Tomas Nordström
Högskolan i Halmstad
Dokumentversion 0.1, 2012-04-01

Beskrivning av porthantering i mikroprocessorn SAM3U som används på vårt labkort SAM3U-EK.

Informationen till detta kompendium är hämtat ifrån följande dokument:

- [SAM3U] ATMEL, AT91SAM, ARM-based Flash MCU -- SAM3U Series, Feb. 2012.
  Speciellt då Kapitel 30. ”Parallel Input/Output Controller (PIO)”. samt
- [SAM3U_GS] ATMEL, ”AT91 ARM Cortex-M3 based Microcontrollers Application Note -- Getting started with SAM3U Microcontrollers”, Aug. 2010

När man använder en ARM processor, som tex i vår SAM3U mikroprocessor, så nås alla in/ut-portar genom att läsa alt. skriva till en speciell adress. Principen att kunna skriva till resp. läsa från en port precis som om det vore en del av själva minnet kallas *minnesmappad I/O*. Huvudskälet till detta är för att spara instruktioner. Annars skulle man behöva specialinstruktioner för att nå varje port på chipet. Alltså nås den fysiska porten precis som om den vore vilken minnesplats som helst!

Hur och var i minnet en viss in/ut-port återfinns beror på vilken mikroprocessor man använder.

SAM3U har tre styrenheter (PIOA, PIOB, och PIOC) för portar ”Parallel Input/Output Controller (PIO)” som var och en styr en 32-bitars port. Varje pinne på porten kan användas för generell I/O eller till en eller två inbyggda I/O enheter. Varje pinne kan också konfigureras att fungera som in- respektive utport. (Man kan egentligen alltid läsa av porten, dvs den signal som ligger på kretsens pinnar, även om in pinne är konfigurerad som utport. *Men om man väntar sig en insignal på en pinne får man aldrig konfigurera den som utport, då det kan leda till kortslutning och kretsen brinner upp*).

För att sätta upp en port finns 12 huvudregister där man vill kunna styra varje bit (som representerar en pinne ”pad” på kretsen). Dessutom finns 4 register som styr port funktionalitet men inte behöver styrning på bitnivå.

Dessa huvudregister kan normalt inte skrivas till direkt i SAM3U, utan bitars sätts respektive nollställs via två speciella styrregister (”enable” och ”disable” register). Detta gör det möjligt att ändra en speciell bit/pinne på en port utan att känna till värdet på de andra bitarna[1]. Vid skrivning till dessa (”enable” och ”disable” register

[1] I andra mikroprocessorer så kan man behöva först göra en läsning av portens nuvarande värde, för att sedan modifiera den bit man är ute efter att ändra, och sist göra en skrivning av hela portens alla bitar. Vilket då kräver fler instruktioner än vad som behövs med denna ”tre-register” konstruktion. Nackdelen är att man inte kan

sätts respektive släcks bitar i huvudregistret (vid skrivning till dessa två register så anger en 1:a att motsvarande bit ska sättas eller släckas, medan en 0:a har ingen effekt). Vad detta huvudregister innehåller kan avläsas via en tredje minnesposition kallat statusregister.

Konfigurering av utportar

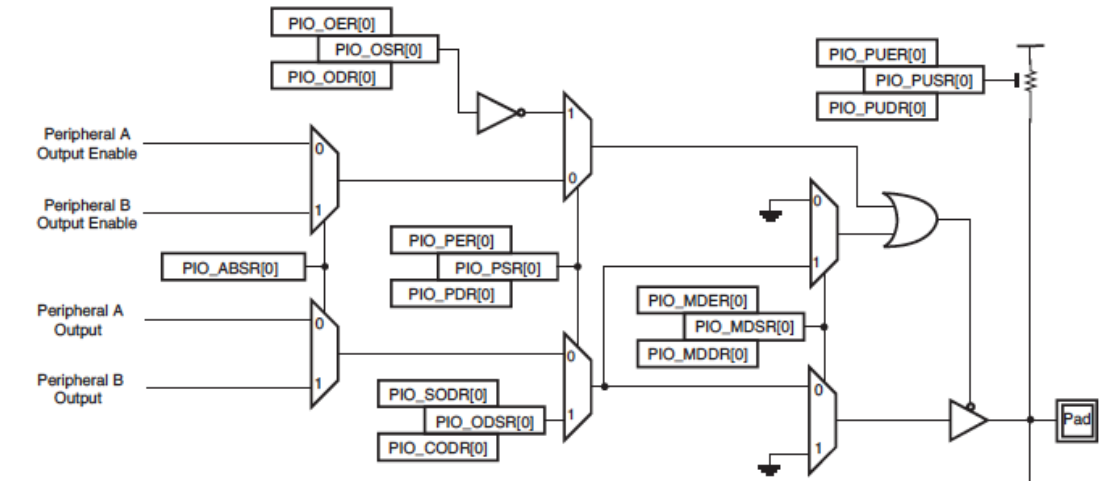
För att konfigurera utportar så finns följande *huvudregister* (statusregistret inom parantes).

