Procesory Blackfin. Część 2

Wykład 8

Projektowanie cyfrowych układów elektronicznych

Mgr inż. Łukasz Kirchner
lukasz.kirchner@cs.put.poznan.pl
http://www.cs.put.poznan.pl/lkirchner

Plan wykładu

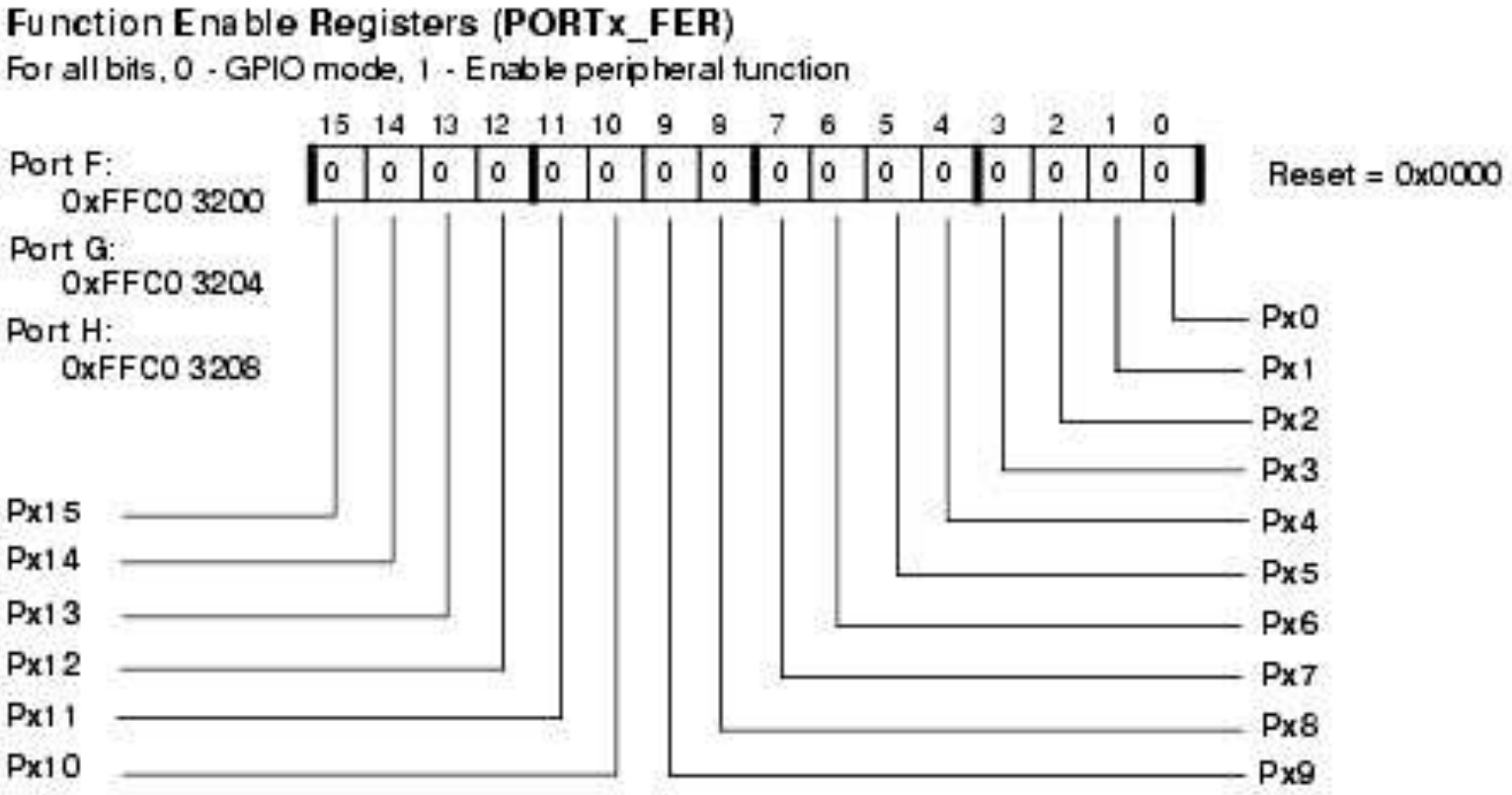
- Urządzenia peryferyjne i interfejsy wejścia/wyjścia procesorów Blackfin
- Konfiguracja elektryczna procesorów Blackfin
- Optymalizacja kodu

Bibliografia

- http://www.analog.com/processors/blackfin/
- http://www.blackfin.org/phorum/index.php

