Digitala projekt, EDI022
Rapport
Handledare: Bertil Lindvall

Simon Lindgren, dt05sl0
Marthin Nielsen, dt05mn7

18 maj 2009

Innehåll

Sammanfattning

This project consists of constucting a radio controlled car with an external remote control. The car has the functionality of finding back to its start position by the simple push of a button on the remote control. The communiction between the remote and the car is radio and this proved to be a tough task because of all the noice in the enviroment and how to make sure that the signals came through intact. In this report we handle all from ideas, construction and problem that we encounted along the way.

1 Inledning

I brist på erfarenhet inom design av hårdvara tog det mycket längre tid än beräknat att utveckla samt bygga hårdvaran. Den absolut största utmaningen var att veta hur och var man skulle starta utvecklingen. Det praktiska så som lödning och virrning var även nytt för oss men det lärde man sig snabbt.

2 Krav

Att veta vilka krav man skulle sätta upp var ingen självklarhet, men efter en veckas diskussion om vad som skulle konstrueras bestämde vi oss för att bygga en radiostyrd bil som skulle kunna hitta tillbaka till startpositionen, dessa är kraven vi satte.

• **Krav 1:** Fordonet skall kunna förflyttas av användaren via fjärrstyrning.

• **Krav 2:** Fordonet skall kunna hitta tillbaka till startpunkten på användarens signal. Kommer dock finnas begränsningar på hur långt bilen kan köra tillbaka då det kan uppkomma begränsningar över hur mycket som kan sparas i minnet.

• **Krav 3:** Fordonet skall kunna styras 8m ifrån fjärrkontrollen, i utrymme utan störningar.

• **Krav 4:** Svarstiden på fordonet får inte överstida 1 sekund.

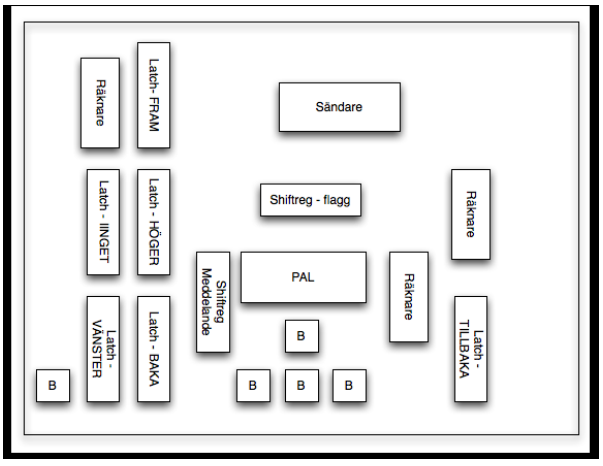
3 Konstruktion

3.1 Fjärrkontroll

Vi insåg att ett problem med signalerna skulle kunna bli att det skulle finnas störningar i luften och att signalerna pga det skulle kunna bli korrupta, och ifall vi hade otur skulle det kunna leda till att signalerna skulle kunna se ut som en annan signal. T ex att en framåt-signal skulle kunna se ut som en bakåt. För att läsa detta försökte vi definiera signalerna så att de var så olika som möjligt, dvs att minst är ett par bitar skillnad mellan varje signal. Detta gjorde att bitmönstret som skickas för varje signal var tvunget att bli minst 8 bitar långt. För att minska våra problem med fjärrkontrollen gjorde vi den helt och hållet i hårdvara utan en processor. Istället för att använda en processor använde vi en PAL för att styra vilken signal som skulle genereras från varje knapp tillsammans med 6 latchar som innehöll de olika signalerna. Det enda som PALen behövde göra då för att skicka ut signalerna var att säga till vilken latch som skulle ladda shiftregisterna samt hur snabbt utsignalen skulle skickas.