PIO Enable (PIO_PSR)
Output Enable (PIO_OSR)
Multi-driver Enable (PIO_MDSR)
Pull-up Enable (PIO_PUSR)
Output Data Register (PIO_ODSR)

Samt styrregister

AB Select Register (PIO_ABSR)

I/O Line Control Logic



PIO Enable (PIO_PSR)

PIO Enable/Disable/Status Register (PIO_PER, PIO_PDR, PIO_PSR)

PIO enable styr om pinnen ska ha generell funktion (där vi kan styra den att göra det vi vill) eller om den är annan (inbyggd I/O) funktion [=0 inbyggd I/O, =1 vår I/O].

Output Enable (PIO_OSR)

Output Enable/Disable/Status Register (PIO_OER, PIO_ODR, PIO_OSR)

Output Enable talar om detta är en utport och att denna pinne drivs av porten (annars sätts den i tristate och porten kan användas som inport). [=0 tristate/inport, =1 utport].

---

sätta och släcka bitar samtidigt, därför finns funktionen synkron skrivning till PIO_ODSR, om sätter upp den funktionen (jmf 30.5.5 i [SAM3U]).

Output Data Register (PIO_ODSR)

Output Data Register (PIO_ODSR)

PIO_SODR (Set Output Data Register) och PIO_CODR (Clear Output Data Register) sätter respektive nollställer PIO_ODSR (Output Data Status Register), som representerar de data som sänds ut på respektive I/O pinnar. [=0 nolla ut på porten, =1 etta ut på porten].

Multi-driver Enable (PIO_MDSR)

Multi-driver Enable/Disable/Status Register (PIO_MDER, PIO_MDDR, PIO_MDSR)

Multi-driver Enable sätter porten att använda ”open drain” så att flera kretsar kan driva samma ledning. Man behöver då ett pullupmotstånd antingen externt eller internt. [=0 normal drivning, =1 ”open drain” drivning].

Pull-up Enable (PIO_PUSR)

Pull-up Enable/Disable/Status Register (PIO_PUER, PIO_PUDR, PIO_PUSR)

Ett inbyggt pullup motstånd kan slås på respektive av med detta huvudregister. Används ofta på inportar där insignalen kan vara ”öppen” eller i tri-state och man vill ha ett tydligt logiskt värde (hög) för detta fall. [=0 med pullupmotstånd, =1 utan pullupmotstånd].

AB Select Register (PIO_ABSR)

AB Select Register (PIO_ABSR)

Det kan finnas upp till två interna I/O enheter som är kopplad till en viss ports pinnar och med detta register så styr man vilken av de två enheterna som ska användas. [=0 enhet A, =1 enhet B].

Sammanfattning utport

För att sätta upp en enkel utport så bör man göra PIO Enable (PIO_PER) och Output Enable (PIO_OER), samt Pull-up Disable (PIO_PUDR). Sedan så kan man via PIO_SODR (Set Output Data Register) and PIO_CODR (Clear Output Data Register) ändra på portens utdata.

