

initmaker by Bob Alkire June 20, 2018 rev 0.1

Scripted program that creates initialization files for the Atmel/Microchip ATSAMD5x processor.

This is for developing standalone C applications on the ATSAMD5x processor.

A standalone C application requires startup code provided by Microchip/Atmel in ASF4 as `startup_samd51.c`. This file has interrupt vector tables, `.bss` and `.data` initialization, FPU setup and makes calls `SystemInit()` and then it calls `main()`. `SystemInit()` can be used to initialize the peripherals. The SAMD51 has a rich but very complex set of peripherals. Microchips answer to this was `start.atmel.com`. My answer was `initmaker`. `Initmaker` requires `bash` shell and `gnu awk`. It has been tested to work under `Cygwin` and `Linux`.

It generates code to initialize and define GPIO, Timer/Counter, SERCOM (UART, SPI, I2C), Clocking, TC/TCC, events, DMA, ADC and Interrupts. It does not do the more complex peripherals which have their own initialization and there are sections I just haven't gotten to.

`Initmaker` uses macro substitution of template files directed by a configuration file to generate a `SystemInit()` procedure in a `.c` and `.h` file. `SystemInit()` is called by the startup code prior to calling `main`. The configuration file, `<filename>.cfg`, uses standard `.ini` style syntax to define sections or peripherals, and key/value properties to describe the peripherals. It uses template files which are mix of C and a non-C macro language.

This generates c source and h include files if they aren't already there or it modifies the existing files. It uses doxygen group tags and ONLY replaces what is between those tags. There is a tag for external code to be placed inline with a command line directive. While it is possible to alter the code outside the tags, and update it, the extended file mechanism insures the code is not lost.

Chip Libraries

`Initmaker` relies on my own chip libraries. These are corrected files from the ASF4 includes, includes/components, includes/instances and includes/pio directories. I also translated the hri includes to simplify and clean up the code. The `readme.txt` file with `initmaker` has details of what was changed.

The files that have a beta release have been tested, the ones with alpha have been translated and not tested.

Installation

Since it is a script file it can go anywhere; however, having a path name to it is a good idea.

`initmaker.sh` is the main script which calls all of the others. The other scripts cannot easily be run standalone as the sequence of scripts is important and variables are set up in `initmaker.sh`. I usually use an environment variable, say `SAMD51` that points to the top and have the makefile work relative to that variable.

Execution

Usage: scripts/initmaker.sh <config file> <c source file> <h include file> {options}

options:

–v or –verbose for verbose progress output.

–x or –extended <filename> to include text inside of the SystemInit() procedure, for additional initialization outside of the initmaker scope.

File	Filetype
<config file>	.cfg
<source file>	.c
<include file>	.h
<extended file>	.c, .h

Blinky Example

See the readme.txt in the example directory for how the example is built.

Extended file

This is a C file that is straight inline c code. The code of this file is copied to the region between the doxygen ExtendedInit group tag (just before NVIC code). There are examples in the

Config file

The config file syntax is similar to the old style INI file format. I used this instead of XML because, well XML is a step in the wrong direction.

This has sections which identify peripherals and key/value pairs, one per line, for the peripheral properties.

The section is [<section>] and the key value pairs are <key> = <value>

<section> identifies not only sections such as BOARD but the peripherals. The peripheral naming comes from the datasheet names such as NVMCTRL or MCLK. If there are more than one instance of a peripheral then the name is followed by a unit number such as SERCOM0 , GLCK2 and so on. Sections are not case-sensitive.

Table 1 Section List

BOARD
NVMCTRL
XOSC32K
GPIO
PINS

EIC
OSCULP32K
SYSTICK
TC0 – TCn
TCC0 – TCCn
XOSC0 – XOSC1
GCLK0 – GCLK11
DFLL
DPLL0 – DPLL1
DMA0 – DMA31

Properties that follow the section belong to that section until end of file or a new section is declared.

Properties are categorized into three parts: properties that are required and must be present in the config file for that section, properties that are optional and properties that are derived and should not be used in the config file. Derived properties are generated from scripts automatically.