Tabell över signalerna på latcharna

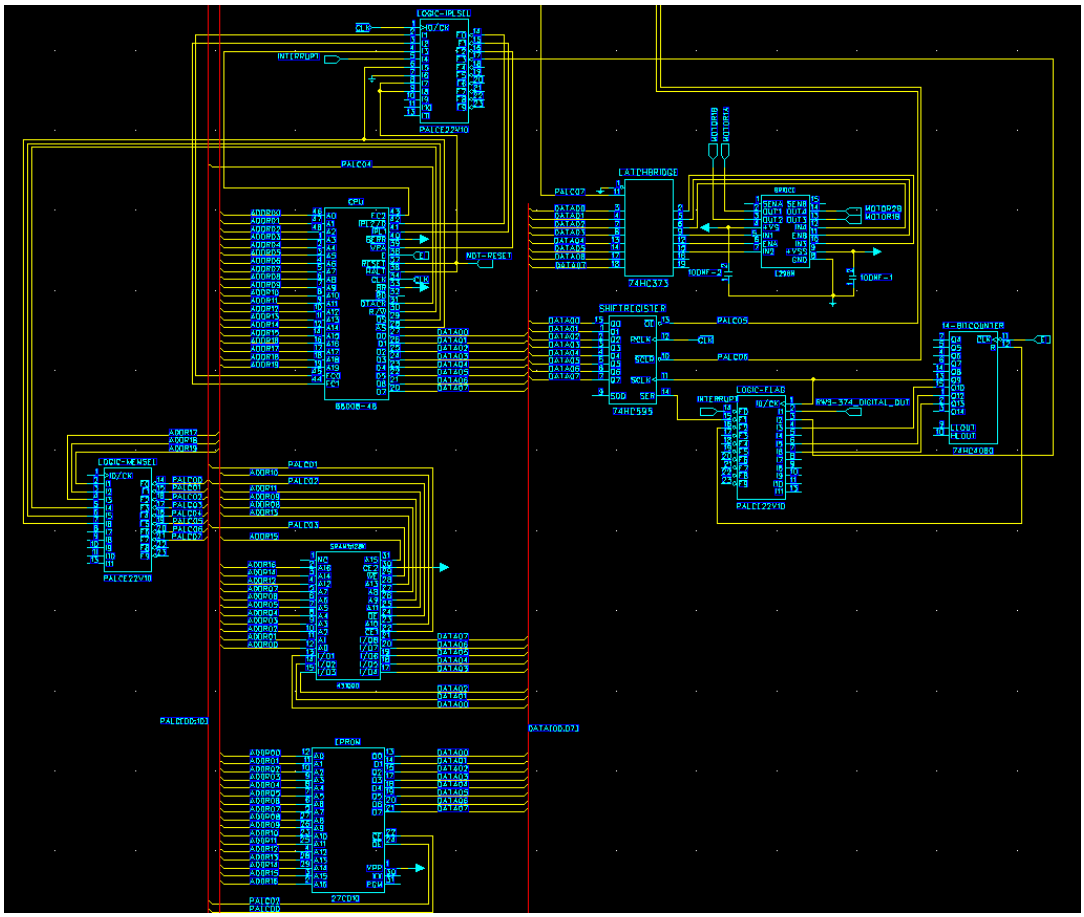
Signaler	*Bitmönster*
Flagga	0111 1110
Fram	0001 0001
Bak	0001 1010
Vänster	0011 1011
Höger	0100 1010
Tillbaka	1011 1100
Inget/Stå still	0000 0000



Figur 1: Fjärrkontroll

3.2 Bilen

Bilen har en mottagare som lyssnar på de frekvenserna som fjärrkontrollen skickar ut. Signalerna går till en PAL som synkroniserar och hämtar ut meddelandet från fjärrkontrollen. När meddelandet har kommit skickar den även en avbrottsignal till processorn så att den kan ta han om meddelandet. Processorn sparar meddelandet i ett SRAM för att kunna hitta tillbaka längs den vägen som den kör. Den lägger även en signal på en latch till motorn så att signalen hålls, fast data på databussen ändras.



Figur 2: Blockschema av bilen

4 Hårdvara

4.1 Processor

Processorn är en Motorola M68008, och var det givna valet för projektet. Den har möjlighet till att lagra mer information eftersom den har 20 ”adresspinnar” vilket ger 1 MB minneskapacitet. Klockans frekvens ligger på 10MHz.

4.2 Minnen

SRAMet som sparar undan signalerna heter 431000. Det är 128kB stort och används fär att spara undan alla signalerna. Detta gör att ifall batteriet skulle ta slut så kommer informationen om var bilen har kört att försvinna och bilen skulle inte kunna hitta tillbaka.