Interfejsy I/O – GPIO

- PORTx_FER



Interfejsy I/O – GPIO

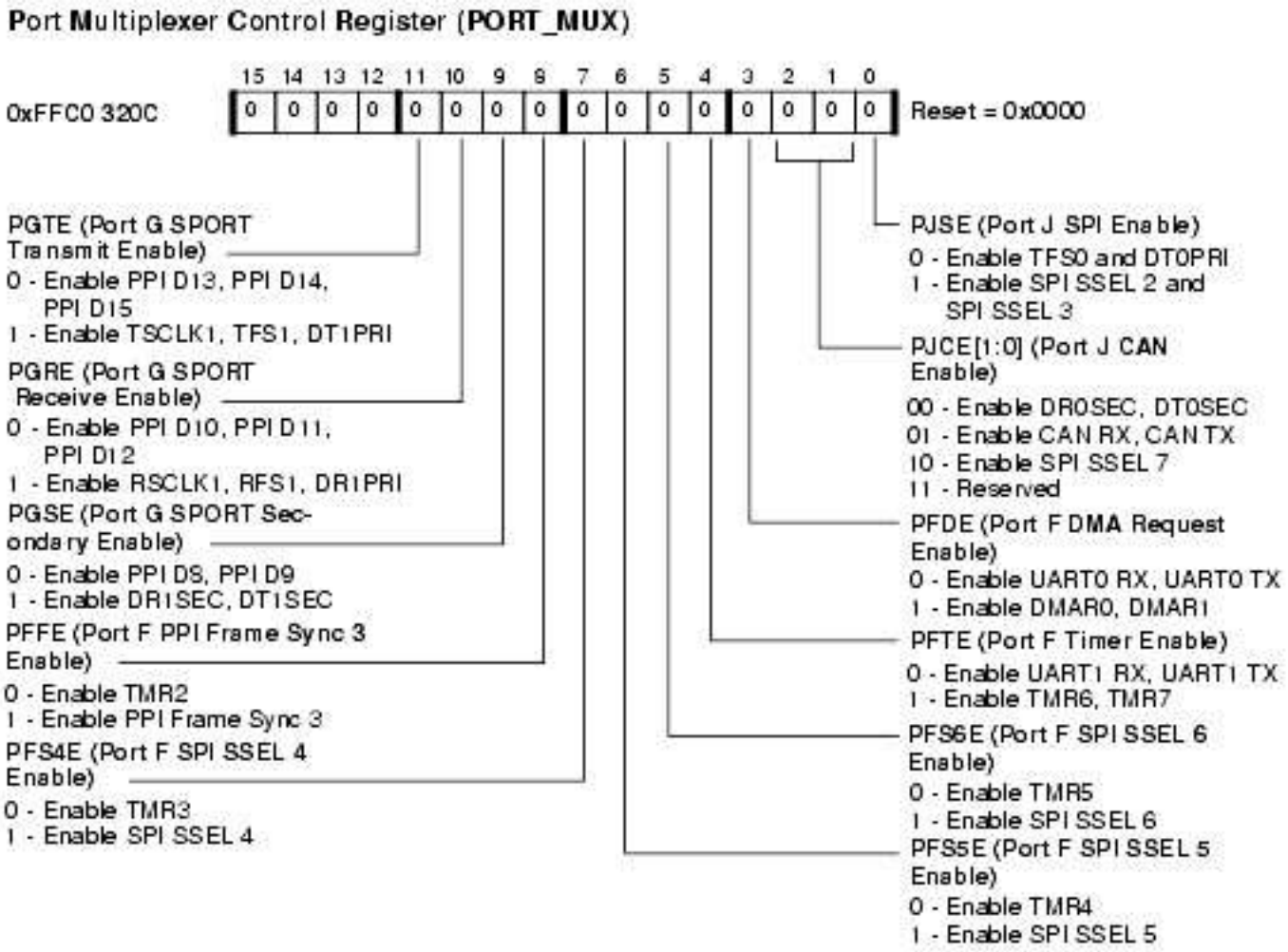
- PORTx_DIR – 0-input 1-output
- PORTxIO_INEN – 0-inp buffer disabled 1-enabled
- PORTxIO_EDGE – 0-level 1-edge
- PORTxIO_POLAR – 0-high/rising 1-low/falling
- PORTxIO_CLEAR – czyści zapisane bity
- PORTxIO_SET – ustawia zapisane bity
- PORTxIO_TOGGLE – przełącza zapisane bity

Interfejsy I/O – GPIO

- PORTxIO

POLAR	EDGE	BOTH	Effect of MMR Settings
0	0	X	Pin that is high reads as 1; pin that is low reads as 0
0	1	0	If rising edge occurred, pin reads as 1; otherwise, pin reads as 0
1	0	X	Pin that is low reads as 1; pin that is high reads as 0
1	1	0	If falling edge occurred, pin reads as 1; otherwise, pin reads as 0
X	1	1	If any edge occurred, pin reads as 1; otherwise, pin reads as 0

Interfejsy I/O – PORT_MUX



Interfejsy I/O – UART

- Universal Asynchronous Receiver/Transmitter:
- Port zgodny ze standardem RS-232C
- 5-8 bitów danych
- 1 lub 2 bity stopu
- Wszystkie tryby bitu parzystości
- Kolejki FIFO
- Brak sprzętowego wsparcia dla kontroli przepływu
- Na płycie ewaluacyjnej BF-537 piny kontroli przepływu NIE SĄ PODŁĄCZONE

Interfejsy I/O – UART

UARTx_THR	**Transmit Holding registers**
UARTx_RBR	**Receive Buffer registers**
UARTx_DLL	**Divisor Latch Low Byte registers**
UARTx_DLH	**Divisor Latch High Byte registers**
UARTx_IER	Interrupt Enable registers
UARTx_IIR	Interrupt Identification registers
UARTx_LCR	**Line Control registers**
UARTx_MCR	Modem Control registers
UARTx_LSR	Line Status registers
UARTx_SCR	8-bit Scratch register
UARTx_GCTL	Global Control registers

