DEZVOLTAREA APLICATIILOR CU MICROCONTROLERUL XMC 4500

DESCRIEREA ARHITECTURII SI A MEDIULUI INTEGRAT DE DEZVOLTARE DAVE

Desfasurarea lucrarii

1. Studiul arhitecturii familiei de microcontrolere XMC 4500 si arhitecturii placii de dezvoltare RELAX 4500
2. Modul de creare a unui proiect nou in DAVE. Exemplu de proiect simplu pentru porturile de intrare-iesire
3. Studiul programului exemplificat (organigrama generala, utilizarea intreruperilor, modul de programare a porturilor IO, analiza programului)
4. Executia pas cu pas si depanarea programului (stabilirea configuratiei pentru depanator, lansarea depanatorului si vizualizarea resurselor programului)
5. Modificarea programului (alte functionalitati)

Teme

1. Prin studierea schemelor electrice, sa se verifice corespondenta dintre butoane si LED-uri si porturile IO asociate.
2. Pentru toate porturile IO, sa se verifice modul de configurare.
3. Sa se realizeze un program care implementeaza o telecomanda cu urmatoarele specificatii:

- apasarea butonului Button 2 – comuta intre modul de lucru: TV sau DVD
- LED-ul LED 2 – stins semnifica mod de lucru TV si aprins semnifica mod de lucru DVD
- pe LED-ul LED 1 se transmite comanda – 3 pulsuri daca se comanda TV si 5 pulsuri daca se comanda DVD. Durata pulsurilor este de 0.5 sec cu pauza intre pulsuri de 1 sec.

Arhitectura microcontrolelor XMC4500

Familia de microcontrolere XMC 4500 combina functionalitatea procesorului ARM Cortex-M4 cu periferice si memorie in acelasi chip si are urmatoarele facilitati:

Subsistemul CPU

1. Nucleul procesorului
   - processor ARM Cortex-M4 pe 32 de biti
   - set de instructiuni pe 16 sau 32 de biti
   - instructiuni DSP/MAC
   - timer de sistem pentru suport de sistem de operare
2. Unitate de virgula mobila
3. Unitate de protectie a memoriei
4. Controler de intreruperi inlantuite
5. Doua blocuri de transfer DMA
6. Unitate de cerere a evenimentelor (pentru servicii interne sau externe)
7. Bloc de detectie a erorilor multiple (CRC)

Memorii on chip

- 16 ko ROM (boot)
- 64 ko memorie de program de mare viteza
- 64 ko memorie de date de mare viteza
- 32 ko memorie de mare viteza pentru comunicatie
- 1024 ko memorie flash cu 4 ko memorie cache

Dispozitive periferice pentru comunicatie

- modul Ethernet 10/100 Mbit
- modul USB
- interfata CAN (Controller Area Network)
- 6 interfete seriale (configurabile in diferite standarde seriale)
- interfata pentru comunicarea om-masina (LED si touch)
- interfata pentru carduri de memorie externa (SD si SDMMC)
- bus extern pentru conectarea unor memorii externe

Periferice pentru semnale analogice

- 4 convertoare ADC pe 12 biti cu cite 8 canale fiecare
- Demodulator Sigma Delta cu 4 canale
- 1 convertor DAC pe 12 biti cu 2 canale

Periferice pentru control industrial

- 2 unitati de captura si comparare pentru controlul motoarelor

- 4 unitati de captura si comparare folosite ca timere de uz general
- 2 interfete de determinare a pozitiei
- timer de tip *watchdog*
- senzori de temperature
- ceas de timp real
- unitate de control a sistemului

Linii de intrare – iesire

- modul pentru porturi programabile
- adresabilitate pe bit
- intrari tri-state
- interfata de test JTAG (Joint Test Action Group)
- suport pentru depanare

Figura 1 ilustreaza blocurile functionale si modul lor de conectare pentru un sistem cu XMC 4500.

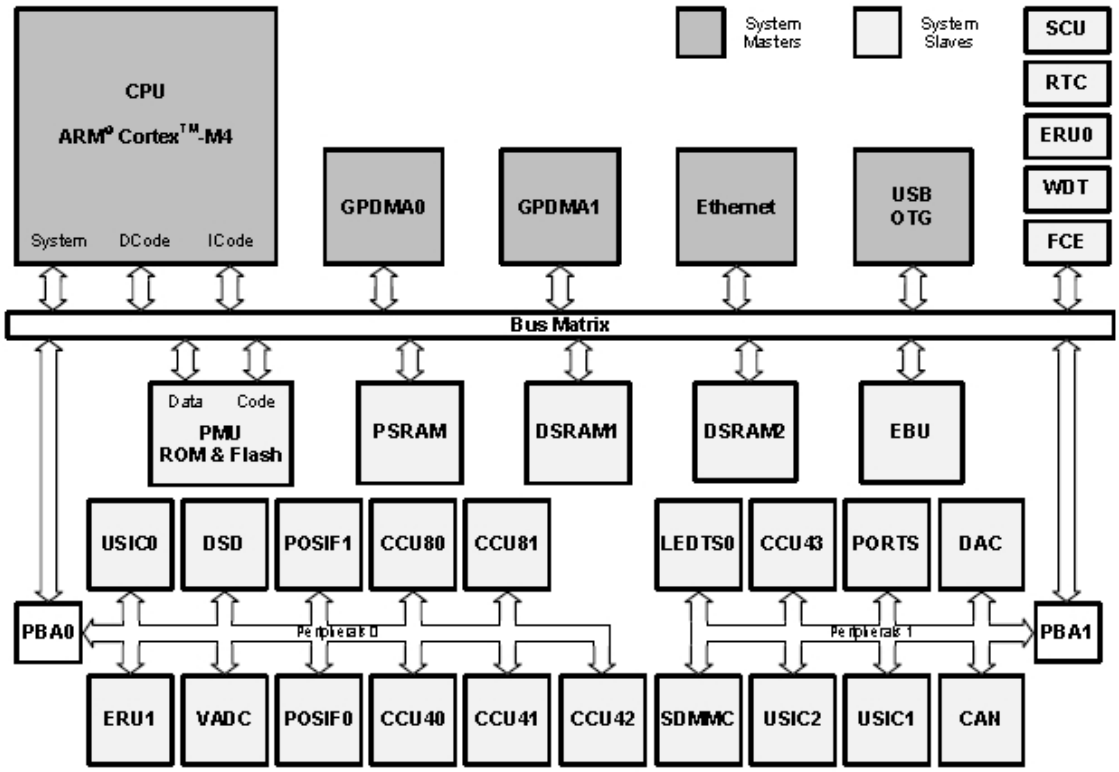


Figura 1.

SCU – System Control Unit
RTC – Real Time Clock
WDT – Watchdog
FCE – Flexible CRC Engine
GPDMA – General Purpose DMA
USB OTG – Universal Serial Bus On The Go
PMU – Protected Memory Unit
PSRAM – Program SRAM
DSRAM – Data SRAM
EBU – External Bus Unit
DSD – Delta Sigma Demodulator
POSIF – Position Interface
CCU – Capture Compare Unit
LEDTS – LED and Touch Sense (Control Unit)
PORTS – porturi IO
DAC – convertor digital analogic
PBA – Peripheral Bridge AHB to AHB (Advanced High-performance Bus)
ERU – Event Request Unit
VADC – Versatile ADC (convertor analog digital)
USIC – Universal Serial Interface Channel
CAN – Controller Area Network

Arhitectura procesorului ARM Cortex-M4

Caracteristici principale:

- arhitectura Harvard
- pipe-line cu 3 stagii
- set de instructiuni eficient - prelucrari in virgula fixa intr-un singur ciclu, operatii SIMD (Single Instruction Multiply Data) de inmultire si inmultire cu acumulare, logica de saturare si operatii de impartire hardware
- set de instructioni simplificat si comprimat (Thumb) pentru cresterea densitatii codului si a vitezei
- densitate mare a codului
- controler de intreruperi configurabil (64 de intreruperi)
- mod *sleep*

Blocurile functionale ale procesorului ARM Cortex-M4 sint prezentate in figura 2:

- Nucleul procesorului (Processor Core)
- Unitatea de virgula mobila (Floating Point Unit)
- Controlerul de intreruperi (Nested Vectored Interrupt Controler)
- Unitatea de protectie a memoriei (Memory Protection Unit)