Most properties are used for straight substitution in the template file. New properties can be added to the template file and used in the config file without changes to the script files. There are modifications that can be made, for example, making it upper case with toupper() function in the template. (note sections cannot be added without modifying the script file).

How it works

initmaker reads the configuration file and creates an internal dictionary of properties. It then reads in one or more template files and does replacement of named tags found in that dictionary and stores it into a temporary file. It then reads the generated code files, finds the appropriate doxygen tags to remove the old code and replace it with the new. There is a script and templates for each category of components.

The scripts also add conditional inclusion or exclusion of lines of text, looping over lists and basic functions to assist in the conversion. Some properties are generated within the scripts derived from other properties. For example, the reference clock frequency of a DPLL requires a lookup and calculation from the reference source. If the reference source is GCLK, the clock frequency is the input frequency of the GCLK divided by its divisor. A function will convert the clock as an integer in Hz to MHz or KHz for better readability. The macro language has to use syntax that is not confused with normal C syntax albeit a bit weird.

The awk scripts are about as large as I am comfortable making them and so the macro language is terse. There will be more scripts and templates as I get sections running. The more complex peripherals like Ethernet, USB, CAN and SDHC will not get scripts as they tend to have more complex initialization.

Macro language

Macro are parsed by initmaker and not passed as text to the output file. They occupy one line of text and must start at the beginning of the line. The remaining text on a macro line after parsing is discarded. It can have only one macro statement per line and no translated code.

Basic macro replacement – The replacement tag is property surrounded by ‘%’

`%<property>|<function>(<property>)%`

eg. out is a property of gpio in the config file which if assigned to a signal name.

[GPIO]

out=REDLED

So in the template file `gpio_set_dir_out(%out%);`, will generate `gpio_set_dir_out(REDLED);`

The list of functions that can be used inside of replacement tags are:

frequency - convert integer in hertz to MHz or KHz units

groupof – extract the group letter from port name ie PA01 -> A

designatorof – extract the group name from an instance name SERCOM3 -> SERCOM

unitof - extract designator from instance name SERCOM3 -> 3, or port name PB31 -> 31

toupper – convert to upper case

Define a macro by <name>. A template file can have multiple macros and this identifies the start and end of the macro. The script must recognize the macro name otherwise it is ignored.

`#defmacro <name>`

`#endmacro`

Conditionals

Conditionals are <macro command> <expression>

<expression> ::= <factor>|<expression><op><factor>

<factor> ::= <integer> | <property> | <string> | '(' <expression> ')'

<op> is:

& - boolean and,

| - boolean or

== equality (and only operator that works on a string)

! - boolean not.

parenthesis can be used to control precedence.

conditional if. If the <expression> evaluates true, then pass the statements to #fi to the output otherwise discard them.

#iftrue <expression>

<statements>

#fi

There is also a test whether a property exist.

#ifdefined <property>

<statements>

#fi

Or if it the property doesn't exist

#ifundefined <property>

<statements>

#fi

conditional if with alternate. If the <expression> evaluate true, pass <statements A> else if the <expression> is false pass <statements B>.

#iftrue <expression>

<statements A>

#otherwise

<statements B>

#fi

loop construct. Repeat passing <statements> to output while doing macro replacement on the <statements> for %key% and %value%. The <list> is created by the script as a key value pair and assigned a name. Each pair is updated after each iteration. The loop can be read as a property to test if the list is defined.

```
#foreach <list>
<statements>
#endfor
```

#isr, #var and #evt macro tag

These macro tags allow the individual scripts to build up a database for interrupts, variables and events. These macros cannot be used within the <<< >>> brackets.