Interfejsy I/O – UART

- Jak włączyć:
  - PORTF_FER – włączyć piny UART
  - PORT_MUX – ustawić multiplekser
  - UART0_DL, UART0_GCTL, UART0_LCR
    - Dzielnik zegara
    - Bity stopu, danych, parzystości…

Interfejsy I/O – UART

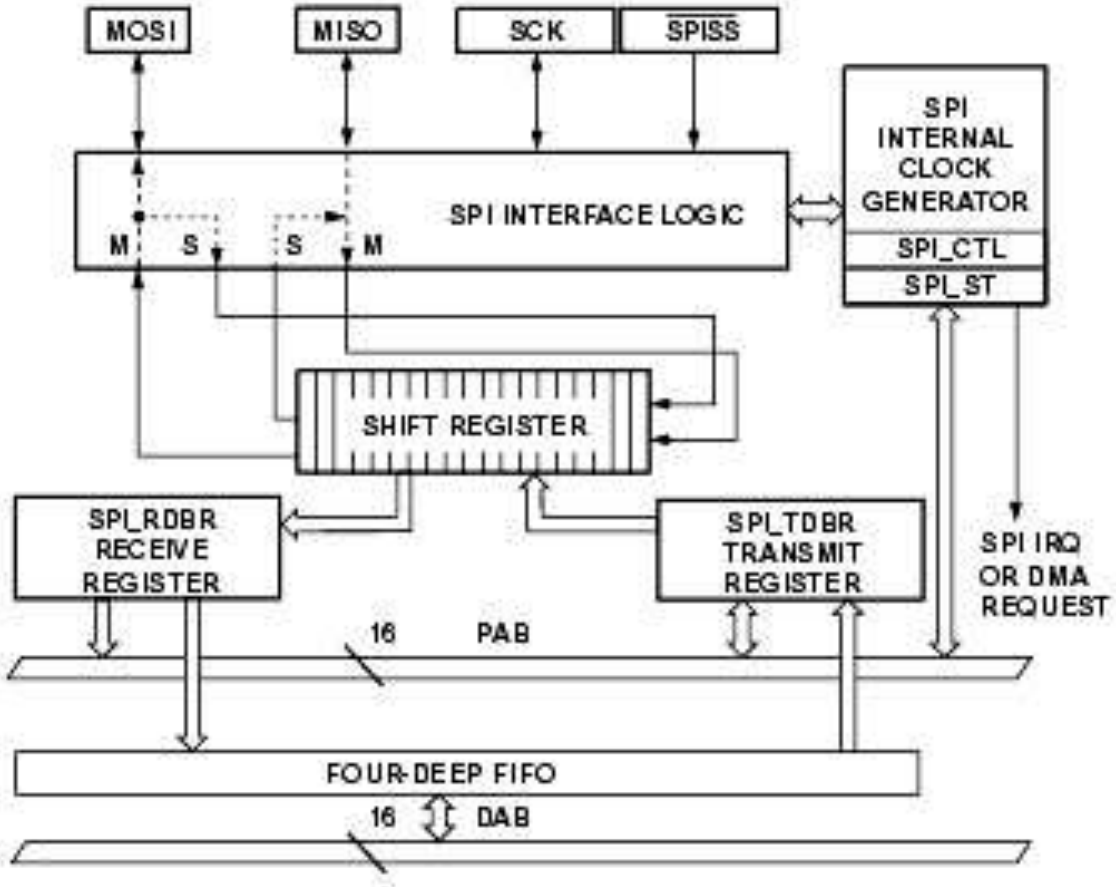
- BF-537, UART0:

```
BIT_SET(*pPORTF_FER,0); //RX
BIT_SET(*pPORTF_FER,1); //TX
BIT_CLEAR(*pPORT_MUX, 3); // RX/TX
*pUART0_GCTL = UCEN; // enable UART clock
*pUART0_LCR = DLAB;
*pUART0_DLL = divisor;
*pUART0_DLH = divisor>>8;
*pUART0_LCR = WLS(8);
// Clear DLAB again and set UART frame to:
// 8 bits, no parity, 1 stop bit.
```

Interfejsy I/O – SPI

- Serial Peripheral Interface:
  - Full-duplex, transfer synchroniczny
  - Słowa 8 lub 16 bit
  - Programowalny tryb MSB/LSB
  - Praca w trybie Master lub Slave
  - Dedykowane linie MISO,MOSI,SCK
    - Master-In-Slave-Out
    - Master-Out-Slave-In
    - Serial-Clock
  - Sugerowane linie dla SS (Slave-Select)

Interfejsy I/O – SPI



Interfejsy I/O – SPI

SPI_CTL	Control register
SPI_FLG	Flag register
SPI_STAT	Status register
SPI_TDBR	Transmit Data Buffer register
SPI_RDBR	Receive Data Buffer register
SPI_BAUD	Baud Rate register
SPI_SHADOW	RDBR Shadow register

Interfejsy I/O – SPI

- Jak włączyć:

```
BIT_SET(*pPORTF_FER,11); // MOSI
BIT_SET(*pPORTF_FER,12); // MISO
BIT_SET(*pPORTF_FER,13); // SCK
BIT_SET(*pPORTF_FER,4);  // SSEL6
BIT_SET(*pPORT_MUX,5); // SSEL6
BIT_SET(*pSPI_FLG,6); // SSEL6
```