Konfigurering av inportar

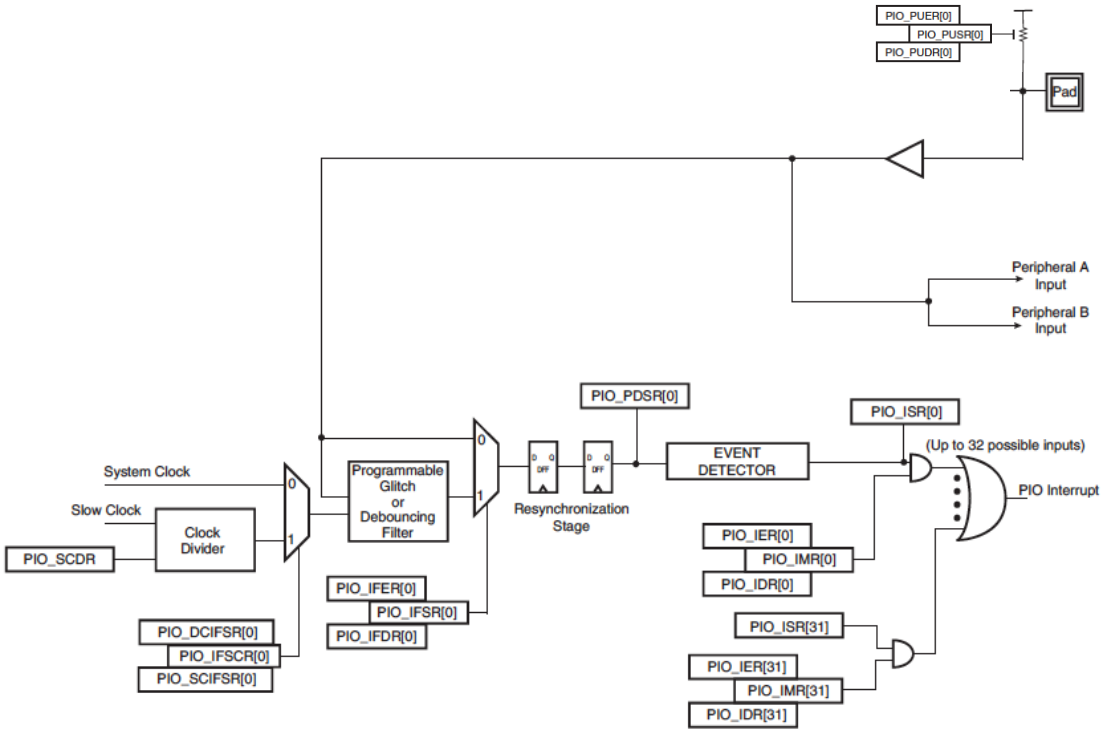
Det viktigaste registret för en inport är PIO_PDSR (Pin Data Status Register) som innehåller, det klockade, värdet av portens pinnar. Detta oberoende av hur portens styrenheten med alla dess register har satts upp.

För att konfigurera inportar så finns följande *huvudregister* (statusregistret inom parantes):

Pull-up Enable (PIO_PUSR)[2]
Interrupt Enable (PIO_IMR)
System Clock Glitch Input Filter Select Register (PIO_IFSCR)
Glitch Input Filter Enable (PIO_IFSR)

Samt register

Interrupt Status Register (PIO_ISR)
Slow Clock Divider Debouncing Register (PIO_SCDR)



[2] notera att den är samma som (och delas med) utporten

Pull-up Enable (PIO_PUSR)
Pull-up Enable/Disable/Status Register (PIO_PUER, PIO_PUDR, PIO_PUSR)

Ett inbyggt pullup motstånd kan slås på respektive av med detta huvudregister. Används ofta på inportar där insignalen kan vara ”öppen” eller i tri-state och man vill ha ett tydligt logiskt värde (hög) för detta fall. [=0 med pullupmotstånd, =1 utan pullupmotstånd].