4.3 Sändare

För kommunikationen mellan enheterna användes radio på frekvens 433.92MHz. Från fjärrkontrollen till bilen användes en sändare som heter TV433N och skickar data seriellt. Sändaren kan skicka med 8 kbps vilket passar utmärkt eftersom informationen skickas med 2kbps.

4.4 Mottagare

RX433 heter mottagare modulen, och kör på 433.92 MHz. Mottagaren är seriell och maximal bithastighet på 4.8kbps vilket är väl inom vår gräns.

4.5 Delay Element

Vi använde en komponent som fördröjer en signal mellan 20-30ns (HC31). Den används för att fördröja klockan till PALen som synkroniserar informationen mellan fjärrkontrollen och bilen. Under tiden som de klocksignalerna skiljer sig åt studerar PALen signalen från mottagaren och ser ifall den överensstämmer med flaggan. Pga av störningar och att PALen inte skall hinna läsa på fel sida av en klockpuls får denna signalen inte vara längre än 30 ns.

4.6 Shiftregister

74LS166 och 74HC595 heter shiftregisterna som vi använder för att shifta mellan seriell och parallell

Vi använder två olika sorters shiftregister. Eftersom vi hanterar all trafik parallelt så måste fjärrkontrollen ha en shiftregister (74LS166) för att omvandla en parallell signal till seriell. Av samma anledning behöver bilen ett

shiftregister (74hc595) för att omvandla den seriella signalen från mottagaren till parallell.

4.7 Pal

Vi har totalt 4 st PALCE 22v10, varav 3 st till bilen och en till fjärrkontrollen. Fjärrkontrollens PAL tar i princip hand om två saker, en av dem är att den bestämmer vilken signal som skall skickas ut beroende på vilka knappar som hålls nedtryckta. Den kontrollerar även hastigheten på hur snabbt shiftregisterna skall laddas. Eftersom vi nu skickar 16 bitar i taget med flaggan samt meddelandet så laddas de var 16e bit.

I bilen finns som sagt 3 st PALs. , Den första lyssnar efter en flagga och är till för att synkronisera meddelanden till bilen. Eftersom vi använder en PAL till detta behöver inte processorn läsa av vad som finns på ingången varje bit och det finns pga det utrymme för processorn att gära andra beräknar under tiden. På grund av detta finns det utrymme för bilen att processa mer information vid ev. utbyggnad än vad den gör för tillfället.

Andra PALen sköter avbrotten till processorn. Den får en signal från första PALen om att det finns ett meddelande på shiftregistret som kan avläsas. Det skickar en signal om det till processorn och ger efter det ett ack tillbaka till första PALen efter processorn tagit hand om avbrottet.

Sista PALen tar endast han om trafiken via adressbusen samt databusen. Det skickar bestämmer utifrån vilken adressbussen vilken enhet som skall aktiveras och skickar sedan tillbaka ett ack till processorn.

4.8 Latch

På fjärrkontrollen är latcharna hårdkodade med de olika bitmönsterna som representerar de olika kommandona som användaren kan skicka. Det gör att man enkelt kan ändra signalerna genom att vira om trådarna till latcharna. På bilen använder vi även en latch för att hålla signalen till motorn.

4.9 Drivkrets (H-Brygga)

H-bryggan, L298, används för att driva motorerna. Fyrkantspulsen som skickats till L298 är för svag för att själv driva motorerna utan den fungerar som en styrsignalsom reglerar när ström skall skickas via spänningskällan, batteri eller spänningkuber, till motorerna.

Vi använder en L298 för att driva motorerna. Man kan till den koppla in en extern strömkälla utifall man extra kraft till motorerna. Bryggan tillåter att man använder 2 hjul och man kan styra båda hjulen individuellt, antingen till vänster eller höger. Man kan även låta den rulla fritt eller bromsa.

5 Mjukvara

Vi har skrivet ett program i c till bilen. Programmets syfte är att styra motorerna samt spara ner vägen som den kör för att kunna spela upp det i efterhand. All kod finns att läsa i appendix.