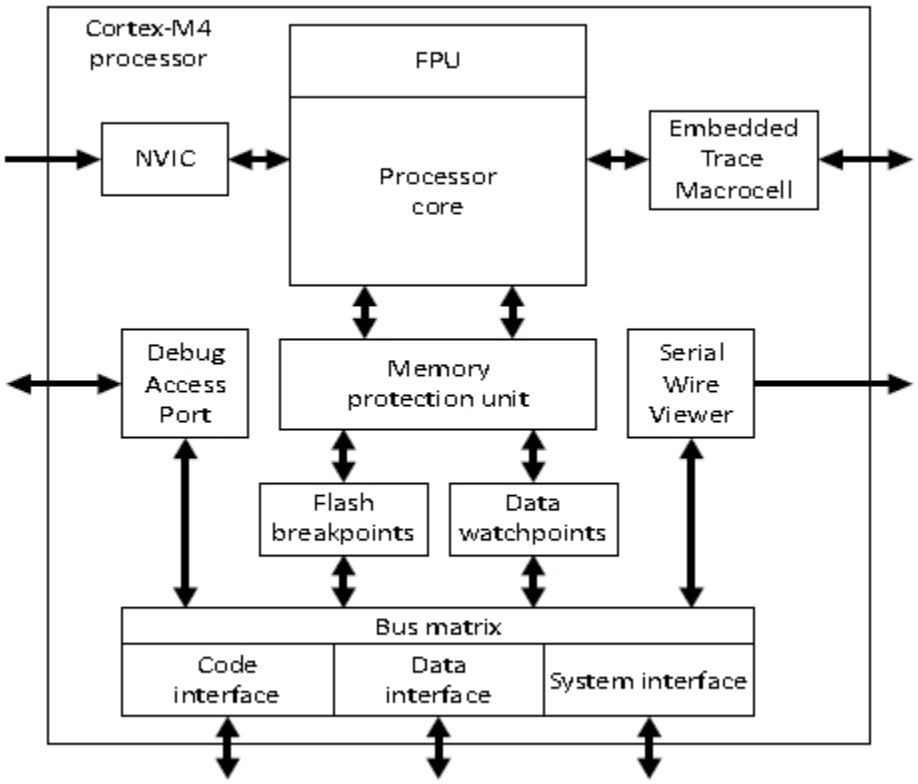


Figura 2

Modelul de programare

Exista urmatoarele moduri de lucru ale procesorului ARM – Cortex M4:

- modul thread – in care se executa un program de aplicatie. In acest mod se intra imediat dupa reset
- modul handler – se trateaza exceptii. Dupa tratarea unei exceptii se revine in modul thread.

Programele se executa pe doua niveluri de privilegiere astfel:

- **modul neprivilegiat** (programele nu pot executa anumite instructiuni de acces la registre speciale, nu se poate folosi timer-ul de sistem si blocul de control al sistemului, pot avea acces restrictionat la memorie)
- **modul privilegiat** (se intra pe acest nivel din modul thread daca actualizeaza adecvat un registru de control; modul hanller este intotdeauna pe nivelul privilegiat)

Registrele procesorului

Toate registrele sint prezentate in figura 3.

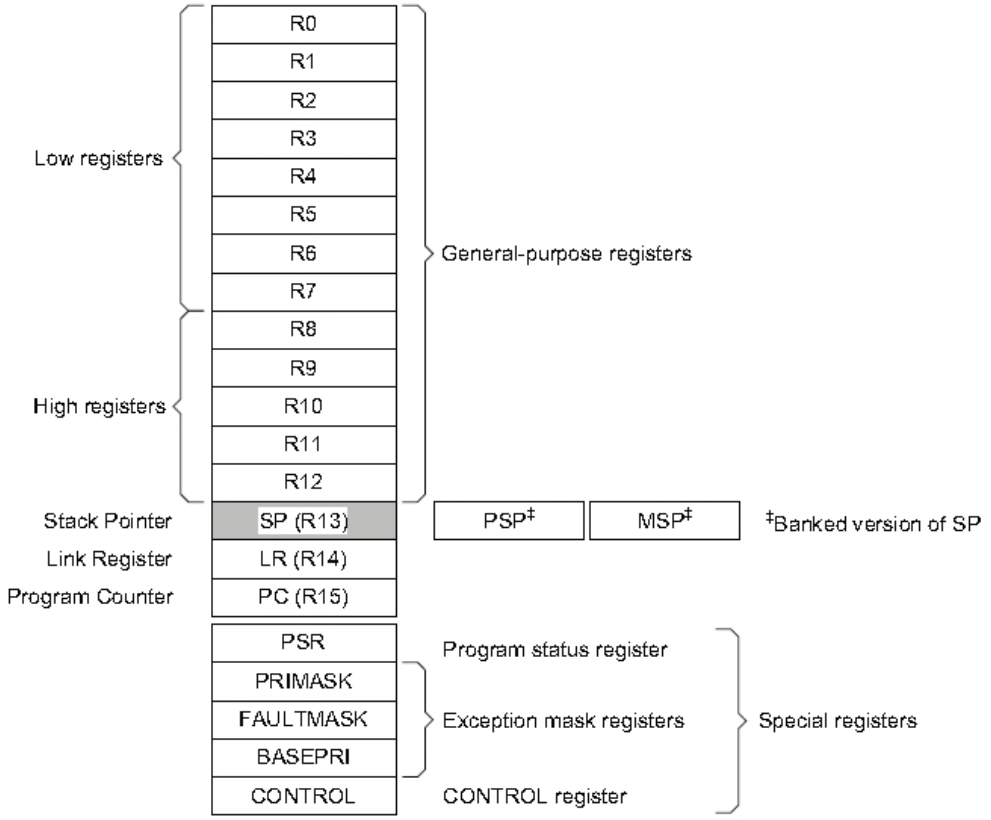


Figura 3.

Se impart in registre de date (generale) pe 32 de biti , pentru controlul programului si al stivei si registre speciale.
Registrele speciale indica starea procesorului dupa cum urmeaza:

PSR - Program Status Register
ASPR - Application Program Status
IPSR - Interrupt Program Status
EPSR - Execution Program Status
PRIMASK - Priority Mask Register on
FAULTMASK - Fault Mask Register
BASEPRI - Base Priority Mask Register
CONTROL - CONTROL register

Modelul de memorie

Harta memoriei este ilustrata in figura 4.

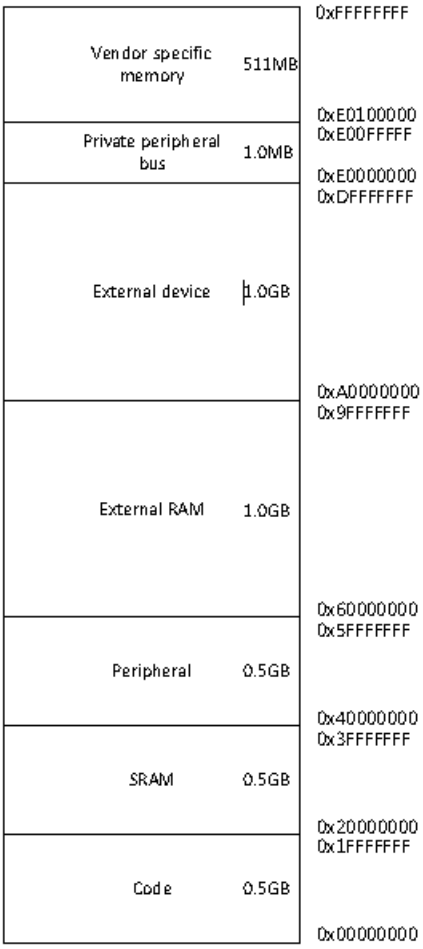


Figura 4.

Exista zone de memorie rezervate pentru cod, date si dispozitive de intrare – iesire. Prin modul de programare al blocului MPU fiecare zona de memorie poate avea attribute specifice:

- **Normal** – se pot re-ordona accesele la memorie pentru cresterea eficientei sau executa citiri speculative din memorie
- **Device** – se conserva ordinea acceselor la memorie, relativ la alte accese de tip **Device** sau **Strongly-ordered**
- **Strongly-ordered** – se conserva ordinea acceselor la memorie, relativ la toate tipurile de acces
- **Execute Never** – nu se pot accesa instructiuni

.

Porturile de intrare – iesire

Exista 16 porturi de intrare – iesire, **PCn** (n=0-15) cu structura prezentata in figura 5 . In structura unui port IO se observa driverele de intrare si iesire (sectiunea *pad*) si sectiunea de configurare pe biti (*slice*). In modul normal, o intrare este citita pe pinul **Pn_IN** iar o iesire este generata pe pinul **Pn_OUT**. In modul alternativ, se pot conecta direct semnale de la dispositive periferice cu portul **PCn**.