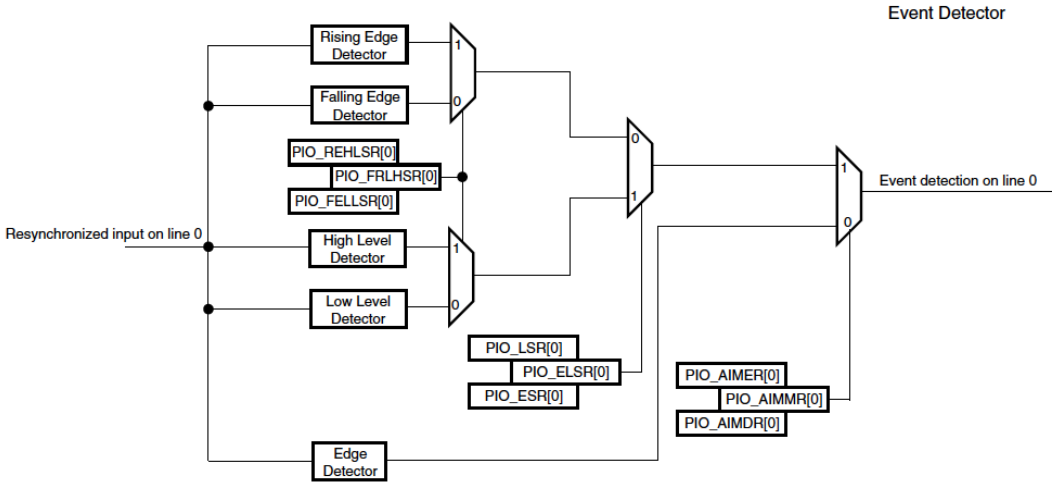
Styrning av avbrott på inport

Interrupt Enable (PIO_IMR)
Interrupt Enable/Disable/Mask Register (PIO_IER, PIO_IDR, PIO_IMR)

PIO_IER (Interrupt Enable Register) och PIO_IDR (Interrupt Disable Register) slår på respektive av om avbrott initieras vid tillståndsförändring/flank (edge) eller ett viss nvå (level) på en inport. Nuvarande värde kan läsas ifrån PIO_IMR (Interrupt Mask Register). [=0 inget avbrott tillåts, =1 kan generera avbrott].

Om avbrott tillåts så kan man styra vad som ska trigga avbrottet styrs via 3 huvudregister till. Se sid 517 i [SAM3U].

Figure 30-7. Event Detector on Input Lines (Figure represents line 0)



Interrupt Status Register (PIO_ISR)
Interrupt Status Register (PIO_ISR)

När en ingång upptäcker en avbrottshändelse (flank eller nivå) på en I/O pinne så sätts motsvarande bit i PIO_ISR (Interrupt Status Register). Om sedan den motsvarande biten i PIO_IMR är satt så generas ett processoravbrott. Alla 32 avbrottssignalerna ifrån en port ORas samman så att en enda avbrottssignal går till NVIC (Nested Vector Interrupt Controller). Men vilken pinne som gav avbrott kan ju avläsas i PIO_ISR.

Enligt [SAM3U_GS], sid 6, så ligger avbrotten för PIOA till PIOC på position 27 till 29 i avbrottsvektorn.
(Ska nog kollas för jag har även sett andra värden i viss kod)

Styrning av ”debouncing” filter

Slow Clock Divider Debouncing Register (PIO_SCDR)
System Clock Glitch Input Filter Select Register (PIO_IFSCR)
Glitch Input Filter Enable (PIO_IFSR)

Att beskrivas senare…
Men ett användarexempel finns som 3.2.6.6 i [SAM3U_GS].

Adresser till PIO

Basadressen för de tre portarna är (jmf Figure 8-1 i [SAM3U]