The #isr is interrupt information from each macro and is used as a list by NVIC macro. The format for the line is:

```
#isr <description> <NVIC number> | NA <handler name>
```

example

```
#isr SERCOM%unit%_UART_DRE SERCOM%unit%_0_IRQn SERCOM%unit%_0_Handler
```

#var macro tag

The #var collects variable information from each macro and is injected after the void SystemInit() declaration.

```
#var <c variable declaration>
```

example

```
#var int i;
```

#evt macro tag

The #evt macro tag defines properties needed for the evsys genevent.sh macro. The format for generator:

```
#evt gen <generator name> <generator source> <path> <edge> <sync_source>
```

for event

```
#evt event <event name> <user>
```

for software generator

```
#evt swgen <event name>
```

Debugging

When running config, you will get error message that the macro translator finds. There is room for improvement here.

Because it is dumb macro expansion, it will not catch a lot of errors. Some show up as error in compile, such as required properties that are missing. It's also easy to forget a section. If the section name is mistyped it will ignore all the properties and will not generate it section. Check the initialization code to insure all the parts are in there that you need.

I have had code wedge the processor especially misguided CPU clocking. I've recovered by using the DSU erase chip feature. From JLink commander -> w1 0x41002000, 0x10.

Section and Properties

Section BOARD

Properties

Required:

processor=<processor> processor name as defined in datasheet, required
<processor> is ATSAM<product series><pin count><flash density><device variant>
ie ATSAM51J20A

Optional:

project=<text> project name line
description=<text> description of project line
copyright=<text> copyright line
author=<text> author name line
legal=<filename> filename of license in template directory
Derived:
filename derived from source or include filename>
date derived from current computer date
includefile <derived> filename of include file
tag <derived> include filename to uppercase and '.' converted '_'

Example

[board]

```
project=foobar1
description=Board file for foobar LED flashing
processor=ATSAMD51J20A
author=Arthur T. Fischell
copyright=Copyright © 2018, Art. T. Fischell Industries
license=bsd3.txt
```

Section NVMCTRL

Properties

Optional

wait_states=<integer> number of wait states default: 0

Example

[NVMCTRL]

wait_states=0

Section XOSC32K

Properties

required:

out_frequency=<integer> frequency in Hz

en32k=1 enable 32KHz output select en32k or en1k

en1k=1 enable 1KHz output select en32k or en1k

optional

hs=1 select highspeed crystal default: standard crystal

Example

[XOSC32K]

out_frequency=32768

en32K=1

Section OSCULP32K

Properties

None

Example

[OSCULP32K]

Section XOSCn

Properties

required

out_frequency=<integer> frequency in Hz

optional

runstdby=1 oscillator runs in standby mode

ondemand=1 oscillator only runs when downstream devices requires it

Example

[XOSC0]

out_frequency=12000000

Section DPLLn

Properties

required

ref_source=XOSCn | XOSC32K | GCLKn must be between 32K and 3.2MHz

out_frequency=<integer> output frequency in Hz 96MHz to 200MHz

optional

runstdby=1 oscillator runs in standby mode

ondemand=1 oscillator only runs when downstream devices requires it

integeronly=1 force the DPLL divider to not use any fractional component. This is much less jitter but not accurate frequency.

Example

;Enables DPLL0 with output frequency of 120MHz and 12MHz external oscillator XOSC0 divided by 6 (2MHz input reference)

[DPLL0]

ref_source=xosc0

out_frequency=120000000

div=6

integeronly=1

Section PINS

Properties

<pin alias>=<port>

<pin alias> is an identifier (begins with letters, and can contain letters, numbers and underscore)

This can be used in place of the <port name> to better describe the board usage of the pin.

<port> is an identifier which starts with 'P', and a group letter A, B, C, D... followed by the pin number 00 to 31 (decimal with leading zeroes). ie PA00. Depending on the package, these identify the connection pin to the microcontroller.

This will generate a macro in the include file of the <pin alias> to a number representing the <port> used by the driver. The <pin alias> is used by the other peripherals to map to the port and can have more descriptive names than portnames.

Example

[PINS]

XIN=PA14

XOUT=PA15

LEDRED=PA07

Section GPIO

GPIO must have one of out, in, pin or eic property to describe the GPIO.