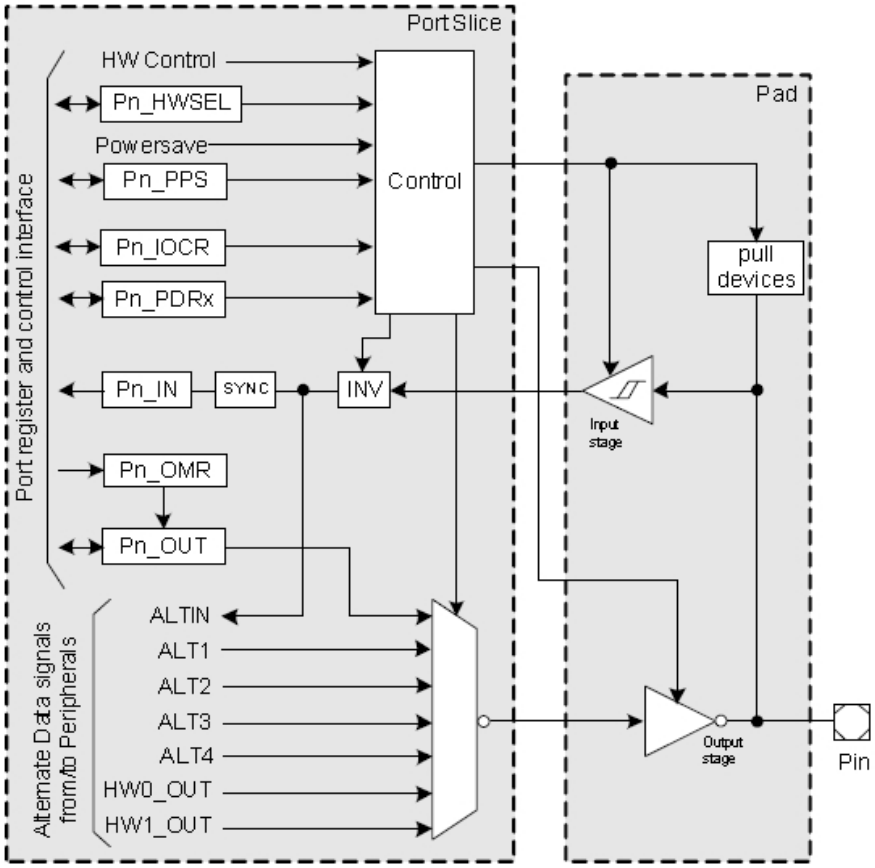


Figura 5.

Atunci cind un port este configurat ca intrare/iesire trebuie scrise urmatoarele registre:

Registrele **Pn_IOCR0**, **Pn_IOCR4**, **Pn_IOCR8** si **Pn_IOCR12** – configureaza bitii 0-3, 4-7, 8-11 si respectiv 12-15 ai portului **Pn** astfel (reprezentare pentru **Pn_IOCR0**, celelelte registre sint similare, astfel **PCx** cu x = 0-3 pentru **Pn_IOCR0,** x = 4-7 pentru **Pn_IOCR4,** x = 8-11 pentru **Pn_IOCR8 si** x = 12-15 pentru **Pn_IOCR12**) ca in figura 6.

31 27	26 24	23 19	18 16	15 11	10 8	7 3	2 0
PC3	**0**	**PC2**	**0**	**PC1**	**0**	**PC0**	**0**
rw	r	rw	r	rw	r	rw	r

PC0, **PC1,** **PC2,** **PC3**	[7:3], [15:11], [23:19], [31:27]	rw	**Port Control for Port n Pin 0 to 3** This bit field determines the Port n line x functionality (x = 0-3) according to the coding table
0	[2:0], [10:8], [18:16], [26:24]	r	**Reserved** Read as 0; should be written with 0.

Standard PCx Coding

PCx[4:0]	I/O	Output Characteristics	Selected Pull-up / Pull-down / Selected Output Function
$0X000_B$	Direct Input	–	No input pull device connected
$0X001_B$			Input pull-down device connected
$0X010_B$			Input pull-up device connected
$0X011_B$			No input pull device connected; Pn_OUTx continuously samples the input value
$0X100_B$	Inverted Input	–	No input pull device connected
$0X101_B$			Input pull-down device connected
$0X110_B$			Input pull-up device connected
$0X111_B$			No input pull device connected; Pn_OUTx continuously samples the input value
10000_B	Output (Direct Input)	Push-pull	General-purpose output
10001_B			Alternate output function 1
10010_B			Alternate output function 2
10011_B			Alternate output function 3
10100_B			Alternate output function 4
10101_B			Reserved.
10110_B			Reserved.
10111_B			Reserved.
11000_B		Open-drain	General-purpose output
11001_B			Alternate output function 1
11010_B			Alternate output function 2
11011_B			Alternate output function 3
11100_B			Alternate output function 4
11101_B			Reserved.
11110_B			Reserved.
11111_B			Reserved.

Figura 6 .

Registrele **Pn_PDR0/1** ofera posibilitatea de a selecta viteza de variatie a iesirii (slew rate) si curentul de iesire (driver strength) pentru pinii configurati ca iesire.

Registrul **Pn_PDR0** are urmatoarea configuratie (figura 7):

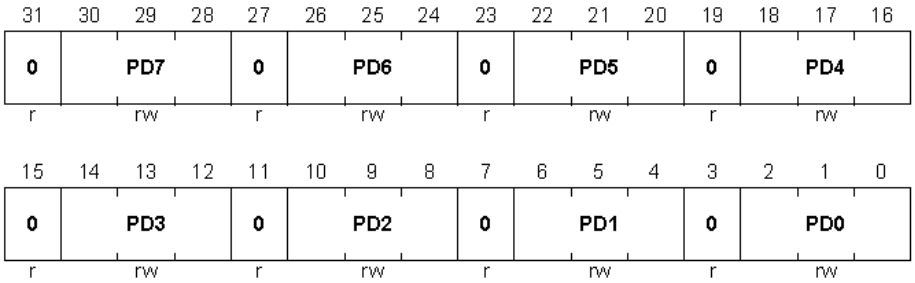


Figura 7

Registrul **Pn_PDR1** are urmatoarea configuratie (figura 8):

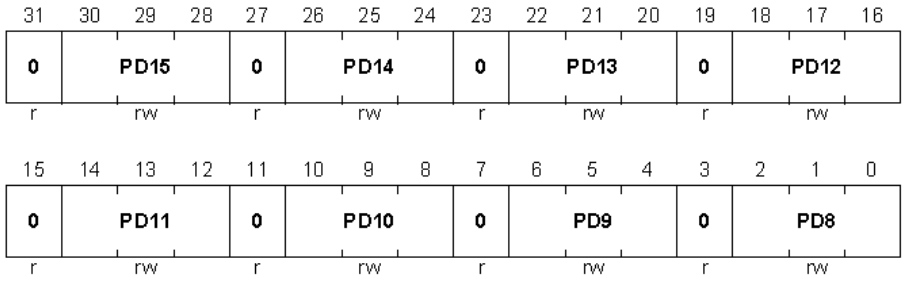


Figura 8

In descrierile de mai sus **PDi** (i = 0-15) reprezinta bitii de configurare pentru pinul **i** al portului **Pn.**

Selectia modului de lucru pentru sectiunea pad este data de figura 9:

Pad Driver Mode Selection

Pad Class	PDx.2	PDx.1	PDx.0	Functionality
A1	X	X	0	Medium driver
			1	Weak driver
A1+	0	X	0	Strong driver soft edge
	0	X	1	Strong driver slow edge
	1	X	0	Medium driver
	1	X	1	Weak driver
A2	0	0	0	Strong driver, sharp edge
	0	0	1	Strong driver, medium edge
	0	1	0	Strong driver, soft edge
	0	1	1	Reserved
	1	0	0	Medium driver
	1	0	1	
	1	1	0	Reserved
	1	1	1	Weak driver

Figura 9

Registrul **Pn_OMR** modifica iesirea si are urmatoarea configuratie (figura 10):

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PR 15	**PR 14**	**PR 13**	**PR 12**	**PR 11**	**PR 10**	**PR 9**	**PR 8**	**PR 7**	**PR 6**	**PR 5**	**PR 4**	**PR 3**	**PR 2**	**PR 1**	**PR 0**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PS 15	**PS 14**	**PS 13**	**PS 12**	**PS 11**	**PS 10**	**PS 9**	**PS 8**	**PS 7**	**PS 6**	**PS 5**	**PS 4**	**PS 3**	**PS 2**	**PS 1**	**PS 0**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Function of the Bits PRx and PSx

PRx	PSx	Function
0	0	Bit Pn_OUT.Px is not changed.
0	1	Bit Pn_OUT.Px is set.
1	0	Bit Pn_OUT.Px is reset.
1	1	Bit Pn_OUT.Px is toggled.