Interfejsy I/O – SPI

```
*pSPI_BAUD = divisor;
*pSPI_CTL = BIT(0) | BIT(3) | BIT(10)
  | BIT(11) | BIT(12) | BIT(14);

  // [0]:  start transfer on write
  // [3]:  get more data
  // [10]: CPHA
  // [11]: active LOW
  // [12]: master
  // [14]: SPI enable
```

Interfejsy I/O – SPORT

- Serial Port:
  - Synchroniczny transfer danych
  - Half-duplex
  - Synchronizowany zegarem wewnętrznym lub zewnętrznym
  - Programowalny tryb MSB/LSB
  - Programowalna szerokość słowa 3-32bit
  - Kompandorowanie dla zastosowań telefonicznych
  - Kolejki FIFO

Interfejsy I/O – SPORT

- Przykład:

```
BIT_CLEAR(*pPORT_MUX,0);
BIT_CLEAR(*pPORT_MUX,1);
BIT_CLEAR(*pPORT_MUX,2);
// Sport0 transmit configuration
// External CLK, External Frame sync, MSB first, Active Low
// 24-bit data, Secondary side enable, Stereo frame sync enable
*pSPORT0_TCR1 = TFSR | TCKFE;
*pSPORT0_TCR2 = SLEN_24 | TSFSE;
*pSPORT0_TCLKDIV = 0x0013;
*pSPORT0_TFSDIV = 0x001F;

// Sport0 receive configuration
// External CLK, External Frame sync, MSB first, Active Low
// 24-bit data, Secondary side enable, Stereo frame sync enable
*pSPORT0_RCR1 = RFSR | RCKFE;
*pSPORT0_RCR2 = SLEN_24 | RSFSE;
*pSPORT0_RCLKDIV = 0x0013;
*pSPORT0_RFSDIV = 0x001F;
```

Interfejsy I/O – CAN

- Controlled Area Network:
  - Spełnia standard CAN 2.0B
  - Wsparcie dla identyfikatorów:
    - Standardowych 11-bitowych
    - Rozszerzonych 29-bitowych
  - Maksymalny transfer: 1Mbit/s
  - 32 skrzynki pocztowe (8 wysyłanie, 8 odbiór, 16 konfigurowalnych)

Interfejsy I/O – PPI

- Parallel Peripheral Interface:
  - Half-duplex
  - Dwukierunkowe linie danych
  - Maksymalnie 16-bitowa szyna danych
  - Możliwość multipleksacji do 8-bitów
  - Kompatybilny z BT.656
  - Może być wykorzystany do dowolnej praktycznej transmisji równoległej
  - Wyzwalanie wewnętrzne lub zewnętrzne
  - Kolejki FIFO

Interfejsy I/O – Ethernet MAC

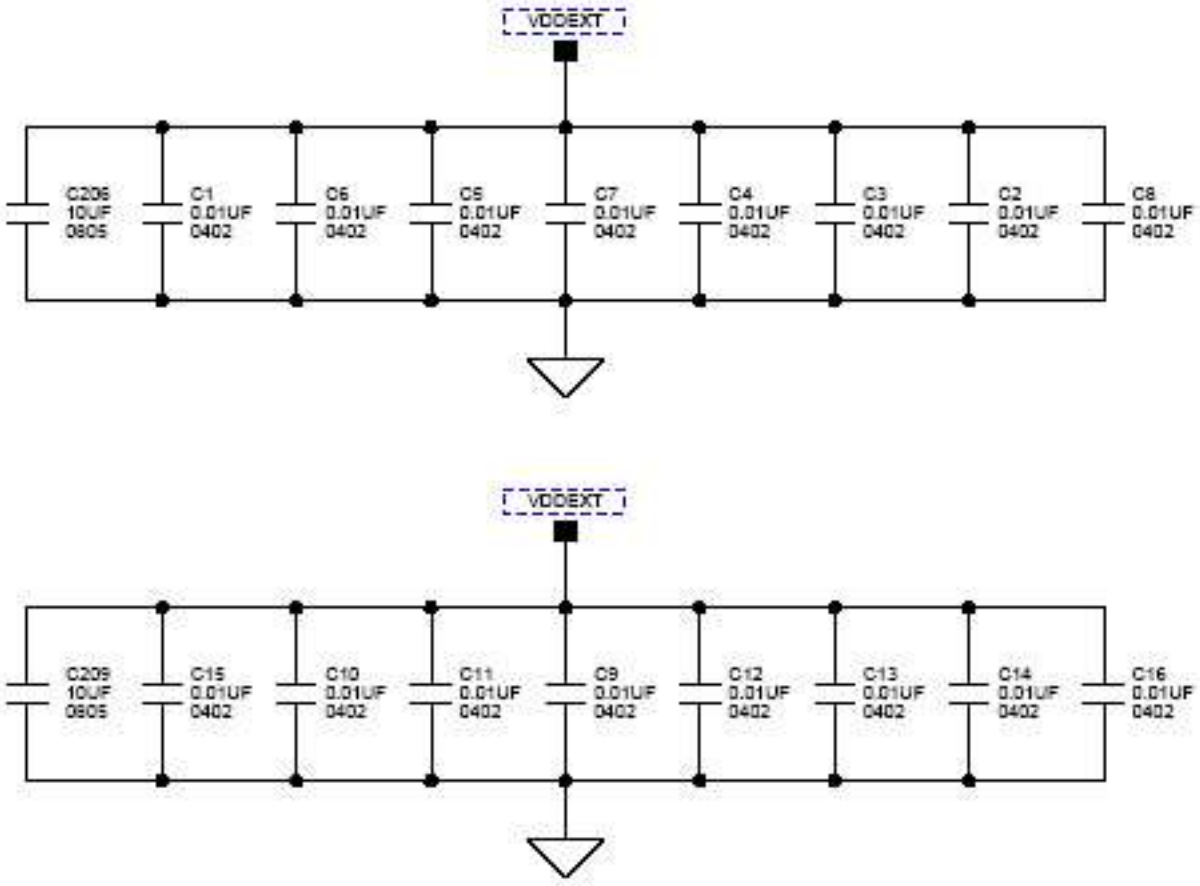
- Ethernet Media Access Control:
  - Interfejs pomiędzy standardem interfejsem standardu MII/RMII a procesorem Blackfin
    - Marvell Gigabit Ethernet (88E8001 PHY) w trybie 100Mbit/s
    - SMSC LAN83C185 10/100
  - Full- lub half-duplex
  - 10Mbit/s lub 100Mbit/s
  - Walidacja sum kontrolnych TCP/IP
  - Wake-ON-LAN
  - Kolejki FIFO