6 Problem och lösningar på konstruktion

Ett problem har varit att det inte finns högnivå debuggning till mjukvaran vilket gjorde det mycket mer komplicerat att hitta felen i koden. Detta ledde till att utvecklingen av mjukvaran tog mycket längre tid än beräknat. Ett exempel på detta var att vi hade problem med att veta ifall bilen läste på rätt adressen varje gång när vi sa till den att börja köra tillbaka till startpositionen. Att testa gränsvärden var också ett problem utan högnivådebuggning. Ett mindre problem vi har på konstruktionerna är att det inte för tillfället finns en power-knapp för att stänga av strömmen till bilen / fjärrkontrollen.

7 Slutsats

Fjärrkontrollen till bilen fungerar mycket bra. Den genererar en stark och tydlig signal vilket gör att bilens radiostyrning fungerar mycket bra. Den kör tills antingen batterierna tar slut eller att den kommer utanför fjärrkontrollens sändningsradie. Dock finns det problem med att hitta tillbaka, detta beror främst på att vi inte vill begränsa användarens frihet att köra bilen precis som han vill. Ifall användaren är oförsiktig så kan bilens hjul spinna och det gör att det, på vår konstruktion, blir omöjligt att veta var bilen hamnar och kan därför inte hitta tillbaka

Sammanfattningsvis så har projektet har varit mycket lärorikt. I kursen lär man sig allt från de minsta detaljerna i design av hårdvara till högnivåprogrammering. Det mest lärorika har varit att lära sig uppskatta hur lång tid det faktiskt tar att designa och utveckla, samt att få en känsla av vilka vanliga fel som kan uppstå när det inte fungerar som planerat. Att kunna använda sig av ingenjören inom sig för att hitta nya lösningar och vägar runt problemen.