Figura 10

Registrul **Pn_PPS** valideaza starea pinilor portului Pn in modul *power save* astfel (figura 11):

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPS 15	**PPS 14**	**PPS 13**	**PPS 12**	**PPS 11**	**PPS 10**	**PPS 9**	**PPS 8**	**PPS 7**	**PPS 6**	**PPS 5**	**PPS 4**	**PPS 3**	**PPS 2**	**PPS 1**	**PPS 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
PPSx (x = 0-15)	x	rw	**Port n Pin Power Save Bit x** 0_B Pin Power Save of Pn.x is disabled. 1_B Pin Power Save of Pn.x is enabled.
0	[31:16]	r	**Reserved** Read as 0.

Figura 11

Starea pinilor in modul power save este data in conformitate cu configurarea stabilita de registrul **Pn_IOCR** astfel (figura 12)

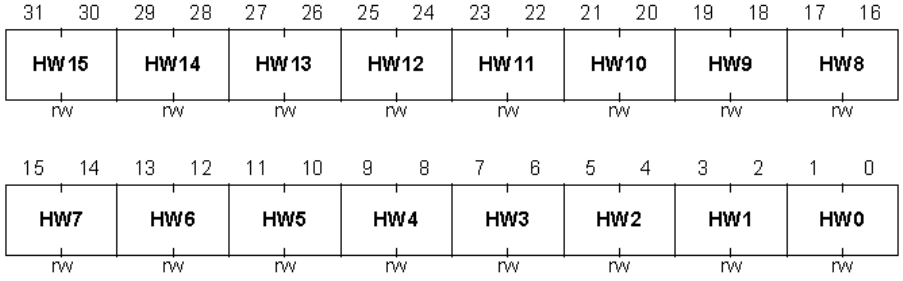
PCx Coding in Deep-Sleep mode

PCx[4:0]	I/O	Deep-Sleep mode and PPSx=1_B
$0X000_B$	Direct Input	Input value=Pn_OUTx
$0X001_B$		Input value=0_B; pull-down deactivated
$0X010_B$		Input value=1_B; pull-up deactivated
$0X011_B$		Input value=Pn_OUTx, storing the last sampled input value
$0X100_B$	Inverted Input	Input value=Pn_OUTx
$0X101_B$		Input value=1_B; pull-down deactivated
$0X110_B$		Input value=0_B; pull-up deactivated
$0X111_B$		Input value=Pn_OUTx, storing the last sampled input value
$1XXXX_B$	Output	Output driver off, Input Schmitt-Trigger off, no pull device active, Input value=Pn_OUTx

Figura 12

Intrarea este citita in registrul **Pn_IN** si iesirea este scrisa in registrul **Pn_OUT**.

Registrul **Pn_HWSEL** asociaza pinii porturilor **Pn** cu semnale externe (pe modul alternativ) ca in figura 13:



Field	Bits	Type	Description
HWx **(x = 0-15)**	[2*x+1: 2*x]	rw	**Port n Pin Hardware Select Bit x** 00_B Software control only. 01_B HW0 control path can override the software configuration. 10_B HW1 control path can override the software configuration. 11_B Reserved.

Figura 13

Placa de evaluare RELAX 4500

Schema bloc a placii de dezvoltare Relax 4500 este prezentata in figura 14.
Este constituita din urmatoarele blocuri functionale:

- Microcontrolerul XMC 4500
- Blocul de depanare (construit cu un al doile microcontroller XMC 4500 si cu interfata USB)
- Interfata Ethernet
- Doua conectoare de 40 de pini (pentru conectare cu alte echipamente)
- Sursa de alimentare
- Doua butoane (conectate pe portul P1 bitii 14 si 15) si doua LED-uri (conectate pe portul P1 bitii 1 si 0) pentru utilizator
- Interfata USB
-

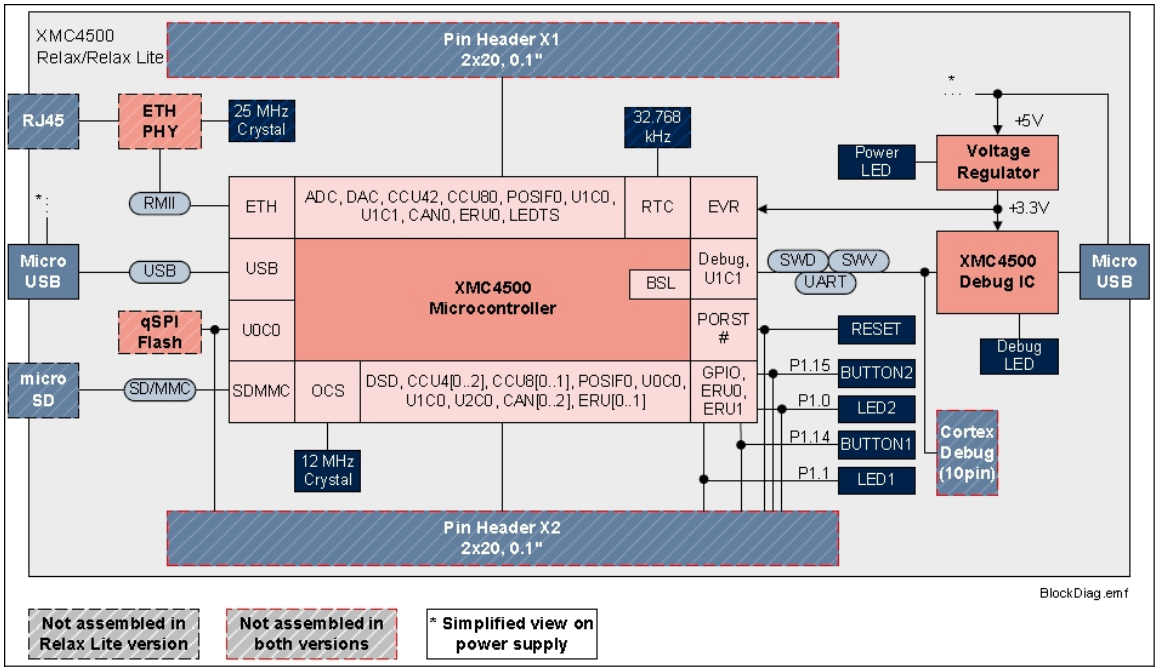
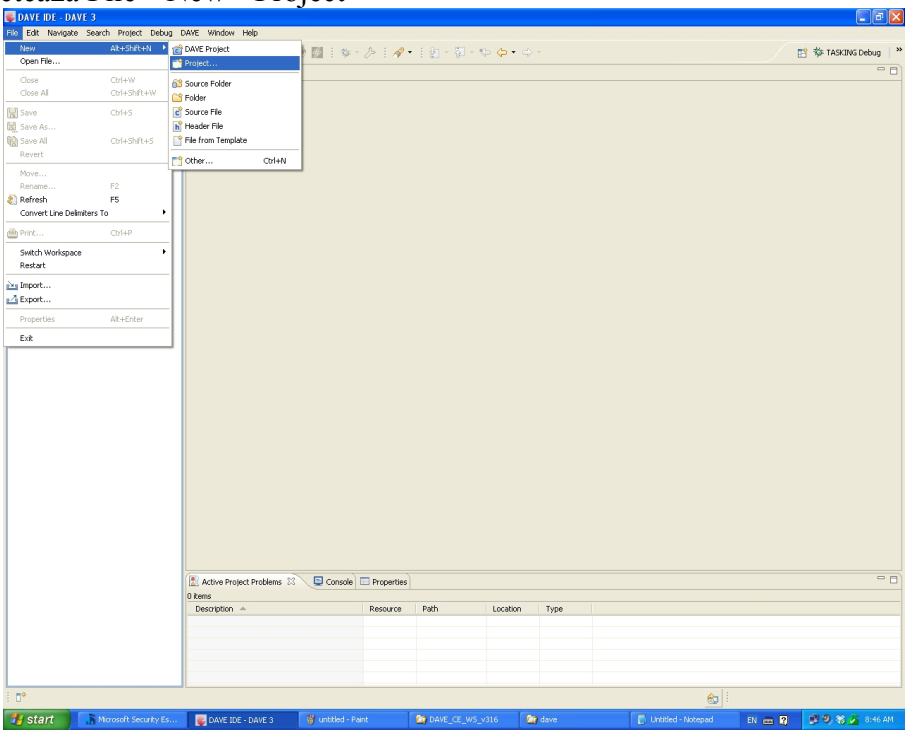


Figura 14.