Interfejsy I/O – Ethernet MAC

- Moduł zapewnia dostęp do pakietów warstwy Ethernet-owej na poziomie pamięci
- Oprogramowanie ADI umożliwia wykorzystanie standardowych gniazd:
- – recv, send, listen, accept, bind…

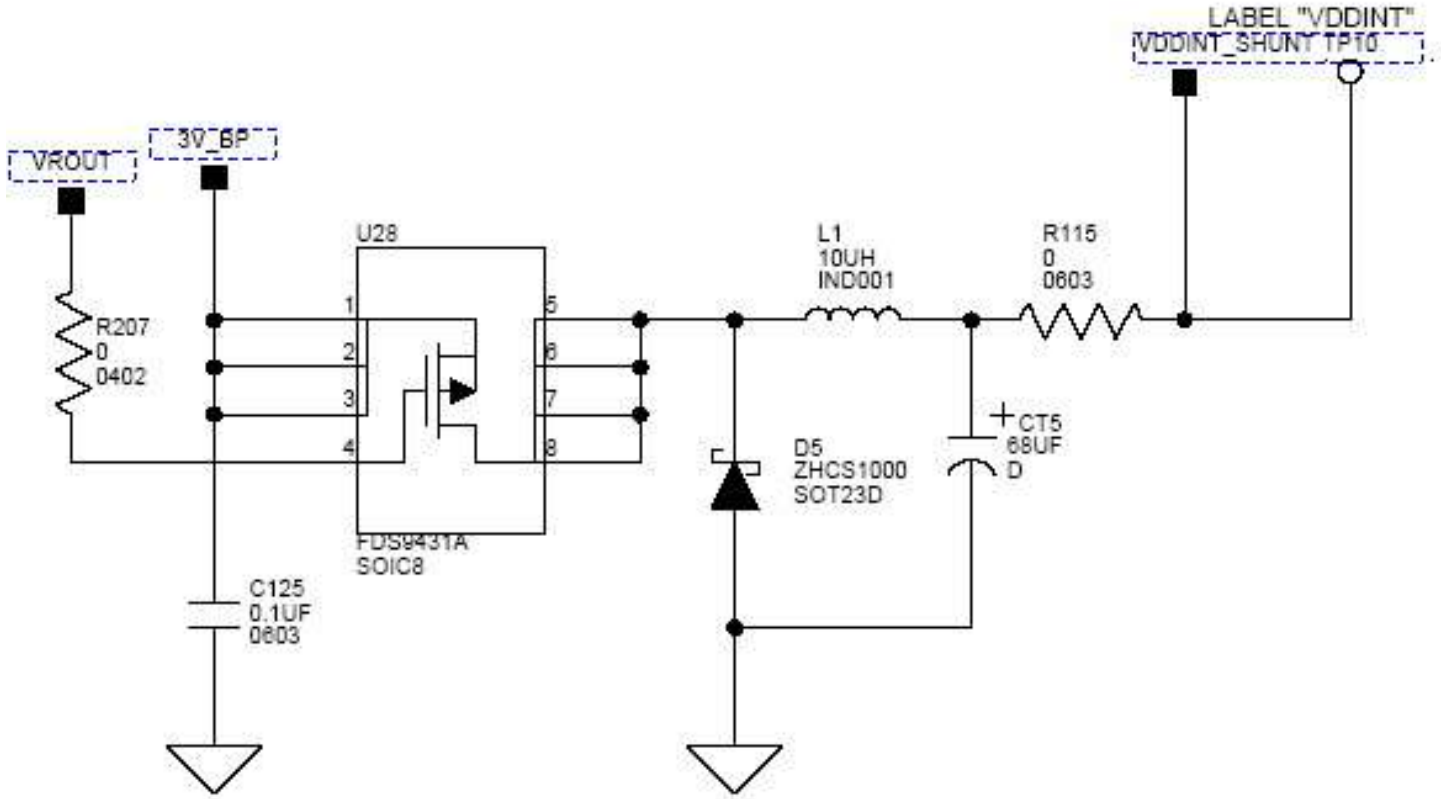
Konfiguracja elektryczna

- Filtracja napięcia „systemowego”