port	basadress
PIOA	0x400E0C00
PIOB	0x400E0E00
PIOC	0x400E1000

Table 30-2. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	PIO Enable Register	PIO_PER	Write-only	–
0x0004	PIO Disable Register	PIO_PDR	Write-only	–
0x0008	PIO Status Register	PIO_PSR	Read-only	(1)
0x000C	Reserved			
0x0010	Output Enable Register	PIO_OER	Write-only	–
0x0014	Output Disable Register	PIO_ODR	Write-only	–
0x0018	Output Status Register	PIO_OSR	Read-only	0x0000 0000
0x001C	Reserved			
0x0020	Glitch Input Filter Enable Register	PIO_IFER	Write-only	–
0x0024	Glitch Input Filter Disable Register	PIO_IFDR	Write-only	–
0x0028	Glitch Input Filter Status Register	PIO_IFSR	Read-only	0x0000 0000
0x002C	Reserved			
0x0030	Set Output Data Register	PIO_SODR	Write-only	–
0x0034	Clear Output Data Register	PIO_CODR	Write-only	
0x0038	Output Data Status Register	PIO_ODSR	Read-only or(2) Read-write	–
0x003C	Pin Data Status Register	PIO_PDSR	Read-only	(3)
0x0040	Interrupt Enable Register	PIO_IER	Write-only	–
0x0044	Interrupt Disable Register	PIO_IDR	Write-only	–
0x0048	Interrupt Mask Register	PIO_IMR	Read-only	0x00000000
0x004C	Interrupt Status Register(4)	PIO_ISR	Read-only	0x00000000
0x0050	Multi-driver Enable Register	PIO_MDER	Write-only	–
0x0054	Multi-driver Disable Register	PIO_MDDR	Write-only	–
0x0058	Multi-driver Status Register	PIO_MDSR	Read-only	0x00000000
0x005C	Reserved			
0x0060	Pull-up Disable Register	PIO_PUDR	Write-only	–
0x0064	Pull-up Enable Register	PIO_PUER	Write-only	–
0x0068	Pad Pull-up Status Register	PIO_PUSR	Read-only	0x00000000
0x006C	Reserved			

Table 30-2. Register Mapping (Continued)

Offset	Register	Name	Access	Reset
0x0070	Peripheral AB Select Register[5]	PIO_ABSR	Read-Write	0x00000000
0x0074 to 0x007C	Reserved			
0x0080	System Clock Glitch Input Filter Select Register	PIO_SCIFSR	Write-Only	–
0x0084	Debouncing Input Filter Select Register	PIO_DIFSR	Write-Only	–
0x0088	Glitch or Debouncing Input Filter Clock Selection Status Register	PIO_IFDGSR	Read-Only	0x00000000
0x008C	Slow Clock Divider Debouncing Register	PIO_SCDR	Read-Write	0x00000000
0x0090 to 0x009C	Reserved			
0x00A0	Output Write Enable	PIO_OWER	Write-only	–
0x00A4	Output Write Disable	PIO_OWDR	Write-only	–
0x00A8	Output Write Status Register	PIO_OWSR	Read-only	0x00000000
0x00AC	Reserved			
0x00B0	Additional Interrupt Modes Enable Register	PIO_AIMER	Write-Only	–
0x00B4	Additional Interrupt Modes Disables Register	PIO_AIMDR	Write-Only	–
0x00B8	Additional Interrupt Modes Mask Register	PIO_AIMMR	Read-Only	0x00000000
0x00BC	Reserved			
0x00C0	Edge Select Register	PIO_ESR	Write-Only	–
0x00C4	Level Select Register	PIO_LSR	Write-Only	–
0x00C8	Edge/Level Status Register	PIO_ELSR	Read-Only	0x00000000
0x00CC	Reserved			
0x00D0	Falling Edge/Low Level Select Register	PIO_FELLSR	Write-Only	–
0x00D4	Rising Edge/ High Level Select Register	PIO_REHLSR	Write-Only	–
0x00D8	Fall/Rise - Low/High Status Register	PIO_FRLHSR	Read-Only	0x00000000
0x00DC	Reserved			
0x00E0	Lock Status	PIO_LOCKSR	Read-Only	0x00000000
0x00E4	Write Protect Mode Register	PIO_WPMR	Read-write	0x0
0x00E8	Write Protect Status Register	PIO_WPSR	Read-only	0x0
0x00EC to 0x00F8	Reserved			
0x0100 to 0x0144	Reserved			

Notes:
1. Reset value of PIO_PSR depends on the product implementation.
2. PIO_ODSR is Read-only or Read/Write depending on PIO_OWSR I/O lines.
3. Reset value of PIO_PDSR depends on the level of the I/O lines. Reading the I/O line levels requires the clock of the PIO Controller to be enabled, otherwise PIO_PDSR reads the levels present on the I/O line at the time the clock was disabled.
4. PIO_ISR is reset at 0x0. However, the first read of the register may read a different value as input changes may have occurred.
5. Only this set of registers clears the status by writing 1 in the first register and sets the status by writing 1 in the second register.