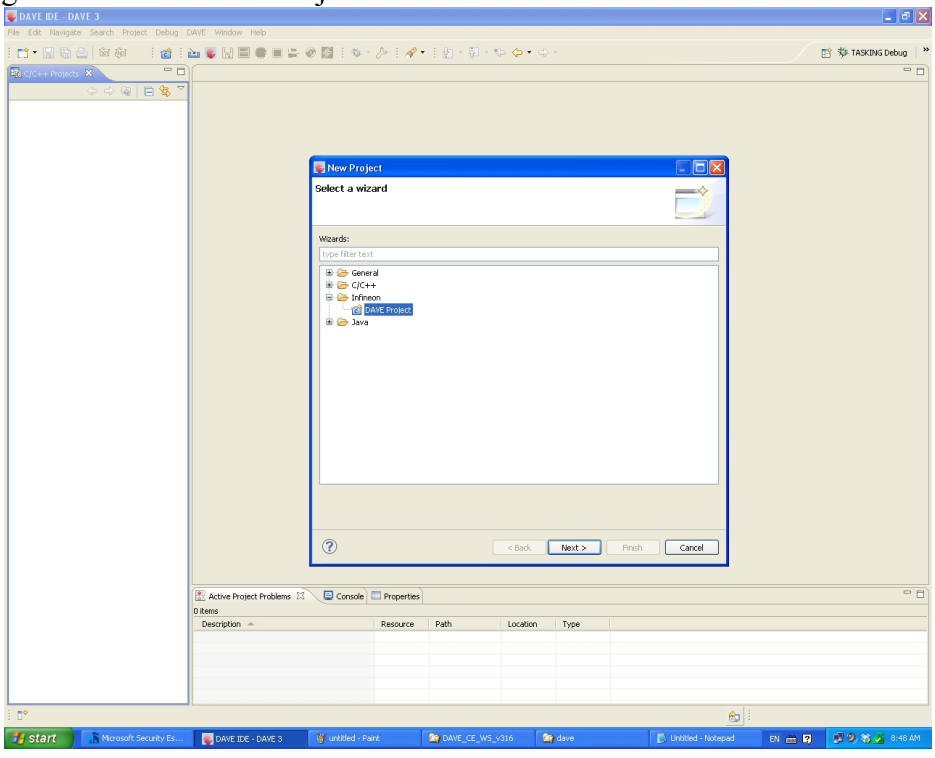
Schemele electrice ale placii **Relax 4500** sint ilustrate in anexa.

Crearea unui program cu ajutorul mediului de dezvoltare DAVE

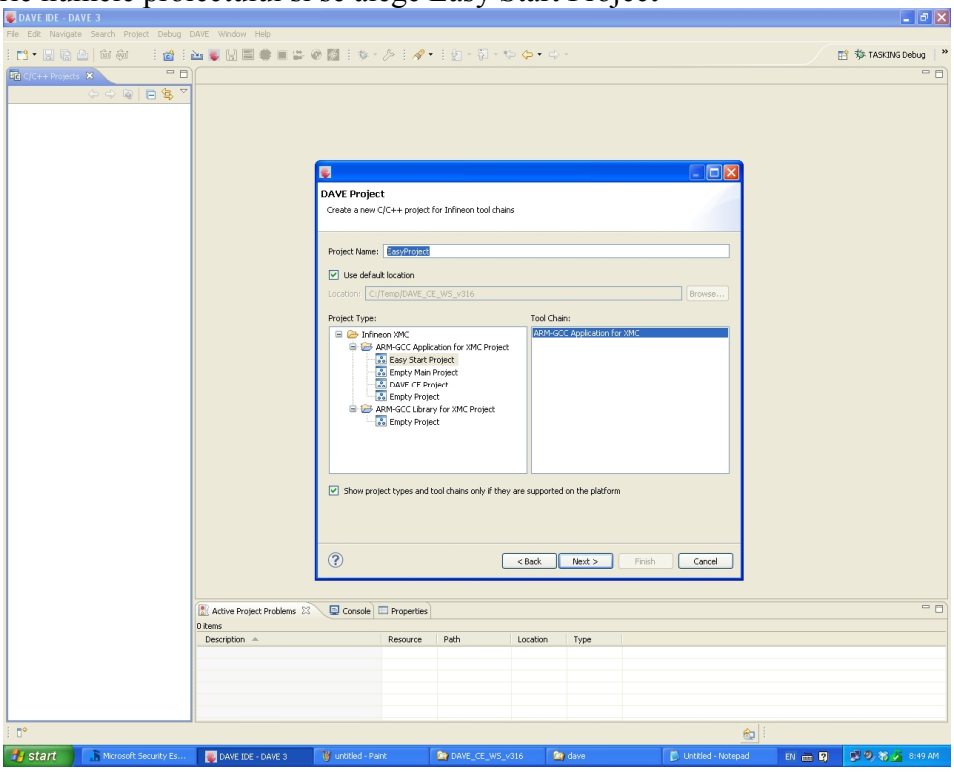
1. Se selecteaza File->New->Project



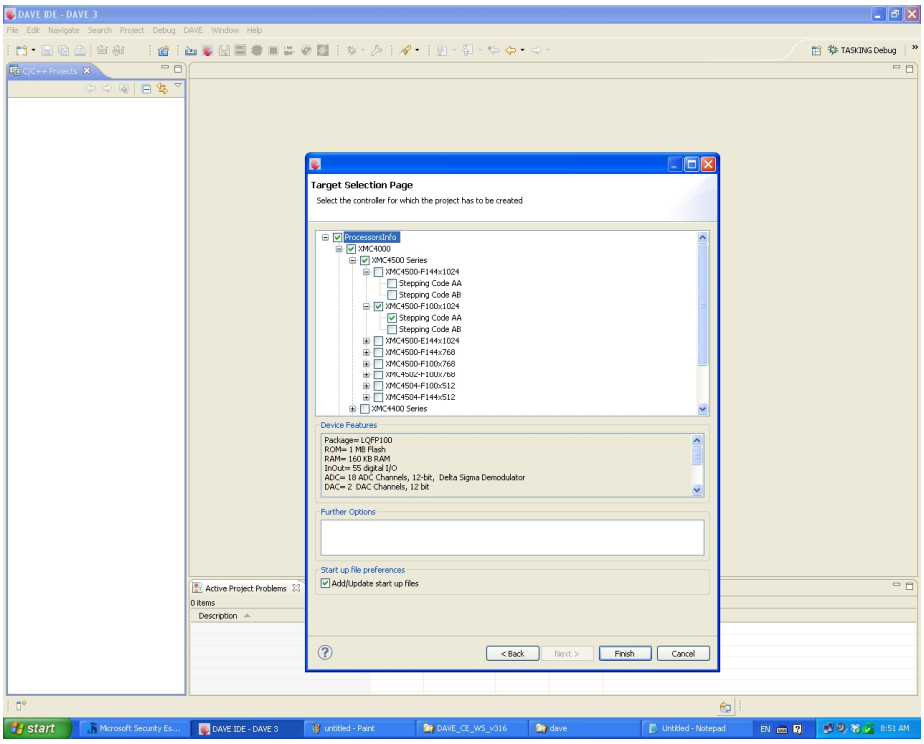
2. Se alege Infineon->DAVE Project



3. Se scrie numele proiectului si se alege Easy Start Project

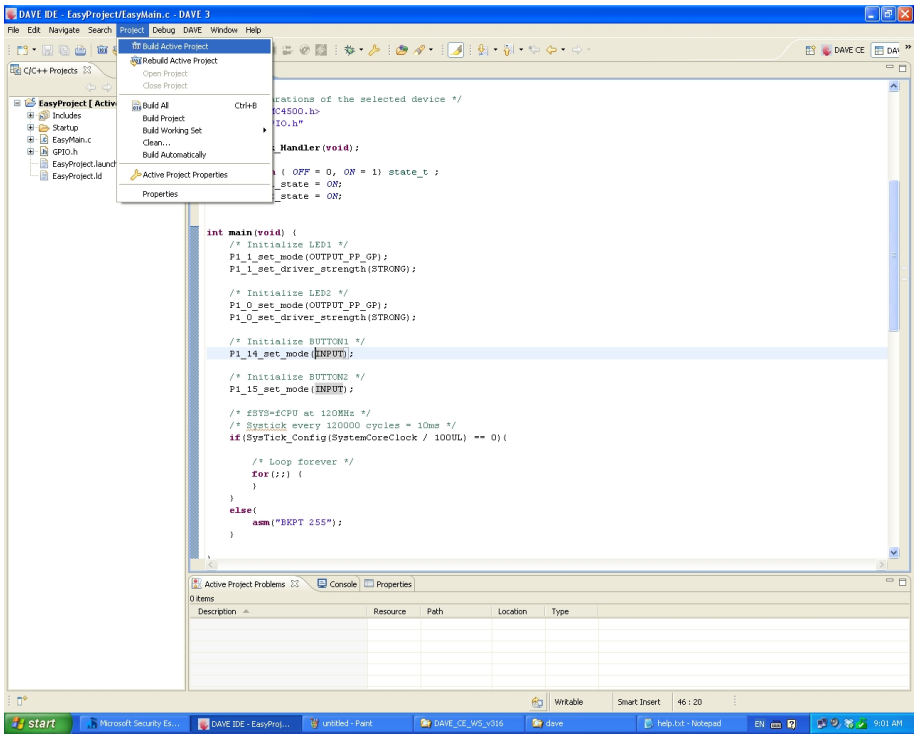


4. Se alege tipul de microcontroler (XMC 4500 series - F100x1024) si se incheie procedura de creare a unui proiect cu butonul Finish.