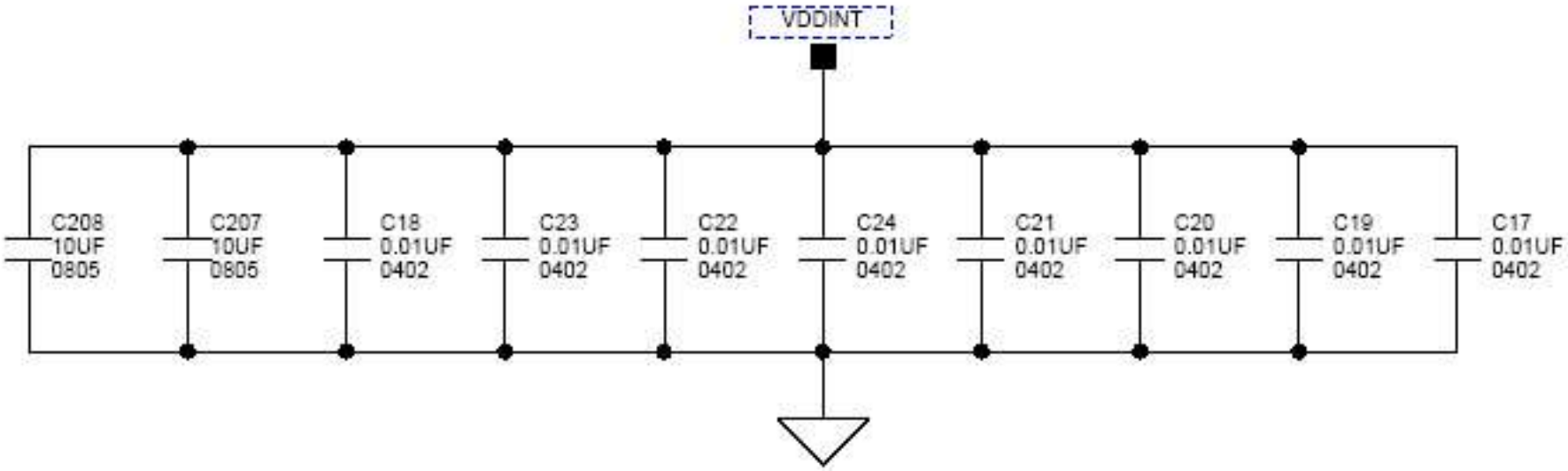
Konfiguracja elektryczna

- Regulator napięcia rdzenia



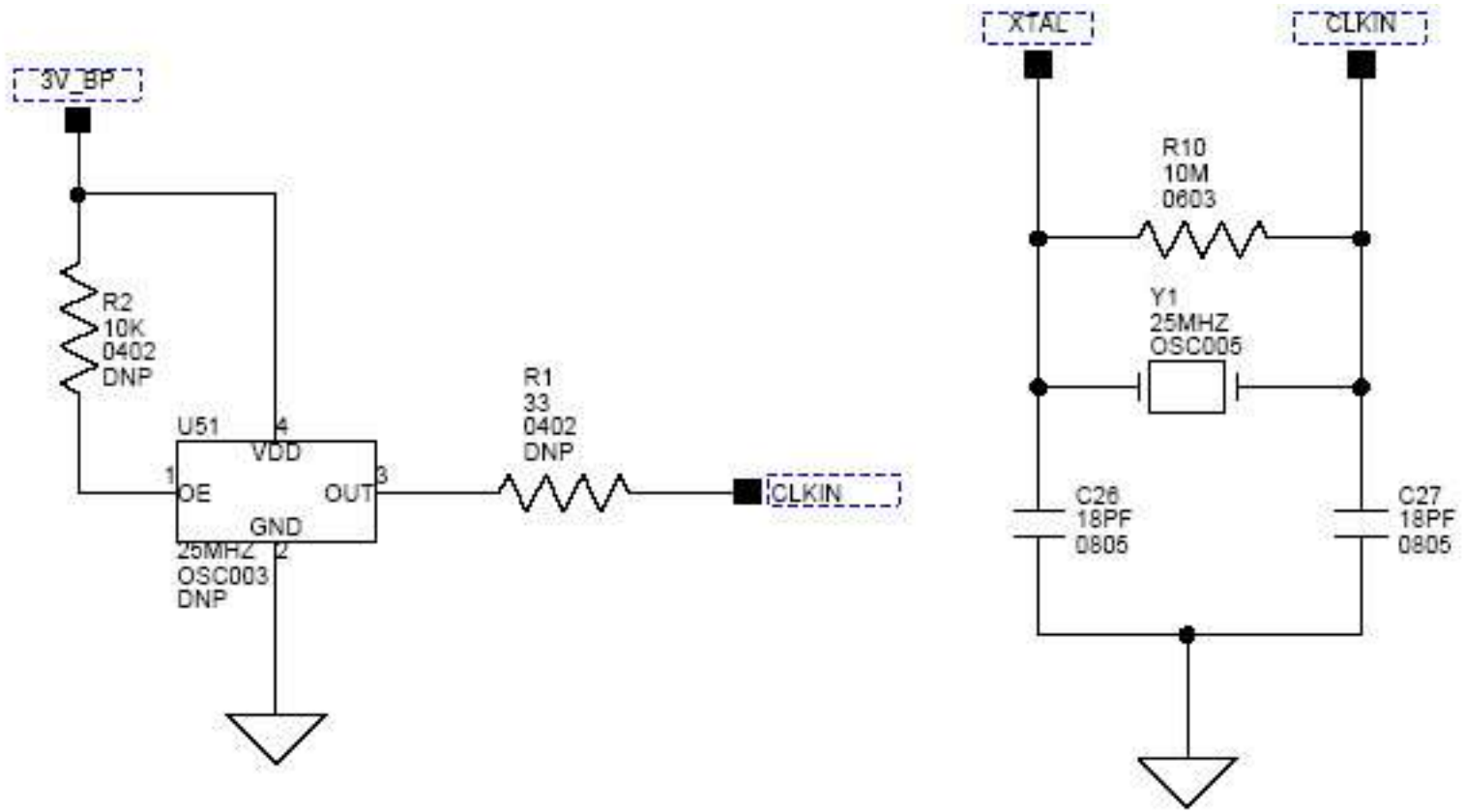
Konfiguracja elektryczna

- Filtracja zasilania rdzenia



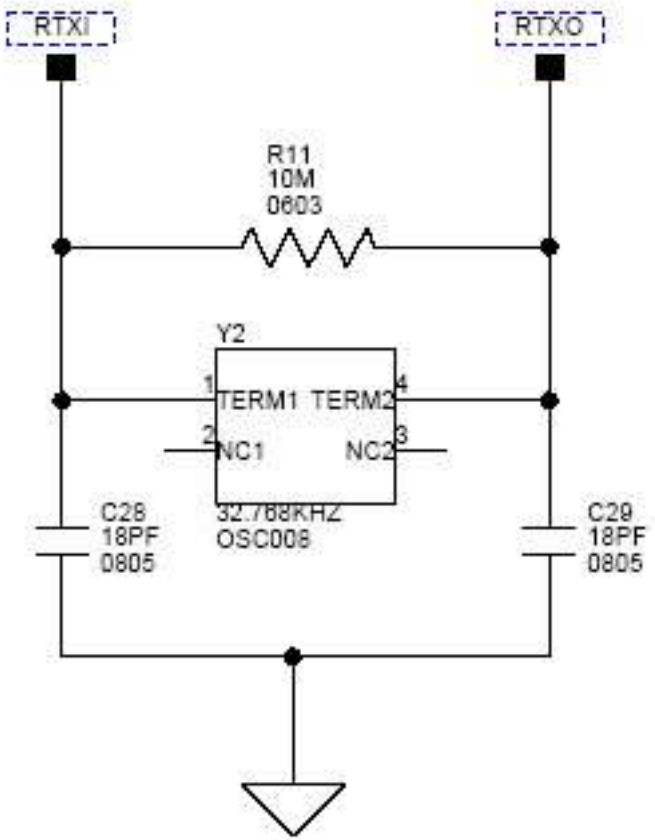
Konfiguracja elektryczna

- Zegar



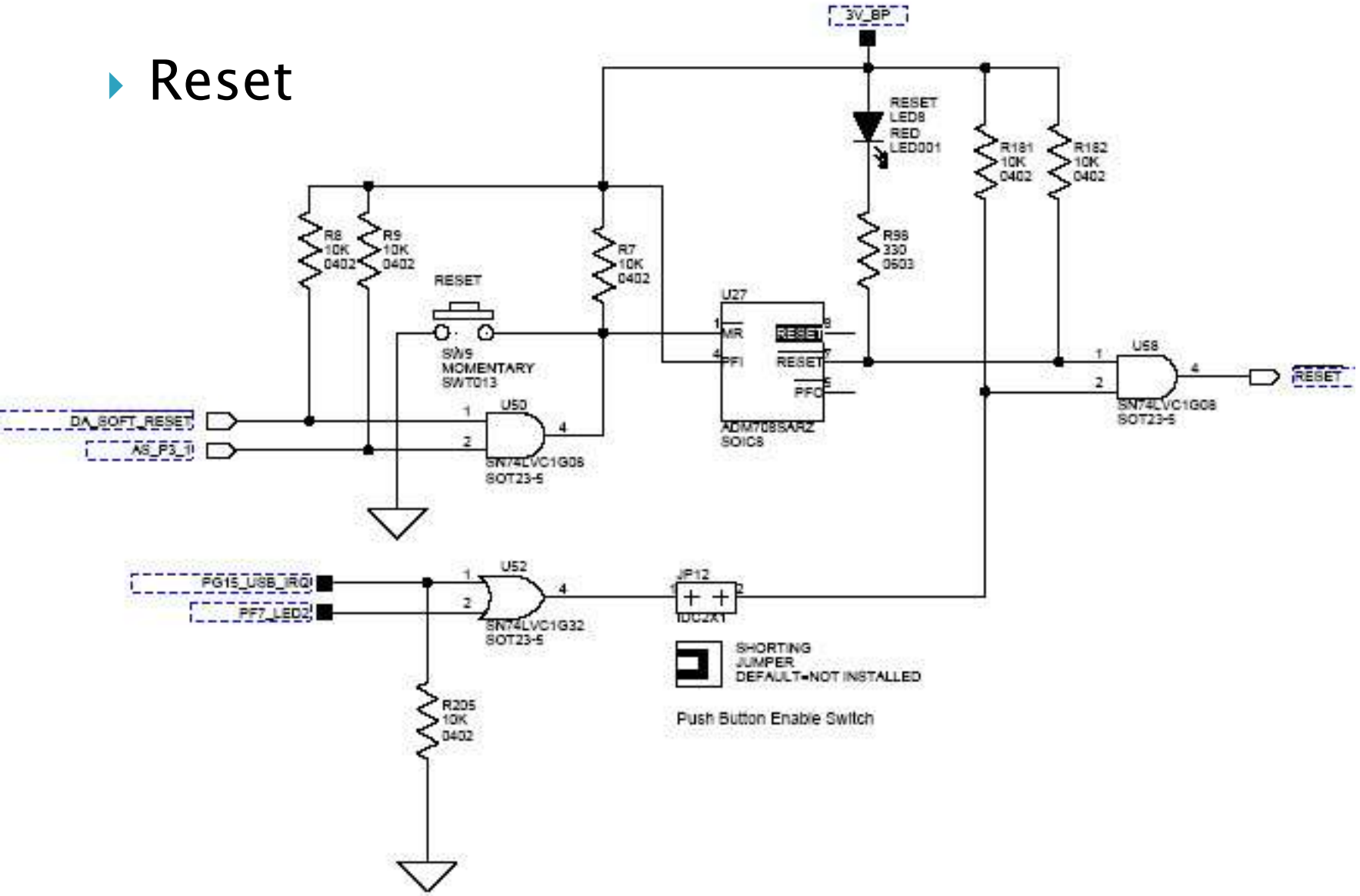
Konfiguracja elektryczna

- Zegar RTC



Konfiguracja elektryczna

- Reset



Optymalizacja kodu

- Typy danych

char	8-bit signed integer
unsigned char	8-bit unsigned integer
short	16-bit signed integer
unsigned short	16-bit unsigned integer
int	32-bit signed integer
unsigned int	32-bit unsigned integer
long	32-bit signed integer
unsigned long	32-bit unsigned integer

- Brak wsparcia sprzętowego: float, double
  - Oba 32-bitowe IEEE754

Optymalizacja kodu

- Zalecenia:
- Unikać arytmetyki zmiennoprzecinkowej
- Unikać operacji dzielenia i modulo
  - Kompilator automatycznie wyłapuje 2^N
- Inicjalizować stałe **statycznie**
  - static int val = 3;
- Dbać o „alignowanie” pamięci