Properties

required

out=<pin alias> define a <pin alias> as a GPIO output pin.

in=<pin alias> define a <pin alias> as a GPIO input pin

pin=<pin alias> define a <pin alias> to be muxed to an alternate function and not GPIO.

eic=<pin alias> describes a pin as an external input. EIC section must be defined.

function=<function> must be defined with pin and describes alternate pin function. see alternate function table.

optional

interrupt=1 used with eic, enables NVIC interrupt (not eic interrupt) for eic

generator=<identifier> used with eic, enables an event generator to channel <identifier>

path=asynchronous|synchronous|resynchronized event synchronization path

edge=falling|rising|both|none set event edge detection for synchronous and resynchronized

sync_source=<clock source> clock source for event generator if synchronous or resynchronized

sense= falling | rising | high | low | both. Must be described with eic pin description

debounce=1 for eic property (optional)

initial=1|0 for out property, defines whether the GPIO is initialized high or low (optional)

drvstr=<pin alias> if defined sets output as high current (optional)

pullup=<pin alias> for in and out properties, if defined causes pullup to be enabled (optional)

pulldown=<pin alias> for in and out properties, if defined causes pulldown to be enabled (optional)

event<0-3>=<identifier> used with out property, enables as an event to channel <identifier>

evact<0-3>=out|set|clr|tgl for event out property. sets the output pin action

alternate functions

AC	PTC	TC1
ADC0	QSPI	TC2
ADC1	SDHC0	TC3
CAN0	SDHC1	TC4
CCL	SERCOM0	TC5
CM4	SERCOM1	TC6
EIC	SERCOM2	TC7
GCLK	SERCOM3	TCC0
GMAC	SERCOM4	TCC1
I2S	SERCOM5	TCC2
PCC	SERCOM7	USB
PDEC	TC0	VREF

Section EIC

properties

required

ref_source=<clock source> <clock source>(GCLKn) for EIC peripheral.

Must be used if any of the GPIO pins are described as external interrupts, etc.

example

[EIC]

ref_source=GCLK4

[GPIO]

eic=BUTTON

sense=falling

debounce=1

event example

[EIC]

ref_source=GCLK4

[GPIO]

eic=BUTTON

sense=falling

debounce=1

generator=BUTTONEVENT

edge=falling

Section SYSTICK

Properties

required

period=<integer> configure systick with <integer> millisecond period

Example

[SYSTICK]

period=10

Section TCn

Properties

required

ref_source=<clock source> clock source for timer counter GCLKn

mode=32 | 16 | 8 number of bits for counter

prescaler=<integer> 1,2,4,8,16,64,256, or 1024

presync=GCLK|PRESC|RESYNC Prescaler and Counter Synchronization

wavegen=NFRQ | MFRQ | NPWM | MPWM

optional

count=<integer> initial count

cc0=<integer> match compare value 0

cc1=<integer> match compare value 1

swgen=<identifier> Timer input event is software generated only. <identifier> matches event <identifier>

oneshot=1 if defined, sets one shot mode

Note: if 32bit counter, TCn is paired even/odd ie TC0 and TC1

event=<identifier> event channel name

evact= off|retrigger|count|start|stamp|ppw|pwp|pw input event action on timer counter

tcinv=1 event only, invert incoming event

gen_ovf=<identifier> event generator channel name

path_ovf=asynchronous|synchronous|resynchronized event synchronization path

edge_ovf=falling|rising|both|none set event edge detection for synchronous and resynchronized

sync_source_ovf=<clock source> clock source for event generator if synchronous or resynchronized

gen_mc0=<identifier> event generator channel name

path_mc0=asynchronous|synchronous|resynchronized event synchronization path

edge_mc0=falling|rising|both|none set event edge detection for synchronous and resynchronized

sync_source_mc0=<clock source> clock source for event generator if synchronous or resynchronized

gen_mc1=<identifier> event generator channel name

path_mc1=asynchronous|synchronous|resynchronized event synchronization path

edge_mc1=falling|rising|both|none set event edge detection for synchronous and resynchronized

sync_source_mc1=<clock source> clock source for event generator if synchronous or resynchronized