Trecerea de la o etapa la alta se realizeaza cu butonul Next. Se poat ereveni la o etapa anterioara cu butonul Back.

Se deschide fereasta cu codul programului, EasyMain.c.



Se genereaza codul executabil cu comanda Build.

Programul EasyStart citeste butoanele **BUTTONi** si aprinde LED-urile **LEDi** (i = 1,2)**,** de pe placa **Relax 4500**, conform grafului din figura 15.

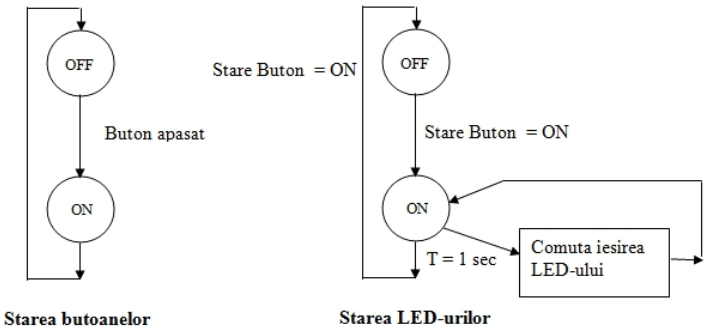


Figura 15.

Expirarea de timp de T = 1 secunda este realizata cu ajutorul unui contor care se incrementeaza la fiecare intrerupere generata de ceasul de sistem la fiecare 10 ms.

Codul programului este :

```
/* SFR declarations of the selected device */
#include <XMC4500.h>
#include "GPIO.h"

void SysTick_Handler(void);

typedef enum { OFF = 0, ON = 1} state_t ;
state_t led1_state = ON;
state_t led2_state = ON;


int main(void) {
      /* Initialize LED1 */
      P1_1_set_mode(OUTPUT_PP_GP);
      P1_1_set_driver_strength(STRONG);

      /* Initialize LED2 */
      P1_0_set_mode(OUTPUT_PP_GP);
      P1_0_set_driver_strength(STRONG);

      /* Initialize BUTTON1 */
      P1_14_set_mode(INPUT);

      /* Initialize BUTTON2 */
      P1_15_set_mode(INPUT);

      /* fSYS=fCPU at 120MHz */
      /* Systick every 120000 cycles = 10ms */[1]
      if(SysTick_Config(SystemCoreClock[2] / 100UL) == 0){

            /* Loop forever */
            for(;;) {
            }
      }
      else{
            asm("BKPT 255");
      }

}
```

[1] Oscilatorul este pe 12 MHz (vezi schemele electrice). Frecventa CPU este obtinuta prin multiplicarea frecventei oscilatorului cu 10, iar intreruperile de system (SysTick) sint generate la frecventa oscilatorului. Perioada intreruperilor de sistem este $T = \frac{1}{12.10^6}$.

Rezulta ca 12000 de ciclii au o durata de $T_1 = 120000.T = \frac{12.10^4}{12.10^6} = 10^{-2}\ \text{sec} = 10ms$

[2] `SystemCoreClock` este initializata cu 12000000

```
void SysTick_Handler(void) {
      static uint32_t ticks = 0UL;
      static state_t button1_state = OFF;
      static state_t button2_state = OFF;

      ticks++;

      /* Read BUTTON1, update state if pressed */
      if(P1_14_read() == 0UL){
            button1_state = ON;
      }
      else{
            if(button1_state == ON){
                  if(led1_state == ON){
                        led1_state = OFF;
                  }
                  else{
                        led1_state = ON;
                  }
            }
            button1_state = OFF;
      }

      /* Read BUTTON2, update state if pressed */
      if(P1_15_read() == 0UL){
            button2_state = ON;
      }
      else{
            if(button2_state == ON){
                  if(led2_state == ON){
                        led2_state = OFF;
                  }
                  else{
                        led2_state = ON;
                  }
            }
            button2_state = OFF;
      }
      /* Toggle every 1s */
      if(ticks == 100UL){
            if(led1_state == ON){
                  /* Toggle LED1 */
                  P1_1_toggle();
            }

            if(led2_state == ON){
                  /* Toggle LED2 */
                  P1_0_toggle();
            }
            ticks = 0UL;
      }
}
```

Exemplu de programare a unui port de intrare (P1 bitul 14, intrare)

```
__STATIC_INLINE void P1_14_set_mode(uint8_t mode){
    PORT1->IOCR12 &= ~0x00f80000UL;
    PORT1->IOCR12 |= mode << 16;
}

__STATIC_INLINE void P1_14_set_driver_strength(uint8_t strength){
    PORT1->PDR1 &= ~0x07000000UL;
    PORT1->PDR1 |= strength << 24;
}

__STATIC_INLINE void P1_14_set_hwsel(uint32_t config){
    PORT1->HWSEL &= ~0x30000000UL;
    PORT1->HWSEL |= config << 28;
}

__STATIC_INLINE void P1_14_set(void){
    PORT1->OMR = 0x00004000UL;
}

__STATIC_INLINE void P1_14_reset(void){
    PORT1->OMR = 0x40000000UL;
}

__STATIC_INLINE void P1_14_toggle(void){
    PORT1->OMR = 0x40004000UL;
}

__STATIC_INLINE uint32_t P1_14_read(void){
    return(PORT1->IN & 0x00004000UL);
}

__STATIC_INLINE void P1_15_set_mode(uint8_t mode){
    PORT1->IOCR12 &= ~0xf8000000UL;
    PORT1->IOCR12 |= mode << 24;
}

__STATIC_INLINE void P1_15_set_driver_strength(uint8_t strength){
    PORT1->PDR1 &= ~0x70000000UL;
    PORT1->PDR1 |= strength << 28;
}

__STATIC_INLINE void P1_15_set_hwsel(uint32_t config){
    PORT1->HWSEL &= ~0xc0000000UL;
    PORT1->HWSEL |= config << 30;
}

__STATIC_INLINE void P1_15_set(void){
    PORT1->OMR = 0x00008000UL;
}
```

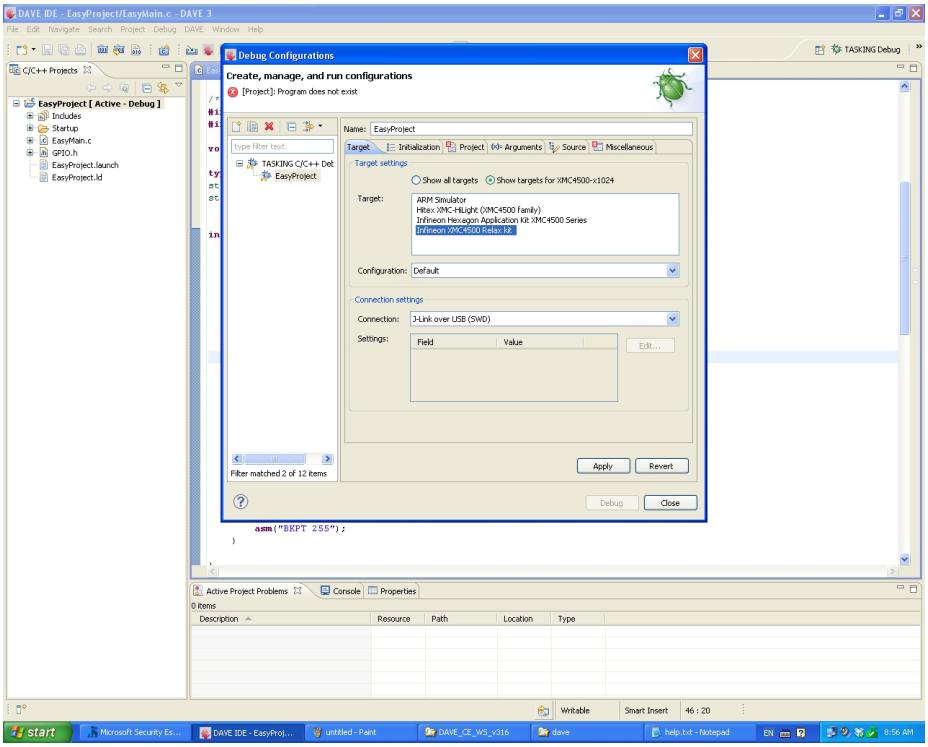
```
__STATIC_INLINE void P1_15_reset(void){
    PORT1->OMR = 0x80000000UL;
}

__STATIC_INLINE void P1_15_toggle(void){
    PORT1->OMR = 0x80008000UL;
}

__STATIC_INLINE uint32_t P1_15_read(void){
    return(PORT1->IN & 0x00008000UL);
}
```

