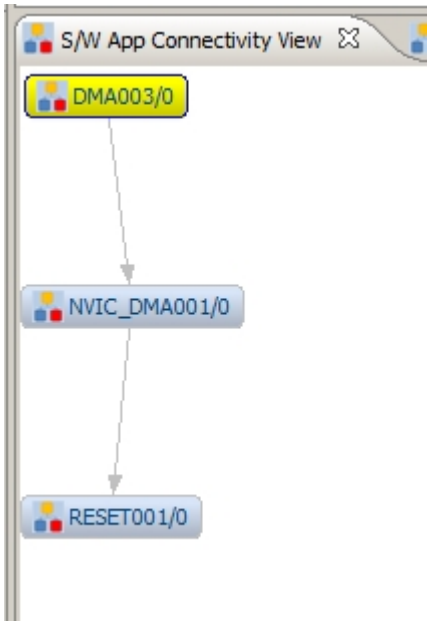


Figura 5

Blocul **DMA003** se programeaza grafic astfel (figura 6):

- transfer memorie-memorie, declansat software, dimensiunea blocului 10 cuvinte de cite 32 de biti, 1 singur burst, adresa sursa – *Array1*, adresa destinatie – *Array2*.

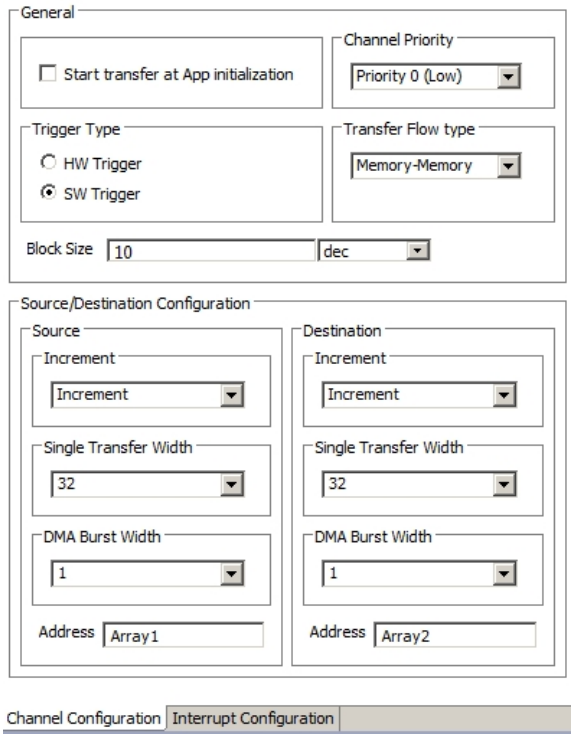


Figura 6

Intreruperile se programeaza tot din meniul grafic (intrerupere la terminarea transferului), ca in figura 7:

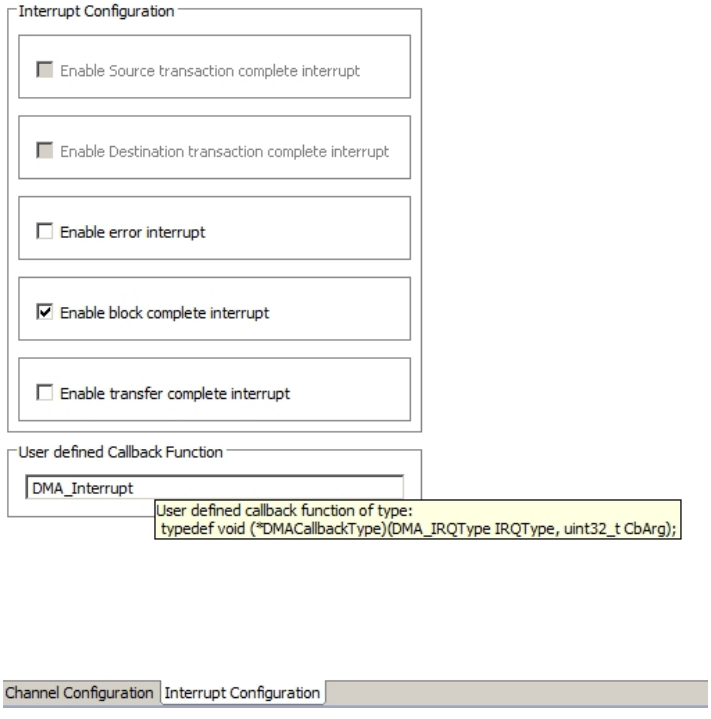


Figura 7

Adresa de servire este *DMA_Interrupt*.

Dupa selectarea grafica a configurarii DMA si a intreruperilor si generarea codului, in programul principal se vor completa liniile de cod pentru definirea tablourilor pentru transfer DMA (*Array1* si *Array2*) , se va defini rutina de servire a intreruperilor *DMA_Interrupt* , se vor seta parametii de transfer si se va initia transferul DMA.

```
#include <DAVE3.h>                //Declarations from DAVE3 Code Generation
(includes SFR declaration)

uint32_t Array1[10]={1,2,3,4,5,6,7,8,9,10};
uint32_t Array2[10];
int v = 0;


void DMA_Interrupt(DMA_IRQType IRQType,uint32_t CbArg)
{
      asm("BKPT 255");
v=1;
}


int main(void)
{
//    status_t status;          // Declaration of return variable for DAVE3
APIs (toggle comment if required)
       DMA003_ChannelConfigType ChConfig;

      DAVE_Init();              // Initialization of DAVE Apps


           DMA003_SetChannelTransferParams(&DMA003_Handle0,&ChConfig);

           DMA003_StartTransfer(&DMA003_Handle0);



      while(1)
      {

      }
      return 0;
}
```

Dupa executia programului se va verifica in depanator (pentru **Relax 4500**) ca tabloul *Array2* a contine valorile din tabloul *Array1* si variabila *v* = 1 (transferul DMA a fost incheiat)