A C-Kod

```
/*************************************************************************
                           test.c
                           ------
   Detta ar huvudprogrammet som ar skrivet i ANSI C. Exekveringen av hela
   programpaketet borjar i pmain.68k (lage __main).

   exp5() anropas fran assemblyprogrammet exp5.68k vid avbrott.

   _avben() anropar avben.68k vilket tillater avbrott fran PI/T.
*************************************************************************/

#define EPROM_ADDR 0x00000
#define SRAM_ADDR 0x20000
#define RECIEVER_CLEAR_ADDR 0x40000
#define RECIEVER_READ_ADDR 0x60000
#define ENGINE_ADDR 0x80000

#define MEMORY_MAX_SIZE 0x01000
#define MEMORY_START_POS 0x00000
#define MAX_TIME 0xFF
#define ERROR 0x1
#define OK 0x0

/* engine control code = XX XX IN2 ENA IN1 IN4 ENB IN3 */

/* turn left on both wheels */
#define FORWARD_CODE 0x11
#define FORWARD 0x1B
/* turn right on both wheels */
#define BACK_CODE 0x1A
#define BACK 0x36
/* turn left on right wheel, free on left wheel */
#define LEFT_CODE 0x3B
#define LEFT_FORWARD 0x18
/* turn left on right wheel, free on left wheel */
#define LEFT_BACKWARD 0x06
/* turn left on left wheel, free running on right wheel */
#define RIGHT_CODE 0x4A
#define RIGHT_FORWARD 0x03
/* turn left on left wheel, free running on right wheel */
#define RIGHT_BACKWARD 0x30

/* free running on both wheels */
#define FREE_CODE 0x00
#define FREE 0x00
/* stop both wheels */
#define STOP 0x12
```

```
/* Go back */
#define GO_BACK_CODE 0xBC
/* Undefined code */
#define UNDEFINED_CODE 0xFF

unsigned short int *eprom = (unsigned short int *) EPROM_ADDR;
unsigned short int *sram = (unsigned short int *) SRAM_ADDR;
unsigned short int *reciever_read = (unsigned short int *) RECIEVER_READ_ADDR;
unsigned short int *reciever_clear = (unsigned short int *) RECIEVER_CLEAR_ADDR;
unsigned short int *engine = (unsigned short int *) ENGINE_ADDR;

long memory_pos;
unsigned short int time;
unsigned short int curr_signal;
unsigned short int prev_signal;
unsigned short int signals[MEMORY_MAX_SIZE];

/* Moves the car in curr_direction */
void move()
{
switch(curr_signal)
{
case FORWARD_CODE:
*engine = FORWARD;
break;
case BACK_CODE:
*engine = BACK;
break;
case RIGHT_CODE:
*engine = RIGHT_FORWARD;
break;
case LEFT_CODE:
*engine = LEFT_FORWARD;
break;
case FREE_CODE:
*engine = FREE;
break;
default:
*engine = FREE;
}
}

/* Moves the car in the opposite direction to curr_signal */
void move_opposite()
{
switch(curr_signal)
{
case FORWARD_CODE:
*engine = BACK;
break;
```

```
case BACK_CODE:
*engine = FORWARD;
break;
case RIGHT_CODE:
*engine = RIGHT_BACKWARD;
break;
case LEFT_CODE:
*engine = LEFT_BACKWARD;
break;
case FREE_CODE:
*engine = FREE;
break;
default:
*engine = FREE;
}
}

/* Stops the car */
void stop()
{
*engine = FREE;
}

void read_signal()
{
prev_signal = curr_signal;
curr_signal = *reciever_read;
}

/* Checks if we dont have any more sram to use */
unsigned short int out_of_memory()
{
if(memory_pos < MEMORY_START_POS || memory_pos > MEMORY_MAX_SIZE)
{
stop();
return ERROR;
}
return OK;
}

void load_signal()
{
prev_signal = curr_signal;
curr_signal = signals[memory_pos - 2];
}

void load_time()
{
time = signals[memory_pos - 1];
}
```

```
void save_signal()
{
signals[memory_pos - 2] = curr_signal;
}

void save_time()
{
signals[memory_pos - 1] = time;
}

/* Stores the signal in the sram and the time */
void push_signal()
{
if(curr_signal != prev_signal || time >= MAX_TIME )
{
memory_pos = memory_pos + 2;
if(out_of_memory() == ERROR)
{
memory_pos = memory_pos - 2;
return;
}
time = 0;
save_signal();
}
time++;
save_time();
}

/* Move back towards the start point,
returns 1 if signal should be changed*/
void pop_signal()
{
if(time == 0)
{
memory_pos = memory_pos - 2;
if(out_of_memory() == ERROR)
{
memory_pos = memory_pos + 2;
return;
}
load_signal();
load_time();
}
time--;
}

/* Reads and control if the signal is go back ,
and if it isnt it stores the signal in the sram*/
void control_signal()
```

```
{
read_signal();
if(curr_signal == GO_BACK_CODE)
{
curr_signal = prev_signal;
pop_signal();
move_opposite();
} else {
push_signal();
move();
}
}

/* Avbrottsprogram */
void exp5()
{
control_signal();
}

/* The main method */
void main()
{
curr_signal = UNDEFINED_CODE;
prev_signal = FREE_CODE;

*engine = FREE;
memory_pos = MEMORY_START_POS;
time = 0;

_avben(); /* allow interupts */
greta:
goto greta; /* evig loop */
}
```