Exemplu de programare a unui port de iesire (P1 bitul 0, iesire)

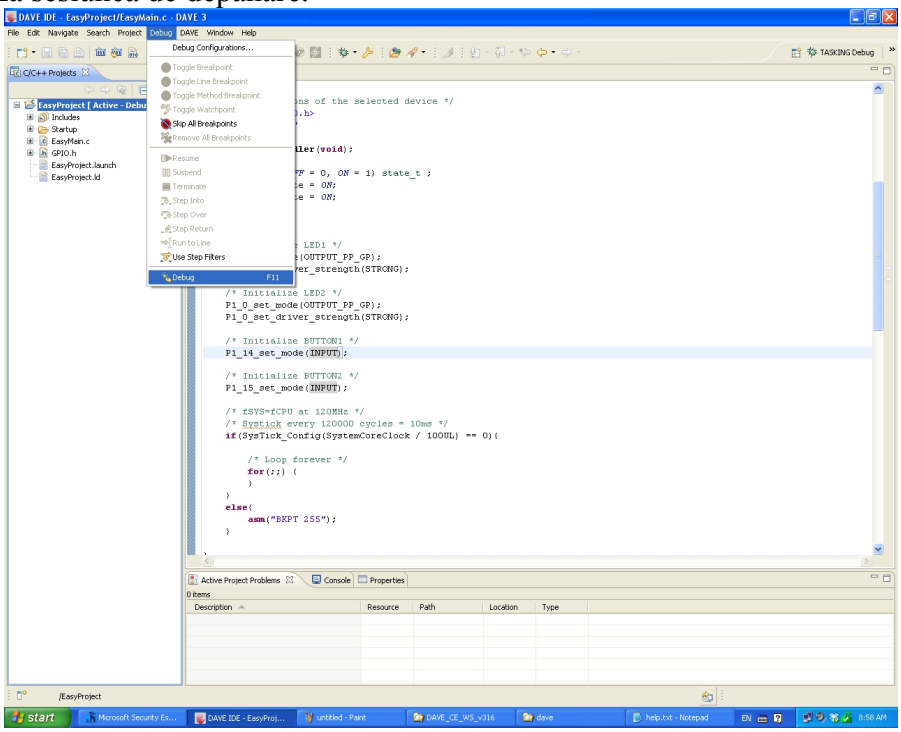
```
__STATIC_INLINE void P1_0_set_mode(uint8_t mode){
    PORT1->IOCR0 &= ~0x000000f8UL;
    PORT1->IOCR0 |= mode << 0;
}

__STATIC_INLINE void P1_0_set_driver_strength(uint8_t strength){
    PORT1->PDR0 &= ~0x00000007UL;
    PORT1->PDR0 |= strength << 0;
}

__STATIC_INLINE void P1_0_set_hwsel(uint32_t config){
    PORT1->HWSEL &= ~0x00000003UL;
    PORT1->HWSEL |= config << 0;
}

__STATIC_INLINE void P1_0_set(void){
    PORT1->OMR = 0x00000001UL;
}

__STATIC_INLINE void P1_0_reset(void){
    PORT1->OMR = 0x00010000UL;
}

__STATIC_INLINE void P1_0_toggle(void){
    PORT1->OMR = 0x00010001UL;
}

__STATIC_INLINE uint32_t P1_0_read(void){
    return(PORT1->IN & 0x00000001UL);
}
```

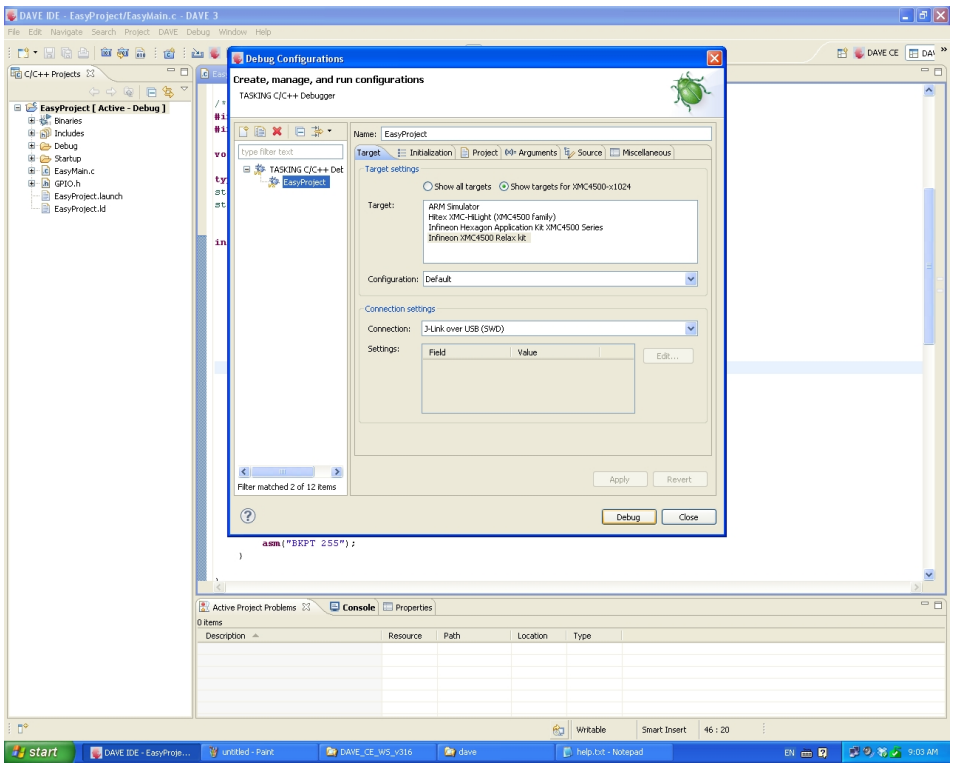
Depanarea programlui

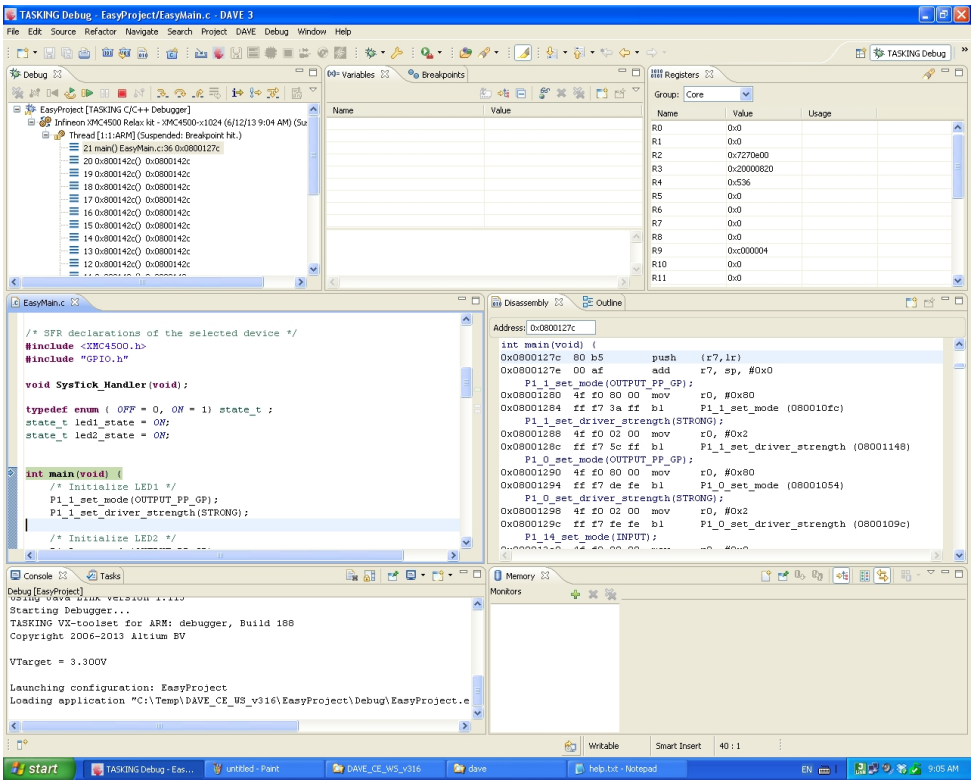
Se alege o configuratie pentru depanator. Se apasa butoanele Apply si Close, pentru a finaliza alegerea.