use_inten=1 if defined enables local interrupt in system init

int_ovf=1 enable OVF interrupt

int_err=1 enable ERR interrupt

int_mc0=1 enable MC0 interrupt

int_mc1=1 enable MC1 interrupt

Example

[TC0]

ref_source=gclk5

mode=32

prescaler=1

wavegen=NFRQ

count=0

Example with event

[TC0]

ref_source=gclk5

mode=32

prescaler=1

wavegen=MFRQ

count=0
cc0=500000
gen_ovf=TIMEOUT_EVENT
path_ovf=synchronous
edge_ovf=both
sync_source_ovf=GCLK5

Section TCCn

Properties

required

ref_source=<clock source> clock source for timer counter GCLKn

wavegen=nfrq|mfrq|npwm|dscritical|dsbottom|dsboth|dstop

optional

faulta_src=disable|enable|invert|altfault

faulta_filtval=<integer> 0-15

faulta_blankval=<integer> 0-255

faulta_keep=1

faulta_qual=1

faulta_restart=1

faulta_blankpresc=1

faulta_halt= disable|hw|sw|nr

faulta_chsel=<integer> 0-3

faulta_blank=start|rise|fall|both

faulta_capture=disable|capt|captmin|captmax|locmin|locmax|deriv0|captmark

nre<0-7>=1

nrv<0-7>=1

inven<0-7>=1

filterval0=<integer> 0-15

filterval1=<integer> 0-15

fddbd=1

dbgrun=1

otmx=<integer> 0-3 see table 49-4

dtien<0-3>=1 Dead-time Insertion Generator x Enable

dtls=<integer> low side dead time 0-255

dths=<integer> high side dead time 0-255

ramp=ramp1|ramp2|ramp2a|ramp2c

ciccen<0-3>=1

pol<0-5>=1

swap<0-3>=1

prescaler=<integer> 1,2,4,8,16,64,256, or 1024

prescsync=GCLK|PRESC|RESYNC Prescaler and Counter Synchronization

count=<integer> initial count

cc0=<integer> match compare value 0

cc1=<integer> match compare value 1

cc2=<integer> match compare value 2

cc3=<integer> match compare value 3

cc4=<integer> match compare value 4

cc5=<integer> match compare value 5

swgen=<identifier> Timer input event is software generated only. <identifier> matches event <identifier>
 oneshot=1 if defined, sets one shot mode
 event0=<identifier> event channel name
 evact0= off|retrigger|countev|start|inc|count|stamp|fault
 event1=<identifier>
 evact1=off|retrigger|dir|stop|dec|ppw|pwp|fault
 cntsel=begin|end|between|boundary
 tcinv<0-1>=1
 event_mc<0-5>=<identifier>
 gen_mc<0-5>=<identifier> event generator channel name
 path_mc<0-5>=asynchronous|synchronous|resynchronized event synchronization path
 edge_mc<0-5>=falling|rising|both|none set event edge detection for synchronous and resynchronized
 sync_source_mc<0-5>=<clock source> clock source for event generator if synchronous or resynchronized
 gen_ovf=<identifier> event generator channel name
 path_ovf=asynchronous|synchronous|resynchronized event synchronization path
 edge_ovf=falling|rising|both|none set event edge detection for synchronous and resynchronized
 sync_source_ovf=<clock source> clock source for event generator if synchronous or resynchronized
 gen_trg=<identifier> event generator channel name
 path_trg=asynchronous|synchronous|resynchronized event synchronization path
 edge_trg=falling|rising|both|none set event edge detection for synchronous and resynchronized
 sync_source_trg=<clock source> clock source for event generator if synchronous or resynchronized
 gen_cnt=<identifier> event generator channel name
 path_cnt=asynchronous|synchronous|resynchronized event synchronization path
 edge_cnt=falling|rising|both|none set event edge detection for synchronous and resynchronized
 sync_source_cnt=<clock source> clock source for event generator if synchronous or resynchronized
 use_inten=1 if defined enables local interrupt in system init
 int_ovf=1 enable ovf interrupt
 int_trg=1 enable trg interrupt
 int_cnt=1 enable cnt interrupt
 int_err=1 enable err interrupt
 int_ufs=1 enable ufs interrupt
 int_dfs=1 enable dfs interrupt
 int_faulta=1 enable faulta interrupt
 int_faultb=1 enable faultb interrupt
 int_fault0=1 enable fault0 interrupt
 int_fault1=1 enable fault1 interrupt
 int_mc<0-5>=1 enable mc0 to mc5 interrupt