- Zapis/odczyt tylko 16-bitowych słów z pamięci zewnętrznej

Optymalizacja kodu

- Adresowanie tablic:

```
void va_ind( short a[], short b[],
short out[], int n)
{
 int i;
 for (i = 0; i < n; ++i)
 out[i] = a[i] + b[i];
}
```

```
void va_ptr( short a[], short b[],
short out[], int n)
{
 int i;
 short *pout = out, *pa = a, *pb = b;
 for (i = 0; i < n; ++i)
 *pout++ = *pa++ + *pb++;
}
```

Optymalizacja kodu

- Adresowanie tablic
  - Lepiej zacząć od zapisu indeksowego
  - W zewnętrznych pętlach używać zapisu indeksowego
  - Używać trybu indeksowego dla buforów cyklicznych
  - **Jednocześnie – pętle przetwarzania ze skomplikowanym adresowaniem – lepszy tryb wskaźnikowy**

Optymalizacja kodu

- Nie rozwijać pętli! (ładowanie 16-bit)

```
void va1(short a[], short b[],
short c[], int n)
{
 int i;
 for (i = 0; i < n; ++i) {
  c[i] = b[i] + a[i];
 }
}
```

```
void va2(short a[], short b[],
short c[], int n)
{
 short xa, xb, xc, ya, yb, yc;
 int i;
 for (i = 0; i < n; i+=2) {
  xb = b[i]; yb = b[i+1];