B PAL

```
Title InteruptCTRL
Pattern  It-io dev. 046
Revision  3.0
Author Simon Lindgren
Date 2009-04-22

device 22v10

CLK 1 'Clock'
FC0 2 'FC0'
FC1 3 'FC1'
FC2 4 'FC2'
```

```
Int1 5 'Interupt 1'
AS 6 'AS active low'
logicZero 7 'GND'
HALT 8 'Halt, active low'
RESET 9 'Reset, active low'
NoIn3 10 'No input here'
NoIn4 11 'No input here'
GND 12
NoIn5 13 'No input here'
IPL02 14 'IPL02 active low'
IPL1 15 'IPL1 active low'
VPA 16 'VPA active low'
Int1Ack 17 'Ack for interupt 1'
VCC 24

start

'only interupt 5 is used'

IPL02 /= Int1;

IPL1 /= logicZero;

VPA /= FC0*FC1*FC2;

'Warning, could be to fast for the processor.'
Int1Ack = Int1*FC0*FC1*FC2+/RESET*/HALT;


end
```

```
Title InteruptCTRL
Pattern  It-io dev. 046
Revision  3.0
Author Simon Lindgren
Date 2009-04-22

device 22v10

CLK 1 'No input here'
ADDR19 2 'ADDR19'
ADDR18 3 'ADDR18'
ADDR17 4 'ADDR17'
RW 5 'Read active high, write active low'
DS 6 'DS active low'
AS 7 'AS active low'
NoIn1 8 'No input here'
NoIn2 9 'No input here'
NoIn3 10 'No input here'
NoIn4 11 'No input here'
```

```
GND 12
NoIn7 13 'No input here'
CSEPROM 14 'CE EPROM, active low '
CSSRAM 15 'CE sram, active low '
OE 16 'OE on memory, active low '
WE 17 'WE on SRAM, active low '
DTACK 18 'DTACK active low'
OESR 19 'OE on ShiftReg, active low'
CLRSR 20 'Clear on ShiftReg, active low'
LE 21 'LE on the latch, load new commands to the bridge'
State1 22 'State one'
State2 23 'State two'
VCC 24

start

'0x00000 is eprom, 0x20000 sram, 0x40000 clear reciever, ''0x60000 read reciever, 0x80000

'00X, 001 is SRAM, 000 is EPROM'
CSSRAM /= /ADDR19*/ADDR18*ADDR17*/AS;

CSEPROM /= /ADDR19*/ADDR18*/ADDR17*/AS;

OE /= RW*/DS;

WE /= /RW*/DS;

DTACK /= /CSSRAM + /CSEPROM + /OESR + /CLRSR + /LE;

'01X, 011 read from SR, and 010 clear the SR.'
OESR /= /ADDR19*ADDR18*ADDR17*/AS*/OE;

CLRSR /= /ADDR19*ADDR18*/ADDR17*/AS*/OE;

'100'
LE = ADDR19*/ADDR18*/ADDR17*/AS*/WE;

end


Title RadioCTRL
Pattern  It-io dev. 046
Revision  3.0
Author Simon Lindgren
Date 2009-04-06

device 22v10

BitCLK 1 'Clock for BAUD rate'
Forward 2 'Forward input'
```

```
Left 3 'Left input'
Right 4 'Right input'
Back 5 'Back input'
Return 6 'Return input'
ByteCLK 7 'Clock for load'
NoIn1 8 'No input here'
NoIn2 9 'No input here'
NoIn3 10 'No input here'
NoIn4 11 'No input here'
GND 12
NoIn5 13 'No input here'
ForwardOE 14 'Forward output enable active low'
LeftOE 15 'Left output enable active low'
RightOE 16 'Right output enable active low'
BackOE 17 'Back output enable active low'
ReturnOE 18 'Return output enable active low'
NothingOE 19 'Nothing output enable active low'
LoadEN 20 'Load enable for Shift Reg'
State1 21 'State one'
State2 22 'State two'
State3 23 'State three'
VCC 24

start

ForwardOE /= Forward*/ForwardOE + Forward*/Left*/Right*/Back*/NothingOE;

'Forward OE is active low'
'Forward is enable when Forward button is pressed and we are in Forward enable'
'Forward should be enabled when we are in Nothing enabled and Forward (and also Return) is

LeftOE /= Left*/LeftOE + Left*/Right*/Back*/NothingOE;

'Left OE is active low'
'Left is enable when Left button is pressed and we are in Left enable'
'Left should be enabled when we are in Nothing enabled and Left (and also Return and/or Fo

RightOE /= Right*/RightOE + Right*/Back*/NothingOE;

'Right OE is active low'
'Right is enable when Right button is pressed and we are in Right enable'
'Right should be enabled when we are in Nothing enabled and Right (and also Return, Forwar

BackOE /= Back*/BackOE + Back*/NothingOE;

'Back OE is active low'
'Back is enable when Back button is pressed and we are in Back enable'