Example

[TCC0]

ref_source=gclk5
 prescaler=1
 wavegen=NFRQ

Example with event

[TC0]

ref_source=gclk5
 mode=32
 prescaler=1


```
wavegen=MFRQ
count=0
cc0=500000
generator=TIMEOUT_EVENT
path=synchronous
edge=both
sync_source=GCLK5
gen_source=mc0
```

Section GCLKn

Properties

required

ref_source=<clock source> <clock source> is XOSC0, XOSC1, DPLL0, DPLL1, XOSC32K, OSCULP32K

ref_frequency=<integer> derived frequency of ref_source

div=<integer> <clock source> is divided by <integer>

optional

out=<pin alias> GCLK output is muxed to external pin <pin alias>

in=<pin alias> GCLK input is muxed to external pin <pin alias>

ext_frequency=<integer> input frequency in Hz

idc=1 if defined, tries to get 50% duty cycle

oov=0|1 state of out <pin alias>

runstdby=1 if defined, GCLK runs while in standby mode

Example

[GCLK6]

ref_source=DPLL0

div=10

out=CLKOUT

oov=0

Section DMAn

Properties

required:

source=<peripheral> peripheral requesting DMA, event, software or disable.

channel=<channel> channel of the peripheral requesting DMA, this isn't required for event, software or disable

trigact= BURST | BLOCK | TRANSACTION

burstlen=<integer>

threshold=<integer>

optional:

event=<identifier> event channel name

evact= noact|trig|ctrig|cblock|suspend|resume|sskip|incpri input event action on DMA

generator=<identifier> event generator channel name

path=asynchronous|synchronous|resynchronized event synchronization path

edge=falling|rising|both|none set event edge detection for synchronous and resynchronized

sync_source=<clock source> clock source for event generator if synchronous or resynchronized

gen_source= disable|block|beat|trigact generator event source from DMA

derived:

trigsrc=<source>_<channel> derived from source and channel

Note if source is software, event or disable, then channel is not required

For example, source can be sercom0 and channel is rx, then the derived trigsrc is SERCOM0_RX

Source_Channel

DISABLE	SERCOM5_RX	TCC1_MC0	TC0_MC1	TC6_OVF	I2S_TX0
SOFTWARE	SERCOM5_TX	TCC1_MC1	TC1_OVF	TC6_MC0	I2S_TX1
RTC_TIMESTAMP	SERCOM6_RX	TCC1_MC2	TC1_MC0	TC6_MC1	PCC_RX
DSU_DCC0	SERCOM6_TX	TCC1_MC3	TC1_MC1	TC7_OVF	AES_WR
DSU_DCC1	SERCOM7_RX	TCC2_OVF	TC2_OVF	TC7_MC0	AES_RD
SERCOM0_RX	SERCOM7_TX	TCC2_MC0	TC2_MC0	TC7_MC1	QSPI_RX
SERCOM0_TX	CAN0_DEBUG	TCC2_MC1	TC2_MC1	ADC0_RESRDY	QSPI_TX
SERCOM1_RX	CAN1_DEBUG	TCC2_MC2	TC3_OVF	ADC0_SEQ	
SERCOM1_TX	TCC0_OVF	TCC3_OVF	TC3_MC0	ADC1_RESRDY	
SERCOM2_RX	TCC0_MC0	TCC3_MC0	TC3_MC1	ADC1_SEQ	
SERCOM2_TX	TCC0_MC1	TCC3_MC1	TC4_OVF	DAC_EMPTY0	
SERCOM3_RX	TCC0_MC2	TCC4_OVF	TC4_MC0	DAC_EMPTY1	
SERCOM3_TX	TCC0_MC3	TCC4_MC0	TC4_MC1	DAC_RESRDY0	
SERCOM4_RX	TCC0_MC4	TCC4_MC1	TC5_OVF	DAC_RESRDY1	
SERCOM4_TX	TCC0_MC5	TC0_OVF	TC5_MC0	I2S_RX0	
SERCOM5_RX	TCC1_OVF	TC0_MC0	TC5_MC1	I2S_RX1	