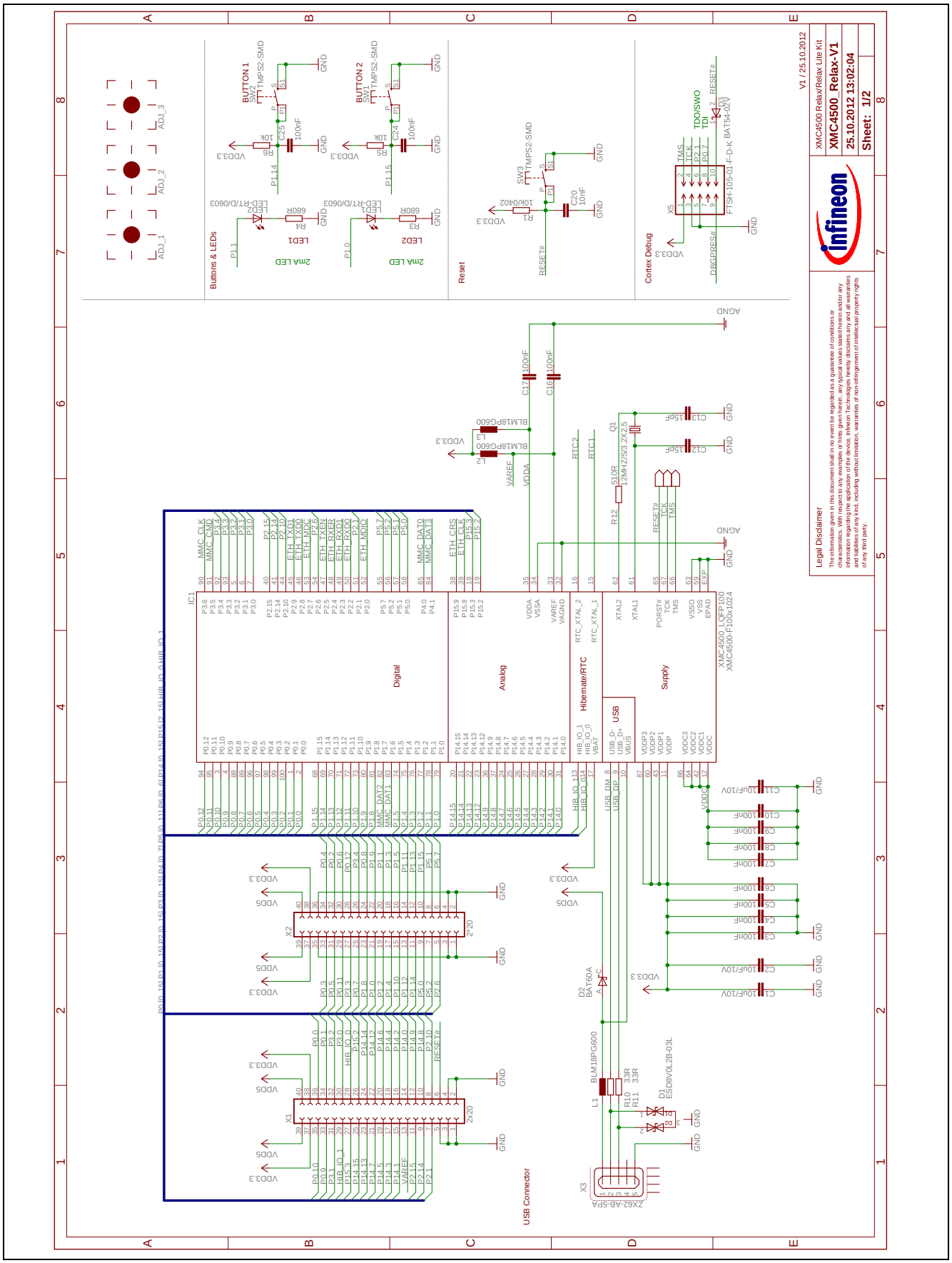
Se lanseaza sesiunea de depanare.



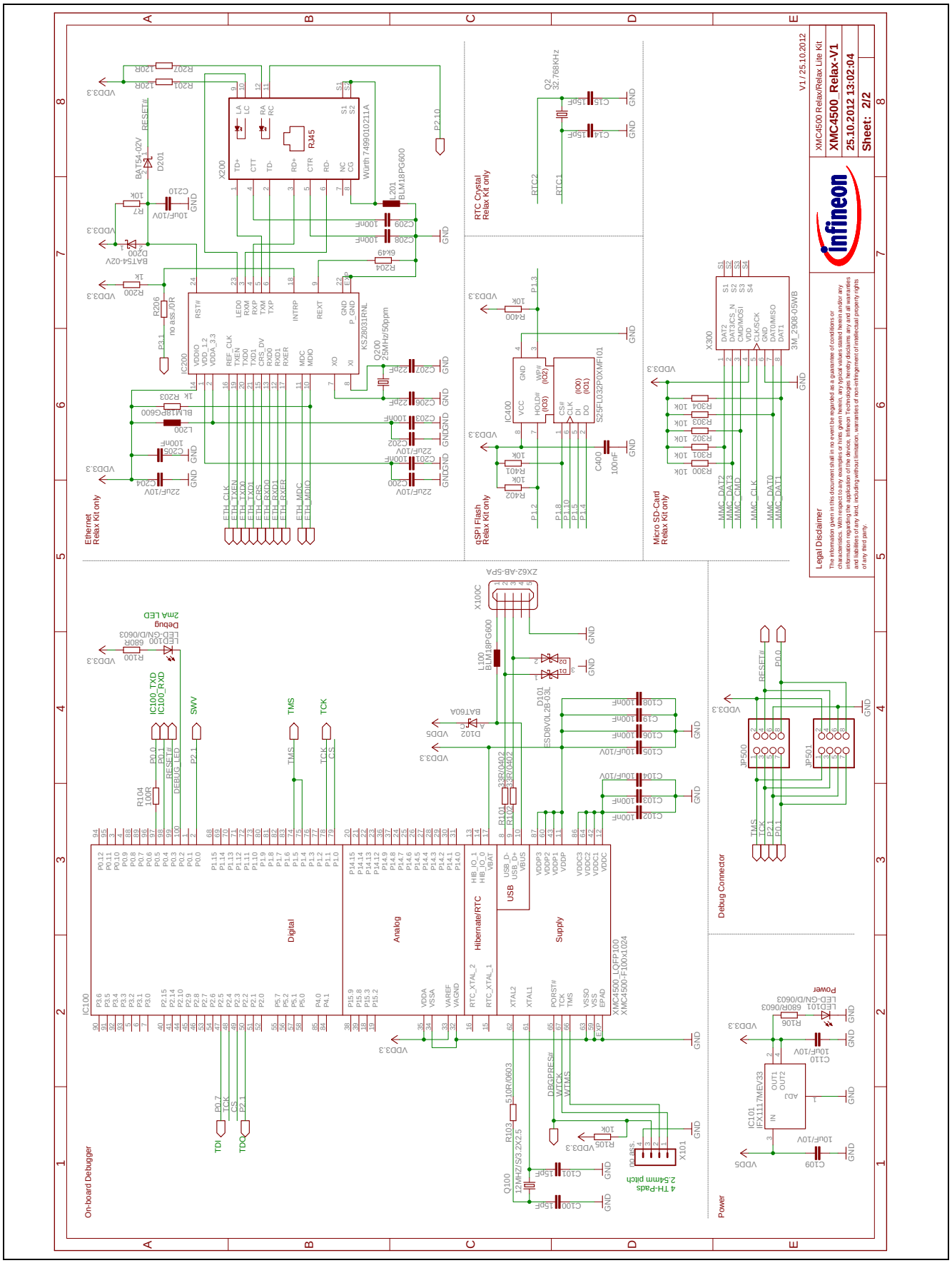




Se ruleaza programul pas cu pas si se vizualizeaza resursele.



CPU, Pin Headers, Buttons, LEDs, Reset



On-board Debugger, Power Supply, Ethernet, Quad-SPI Memory, SD Card Slot, RTC Crystal