'Back should be enabled when we are in Nothing enabled and Back (and also Return, Forward,

ReturnOE /= Return*/ReturnOE + /Forward*/Left*/Right*/Back*Return*/NothingOE;
```

```
'Return OE is active low'
'Return is enabled when Return button is pressed and we are in Return enable'
'Return should be enabled when we are in Nothing enabled and Return is pressed

NothingOE /= ForwardOE*LeftOE*RightOE*BackOE*ReturnOE;

'Nothing OE is active low'
'Nothing OE is enabled when theres nothing else enabled'

LoadEN /= State1*/State2 + /State1*State2;

'LoadEN is active low'
'LoadEN enables when we State1 xor State2 is high'

State1 = /State3*/State2*/State1*/BitCLK*ByteCLK + /State3*/State2*State1*/BitCLK
+ /State3*State2*/State1*/BitCLK + /State3*State2*State1*ByteCLK
+ State3*/State2*/State1*ByteCLK + State3*State2*State1*ByteCLK;

'Describes: Jump to 001, stay in 001, jump to 011, stay in 011, jump to 111, stay in 111'

State2 = /State3*/State2*State1*BitCLK + /State3*State2*/State1*BitCLK
+ /State3*State2*/State1*/BitCLK + /State3*State2*State1*ByteCLK
+ State3*/State2*/State1*ByteCLK + State3*State2*State1*ByteCLK;

'Describes: Jump to 010, stay in 010, jump to 011, stay in 011, jump to 111, stay in 111'

State3 = /State3*State2*State1*/ByteCLK + State3*/State2*/State1*/ByteCLK
+ State3*/State2*/State1*ByteCLK + State3*State2*State1*ByteCLK;

'Describes: Jump to 100, stay in 100, jump to 111, stay in 111'

'We have 6 states, State3/State2/State1'
'First state 000 are we in when ByteCLK is low first time'
'We jump to the second state 001 from 000 when ByteCLK is high and BitCLK is low'
'We jump from the second state to the third state 010 when ByteCLK is high and BitCLK is h
'We jump from the third to the fourth state 011 when ByteCLK is high and BitCLK is low'
'We jump from the fourth state to the fifth state 100 when ByteCLK is low'
'We jump from the fifth state to the sixth state 111 when ByteCLK is high'
'We jump from the sixth state to the first state 000 when ByteCLK is low'

end
```

```
Title InteruptCTRL
Pattern  It-io dev. 046
Revision  3.0
Author Simon Lindgren
Date 2009-04-22

device 22v10
```

```
BitCLK 1 'QI ~2000hz'
InSER 2 'RWS/374 Digital Input'
Int1Ack 3 'Ack for interupt 1'
BitCLK2 4 'QJ ~1000hz'
BitCLK4 5 'QK ~500hz'
BitCLK8 6 'QL ~250hz'
BitCLK16 7 'QM ~125hz'
DelayCTR 8 'No input here'
NoIn7 9 'No input here'
NoIn8 10 'No input here'
NoIn9 11 'No input here'
GND 12
NoIn10 13 'No input here'
Int1 14 'Interupt 1 '
OutSER 15 'Signal that goes to the ShiftReg'
State3 16 'State one'
Error 17 'State two'
HighCLK 18 'State three'
State4 19 'State four'
State5 20 'State five'
State6 21 'State six'
State7 22 'State seven'
ResetCLK 23 'Reset the clock if we get error'
VCC 24

start


'If we get wrong flag Error goes high'
'Max 12 additions on this pin (21).'
'Checks the pattern 111 1110'
Error = /*BitCLK16*/BitCLK8*/BitCLK4*BitCLK2*/BitCLK*DelayCTR*/InSER
+ /*BitCLK16*/BitCLK8*BitCLK4*/BitCLK2*/BitCLK*DelayCTR*/InSER
+ /*BitCLK16*/BitCLK8*BitCLK4*BitCLK2*/BitCLK*DelayCTR*/InSER
+ /*BitCLK16*BitCLK8*/BitCLK4*/BitCLK2*/BitCLK*DelayCTR*/InSER
+ /*BitCLK16*BitCLK8*/BitCLK4*BitCLK2*/BitCLK*DelayCTR*/InSER
+ /*BitCLK16*BitCLK8*BitCLK4*/BitCLK2*/BitCLK*DelayCTR*/InSER
+ /*BitCLK16*BitCLK8*BitCLK4*BitCLK2*/BitCLK*DelayCTR*InSER
+ Error*HighCLK;

HighCLK = BitCLK2 + BitCLK4 + BitCLK8 + BitCLK16;

'Reset when we get an error until we have a 0 as insignal'
ResetCLK = Error + ResetCLK*InSER;

OutSER = BitCLK16*InSER;

Int1 = BitCLK16*BitCLK8*BitCLK4*BitCLK2*BitCLK*/DelayCTR + Int1*/Int1Ack;
```

```
end
```