  xa = a[i]; ya = a[i+1];
  xc = xa + xb; yc = ya + yb;
  c[i] = xc; c[i+1] = yc;
 }
}
```

Optymalizacja kodu

- Dalsze uwagi do pętli
- Nie obracać pętli
- Unikać „aliasów”
- Unikać zależności
  - for (i = 0; i < n; ++i) a[i] = b[i] * a[c[i]];

Optymalizacja kodu

- Wektoryzacja pętli

```
void copy(short *a, short *b)
{
 int i;
 #pragma vector_for
 for (i=0; i<100; i++){
  a[i] = b[i];
 }
}
```

Optymalizacja kodu

- Bufory cykliczne
  - Umożliwiają uniknąć kopiowania pamięci
  - Lub operacji modulo/iloczynu logicznego
  - Zazwyczaj kompilator sam wykorzystuje konstrukcje
- Indeksowanie % N
  - CIRCULAR_BUF[i%1000] = xyz;

Optymalizacja kodu

- Fractional 16

```
int sp(short *a, short *b)
{
 int i;
 int sum=0;
 for (i=0; i<100; i++) {
  sum += ((a[i]*b[i]) >> 15);
 }
 return sum;
}
```

```
#include <fract.h>
fract32 sp(fract16 *a,
fract16 *b)
{
 int i;
  fract32 sum=0;
  for (i=0; i<100; i++) {
  sum = add_fr1x32(sum,
  mult_fr1x32(a[i],b[i]));
 }
 return sum;
}
```

- Dziękuję za uwagę!