example

[DMA0]

source=sercom3

channel=tx

action=burst

burstlen=1

threshold=1

note that trigsrc is SERCOM3_TX

Section SERCOM

SERCOM can be UART, USART, SPI Master, SPI Slave, I²C Master, or , I²C Slave. The SERCOM mode of operation is selected by the type property. USART, SPI and , I²C Slave are not implemented yet. Note that only NVIC interrupts can be enabled. You must supply the local interrupt enable and interrupt service routine elsewhere.

UART Properties

required

type=uart select SERCOM type as uart.

baudrate=<integer> baudrate

ref_source=<clock source> gclk clocksource maximum 100MHz

slow_source=<clock source> gclk clocksource maximum 12MHz

txd=<pin name> transmit data pin name (from pin section)

rxid=<pin name> receive data pin name (from pin section)

rts=<pin name> request to send pin name (from pin section)

cts=<pin name> clear to send pin name (from pin section)

optional

sampr=<integer> set sample rate (from datasheet) default: 16X oversample

sampa=<integer> set sample adjustment (from datasheet) default:7-8-9

form=<integer> set form (from datasheet) default: usart frame (unused for uart)

ibon=1 immediate buffer overflow notification

rxinv=1 receive input invert

txinv=1 transmit output invert

runstandby=1 run in standby

msbfirst=1 MSB is shifted out first if defined, otherwise LSB is shifted out first

size=<integer> character size in bits (5 to 9 bits)

dre_irq=1 enables NVIC data register empty interrupt

txc_irq=1 enables NVIC transmit complete interrupt

rxrc_irq=1 enables NVIC receive complete interrupt

err_irq=1 enables NVIC error, receive break, clear to send input change and receive start interrupt

derived

chsize is derived from charsize

apb is derived from SERCOM unit to select to correct clock source

unit is derived from SERCOM section number

rxpo is derived from pin assignments

txpo is derived from pin assignments

SPI Master Properties

required

type=spim select SERCOM type as SPI Master.

baudrate=<integer> baudrate in bits per second

ref_source=<clock source> gclk clocksource maximum 100MHz

slow_source=<clock source> gclk clocksource maximum 12MHz

miso=<pin name> transmit data pin name (from pin section)

sck=<pin name> request to send pin name (from pin section)

optional

cs=<pin name> use hardware generated chip select at pin name (from pin selection)

mosi=<pin name> receive data pin name (from pin section)

cpol=1 set clock polarity to high when idle, For 0 or undefined, clock is low when idle

cpha=1 set clock phase sample rising edge, change falling edge, For 0 or undefined clock is sample falling edge, change rising
form=<integer> set form (from datasheet) default: SPI Frame
runstdby=1 run in standby
ibon=1 immediate buffer overflow notification
dord=1 MSB is shifted out first if defined, otherwise LSB is shifted out first
len=<integer> set 32 bit mode with length bytes
icspace=<integer> set intercommunication spacing to icspace bit times
chsize=<integer> character size in bits (8 or 9 bits, 8 bit default)
dre_irq=1 enables NVIC data register empty interrupt
txc_irq=1 enables NVIC transmit complete interrupt
rxr_irq=1 enables NVIC receive complete interrupt
err_irq=1 enables NVIC error, receive break, clear to send input change and receive start interrupt
derived
chsize is derived from size
apb is derived from SERCOM unit to select to correct clock source
unit is derived from SERCOM section number
dipo is derived from pin assignments
dopo is derived from pin assignments

I2C Master Properties

required
type=i2cm select SERCOM type as I2C Master.
baudrate=<integer> baudrate in bits per second
ref_source=<clock source> gclk clocksource maximum 100MHz
slow_source=<clock source> gclk clocksource maximum 12MHz
sda=<pin name> transmit data pin name (from pin section)
scl=<pin name> receive data pin name (from pin section)
optional
runstandby=1 run in standby
mb_irq=1 enables NVIC master on bus interrupt
sb_irq=1 enables NVIC slave on bus interrupt
err_irq=1 enables NVIC error interrupt
derived
apb is derived from SERCOM unit to select to correct clock source
unit is derived from SERCOM section number
sda_port, sda_pad, sda_mux derived from <pin name> port
scl_port, scl_pad, scl_mux derived from <pin name> port

Section DFLL

Properties

required

ref_source=<clock source>

out_frequency=<integer> frequency in Hz (DFLL is 48000000)

cstep=<integer> course step

fstep=<integer> fine step

optional

mode=1 if in closed loop mode only, otherwise it is openloop

waitlock=1 wait for lock before output clock

stable=1 calibration register value will be fixed after fine lock

llaw=1 lose lock after wake

usbcrm=1 USB clock recovery mode

ccdis=1 chill cycle disable

qldis=1 quick lock disable

bpckc=1 bypass course lock

ondemand=1 dfll only runs when peripheral requires clock

runstdby=1 run in standby

course=<integer>

fine=<integer>

Example

[DFLL]

ref_source=gclk3

out_frequency=48000000

cstep=10

fstep=10

course=7

fine=128

mode=1

Section ADC

Section is [ADC0] or [ADC1]

Properties

required

ref_source=<clock source>

leftadj=1 left adjust if one, right adjust if undefined

freerun=1 free running if one, one shot if undefined

corren=1 gain and offset correction enabled if one

ressel=<integer> 8, 10, 12, 16 bit resolution

winmode=<integer> 0-4 See Winmode table

winss=1 window single sample

refsel=intref, intvcc0, intvcc1, arefa, arefb or arefc reference selection

refcomp=1 offset compensation

flushai=1 flush and new conversion triggered on incoming event

startei=1 new conversion triggered on incoming event

flushinv=1 flush event input source is inverted

startinv=1 start conversion even input inverted

resrdy=1 event occurs when result ready

winmoneo=1 event occurs from window monitor

muxpos=ain[0-23], scaledcorevcc, scaledvbat, scalediovcc, bandgap, ptat, ctat, DAC select positive input source

muxneg=ain[0-7], gnd select negative input source

diffmode=1 differential mode is enabled

dseqstop=1 stops DMA sequence

samplenum=<integer> 1,2,4,8,16,32,64, 128, 256, 512 or 1024 how many samples averaged

adjres=<integer> division coefficient for sample average 2^n

samplen=<integer> sampling time length, sampling time = (samplen+1)*CLKadc

offcomp=1 offset compensation enable

winut=<integer> window monitor upper threshold

winlt=<integer> window monitor lower threshold

gaincorr=<integer> gain correction value

offsetcorr=<integer> offset correction value

dbgrun=1 adc continues to run after debugger halt

dualsel=both (triggers both adcs) or interleave (alternately triggers adcs)

slaveen=1 enables adc1 as slave to adc0

runstdby=1 runs when in sleep mode

ondemand=1 adc only runs when requested by a peripheral

prescaler=2,4,8,16,32,64,128,256 clock divider

r2r=1 rail to rail operation enabled, only in differential mode

overrun_irq=1 use overrun interrupt

winmon_irq=1 use window monitor interrupt

resrdy_irq=1 use result ready interrupt

Winmode

0 – disabled

1 – Result > WINLT

2 – Result < WINUT

3 – WINLT < Result < WINUT

4 – !(WINLT < Result < WINUT)

Example

[ADC0]

ref_source=gclk5

ressel=12

refsel=intvcc1

muxpos=ain2

muxneg=gnd